django-elasticsearch-dsl-drf Documentation

Release 0.2.6

Artur Barseghyan <artur.barseghyan@gmail.com>

Jul 11, 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 document models.

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`, `ids`, `endswith`, `contains`, `wildcard`, `exists`, `exclude`, `isnull`, `range`, `in`, `term` and `terms` is implemented.
- *Faceted search filter backend*.
- *Pagination (Page number and limit/offset pagination)*.

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
```

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

Documentation

Contents:

Table of Contents

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

Installation

1. Install latest stable version from PyPI:

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

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

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

    # Django Elasticsearch integration
    'django_elasticsearch_dsl',

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

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

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

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

Example app

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

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

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

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

Sample models

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

Required imports

Imports required for model definition.

```
import json

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

from six import python_2_unicode_compatible
```

Book statuses

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

```
    (BOOK_PUBLISHING_STATUS_CANCELLED, "Cancelled"),
    (BOOK_PUBLISHING_STATUS_REJECTED, "Rejected"),
)
BOOK_PUBLISHING_STATUS_DEFAULT = BOOK_PUBLISHING_STATUS_PUBLISHED
```

Publisher model

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

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

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

        ordering = ["id"]

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

Author model

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

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

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

        ordering = ["id"]

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

Tag model

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

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

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

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

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

Book model

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

Admin classes

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

```
from django.contrib import admin

from .models import *


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

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


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

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


@admin.register(Publisher)
class PublisherAdmin(admin.ModelAdmin):
    """Publisher admin."""

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


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

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

Create database tables

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

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

Fill in some data

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

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

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

Sample document

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

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

Simply said, Document in Elasticsearch is similar to Model in Django.

Often, complicated SQL model structures are flatterned in Elasticsearch indexes. Complicated 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.

Required imports

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

from books.models import Book
```

Index definition

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

Custom analyzers

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

Document definition

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

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

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

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

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

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

    publication_date = fields.DateField()
```

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

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

    price = fields.FloatField()

    pages = fields.IntegerField()

    stock_count = fields.IntegerField()

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

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

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

Syncing Django's database with Elasticsearch indexes

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

Full database sync

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

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

1. Create Elasticsearch indexes:

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

2. Sync the data:

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

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

Sample partial sync (using custom signals)

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

Required imports

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

from django_elasticsearch_dsl.registries import registry
```

Update book index on related model change

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

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

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

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

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

Update book index on related model removal

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

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

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

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

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

Sample serializer

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

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

Required imports

```
import json

from rest_framework import serializers
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from .documents import BookDocument
```

Serializer definition

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

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

    tags = serializers.SerializerMethodField()

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

        # Specify the correspondent document class
        document = BookDocument

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

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

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

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

    id = serializers.IntegerField(read_only=True)

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

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

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

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

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

ViewSet definition

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

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

Required imports

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

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

ViewSet definition

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

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

```
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'stock_count': {
            'field': 'stock_count',
            # Note, that we limit the lookups of `stock_count` field in
            # this example, to `range`, `gt`, `gte`, `lt` and `lte`
            # filters.
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_GT,
                LOOKUP_QUERY_GTE,
                LOOKUP_QUERY_LT,
                LOOKUP_QUERY_LTE,
            ],
        },
        'tags': {
            'field': 'tags',
            # Note, that we limit the lookups of `tags` field in
            # this example, to `terms, `prefix`, `wildcard`, `in` and
            # `exclude` filters.
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
        'tags.raw': {
            'field': 'tags.raw',
            # Note, that we limit the lookups of `tags.raw` field in
            # this example, to `terms, `prefix`, `wildcard`, `in` and
            # `exclude` filters.
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
    }
    # Define ordering fields
    ordering_fields = {
        'id': 'id',
        'title': 'title.raw',
        'price': 'price.raw',
        'state': 'state.raw',
        'publication_date': 'publication_date',
    }
    # Specify default ordering
    ordering = ('id', 'title', 'price',)
```

URLs

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

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

Required imports

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

from .views import BookDocumentView
```

Router definition

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

URL patterns

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

Check what you've done so far

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

URLs

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

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

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

Test in browser

Open the following URL in your browser.

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

Perform a number of lookups:

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

Development and debugging

Looking for profiling tools for Elasticsearch?

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

Installation

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

Configuration

Change your development settings in the following way:

```
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',
)
```

Filter usage examples

Example usage of filtering backends.

Contents:

Table of Contents

Search

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

Multiple search terms are joined with `OR`.

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

Search in all fields

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

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

Search a single term on specific field

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

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

Search for multiple terms

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

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

Search for multiple terms in specific fields

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

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

Filtering

Supported lookups

Native

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

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

term

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

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

terms

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

range

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

exists

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

prefix

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

wildcard

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

ids

Find documents with the specified type and IDs.

Functional

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

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

contains

Case-insensitive containment test.

in

In a given list.

gt

Greater than.

gte

Greater than or equal to.

lt

Less than.

lte

Less than or equal to.

startswith

Case-sensitive starts-with.

endswith

Case-sensitive ends-with.

isnull

Takes either True or False.

exclude

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

Usage examples

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

- models
- documents
- serializers
- views

Additionally, see:

- Basic usage examples
- Advanced usage examples
- Misc usage examples

Basic usage examples

Basic Django REST framework integration example

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

- models
- documents
- serializers
- views

Contents:

Table of Contents

Example app

Sample models

books/models.py:

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

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

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

        ordering = ["id"]

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

Sample document

search_indexes/documents/publisher.py:

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

from books.models import Publisher

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

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

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

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

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

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

Sample serializer

search_indexes/serializers.py:

```
import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

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

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

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

Sample view

search_indexes/views.py:

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

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

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

    document = PublisherDocument
    serializer_class = PublisherDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
```

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

Usage example

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

Sample queries

Search

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

Multiple search terms are joined with `OR`.

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

Search in all fields

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

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

Search a single term on specific field

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

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

Search for multiple terms

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

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

Search for multiple terms in specific fields

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

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

Filtering

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

Multiple filter terms are joined with `AND`.

Filter documents by single field

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

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

Filter documents by multiple fields

Filter documents by `city` "Yerevan" and "Groningen".

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

Filter document by a single field

Filter documents by (field `country`) "Armenia".

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

Filter documents by multiple fields

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

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

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

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

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

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

Filter documents by a word part of a single field

Filter documents by a part word part in single field (`city`) "ondon".

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

Ordering

The − prefix means ordering should be descending.

Order documents by field (ascending)

Filter documents by field `city` (ascending).

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

Order documents by field (descending)

Filter documents by field `country` (descending).

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

Order documents by multiple fields

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

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

Advanced usage examples

Advanced Django REST framework integration examples.

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

- models
- documents
- serializers
- views

Contents:

Table of Contents

Example app

Sample models

books/models.py:

```
import json

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

from six import python_2_unicode_compatible

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


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

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

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

        ordering = ["id"]

    def __str__(self):
```

```
        return self.name


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

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

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

        ordering = ["id"]

    def __str__(self):
        return self.name


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

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

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

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

    def __str__(self):
        return self.title


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

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

    class Meta(object):
```

```
        """Meta options."""

        ordering = ["isbn"]

    def __str__(self):
        return self.title

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

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

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

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

Sample document

search_indexes/documents/book.py:

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

from books.models import Book

# Name of the Elasticsearch index
BOOK_INDEX = Index('book')
# See Elasticsearch Indices API reference for available settings
BOOK_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"]
)


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

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

    title = fields.StringField(
```

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

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

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

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

    publication_date = fields.DateField()

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

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

    price = fields.FloatField()
```

```
    pages = fields.IntegerField()

    stock_count = fields.IntegerField()

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

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

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

Sample serializer

search_indexes/serializers.py:

```
import json

from rest_framework import serializers

class TagSerializer(serializers.Serializer):
    """Helper serializer for the Tag field of the Book document."""

    title = serializers.CharField()

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

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


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

    id = serializers.SerializerMethodField()

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

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

```
    tags = serializers.SerializerMethodField()

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

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

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

Sample view

search_indexes/views.py:

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

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


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

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

```
        OrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'description',
        'summary',
    )
    # Define filtering fields
    filter_fields = {
        'id': {
            'field': '_id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
            ],
        },
        'publisher': 'publisher.raw',
        'publication_date': 'publication_date',
        'isbn': 'isbn.raw',
        'tags': {
            'field': 'tags',
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
        'tags.raw': {
            'field': 'tags.raw',
            'lookups': [
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
    }
    # Define ordering fields
    ordering_fields = {
        'id': 'id',
        'title': 'title.raw',
        'price': 'price.raw',
        'state': 'state.raw',
        'publication_date': 'publication_date',
    }
    # Specify default ordering
    ordering = ('id', 'title',)
```

Usage example

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

Sample queries

Search

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

Multiple search terms are joined with `OR`.

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

Search in all fields

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

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

Search a single term on specific field

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

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

Search for multiple terms

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

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

Search for multiple terms on specific fields

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

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

Filtering

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

Multiple filter terms are joined with `AND`.

Filter documents by field

Filter documents by field (`state`) "published".

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

Filter documents by multiple fields

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

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

Filter document by a single field

Filter documents by (field `tag`) "education".

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

Filter documents by multiple fields

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

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

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

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

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

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

Filter documents by a word part of a single field

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

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

Ordering

The - prefix means ordering should be descending.

Order documents by field (ascending)

Filter documents by field `price` (ascending).

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

Order documents by field (descending)

Filter documents by field `price` (descending).

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

Order documents by multiple fields

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

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

Faceted search

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

```
# ...

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
```

Pagination

Page number pagination

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

Example:

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

Limit/offset pagination

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

```
# ...

from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination

# ...

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

    # ...

    pagination_class = LimitOffsetPagination

    # ...
```

Example:

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

Various handy helpers

Contents:

Table of Contents

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']
)
```

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.

0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

0.2.5

2017-07-11

- Fixes in documentation.

0.2.4

2017-07-11

- Fixes in documentation.

0.2.3

2017-07-11

- Fixes in documentation.

0.2.2

2017-07-11

- Fixes in documentation.

0.2.1

2017-07-11

- Fixes in documentation.

0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

0.1.7

2017-06-25

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

0.1.6

2017-06-23

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

0.1.5

2017-06-22

- Implemented `endswith` and `contains` functional filters.
- Added tests for `wildcard`, `exists`, `exclude` and `isnull` filters. Improved `range` filter tests.
- Improve `more_like_this` helper test.
- Improve ordering tests.

- Two additional arguments added to the `more_like_this` helper: `min_doc_freq` and `max_doc_freq`.
- Minor documentation improvements.

0.1.4

2017-06-22

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

0.1.3

2017-06-21

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

0.1.2

2017-06-20

- Minor fixes in tests.

0.1.1

2017-06-20

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

0.1

2017-06-19

- Initial beta release.

django_elasticsearch_dsl_drf package

Subpackages

django_elasticsearch_dsl_drf.filter_backends package

Subpackages

django_elasticsearch_dsl_drf.filter_backends.filtering package

Submodules

django_elasticsearch_dsl_drf.filter_backends.filtering.common module

django_elasticsearch_dsl_drf.filter_backends.filtering.ids module

Module contents

Submodules

django_elasticsearch_dsl_drf.filter_backends.faceted_search module

django_elasticsearch_dsl_drf.filter_backends.mixins module

django_elasticsearch_dsl_drf.filter_backends.ordering module

django_elasticsearch_dsl_drf.filter_backends.search module

Module contents

django_elasticsearch_dsl_drf.tests package

Submodules

django_elasticsearch_dsl_drf.tests.base module

Base tests.

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

Bases: `django.test.testcases.TransactionTestCase`

Base REST framework test case.

`authenticate`()

Helper for logging in Genre Coordinator user.

Returns

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

classmethod setUpClass()
Set up class.

class django_elasticsearch_dsl_drf.tests.base.**BaseTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TransactionTestCase

Base test case.

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

classmethod setUpClass()
Set up class.

django_elasticsearch_dsl_drf.tests.test_faceted_search module

Test faceted search backend.

class django_elasticsearch_dsl_drf.tests.test_faceted_search.**TestFacetedSearch**(*methodName='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.

django_elasticsearch_dsl_drf.tests.test_filtering module

Test filtering backend.

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

Test filtering.

pytestmark = [<MarkDecorator 'django_db' {'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

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.

django_elasticsearch_dsl_drf.tests.test_ordering module

Test ordering backend.

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

Test ordering.

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

classmethod setUpClass()
Set up class.

test_order_by_field()
Order by field.

test_order_by_non_existent_field()
Order by non-existent field.

django_elasticsearch_dsl_drf.tests.test_pagination module

Test ordering backend.

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()
Order by field.

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.

django_elasticsearch_dsl_drf.tests.test_views module

Test views.

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

Test views.

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

classmethod setUpClass()
Set up class.

test_detail_view()
Test detail view.

test_listing_view()
Test listing view.

Module contents

Submodules

django_elasticsearch_dsl_drf.apps module

Apps.

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

Config.

label = 'django_elasticsearch_dsl_drf'

name = 'django_elasticsearch_dsl_drf'

django_elasticsearch_dsl_drf.constants module

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

django_elasticsearch_dsl_drf.helpers module

Helpers.

django_elasticsearch_dsl_drf.helpers.**get_document_for_model**(*model*)

Get document for model given.

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

Returns Document index for the given model.

Return type Subclass of *django_elasticsearch_dsl.DocType*.

django_elasticsearch_dsl_drf.helpers.**get_index_and_mapping_for_model**(*model*)

Get index and mapping for model.

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

Returns Index and mapping values.

Return type tuple.

django_elasticsearch_dsl_drf.helpers.**more_like_this**(*obj*, *fields*, *max_query_terms=25*, *min_term_freq=2*, *min_doc_freq=5*, *max_doc_freq=0*)

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*) –

Returns List of objects.

Return type elasticsearch_dsl.search.Search

Example:

```
>>> from django_elasticsearch_dsl_drf.helpers import more_like_this
>>> from books.models import Book
>>> book = Book.objects.first()
>>> similar_books = more_like_this(
```

```
>>>     book,
>>>     ['title', 'description', 'summary']
>>> )
```

django_elasticsearch_dsl_drf.helpers.**sort_by_list**(*unsorted_dict*, *sorted_keys*)

Sort an OrderedDict by list of sorted keys.

Parameters

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

Returns Sorted dictionary.

Return type collections.OrderedDict

django_elasticsearch_dsl_drf.pagination module

Pagination.

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

Bases: rest_framework.pagination.LimitOffsetPagination

A limit/offset pagination.

Example:

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

get_facets(*facets=None*)

Get facets.

Parameters **facets** –

Returns

get_paginated_response(*data*)

Get paginated response.

Parameters **data** –

Returns

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

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

Bases: django.core.paginator.Page

Page for Elasticsearch.

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

Bases: rest_framework.pagination.PageNumberPagination

Page number pagination.

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

Example:

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

django_paginator_class
alias of *Paginator*

get_facets(*page=None*)
Get facets.

Parameters **page** –

Returns

get_paginated_response(*data*)
Get paginated response.

Parameters **data** –

Returns

paginate_queryset(*queryset*, *request*, *view=None*)
Paginate a queryset.

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

Parameters

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

Returns

class django_elasticsearch_dsl_drf.pagination.**Paginator**(*object_list*, *per_page*, *orphans=0*, *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

django_elasticsearch_dsl_drf.serializers module

Serializers.

class django_elasticsearch_dsl_drf.serializers.**DocumentSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)
Bases: rest_framework.serializers.Serializer

A dynamic DocumentSerializer class.

create(*validated_data*)
Create.

Do nothing.

Parameters **validated_data** –

Returns

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

update(*instance*, *validated_data*)
Update.

Do nothing.

Parameters

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

Returns

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

Metaclass for the DocumentSerializer.

Ensures that all declared subclasses implemented a Meta.

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

Template for the DocumentSerializerMeta.Meta class.

exclude = ()

field_aliases = {}

field_options = {}

fields = ()

ignore_fields = ()

index_aliases = {}

index_classes = ()

search_fields = ()

serializers = ()

django_elasticsearch_dsl_drf.utils module

Utils.

class django_elasticsearch_dsl_drf.utils.**DictionaryProxy**(*mapping*)
Bases: object

Dictionary proxy.

class django_elasticsearch_dsl_drf.utils.**EmptySearch**(***kwargs*)
Bases: object

Empty Search.

django_elasticsearch_dsl_drf.views module

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 = u'id'

get_object()
Get object.

get_queryset()
Get queryset.

pagination_class
alias of PageNumberPagination

Module contents

Integrate Elasticsearch DSL with Django REST framework.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

U