
django-elasticsearch-dsl-drf Documentation

Release 0.20

Artur Barseghyan <artur.barseghyan@gmail.com>

Aug 16, 2019

Contents

1	Documentation	3
2	Prerequisites	5
3	Main features and highlights	7
4	Demo	9
5	Installation	11
6	Quick start	13
7	Testing	15
8	Writing documentation	17
9	License	19
10	Support	21
11	Author	23
12	Project documentation	25
12.1	Dependencies	25
12.2	Installing Elasticsearch	26
12.2.1	Docker	27
12.2.1.1	2.x	27
12.2.1.2	5.x	27
12.2.1.3	6.x	27
12.2.1.4	7.x	27
12.2.2	Vagrant	27
12.2.2.1	2.x	27
12.3	Quick start	27
12.3.1	Installation	29
12.3.2	Example app	30
12.3.2.1	Sample models	30
12.3.2.1.1	Required imports	30
12.3.2.1.2	Book statuses	30

12.3.2.1.3	Publisher model	31
12.3.2.1.4	Author model	32
12.3.2.1.5	Tag model	32
12.3.2.1.6	Book model	32
12.3.2.2	Admin classes	33
12.3.2.3	Create database tables	34
12.3.2.4	Fill in some data	34
12.3.2.5	Sample document	35
12.3.2.5.1	Required imports	35
12.3.2.5.2	Index definition	35
12.3.2.5.2.1	Settings	35
12.3.2.5.2.2	Document index	36
12.3.2.5.3	Custom analyzers	36
12.3.2.5.4	Document definition	36
12.3.2.6	Syncing Django's database with Elasticsearch indexes	38
12.3.2.6.1	Full database sync	38
12.3.2.6.2	Sample partial sync (using custom signals)	38
12.3.2.6.2.1	Required imports	38
12.3.2.6.2.2	Update book index on related model change	38
12.3.2.6.2.3	Update book index on related model removal	39
12.3.2.7	Sample serializer	40
12.3.2.7.1	Required imports	40
12.3.2.7.2	Serializer definition	40
12.3.2.8	ViewSet definition	42
12.3.2.8.1	Required imports	42
12.3.2.8.2	ViewSet definition	42
12.3.2.9	URLs	44
12.3.2.9.1	Required imports	45
12.3.2.9.2	Router definition	45
12.3.2.9.3	URL patterns	45
12.3.2.10	Check what you've done so far	45
12.3.2.10.1	URLs	45
12.3.2.10.2	Test in browser	45
12.3.3	Development and debugging	46
12.3.3.1	Profiling tools	46
12.3.3.1.1	Installation	46
12.3.3.1.2	Configuration	46
12.3.3.2	Debugging	47
12.4	Filter usage examples	47
12.4.1	Search	48
12.4.1.1	Search in all fields	48
12.4.1.2	Search a single term on specific field	48
12.4.1.3	Search for multiple terms	48
12.4.1.4	Search for multiple terms in specific fields	48
12.4.2	Filtering	49
12.4.2.1	Supported lookups	49
12.4.2.1.1	Native	49
12.4.2.1.1.1	term	49
12.4.2.1.1.2	terms	49
12.4.2.1.1.3	range	49
12.4.2.1.1.4	exists	50
12.4.2.1.1.5	prefix	50
12.4.2.1.1.6	wildcard	50
12.4.2.1.1.7	ids	50

12.4.2.1.2	Functional	50
12.4.2.1.2.1	contains	51
12.4.2.1.2.2	in	51
12.4.2.1.2.3	gt	51
12.4.2.1.2.4	gte	51
12.4.2.1.2.5	lt	51
12.4.2.1.2.6	lte	51
12.4.2.1.2.7	startswith	51
12.4.2.1.2.8	endswith	52
12.4.2.1.2.9	isnull	52
12.4.2.1.2.10	exclude	52
12.4.3	Usage examples	52
12.5	Search backends	52
12.5.1	Compound search filter backend	52
12.5.1.1	Sample view	53
12.5.1.2	Sample request	53
12.5.1.3	Generated query	53
12.5.2	Multi match search filter backend	54
12.5.2.1	Sample view	54
12.5.2.2	Sample request	55
12.5.2.3	Generated query	55
12.5.2.4	Options	55
12.5.2.4.1	Type options	56
12.5.2.4.2	Operator options	56
12.5.3	Simple query string filter backend	56
12.5.3.1	Sample view	56
12.5.3.2	Sample request 1	57
12.5.3.3	Generated query 1	57
12.5.3.4	Sample request 2	57
12.5.3.5	Generated query 2	58
12.5.3.6	Sample request 3	58
12.5.3.7	Generated query 3	58
12.5.3.8	Options	59
12.5.3.8.1	Default Operator options	59
12.6	Basic usage examples	59
12.6.1	Example app	60
12.6.1.1	Sample models	60
12.6.1.2	Sample document	61
12.6.1.3	Sample serializer	62
12.6.1.4	Sample view	63
12.6.1.5	Usage example	64
12.6.1.5.1	Sample queries	64
12.6.1.5.1.1	Search	64
12.6.1.5.1.2	Filtering	65
12.6.1.5.1.3	Ordering	65
12.7	Advanced usage examples	66
12.7.1	Example app	68
12.7.1.1	Sample models	68
12.7.1.2	Sample document	70
12.7.1.2.1	Index definition	70
12.7.1.2.1.1	Settings	70
12.7.1.2.1.2	Document index	71
12.7.1.3	Sample serializer	73
12.7.1.4	Sample view	74

12.7.1.5	Usage example	75
12.7.1.5.1	Search	75
12.7.1.5.2	Filtering	76
12.7.1.5.3	Ordering	77
12.7.1.6	Ids filter	78
12.7.1.7	Faceted search	78
12.7.1.8	Post-filter	79
12.7.1.8.1	Sample view	79
12.7.1.8.2	Sample queries	81
12.7.1.9	Geo-spatial features	81
12.7.1.9.1	Filtering	81
12.7.1.9.2	Ordering	82
12.7.1.10	Suggestions	82
12.7.1.10.1	Completion suggesters	82
12.7.1.10.1.1	Document definition	82
12.7.1.10.1.2	Serializer definition	84
12.7.1.10.1.3	ViewSet definition	84
12.7.1.10.1.4	Sample requests/responses	85
12.7.1.10.1.5	Suggestions on Array/List field	87
12.7.1.10.1.6	Sample requests/responses	87
12.7.1.10.1.7	Context suggesters	88
12.7.1.10.1.8	<i>category</i> context	88
12.7.1.10.1.9	<i>geo</i> context	90
12.7.1.10.2	Term and Phrase suggestions	91
12.7.1.10.2.1	Document definition	91
12.7.1.10.2.2	ViewSet definition	93
12.7.1.10.2.3	Sample requests/responses	96
12.7.1.10.2.4	Completion	97
12.7.1.10.2.5	Term	98
12.7.1.10.2.6	Phrase	99
12.7.1.11	Functional suggestions	100
12.7.1.11.1	Document definition	100
12.7.1.11.2	ViewSet definition	101
12.7.1.12	Highlighting	103
12.7.1.13	Pagination	105
12.7.1.13.1	Page number pagination	105
12.7.1.13.2	Limit/offset pagination	105
12.7.1.13.3	Customisations	105
12.8	Nested fields usage examples	106
12.8.1	Example app	107
12.8.1.1	Sample models	107
12.8.1.2	Sample document	111
12.8.1.2.1	Index definition	111
12.8.1.2.1.1	Settings	111
12.8.1.2.1.2	Document index	112
12.8.1.3	Sample serializer	115
12.8.1.4	Sample view	115
12.8.1.5	Usage example	117
12.8.1.5.1	Sample queries	118
12.8.1.5.1.1	Search	118
12.8.1.5.1.2	Nested filtering	118
12.8.1.5.1.3	Nested search	118
12.8.1.5.1.4	Sample models	118
12.8.1.5.1.5	Sample document	119

	12.8.1.5.1.6	Sample view	121
	12.8.1.5.1.7	Sample request	123
	12.8.1.5.1.8	Filtering	123
	12.8.1.5.1.9	Ordering	123
	12.8.1.6	Suggestions	123
	12.8.1.7	Nested aggregations/facets	124
12.9	More like this		127
	12.9.1	Usage example	127
		12.9.1.1 Sample document	127
		12.9.1.2 Sample view	128
		12.9.1.3 Sample request	129
		12.9.1.4 Generated query	129
		12.9.1.5 Options	130
12.10	Global aggregations		130
	12.10.1	Sample view	130
	12.10.2	Sample request	131
	12.10.3	Generated query	131
	12.10.4	Sample response	132
12.11	Configuration tweaks		133
	12.11.1	Ignore certain Elasticsearch exceptions	133
12.12	Source filtering backend		133
12.13	Indexing troubleshooting		134
	12.13.1	Timeout	134
	12.13.2	Chunk size	135
	12.13.3	Use parallel indexing	135
	12.13.4	Limit the number of items indexed at once	136
12.14	FAQ		137
	12.14.1	Questions and answers	137
12.15	frontend demo for django-elasticsearch-dsl-drf		138
	12.15.1	Quick start	138
		12.15.1.1 Install the django requirements	138
		12.15.1.2 Install Elasticsearch requirements	138
		12.15.1.3 Run Elasticsearch	138
		12.15.1.4 Build Elasticsearch index	138
		12.15.1.5 Install React requirements	139
		12.15.1.6 Run Django	139
		12.15.1.7 Run React demo app	139
12.16	Release history and notes		139
	12.16.1	0.20	139
	12.16.2	0.19	139
	12.16.3	0.18	140
	12.16.4	0.17.7	140
	12.16.5	0.17.6	140
	12.16.6	0.17.5	140
	12.16.7	0.17.4	141
	12.16.8	0.17.3	141
	12.16.9	0.17.2	141
	12.16.100.17.1	141
	12.16.110.17	141
	12.16.120.16.3	141
	12.16.130.16.2	142
	12.16.140.16.1	142
	12.16.150.16	142
	12.16.160.15.1	142

12.16.170.15	143
12.16.180.14	143
12.16.190.13.2	143
12.16.200.13.1	143
12.16.210.13	143
12.16.220.12	143
12.16.230.11	144
12.16.240.10	145
12.16.250.9	145
12.16.260.8.4	145
12.16.270.8.3	145
12.16.280.8.2	146
12.16.290.8.1	146
12.16.300.8	146
12.16.310.7.2	146
12.16.320.7.1	147
12.16.330.7	147
12.16.340.6.4	147
12.16.350.6.3	147
12.16.360.6.2	147
12.16.370.6.1	147
12.16.380.6	147
12.16.390.5.1	148
12.16.400.5	148
12.16.410.4.4	148
12.16.420.4.3	148
12.16.430.4.2	148
12.16.440.4.1	149
12.16.450.4	149
12.16.460.3.12	149
12.16.470.3.11	149
12.16.480.3.10	149
12.16.490.3.9	149
12.16.500.3.8	150
12.16.510.3.7	150
12.16.520.3.6	150
12.16.530.3.5	150
12.16.540.3.4	150
12.16.550.3.3	150
12.16.560.3.2	150
12.16.570.3.1	150
12.16.580.3	151
12.16.590.2.6	151
12.16.600.2.5	151
12.16.610.2.4	151
12.16.620.2.3	151
12.16.630.2.2	151
12.16.640.2.1	151
12.16.650.2	151
12.16.660.1.8	152
12.16.670.1.7	152
12.16.680.1.6	152
12.16.690.1.5	152
12.16.700.1.4	152

12.16.710.1.3	152
12.16.720.1.2	153
12.16.730.1.1	153
12.16.740.1	153
12.17 django_elasticsearch_dsl_drf package	153
12.17.1 Subpackages	153
12.17.1.1 django_elasticsearch_dsl_drf.fields package	153
12.17.1.1.1 Submodules	153
12.17.1.1.2 django_elasticsearch_dsl_drf.fields.common module	153
12.17.1.1.3 django_elasticsearch_dsl_drf.fields.helpers module	155
12.17.1.1.4 django_elasticsearch_dsl_drf.fields.nested_fields module	155
12.17.1.1.5 Module contents	158
12.17.1.2 django_elasticsearch_dsl_drf.filter_backends package	161
12.17.1.2.1 Subpackages	161
12.17.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package	161
12.17.1.2.1.2 Submodules	161
12.17.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module	161
12.17.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module	161
12.17.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module	161
12.17.1.2.1.6 Module contents	161
12.17.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package	161
12.17.1.2.1.8 Submodules	161
12.17.1.2.1.9 django_elasticsearch_dsl_drf.filter_backends.filtering.common module	161
12.17.1.2.1.10 django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial module	169
12.17.1.2.1.11 django_elasticsearch_dsl_drf.filter_backends.filtering.ids module	173
12.17.1.2.1.12 django_elasticsearch_dsl_drf.filter_backends.filtering.nested module	175
12.17.1.2.1.13 django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter module	176
12.17.1.2.1.14 Module contents	178
12.17.1.2.1.15 django_elasticsearch_dsl_drf.filter_backends.ordering package	192
12.17.1.2.1.16 Submodules	192
12.17.1.2.1.17 django_elasticsearch_dsl_drf.filter_backends.ordering.common module	192
12.17.1.2.1.18 django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial module	195
12.17.1.2.1.19 Module contents	197
12.17.1.2.1.20 django_elasticsearch_dsl_drf.filter_backends.search package	202
12.17.1.2.1.21 Subpackages	202
12.17.1.2.1.22 django_elasticsearch_dsl_drf.filter_backends.search.query_backends package	202
12.17.1.2.1.23 Submodules	202
12.17.1.2.1.24 django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base module	202
12.17.1.2.1.25 django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match module	202
12.17.1.2.1.26 django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase module	202

12.17.1.2.1.27	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix module	203
12.17.1.2.1.28	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match module	203
12.17.1.2.1.29	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested module	204
12.17.1.2.1.30	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string module	205
12.17.1.2.1.31	Module contents	206
12.17.1.2.1.32	Submodules	210
12.17.1.2.1.33	django_elasticsearch_dsl_drf.filter_backends.search.base module	210
12.17.1.2.1.34	django_elasticsearch_dsl_drf.filter_backends.search.compound module	211
12.17.1.2.1.35	django_elasticsearch_dsl_drf.filter_backends.search.historical module	211
12.17.1.2.1.36	django_elasticsearch_dsl_drf.filter_backends.search.multi_match module	213
12.17.1.2.1.37	django_elasticsearch_dsl_drf.filter_backends.search.query_string module	213
12.17.1.2.1.38	django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string module	213
12.17.1.2.1.39	Module contents	214
12.17.1.2.1.40	django_elasticsearch_dsl_drf.filter_backends.suggester package	217
12.17.1.2.1.41	Submodules	217
12.17.1.2.1.42	django_elasticsearch_dsl_drf.filter_backends.suggester.functional module	217
12.17.1.2.1.43	django_elasticsearch_dsl_drf.filter_backends.suggester.native module	222
12.17.1.2.1.44	Module contents	226
12.17.1.2.2	Submodules	233
12.17.1.2.3	django_elasticsearch_dsl_drf.filter_backends.faceted_search module	233
12.17.1.2.4	django_elasticsearch_dsl_drf.filter_backends.highlight module	236
12.17.1.2.5	django_elasticsearch_dsl_drf.filter_backends.mixins module	237
12.17.1.2.6	Module contents	238
12.17.1.3	django_elasticsearch_dsl_drf.tests package	239
12.17.1.3.1	Submodules	239
12.17.1.3.2	django_elasticsearch_dsl_drf.tests.base module	239
12.17.1.3.3	django_elasticsearch_dsl_drf.tests.data_mixins module	239
12.17.1.3.4	django_elasticsearch_dsl_drf.tests.test_faceted_search module	239
12.17.1.3.5	django_elasticsearch_dsl_drf.tests.test_filtering_common module	240
12.17.1.3.6	django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module	243
12.17.1.3.7	django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module	244
12.17.1.3.8	django_elasticsearch_dsl_drf.tests.test_filtering_nested module	244
12.17.1.3.9	django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module	246
12.17.1.3.10	django_elasticsearch_dsl_drf.tests.test_functional_suggesters module	248
12.17.1.3.11	django_elasticsearch_dsl_drf.tests.test_helpers module	249
12.17.1.3.12	django_elasticsearch_dsl_drf.tests.test_highlight module	249
12.17.1.3.13	django_elasticsearch_dsl_drf.tests.test_more_like_this module	249
12.17.1.3.14	django_elasticsearch_dsl_drf.tests.test_ordering_common module	249
12.17.1.3.15	django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial module	251
12.17.1.3.16	django_elasticsearch_dsl_drf.tests.test_pagination module	251
12.17.1.3.17	django_elasticsearch_dsl_drf.tests.test_pip_helpers module	251
12.17.1.3.18	django_elasticsearch_dsl_drf.tests.test_search module	252

12.17.1.3.19	django_elasticsearch_dsl_drf.tests.test_search_multi_match module . . .	252
12.17.1.3.20	django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module	253
12.17.1.3.21	django_elasticsearch_dsl_drf.tests.test_serializers module	254
12.17.1.3.22	django_elasticsearch_dsl_drf.tests.test_suggesters module	254
12.17.1.3.23	django_elasticsearch_dsl_drf.tests.test_views module	255
12.17.1.3.24	django_elasticsearch_dsl_drf.tests.test_wrappers module	256
12.17.1.3.25	Module contents	256
12.17.2	Submodules	270
12.17.3	django_elasticsearch_dsl_drf.analyzers module	270
12.17.4	django_elasticsearch_dsl_drf.apps module	270
12.17.5	django_elasticsearch_dsl_drf.compat module	271
12.17.6	django_elasticsearch_dsl_drf.constants module	271
12.17.7	django_elasticsearch_dsl_drf.helpers module	271
12.17.8	django_elasticsearch_dsl_drf.pagination module	272
12.17.9	django_elasticsearch_dsl_drf.pip_helpers module	274
12.17.10	django_elasticsearch_dsl_drf.serializers module	274
12.17.11	django_elasticsearch_dsl_drf.utils module	275
12.17.12	django_elasticsearch_dsl_drf.versions module	275
12.17.13	django_elasticsearch_dsl_drf.viewsets module	275
12.17.14	django_elasticsearch_dsl_drf.wrappers module	276
12.17.15	Module contents	277
13	Indices and tables	279
	Python Module Index	281
	Index	283

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

Documentation

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

Make sure to read [FAQ](#).

CHAPTER 2

Prerequisites

- Django 1.11, 2.0, 2.1 and 2.2.
- Python 2.7, 3.5, 3.6, 3.7.
- Elasticsearch 6.x, 7.x. For older versions use `django-elasticsearch-dsl-drf` version 0.18.

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`, `prefix` (same as `startswith`), `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.*
- *Post-filter filter backend.*
- *Nested filtering filter backend.*
- *Highlight backend.*
- *Suggester filter backend.*
- *Functional suggester filter backend.*
- *Pagination* (Page number and limit/offset pagination).
- *Ids filter backend.*
- *Multi match search filter backend.*
- *Simple search query search filter backend.*
- *More-like-this support* (detail action).
- *Global aggregations support.*
- *Source filter backend.*

CHAPTER 4

Demo

A frontend demo (React based) is available. See the [dedicated docs](#) for more information.

To bootstrap evaluation, clone the repository locally and run *docker-compose*.

```
docker-compose up
```

It will set up:

- Elasticsearch on <http://localhost:9200>
- Django REST framework on <http://localhost:8000>
- React on <http://localhost:3000>

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 py37-django21
```

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
```

To run a single test class in a given test module in your working environment type:

```
./runtests.py src/django_elasticsearch_dsl_drf/tests/test_suggesters.  
↪py::TestSuggesters
```

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

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

Writing documentation

Keep the following hierarchy.

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


CHAPTER 9

License

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

CHAPTER 10

Support

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

CHAPTER 11

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

CHAPTER 12

Project documentation

Contents:

Table of Contents

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

12.1 Dependencies

elasticsearch and elasticsearch-dsl

Depending on your `Elasticsearch` version (either 2.x, 5.x, 6.x or 7.x) you should use 2.x, 5.x, 6.x or 7.x versions of the `elasticsearch` and `elasticsearch-dsl` packages accordingly.

Current compatibility matrix is:

This package	Elasticsearch
0.20	6.x, 7.x
0.19	6.x
0.18	2.x, 5.x, 6.x

django-elasticsearch-dsl

You are advised to use the latest version of `django-elasticsearch-dsl`.

As of `django-elasticsearch-dsl-drf` 0.19, support for `Elasticsearch` versions prior 6.x has been dropped.

Django/ Django REST Framework

Initial version of this package was written for `django` 3.6.2.

Starting from `django-elasticsearch-dsl-drf` version 0.18, support for `Django` versions prior 1.11 and `Django REST Framework` versions prior 3.9 has been dropped.

Current compatibility matrix is:

Django	Django REST Framework
1.11	3.9.3
2.0	3.9.3
2.1	3.9.3
2.2	3.9.3

The version 0.17.7 has been tested with the following versions of `Django` and `Django REST Framework`:

Django	Django REST Framework
1.8	3.6.2
1.9	3.6.2
1.10	3.6.2
1.11	3.7.7
2.0	3.7.7
2.1	3.8.2
2.2	3.9.2

12.2 Installing Elasticsearch

For development and testing purposes, it's often handy to be able to quickly switch between different `Elasticsearch` versions. Since this packages supports 2.x, 5.x and 6.x branches, you could make use of the following boxes/containers for development and testing.

Note: As of `django-elasticsearch-dsl-drf` 0.19, support for `Elasticsearch` versions prior 6.x has been dropped.

For all containers/boxes mentioned below, no authentication is required (for `Elasticsearch`).

12.2.1 Docker

12.2.1.1 2.x

```
docker pull elasticsearch:2.4.6
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" elasticsearch:2.4.6
```

12.2.1.2 5.x

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:5.5.3
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" docker.elastic.co/elasticsearch/elasticsearch:5.5.3
```

12.2.1.3 6.x

6.3.2

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:6.3.2
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" docker.elastic.co/elasticsearch/elasticsearch:6.3.2
```

6.4.0

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:6.4.0
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" docker.elastic.co/elasticsearch/elasticsearch:6.4.0
```

12.2.1.4 7.x

7.3.0

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:7.3.0
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" docker.elastic.co/elasticsearch/elasticsearch:7.3.0
```

12.2.2 Vagrant

12.2.2.1 2.x

```
./scripts/vagrant_start.sh
```

12.3 Quick start

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

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

Contents:

Table of Contents

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

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

12.3.1 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'
```

(continues on next page)

(continued from previous page)

```
} ,  
}
```

12.3.2 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  
    # ...  
)
```

12.3.2.1 Sample models

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

12.3.2.1.1 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
```

12.3.2.1.2 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

```

12.3.2.1.3 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 {

```

(continues on next page)

(continued from previous page)

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

12.3.2.1.4 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
```

12.3.2.1.5 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
```

12.3.2.1.6 Book model

books/models/book.py

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

(continues on next page)

(continued from previous page)

```

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

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

    ordering = ["isbn"]

def __str__(self):
    return self.title

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

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

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

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

```

12.3.2.2 Admin classes

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

books/admin.py

```
from django.contrib import admin

from .models import *

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

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

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

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

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

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

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

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

12.3.2.3 Create database tables

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

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

12.3.2.4 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.

12.3.2.5 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 flattened in Elasticsearch indexes/documents. Nested relations are denormalized.

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

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

12.3.2.5.1 Required imports

search_indexes/documents/book.py

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

from books.models import Book
```

12.3.2.5.2 Index definition

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

12.3.2.5.2.1 Settings

Note: In the examples below the `search_indexes.documents.book` and `search_indexes.documents.publisher` are the pythonic file paths to modules where documents are defined.

settings/base.py

Note: In this example, `book` and `publisher` are Elasticsearch index names.

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

settings/testing.py

Note: In this example, `test_book` and `test_publisher` are Elasticsearch index names.

```
# 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',
}
```

12.3.2.5.2.2 Document index

search_indexes/documents/book.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
)
```

12.3.2.5.3 Custom analyzers

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

12.3.2.5.4 Document definition

search_indexes/documents/book.py

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

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

    title = fields.StringField(
        analyzer=html_strip,
```

(continues on next page)

(continued from previous page)

```

        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

```

(continues on next page)

(continued from previous page)

```
)

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

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

12.3.2.6 Syncing Django's database with Elasticsearch indexes

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

12.3.2.6.1 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.

12.3.2.6.2 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.

12.3.2.6.2.1 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
```

12.3.2.6.2.2 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)

```

12.3.2.6.2.3 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.

```

(continues on next page)

(continued from previous page)

```
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)
```

12.3.2.7 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.

12.3.2.7.1 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
```

12.3.2.7.2 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',
```

(continues on next page)

(continued from previous page)

```

        '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:

```

(continues on next page)

(continued from previous page)

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

12.3.2.8 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.

12.3.2.8.1 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.viewsets import BaseDocumentViewSet
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination

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

12.3.2.8.2 ViewSet definition

search_indexes/viewsets/book.py

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

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

(continues on next page)

(continued from previous page)

```

        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',

```

(continues on next page)

(continued from previous page)

```
# 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',)
```

12.3.2.9 URLs

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

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

12.3.2.9.1 Required imports

search_indexes/urls.py

```
from django.conf.urls import url, include
from rest_framework.routers import DefaultRouter

from .views import BookDocumentView
```

12.3.2.9.2 Router definition

search_indexes/urls.py

```
router = DefaultRouter()
books = router.register(r'books',
                        BookDocumentView,
                        basename='bookdocument')
```

12.3.2.9.3 URL patterns

search_indexes/urls.py

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

12.3.2.10 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.

12.3.2.10.1 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)),
    # ...
]
```

12.3.2.10.2 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
```

12.3.3 Development and debugging

12.3.3.1 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.

12.3.3.1.1 Installation

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

12.3.3.1.2 Configuration

Change your development settings in the following way:

settings/dev.py

```
MIDDLEWARE += (
    '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',
)
```

(continues on next page)

(continued from previous page)

```

'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',
)

```

12.3.3.2 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.

12.4 Filter usage examples

Example usage of filtering backends.

Contents:

Table of Contents

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

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

12.4.1 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`.

12.4.1.1 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
```

12.4.1.2 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
```

12.4.1.3 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
```

12.4.1.4 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
```

12.4.2 Filtering

12.4.2.1 Supported lookups

12.4.2.1.1 Native

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

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

12.4.2.1.1.1 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
```

12.4.2.1.1.2 terms

Find documents which contain any of the exact terms specified in the field specified. Note, that multiple values are separated with double underscores __.

```
http://localhost:8000/api/articles/?id=1&id=2&id=3
http://localhost:8000/api/articles/?id__terms=1__2__3
```

12.4.2.1.1.3 range

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

From, to

```
http://localhost:8000/api/users/?age__range=16__67
```

From, to, boost

```
http://localhost:8000/api/users/?age__range=16__67__2.0
```

12.4.2.1.1.4 exists

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

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

12.4.2.1.1.5 prefix

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

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

12.4.2.1.1.6 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 (*)

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

Should match: *delusional insanity*.

12.4.2.1.1.7 ids

Find documents with the specified type and IDs.

```
http://localhost:8000/api/articles/?ids=68__64__58  
http://localhost:8000/api/articles/?ids=68&ids=64&ids=58
```

12.4.2.1.2 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*

12.4.2.1.2.1 contains

Case-insensitive containment test.

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

Should match: *published*, *not published*, *needs polishing*.

12.4.2.1.2.2 in

In a given list.

```
http://localhost:8000/api/articles/?id__in=1__2__3
```

12.4.2.1.2.3 gt

Greater than.

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

12.4.2.1.2.4 gte

Greater than or equal to.

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

12.4.2.1.2.5 lt

Less than.

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

12.4.2.1.2.6 lte

Less than or equal to.

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

12.4.2.1.2.7 startswith

Case-sensitive starts-with.

```
http://localhost:8000/api/articles/?tags__startswith=bio
```

Should match: *biography*, *bio mechanics*

12.4.2.1.2.8 endswith

Case-sensitive ends-with.

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

Should match: *published*, *not published*.

12.4.2.1.2.9 isnull

Takes either True or False.

True

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

False

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

12.4.2.1.2.10 exclude

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

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

12.4.3 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](#)

12.5 Search backends

12.5.1 Compound search filter backend

Compound search filter backend aims to replace old style *SearchFilterBackend*.

12.5.1.1 Sample view

```

from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    CompoundSearchFilterBackend,
    OrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookCompoundSearchBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

    filter_backends = [
        # ...
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        CompoundSearchFilterBackend,
        # ...
    ]

    multi_match_search_fields = (
        'title',
        'description',
        'summary',
    )

    ordering = ('_score', 'id', 'title', 'price',)

```

12.5.1.2 Sample request

```
http://localhost:8000/search/books-compound-search-backend/?search=enim
```

12.5.1.3 Generated query

```

{
  "from": 0,
  "sort": [
    "id",
    "title",
    "price"
  ],
  "size": 23,
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "title": {

```

(continues on next page)

(continued from previous page)

```

        "query": "enim"
    }
},
{
    "match": {
        "description": {
            "query": "enim"
        }
    }
},
{
    "match": {
        "summary": {
            "query": "enim"
        }
    }
}
]
}
}

```

12.5.2 Multi match search filter backend

Document and serializer definition are trivial (there are lots of examples in other sections).

12.5.2.1 Sample view

```

from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    MultiMatchSearchFilterBackend,
    OrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookMultiMatchSearchFilterBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

    filter_backends = [
        # ...
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        MultiMatchSearchFilterBackend,
        # ...
    ]

```

(continues on next page)

(continued from previous page)

```
multi_match_search_fields = {
    'title': {'boost': 4},
    'summary': {'boost': 2},
    'description': None,
}

ordering = ('_score', 'id', 'title', 'price',)
```

12.5.2.2 Sample request

Note: Multiple search params (*search_multi_match*) are not supported. Even if you provide multiple search params, the first one would be picked, having the rest simply ignored.

```
http://localhost:8000/search/books-multi-match-search-backend/?search_multi_
↪match=debitis%20enim
```

12.5.2.3 Generated query

```
{
  "from": 0,
  "query": {
    "multi_match": {
      "query": "debitis enim",
      "fields": [
        "summary^2",
        "description",
        "title^4"
      ]
    }
  },
  "size": 38,
  "sort": [
    "_score",
    "id",
    "title",
    "price"
  ]
}
```

12.5.2.4 Options

All standard multi match query options are available/tunable with help of `multi_match_options` view property.

Selective list of available options:

- operator
- type
- analyzer

- tie_breaker

12.5.2.4.1 Type options

See the [Elasticsearch docs](#) for detailed explanation.

- best_fields
- most_fields
- cross_fields
- phrase
- phrase_prefix

Example

```
class BookMultiMatchSearchFilterBackendDocumentViewSet(DocumentViewSet):

    # ...

    multi_match_options = {
        'type': 'phrase'
    }
```

12.5.2.4.2 Operator options

Can be either `and` or `or`.

12.5.3 Simple query string filter backend

Document and serializer definition are trivial (there are lots of examples in other sections).

12.5.3.1 Sample view

```
from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    SimpleQueryStringSearchFilterBackend,
    OrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookSimpleQueryStringSearchFilterBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

    filter_backends = [
```

(continues on next page)

(continued from previous page)

```

    # ...
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
    SimpleQueryStringSearchFilterBackend,
    # ...
]

simple_query_string_search_fields = {
    'title': {'boost': 4},
    'summary': {'boost': 2},
    'description': None,
}

ordering = ('_score', 'id', 'title', 'price',)

```

12.5.3.2 Sample request 1

Note: Multiple search params (*search_simple_query_string*) are not supported. Even if you provide multiple search params, the first one would be picked, having the rest simply ignored.

```

http://localhost:8000/search/books-simple-query-string-search-backend/?search_simple_
↪query_string="chapter%20II"%20%2Bfender

```

12.5.3.3 Generated query 1

```

{
  "query": {
    "simple_query_string": {
      "query": "\"chapter II\" +fender",
      "default_operator": "and",
      "fields": [
        "title",
        "description",
        "summary"
      ]
    }
  },
  "sort": [
    "_score",
    "id",
    "title",
    "price"
  ],
  "from": 0,
  "size": 1
}

```

12.5.3.4 Sample request 2

Note: Multiple search params (*search_simple_query_string*) are not supported. Even if you provide multiple search params, the first one would be picked, having the rest simply ignored.

```
http://localhost:8000/search/books-simple-query-string-search-backend/?search_simple_
↪query_string="chapter%20II"%20%2B(shutting%20|%20fender)
```

12.5.3.5 Generated query 2

```
{
  "query": {
    "simple_query_string": {
      "query": "\"chapter II\" +(shutting | fender)",
      "default_operator": "and",
      "fields": [
        "title",
        "description",
        "summary"
      ]
    }
  },
  "sort": [
    "_score",
    "id",
    "title",
    "price"
  ],
  "from": 0,
  "size": 2
}
```

12.5.3.6 Sample request 3

Note: Multiple search params (*search_simple_query_string*) are not supported. Even if you provide multiple search params, the first one would be picked, having the rest simply ignored.

```
http://localhost:8000/search/books-simple-query-string-search-backend/?search_simple_
↪query_string=%22Pool%20of%20Tears%22%20-considering
```

12.5.3.7 Generated query 3

```
{
  "query": {
    "simple_query_string": {
      "query": "\"Pool of Tears\" -considering",
      "default_operator": "and",
      "fields": [
        "title",
        "description",
```

(continues on next page)

(continued from previous page)

```

        "summary"
    ]
}
},
"sort": [
    "_score",
    "id",
    "title",
    "price"
],
"from": 0,
"size": 1
}

```

12.5.3.8 Options

All standard multi match query options are available/tunable with help of `simple_query_string_options` view property.

Selective list of available options:

- `default_operator`

12.5.3.8.1 Default Operator options

Can be either `and` or `or`.

Example

```

class BookSimpleQueryStringSearchFilterBackendDocumentViewSet(DocumentViewSet):

    # ...

    simple_query_string_options = {
        "default_operator": "and",
    }

```

12.6 Basic usage examples

Basic Django REST framework integration example

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

- `models`
- `documents`
- `serializers`
- `views`

Contents:

Table of Contents

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

12.6.1 Example app

12.6.1.1 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
```

(continues on next page)

(continued from previous page)

```
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,
    }
```

12.6.1.2 Sample document

search_indexes/documents/publisher.py

```
from django_elasticsearch_dsl import Document, 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(Document):
    """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(
```

(continues on next page)

(continued from previous page)

```

        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 Document

```

12.6.1.3 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:

```

(continues on next page)

(continued from previous page)

```
return {}
```

12.6.1.4 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.viewsets import DocumentViewSet

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

class PublisherDocumentView(DocumentViewSet):
    """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,
```

(continues on next page)

(continued from previous page)

```
}
# Specify default ordering
ordering = ('id', 'name',)
# Define geo-spatial filtering fields
geo_spatial_filter_fields = {
    'location': {
        'lookups': [
            LOOKUP_FILTER_GEO_DISTANCE,
        ],
    },
}
```

12.6.1.5 Usage example

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

12.6.1.5.1 Sample queries

12.6.1.5.1.1 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
```

12.6.1.5.1.2 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
```

12.6.1.5.1.3 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
```

12.7 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

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

- * *Post-filter*
 - *Sample view*
 - *Sample queries*
- * *Geo-spatial features*
 - *Filtering*
 - *Ordering*
- * *Suggestions*
 - *Completion suggesters*
 - *Document definition*
 - *Serializer definition*
 - *ViewSet definition*
 - *Sample requests/responses*
 - *Suggestions on Array/List field*
 - *Sample requests/responses*
 - *Context suggesters*
 - *category context*
 - *geo context*
 - *Term and Phrase suggestions*
 - *Document definition*
 - *ViewSet definition*
 - *Sample requests/responses*
 - *Completion*
 - *Term*
 - *Phrase*
- * *Functional suggestions*
 - *Document definition*
 - *ViewSet definition*
- * *Highlighting*
- * *Pagination*
 - *Page number pagination*
 - *Limit/offset pagination*
 - *Customisations*

12.7.1 Example app

12.7.1.1 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):
```

(continues on next page)

(continued from previous page)

```

    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)

```

(continues on next page)

(continued from previous page)

```

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()]

```

12.7.1.2 Sample document

12.7.1.2.1 Index definition

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

12.7.1.2.1.1 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',
}
```

12.7.1.2.1.2 Document index

search_indexes/documents/book.py

```
from django.conf import settings
from django_elasticsearch_dsl import Document, 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(Document):
    """Book Elasticsearch document."""

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

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

(continues on next page)

(continued from previous page)

```

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."""

```

(continues on next page)

(continued from previous page)

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

12.7.1.3 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',
```

(continues on next page)

(continued from previous page)

```

        'stock_count',
        'tags',
    )
    read_only_fields = fields

    def get_tags(self, obj):
        """Get tags."""
        if obj.tags:
            return list(obj.tags)
        else:
            return []

```

12.7.1.4 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.viewsets import DocumentViewSet

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

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

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'summary',
        'description',
    )

```

(continues on next page)

(continued from previous page)

```

# 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',)

```

12.7.1.5 Usage example

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

12.7.1.5.1 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
```

Search with boosting

It's possible to boost search fields. In order to do that change the `search_fields` definition of the `DocumentViewSet` as follows:

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

    # ...

    # Define search fields
    search_fields = {
        'title': {'boost': 4},
        'summary': {'boost': 2},
        'description': None,
    }

    # Order by `_score` first.
    ordering = ('_score', 'id', 'title', 'price',)

    # ...
```

Note, that we are ordering results by `_score` first.

12.7.1.5.2 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
```

12.7.1.5.3 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
```

12.7.1.6 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
```

12.7.1.7 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(DocumentViewSet):
    """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,
        },
    },
```

(continues on next page)

(continued from previous page)

```

    '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
```

12.7.1.8 Post-filter

The *post_filter* is very similar to the common filter. The only difference is that it doesn't affect facets. So, whatever post-filters applied, the numbers in facets will remain intact.

12.7.1.8.1 Sample view

Note: Note the `PostFilterFilteringFilterBackend` and `post_filter_fields` usage.

search_indexes/viewsets/book.py

```

# ...

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

# ...

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

```

(continues on next page)

(continued from previous page)

```

document = BookDocument
serializer_class = BookDocumentSerializer
lookup_field = 'id'
filter_backends = [
    FilteringFilterBackend,
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
    SearchFilterBackend,
    PostFilterFilteringFilterBackend,
]
# Define search fields
search_fields = (
    'title',
    'summary',
    'description',
)
# 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 post-filter filtering fields
post_filter_fields = {
    'publisher_pf': 'publisher.raw',
    'isbn_pf': 'isbn.raw',
    'state_pf': 'state.raw',
    'tags_pf': {
        'field': 'tags',

```

(continues on next page)

(continued from previous page)

```

        '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',)

```

12.7.1.8.2 Sample queries

Filter documents by field

Filter documents by field (state) “published”.

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

Filter documents by multiple fields

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

```
http://127.0.0.1:8080/search/books/?state_pf__in=published__in_progress
```

12.7.1.9 Geo-spatial features

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

- geojson.io
- [Bounding Box Tool](#)

12.7.1.9.1 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
```

12.7.1.9.2 Ordering

Geo-distance ordering

```
http://localhost:8000/search/publishers/?ordering=location__48.85__2.30__km__plane
```

12.7.1.10 Suggestions

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

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

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

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

12.7.1.10.1 Completion suggesters

12.7.1.10.1.1 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 Document, 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,
```

(continues on next page)

(continued from previous page)

```

        number_of_replicas=1
    )

@INDEX.doc_type
class PublisherDocument(Document):
    """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 Document

```

After that the `name.suggest`, `city.suggest`, `state_province.suggest` and `country.suggest`

fields would be available for suggestions feature.

12.7.1.10.1.2 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',
        )
```

12.7.1.10.1.3 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(DocumentViewSet):
    """The PublisherDocument view."""

    document = PublisherDocument

    # ...
```

(continues on next page)

(continued from previous page)

```

filter_backends = [
    # ...
    SuggesterFilterBackend,
]

# ...

# Suggester fields
suggester_fields = {
    'name_suggest': {
        'field': 'name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
        'options': {
            'size': 20,  # Override default number of suggestions
        },
    },
    '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.

12.7.1.10.1.4 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,

```

(continues on next page)

(continued from previous page)

```

        "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
        }
    ]
}

```

12.7.1.10.1.5 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'**)

12.7.1.10.1.6 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"
    }
  ]
}
```

12.7.1.10.1.7 Context suggesters

Note, that context suggesters only work for *completion* (thus, not for *term* or *phrase*).

12.7.1.10.1.8 *category* context

The completion suggester considers all documents in the index, but it is often desirable to serve suggestions filtered and/or boosted by some criteria. For example, you want to suggest song titles filtered by certain artists or you want to boost song titles based on their genre.

In that case, the document definition should be altered as follows:

Document definition

```
class BookDocument(Document):

    # ...

    title = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
            'suggest_context': fields.CompletionField(
                contexts=[
                    {
                        "name": "tag",
                        "type": "category",
```

(continues on next page)

(continued from previous page)

```

        # The `path` value shall be pointing to an
        # existing field of current document, which shall
        # be used for filtering.
        "path": "tags.raw",
    },
]
),
}
)

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

# ...

```

ViewSet should altered as follows:

ViewSet definition

```

class BookFrontendDocumentViewSet(DocumentViewSet):

    # ...

    # Suggester fields
    suggester_fields = {
        'title_suggest_context': {
            'field': 'title.suggest_context',
            'default_suggester': SUGGESTER_COMPLETION,
            # We want to be able to filter the completion filter
            # results on the following params: tag, state and publisher.
            # We also want to provide the size value.
            # See the "https://www.elastic.co/guide/en/elasticsearch/
            # reference/6.1/suggester-context.html" for the reference.
            'completion_options': {
                'category_filters': {
                    # The `tag` has been defined as `name` value in the
                    # `suggest_context` of the `BookDocument`.
                    'title_suggest_tag': 'tag',
                },
            },
            'options': {
                'size': 10, # By default, number of results is 5.
            },
        },
    }

    # ...

```

And finally we can narrow our suggestions as follows:

Sample request

In the example below we have filtered suggestions by tags “Art” and “Comics” having boosted “Comics” by 2.0.

```
GET http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M&
→title_suggest_tag=Art&title_suggest_tag=Comics__2.0
```

12.7.1.10.1.9 geo context

Geo context allows to get suggestions within a certain distance from a specified geo location.

In that case, the document definition should be altered as follows:

Document definition

```
class AddressDocument(Document):

    # ...

    street = StringField(
        analyzer=html_strip,
        fields={
            'raw': KeywordField(),
            'suggest': fields.CompletionField(),
            'suggest_context': fields.CompletionField(
                contexts=[
                    {
                        "name": "loc",
                        "type": "geo",
                        "path": "location",
                        # You could also optionally add precision value.
                        # However, this is not required and can be
                        # specified in the query during runtime.
                        # "precision": "100km",
                    },
                ],
            ),
        }
    )

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

    # ...
```

ViewSet should altered as follows:

ViewSet definition

```
class BookFrontendDocumentViewSet(DocumentViewSet):

    # ...

    # Suggester fields
    suggester_fields = {
        'street_suggest_context': {
            'field': 'street.suggest_context',
```

(continues on next page)

(continued from previous page)

```

'default_suggester': SUGGESTER_COMPLETION,
# We want to be able to filter the completion filter
# results on the following params: tag, state and publisher.
# We also want to provide the size value.
# See the "https://www.elastic.co/guide/en/elasticsearch/
# reference/6.1/suggester-context.html" for the reference.
'completion_options': {
    'geo_filters': {
        'title_suggest_loc': 'loc',
    },
},
'options': {
    'size': 10, # By default, number of results is 5.
},
},
}

# ...

```

And finally we can narrow our suggestions as follows:

Sample request

In the example below we have filtered suggestions within 8000km distance from geo-point (-30, -100).

```
GET http://localhost:8000/search/addresses-frontend/suggest/?street_suggest_context=L&
↪title_suggest_loc=-30__-100__8000km
```

Same query with boosting (boost value 2.0):

```
GET http://localhost:8000/search/addresses-frontend/suggest/?street_suggest_context=L&
↪title_suggest_loc=-30__-100__8000km__2.0
```

12.7.1.10.2 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.

12.7.1.10.2.1 Document definition

search_indexes/documents/book.py

```

from django.conf import settings

from django_elasticsearch_dsl import Document, 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,

```

(continues on next page)

(continued from previous page)

```

        number_of_replicas=1
    )

@INDEX.doc_type
class BookDocument(Document):
    """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(),

```

(continues on next page)

(continued from previous page)

```

    }
)

# 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 Document

```

12.7.1.10.2.2 ViewSet definition

Note: The suggerster filter backends shall come as last ones.

Suggesters for the view are configured in `suggester_fields` property.

In the example below, the `title_suggest` is the name of the GET query param which points to the `title_suggest` field of the `BookDocument` document. For the `title_suggest` the allowed suggesters are `SUGGESTER_COMPLETION`, `SUGGESTER_TERM` and `SUGGESTER_PHRASE`.

URL shall be constructed in the following way:

```
/search/books/suggest/{QUERY_PARAM}__{SUGGESTER_NAME}={VALUE}
```

Example for completion suggerster:

```
GET http://127.0.0.1:8000/search/books/suggest/?title_suggest__completion=temp
```

However, since we have `default_suggester` defined we can skip the `__{SUGGESTER_NAME}` part (if we want completion suggerster functionality). Thus, it might be written as short as:

```
GET http://127.0.0.1:8000/search/books/suggest/?title_suggest=temp
```

Example for term suggerster:

```
GET http://127.0.0.1:8000/search/books/suggest/?title_suggest__term=tmeporus
```

Example for phrase suggester:

```
GET http://127.0.0.1:8000/search/books/suggest/?title_suggest__phrase=tmeporus
```

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

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

    document = BookDocument
    # serializer_class = BookDocumentSerializer
    serializer_class = BookDocumentSimpleSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
        SuggesterFilterBackend, # This should be the last backend
    ]
    # Define search fields
    search_fields = (
        'title',
        'description',
        'summary',
    )
    # Define filter fields
    filter_fields = {
        'id': {
            'field': 'id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
```

(continues on next page)

(continued from previous page)

```

        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,

```

(continues on next page)

(continued from previous page)

```

        ],
    },
    # 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': {
        'field': 'title.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
            SUGGESTER_TERM,
            SUGGESTER_PHRASE,
        ]
        'default_suggester': SUGGESTER_COMPLETION,
        'options': {
            'size': 10, # Number of suggestions to retrieve.
        },
    },
    'publisher_suggest': 'publisher.suggest',
    'tag_suggest': 'tags.suggest',
    'summary_suggest': 'summary',
}

```

Note, that by default the number of suggestions is limited to 5. If you need more suggestions, add ‘options’ dictionary with *size* provided, as show above.

12.7.1.10.2.3 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

```

(continues on next page)

(continued from previous page)

```
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.
```

12.7.1.10.2.4 Completion

Request

```
GET http://127.0.0.1:8000/search/books/suggest/?title_suggest__completion=temp
```

Response

```
{
  "_shards": {
    "successful": 1,
    "total": 1,
    "failed": 0
  },
  "title_suggest": [
    {
      "length": 4,
      "text": "temp",
      "options": [
        {
          "text": "Tempora voluptates distinctio facere ",
          "_index": "book",
          "_score": 1.0,
          "_id": "1000087",
          "_type": "book_document",
          "_source": {
            "description": null,
            "summary": "Veniam dolores recusandae maxime laborum earum.",
            "id": 1000087,
            "state": "cancelled",
            "authors": [
              "Jayden van Luyssel",
              "Yassin van Rooij",
              "Florian van 't Erve",
              "Mats van Nimwegen",
              "Wessel Keltenie"
            ],
            "title": "Tempora voluptates distinctio facere."
          }
        },
        {
          "text": "Tempore sapiente repellat alias ad corrupti",
          "_index": "book",
          "_score": 1.0,
          "_id": "29",
          "_type": "book_document",
          "_source": {
            "description": null,
```

(continues on next page)

(continued from previous page)

```

        "summary": "Dolores minus architecto iure fugit qui sed.",
        "id": 29,
        "state": "cancelled",
        "authors": [
            "Wout van Northeim",
            "Lenn van Vliet-Kuijpers",
            "Tijs Mulder"
        ],
        "title": "Tempore sapiente repellat alias ad."
    },
    {
        "text": "Temporibus exercitationem minus expedita",
        "_index": "book",
        "_score": 1.0,
        "_id": "17",
        "_type": "book_document",
        "_source": {
            "description": null,
            "summary": "A laborum alias voluptates tenetur sapiente modi.
→",
            "id": 17,
            "state": "cancelled",
            "authors": [
                "Juliette Estey",
                "Keano de Keijzer",
                "Koen Scheffers",
                "Florian van 't Erve",
                "Tara Oversteeg",
                "Mats van Nimwegen"
            ],
            "title": "Temporibus exercitationem minus expedita."
        }
    }
],
"offset": 0
}
]
}

```

12.7.1.10.2.5 Term

Request

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

Response

```

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

```

(continues on next page)

(continued from previous page)

```

"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
  }
]
}

```

12.7.1.10.2.6 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
  }
}

```

12.7.1.11 Functional suggestions

If native suggestions are not good enough for you, use functional suggesters.

Configuration is very similar to native suggesters.

12.7.1.11.1 Document definition

Obviously, different filters require different approaches. For instance, when using functional completion prefix filter, the best approach is to use keyword field of the Elasticsearch. While for match completion, Ngram fields work really well.

The following example indicates Ngram analyzer/filter usage.

search_indexes/documents/book.py

```
from django.conf import settings
from django_elasticsearch_dsl import Document, Index, fields

from elasticsearch_dsl import analyzer
from elasticsearch_dsl.analysis import token_filter

from books.models import Book

edge_ngram_completion_filter = token_filter(
    'edge_ngram_completion_filter',
    type="edge_ngram",
    min_gram=1,
    max_gram=20
)

edge_ngram_completion = analyzer(
    "edge_ngram_completion",
    tokenizer="standard",
    filter=["lowercase", edge_ngram_completion_filter]
)

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(Document):
    """Book 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')
```

(continues on next page)

(continued from previous page)

```

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

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

# ...

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

    model = Book # The model associate with this Document

```

12.7.1.11.2 ViewSet definition

Note: The suggerer filter backends shall come as last ones.

Functional suggesters for the view are configured in `functional_suggester_fields` property.

In the example below, the `title_suggest` is the name of the GET query param which points to the `title.raw` field of the `BookDocument` document. For the `title_suggest` the allowed suggerer is `FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX`. For Ngram match we have the `title_suggest_match` field, which points to `title.edge_ngram_completion` field of the same document. For `title_suggest_match` the allowed suggerer is `FUNCTIONAL_SUGGESTER_COMPLETION_MATCH`.

URL shall be constructed in the following way:

```
/search/books/functional_suggest/?{QUERY_PARAM}__{SUGGESTER_NAME}={VALUE}
```

Example for `completion_prefix` suggerer:

```
GET http://localhost:8000/search/books/functional_suggest/?title_suggest_prefix__
↪completion_prefix=Temp
```

However, since we have `default_suggester` defined we can skip the `__{SUGGESTER_NAME}` part (if we want `completion_prefix` suggerer functionality). Thus, it might be written as short as:

```
GET http://localhost:8000/search/books/functional_suggest/?title_suggest_prefix=Temp
```

Example for `completion_match` suggerer:

```
GET http://localhost:8000/search/books/functional_suggest/?title_suggest_match__
↪completion_match=Temp
```

However, since we have `default_suggester` defined we can skip the `__{SUGGESTER_NAME}` part (if we want `completion_match` suggester functionality). Thus, it might be written as short as:

```
GET http://localhost:8000/search/books/functional_suggest/?title_suggest_match=Temp
```

search_indexes/viewsets/book.py

```
from django_elasticsearch_dsl_drf.constants import (
    # ...
    FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
    FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    SuggesterFilterBackend,
)

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

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        IdsFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
        FacetedSearchFilterBackend,
        HighlightBackend,
        FunctionalSuggesterFilterBackend, # This should come as last
    ]

    # ...

    # Functional suggester fields
    functional_suggester_fields = {
        'title_suggest': {
            'field': 'title.raw',
            'suggesters': [
                FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
            ],
            'default_suggester': FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
            'options': {
                'size': 25,
                'from': 0,
            }
        },
        'title_suggest_match': {
            'field': 'title.edge_ngram_completion',
            'suggesters': [FUNCTIONAL_SUGGESTER_COMPLETION_MATCH],
            'default_suggester': FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
        }
    }
}
```

Note: Note, that in `functional_suggester_fields['title_suggest']['options']` there are two

params: size and from. They control the query size and the offset of the generated functional suggest query.

12.7.1.12 Highlighting

Highlighters enable you to get highlighted snippets from one or more fields in your search results so you can show users where the query matches are.

ViewSet definition

```
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    HighlightBackend,
)

from ..documents import BookDocument
from ..serializers import BookDocumentSimpleSerializer

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

    document = BookDocument
    # serializer_class = BookDocumentSerializer
    serializer_class = BookDocumentSimpleSerializer
    lookup_field = 'id'
    filter_backends = [
        # ...
        HighlightBackend,
    ]

    # ...

    # Define highlight fields
    highlight_fields = {
        'title': {
            'enabled': True,
            'options': {
                'pre_tags': ["<b>"],
                'post_tags': ["</b>"],
            }
        },
        'summary': {
            'options': {
                'fragment_size': 50,
                'number_of_fragments': 3
            }
        },
        'description': {},
    }

    # ...
```

Request

```
GET http://127.0.0.1:8000/search/books/?search=optimisation&highlight=title&
-highlight=summary
```

(continues on next page)

Response

```

{
  "count": 1,
  "next": null,
  "previous": null,
  "facets": {
    "_filter_publisher": {
      "publisher": {
        "buckets": [
          {
            "key": "Self published",
            "doc_count": 1
          }
        ],
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0
      },
      "doc_count": 1
    }
  },
  "results": [
    {
      "id": 999999,
      "title": "Performance optimisation",
      "description": null,
      "summary": "Ad animi adipisci libero facilis iure totam
        impedit. Facilis maiores quae qui magnam dolores.
        Veritatis quia amet porro voluptates iure quod
        impedit. Dolor voluptatibus maiores at libero
        magnam.",
      "authors": [
        "Artur Barseghyan"
      ],
      "publisher": "Self published",
      "publication_date": "1981-04-29",
      "state": "cancelled",
      "isbn": "978-1-7372176-0-2",
      "price": 40.51,
      "pages": 162,
      "stock_count": 30,
      "tags": [
        "Guide",
        "Poetry",
        "Fantasy"
      ],
      "highlight": {
        "title": [
          "Performance <b>optimisation</b>"
        ]
      },
      "null_field": null
    }
  ]
}

```

12.7.1.13 Pagination

12.7.1.13.1 Page number pagination

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

Example:

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

12.7.1.13.2 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(DocumentViewSet):
    """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
```

12.7.1.13.3 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)
```

(continues on next page)

(continued from previous page)

```
__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`.

12.8 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

- *Nested fields usage examples*
 - *Example app*
 - * *Sample models*
 - * *Sample document*
 - *Index definition*
 - *Settings*
 - *Document index*
 - * *Sample serializer*
 - * *Sample view*
 - * *Usage example*
 - *Sample queries*
 - *Search*
 - *Nested filtering*
 - *Nested search*
 - *Sample models*
 - *Sample document*

- *Sample view*
- *Sample request*
- *Filtering*
- *Ordering*
- * *Suggestions*
- * *Nested aggregations/facets*

12.8.1 Example app

12.8.1.1 Sample models

books/models/continent.py

```
from django.db import models

from six import python_2_unicode_compatible

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

    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.
        """
```

(continues on next page)

(continued from previous page)

```

return {
    'lat': self.latitude,
    'lon': self.longitude,
}

```

books/models/country.py

```

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

    name = models.CharField(max_length=255)
    info = models.TextField(null=True, blank=True)
    continent = models.ForeignKey(
        'books.Continent',
        on_delete=models.CASCADE
    )
    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."""

```

(continues on next page)

(continued from previous page)

```

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

```

from django.db import models
from django_elasticsearch_dsl_drf.wrappers import dict_to_obj

from six import python_2_unicode_compatible

@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', on_delete=models.CASCADE)
    latitude = models.DecimalField(
        null=True,

```

(continues on next page)

(continued from previous page)

```

        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,
        }

    @property
    def country_indexing(self):
        """Country data (nested) for indexing.

        Example:

        >>> mapping = {
        >>>     'country': {
        >>>         'name': 'Netherlands',
        >>>         'city': {
        >>>             'name': 'Amsterdam',
        >>>         }
        >>>     }
        >>> }

        :return:
        """
        wrapper = dict_to_obj({
            'name': self.city.country.name,

```

(continues on next page)

(continued from previous page)

```

        'city': {
            'name': self.city.name
        }
    })

    return wrapper

@property
def continent_indexing(self):
    """Continent data (nested) for indexing.

    Example:

    >>> mapping = {
    >>>     'continent': {
    >>>         'name': 'Asia',
    >>>         'country': {
    >>>             'name': 'Netherlands',
    >>>             'city': {
    >>>                 'name': 'Amsterdam',
    >>>             }
    >>>         }
    >>>     }
    >>> }

    :return:
    """
    wrapper = dict_to_obj({
        'name': self.city.country.continent.name,
        'country': {
            'name': self.city.country.name,
            'city': {
                'name': self.city.name,
            }
        }
    })

    return wrapper

```

12.8.1.2 Sample document

12.8.1.2.1 Index definition

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

12.8.1.2.1.1 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',
}
```

12.8.1.2.1.2 Document index

search_indexes/documents/address.py

```
from django.conf import settings

from django_elasticsearch_dsl import Document, 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(Document):
    """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(),
```

(continues on next page)

(continued from previous page)

```

        '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'
                )
            }
        )
    }
)

# Country object
country = fields.NestedField(
    attr='country_indexing',
    properties={
        'name': StringField(
            analyzer=html_strip,
            fields={

```

(continues on next page)

(continued from previous page)

```

        'raw': KeywordField(),
        'suggest': fields.CompletionField(),
    }
),
'city': fields.ObjectField(
    properties={
        'name': StringField(
            analyzer=html_strip,
            fields={
                'raw': KeywordField(),
            },
        ),
    },
),
),
},
)

# Continent object
continent = fields.NestedField(
    attr='continent_indexing',
    properties={
        'name': StringField(
            analyzer=html_strip,
            fields={
                'raw': KeywordField(),
                'suggest': fields.CompletionField(),
            }
        ),
    },
'country': fields.NestedField(
    properties={
        'name': StringField(
            analyzer=html_strip,
            fields={
                'raw': KeywordField(),
            }
        ),
        'city': fields.NestedField(
            properties={
                'name': StringField(
                    analyzer=html_strip,
                    fields={
                        'raw': KeywordField(),
                    }
                )
            }
        )
    }
),
)

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

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

    model = Address # The model associate with this Document

```

12.8.1.3 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',
            'street',
            'house_number',
            'appendix',
            'zip_code',
            'city',
            'country',
            'continent',
            'location',
        )
```

12.8.1.4 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_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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


class AddressDocumentViewSet(DocumentViewSet):
```

(continues on next page)

(continued from previous page)

```

"""The AddressDocument view."""

document = AddressDocument
serializer_class = AddressDocumentSerializer
lookup_field = 'id'
filter_backends = [
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    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',
}
# Nested filtering fields
nested_filter_fields = {
    'continent_country': {
        'field': 'continent.country.name.raw',
        'path': 'continent.country',
    },
    'continent_country_city': {
        'field': 'continent.country.city.name.raw',
        'path': 'continent.country.city',
    },
}
# 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',
}

```

(continues on next page)

(continued from previous page)

```

        '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,
            ],
        },
    }

    # Facets
    faceted_search_fields = {
        'city': {
            'field': 'city.name.raw',
            'enabled': True,
        },
        'country': {
            'field': 'city.country.name.raw',
            'enabled': True,
        },
    }
}

```

12.8.1.5 Usage example

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

12.8.1.5.1 Sample queries

12.8.1.5.1.1 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 "Piccadilly".

```
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
```

12.8.1.5.1.2 Nested filtering

Filter documents by nested field

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

```
http://127.0.0.1:8000/search/addresses/?continent_country=Armenia
```

Filter documents by field (`continent.country.city`) "Amsterdam".

```
http://127.0.0.1:8000/search/addresses/?continent_country_city=Amsterdam
```

12.8.1.5.1.3 Nested search

For nested search, let's have another example.

12.8.1.5.1.4 Sample models

books/models/city.py

```
from django.db import models
from six import python_2_unicode_compatible

@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,
```

(continues on next page)

(continued from previous page)

```

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

```

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)

```

12.8.1.5.1.5 Sample document

documents/city.py

```

from django.conf import settings

from django_elasticsearch_dsl import Document, Index, fields
from django_elasticsearch_dsl_drf.compat import KeywordField, StringField

from books.models import City

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 CityDocument(Document):
    """City Elasticsearch document.

```

(continues on next page)

(continued from previous page)

```

This document has been created purely for testing out complex fields.
"""

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

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

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

info = StringField(analyzer=html_strip)

# *****
# ***** Nested fields for search and filtering *****
# *****

# City object
country = fields.NestedField(
    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')

# *****
# ***** Other complex fields for search and filtering *****
# *****

boolean_list = fields.ListField(
    StringField(attr='boolean_list_indexing')
)

datetime_list = fields.ListField(
    StringField(attr='datetime_list_indexing')
)

float_list = fields.ListField(
    StringField(attr='float_list_indexing')
)

integer_list = fields.ListField(
    StringField(attr='integer_list_indexing')
)

```

(continues on next page)

(continued from previous page)

```

)

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

    model = City # The model associate with this Document

```

12.8.1.5.1.6 Sample view

viewsets/city.py

```

from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
    LOOKUP_FILTER_GEO_POLYGON,
    LOOKUP_FILTER_GEO_BOUNDING_BOX,
    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.viewsets import DocumentViewSet

from ..documents import CityDocument
from ..serializers import CityDocumentSerializer

class CityDocumentViewSet(BaseDocumentViewSet):
    """The CityDocument view."""

    document = CityDocument
    serializer_class = CityDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
        GeoSpatialFilteringFilterBackend,
        GeoSpatialOrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SuggesterFilterBackend,
    ]
    pagination_class = LimitOffsetPagination
    # Define search fields
    search_fields = (
        'name',
        'info',
    )

    search_nested_fields = {

```

(continues on next page)

(continued from previous page)

```

        'country': {
            'path': 'country',
            'fields': ['name'],
        }
    }

    # Define filtering fields
    filter_fields = {
        'id': None,
        'name': 'name.raw',
        'country': '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,
        'name': None,
        'country': 'country.name.raw',
    }

    # Define ordering fields
    geo_spatial_ordering_fields = {
        'location': None,
    }

    # Specify default ordering
    ordering = (
        'id',
        'name.raw',
        'country.name.raw',
    )

    # Suggester fields
    suggester_fields = {
        'name_suggest': {
            'field': 'name.suggest',
            'suggesters': [
                SUGGESTER_COMPLETION,
            ],
        },
        'country_suggest': {
            'field': 'country.name.suggest',
            'suggesters': [
                SUGGESTER_COMPLETION,
            ],
        },
    }
}

```

12.8.1.5.1.7 Sample request

Request

```
GET http://127.0.0.1:8000/search/cities/?search=Switzerland
```

12.8.1.5.1.8 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
```

12.8.1.5.1.9 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
```

12.8.1.6 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 `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
```

12.8.1.7 Nested aggregations/facets

At the moment, nested aggregations/facets are not supported out of the box. Out of the box support will surely land in the package one day, but for now, there's a simple and convenient way of implementing nested aggregations/facets with minimal efforts. Consider the following example.

search_indexes/backends/nested_continents.py

```
from django_elasticsearch_dsl_drf.filter_backends.mixins import (
    FilterBackendMixin,
)
from rest_framework.filters import BaseFilterBackend

class NestedContinentsBackend(BaseFilterBackend, FilterBackendMixin):
    """Adds nesting to continents."""

    faceted_search_param = 'nested_facet'

    def get_faceted_search_query_params(self, request):
        """Get faceted search query params.

        :param request: Django REST framework request.
        :type request: rest_framework.request.Request
        :return: List of search query params.
        :rtype: list
        """
        query_params = request.query_params.copy()
        return query_params.getlist(self.faceted_search_param, [])

    def filter_queryset(self, request, queryset, view):
        """Filter the queryset.

        :param request: Django REST framework request.
        :param queryset: Base queryset.
        :param view: View.
        :type request: rest_framework.request.Request
        :type queryset: elasticsearch_dsl.search.Search
        :type view: rest_framework.viewsets.ReadOnlyModelViewSet
        :return: Updated queryset.
        :rtype: elasticsearch_dsl.search.Search
        """
        facets = self.get_faceted_search_query_params(request)

        if 'continent' in facets:
            queryset \
                .aggs\
                .bucket('continents',
                        'nested',
                        path='continent') \
                .bucket('continent_name',
                        'terms',
                        field='continent.name.raw',
                        size=10) \
                .bucket('counties',
                        'nested',
                        path='continent.country') \
                .bucket('country_name',
                        'terms',
                        field='continent.country.name.raw',
```

(continues on next page)

(continued from previous page)

```

        size=10) \
        .bucket('city',
                'nested',
                path='continent.country.city') \
        .bucket('city_name',
                'terms',
                field='continent.country.city.name.raw',
                size=10)

    return queryset

```

The view will look as follows:

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_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

from ..backends import NestedContinentsBackend
from ..documents import AddressDocument
from ..serializers import AddressDocumentSerializer

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

    document = AddressDocument
    serializer_class = AddressDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FacetedSearchFilterBackend,
        FilteringFilterBackend,
        OrderingFilterBackend,
        SearchFilterBackend,
        GeoSpatialFilteringFilterBackend,
        GeoSpatialOrderingFilterBackend,
        NestedContinentsBackend,
        NestedFilteringFilterBackend,
        DefaultOrderingFilterBackend,
        SuggesterFilterBackend,
    ]

```

(continues on next page)

(continued from previous page)

```

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',
}
# Nested filtering fields
nested_filter_fields = {
    'continent_country': {
        'field': 'continent.country.name.raw',
        'path': 'continent.country',
    },
    'continent_country_city': {
        'field': 'continent.country.city.name.raw',
        'path': 'continent.country.city',
    },
}
# 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',
    },
}

```

(continues on next page)

(continued from previous page)

```

        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'city_suggest': {
        'field': 'city.name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'country_suggest': {
        'field': 'city.country.name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
}

# Facets
faceted_search_fields = {
    'city': {
        'field': 'city.name.raw',
        'enabled': True,
    },
    'country': {
        'field': 'city.country.name.raw',
        'enabled': True,
    },
}

```

12.9 More like this

More like this functionality.

12.9.1 Usage example

12.9.1.1 Sample document

```

from django.conf import settings

from django_elasticsearch_dsl import Document, Index, fields
from django_elasticsearch_dsl_drf.compat import KeywordField, StringField
from django_elasticsearch_dsl_drf.analyzers import edge_ngram_completion

from books.models import Book

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

# See Elasticsearch Indices API reference for available settings
INDEX.settings(

```

(continues on next page)

(continued from previous page)

```

    number_of_shards=1,
    number_of_replicas=1,
    blocks={'read_only_allow_delete': False}
)

@INDEX.doc_type
class BookDocument(Document):

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

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

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

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

    # ...

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

        model = Book  # The model associate with this Document

    def prepare_summary(self, instance):
        """Prepare summary."""
        return instance.summary[:32766]

```

12.9.1.2 Sample view

```

from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,

```

(continues on next page)

(continued from previous page)

```

        IdsFilterBackend,
        OrderingFilterBackend,
        PostFilterFilteringFilterBackend,
        SearchFilterBackend,
    )
    from django_elasticsearch_dsl_drf.viewsets import (
        DocumentViewSet,
        MoreLikeThisMixin,
    )

    from .serializers import BookDocumentSerializer

    class BookMoreLikeThisDocumentViewSet(DocumentViewSet,
                                           MoreLikeThisMixin):
        """Same as BookDocumentViewSet, with more-like-this and no facets."""

        # ...

        document = BookDocument
        lookup_field = 'id'
        serializer_class = BookDocumentSerializer

        # ...

        filter_backends = [
            # ...
            FilteringFilterBackend,
            PostFilterFilteringFilterBackend,
            IdsFilterBackend,
            OrderingFilterBackend,
            SearchFilterBackend,
            # ...
        ]

        # More-like-this options
        more_like_this_options = {
            'fields': (
                'title.mlt',
                'summary.mlt',
                'description.mlt',
            )
        }

```

12.9.1.3 Sample request

```
http://localhost:8000/search/books-more-like-this-no-options/1007587/more_like_this/
```

12.9.1.4 Generated query

```
{
  "query": {
    "more_like_this": {
      "fields": [

```

(continues on next page)

(continued from previous page)

```
        "title.mlt",
        "summary.mlt",
        "description.mlt"
    ],
    "like": {
        "_index": "book",
        "_id": "1007587",
        "_type": "book_document"
    }
}
},
"from": 0,
"size": 14,
"sort": [
    "_score"
]
}
```

12.9.1.5 Options

Pretty much [all Elasticsearch more-like-this options](#) available. You might be particularly interested in the following:

- `min_term_freq`
- `max_query_terms`
- `unlike`
- `stop_words`

12.10 Global aggregations

Global aggregations (facets) are regular aggregations, which are not influenced by the search query/filter. They deliver results similar to *post_filter*.

12.10.1 Sample view

```
from django_elasticsearch_dsl_drf.filter_backends import (
    CompoundSearchFilterBackend,
    DefaultOrderingFilterBackend,
    FacetedSearchFilterBackend,
    OrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookCompoundSearchBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
```

(continues on next page)

(continued from previous page)

```

filter_backends = [
    # ...
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
    CompoundSearchFilterBackend,
    FacetedSearchFilterBackend,
    # ...
]

faceted_search_fields = {
    'state_global': {
        'field': 'state.raw',
        'enabled': True,
        'global': True,  # This makes the aggregation global
    },
}

```

12.10.2 Sample request

```
http://localhost:8000/search/books/?facet=state_global&state=rejected
```

12.10.3 Generated query

```

{
  "from": 0,
  "query": {
    "bool": {
      "filter": [
        {
          "terms": {
            "state.raw": [
              "rejected"
            ]
          }
        }
      ]
    }
  },
  "size": 25,
  "aggs": {
    "_filter_state_global": {
      "aggs": {
        "state_global": {
          "terms": {
            "field": "state.raw"
          }
        }
      }
    },
    "global": {

```

(continues on next page)

(continued from previous page)

```
    }
  },
  "sort": [
    "id",
    "title",
    "price"
  ]
}
```

12.10.4 Sample response

```
{
  "count": 25,
  "next": null,
  "previous": null,
  "facets": {
    "_filter_state_global": {
      "state_global": {
        "buckets": [
          {
            "doc_count": 29,
            "key": "not_published"
          },
          {
            "doc_count": 25,
            "key": "in_progress"
          },
          {
            "doc_count": 25,
            "key": "rejected"
          },
          {
            "doc_count": 21,
            "key": "cancelled"
          },
          {
            "doc_count": 17,
            "key": "published"
          }
        ],
        "sum_other_doc_count": 0,
        "doc_count_error_upper_bound": 0
      },
      "doc_count": 117
    }
  },
  "results": [
    {
      "id": 1007489,
      "title": "Cupiditate qui nulla itaque maxime impedit.",
      "description": null,
      "summary": "Aut recusandae architecto incidunt quaerat odio .",
      "authors": [
        "Evy Vermeulen",
        "Tycho Weijland",

```

(continues on next page)

(continued from previous page)

```

        "Rik Zeldenrust"
    ],
    "publisher": "Overdijk Inc",
    "publication_date": "2014-02-28",
    "state": "rejected",
    "isbn": "978-0-15-184366-4",
    "price": 6.53,
    "pages": 82,
    "stock_count": 30,
    "tags": [
        "Trilogy"
    ],
    "highlight": {},
    "null_field": null,
    "score": null
},
# ...
]
}

```

12.11 Configuration tweaks

12.11.1 Ignore certain Elasticsearch exceptions

```

class BookIgnoreIndexErrorsDocumentViewSet(DocumentViewSet):

    # ...
    ignore = [404]
    # ...

```

12.12 Source filtering backend

Allows to control how the `_source` field is returned with every hit.

By default operations return the contents of the `_source` field unless you have used the `stored_fields` parameter or if the `_source` field is disabled.

You can turn off `_source` retrieval by using the `source` parameter:

```

from django_elasticsearch_dsl_drf.filter_backends import (
    SourceBackend
)
from django_elasticsearch_dsl_drf.viewsets import (
    BaseDocumentViewSet,
)

# Local article document definition
from .documents import ArticleDocument

# Local article document serializer
from .serializers import ArticleDocumentSerializer

```

(continues on next page)

(continued from previous page)

```
class ArticleDocumentView(BaseDocumentViewSet):

    document = ArticleDocument
    serializer_class = ArticleDocumentSerializer
    filter_backends = [SourceBackend,]
    source = ["title"]
```

To disable `_source` retrieval set to False:

```
# ...
source = False
# ...
```

The `source` also accepts one or more wildcard patterns to control what parts of the `_source` should be returned:

```
# ...
source = ["title", "author.*"]
# ...
```

Finally, for complete control, you can specify both *includes* and *excludes* patterns:

```
# ...
source = {
    "includes": ["title", "author.*"],
    "excludes": [ "*.description" ]
}
# ...
```

Note: Source can make queries lighter. However, it can break current functionality. Use it with caution.

12.13 Indexing troubleshooting

When indexing lots of data (millions of records), you might get timeout exceptions.

A couple of possible solutions (complementary) are listed below. All of them are independent and not strictly related to each other. Thus, you may just use one or a couple or all of them. It's totally up to you.

If you want to test what works best for you, use [this test dataset \(Postgres\)](#) containing 1.8 million location records for `search_indexes.documents.location.LocationDocument` document.

12.13.1 Timeout

For re-indexing, you might want to increase the timeout to avoid time-out exceptions.

To do that, make a new settings file (*indexing*) and add the following:

settings/indexing.py

```
from .base import * # Import from your main/production settings.

# Override the elasticsearch configuration and provide a custom timeout
```

(continues on next page)

(continued from previous page)

```
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'localhost:9200',
        'timeout': 60, # Custom timeout
    },
}
```

Then rebuild your search index specifying the indexing settings:

```
./manage.py search_index --rebuild -f --settings=settings.indexing
```

Note, that you may as well specify the timeout in your global settings. However, if you're happy with how things work in production (except for the indexing part), you may do as suggested (separate indexing settings).

12.13.2 Chunk size

Note, that this feature is (yet) *only available in the forked version* [barseghyanartur/django-elasticsearch-dsl](#).

Install it as follows:

```
pip install https://github.com/barseghyanartur/django-elasticsearch-dsl/archive/mjl-
↪index-speedup-2-additions.zip
```

Specify the *chunk_size* param as follows (we set *chunk_size* to 50 in this case):

```
./manage.py search_index --rebuild -f --chunk-size=50
```

12.13.3 Use parallel indexing

Parallel indexing speeds things up (drastically). In my tests I got a speedup boost of 66 percent on 1.8 million records.

Note, that this feature is (yet) *only available in the forked versions* [barseghyanartur/django-elasticsearch-dsl](#), or [mjl/django-elasticsearch-dsl](#).

Install it as follows:

barseghyanartur/django-elasticsearch-dsl fork

```
pip install https://github.com/barseghyanartur/django-elasticsearch-dsl/archive/mjl-
↪index-speedup-2-additions.zip
```

mjl/django-elasticsearch-dsl fork

```
pip install https://github.com/mjl/django-elasticsearch-dsl/archive/mjl-index-speedup.
↪zip
```

In order to make use of it, define set *parallel_indexing* to True on the document meta.

yourapp/documents.py

```
class LocationDocument(Document):

    # ...

    class Meta(object):
```

(continues on next page)

(continued from previous page)

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

model = Location
parallel_indexing = True
```

12.13.4 Limit the number of items indexed at once

This is very close to the `chunk_size` shown above, but might work better on heavy querysets. Instead of processing entire queryset at once, it's sliced instead. So, if you have 2 million records in your queryset and you wish to index them by chunks of 20 thousands at once, specify the `queryset_pagination` on the document meta:

yourapp/documents.py

```
class LocationDocument(Document):

    # ...

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

        model = Location
        queryset_pagination = 50
```

You may even make it dynamic based on the settings loaded. So, for instance, you may have it set to `None` in production (if you were happy with how things were) and provide a certain value for it in the dedicated indexing settings (as already has been mentioned above).

settings/base.py

```
# Main/production settings
ELASTICSEARCH_DSL_QUERYSET_PAGINATION = None
```

settings/indexing.py

```
# Indexing only settings
ELASTICSEARCH_DSL_QUERYSET_PAGINATION = 1000
```

yourapp/documents.py

```
from django.conf import settings

# ...

class LocationDocument(Document):

    # ...

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

        model = Location
        queryset_pagination = settings.ELASTICSEARCH_DSL_QUERYSET_PAGINATION
```

12.14 FAQ

You will find a lot of useful information in the [documentation](#).

Additionally, all raised issues that were questions have been marked as *question*, so you could take a look at the [closed \(question\) issues](#).

12.14.1 Questions and answers

Question

- Is it possible to search sub string in word?
- How to implement partial/fuzzy search?

Answer

Yes. There are many ways doing this in Elasticsearch.

To mention a couple:

- You could use partial matching using NGrams. Partially shown [here](#). [The basic idea](#).
- Use [contains](#) functional filter.

Question

Can we use Django REST Framework `serializers.ModelSerializer` directly?

Answer

No, but you could use `serializers.Serializer`. [Read the docs](#).

Question

How can I order search results overall relevance

Answer

That's `_score`. See the following [example](#).

```
ordering = ('_score', 'id', 'title', 'price',)
```

In the given example, results are sorted by the `score` (which is relevance), then by `id`, `title` and `price`.

Question

How can I separate my development/production/acceptance/test indexes?

Answer

It's documented [here](#).

Question

How can I sync my database with Elasticsearch indexes.

Answer

It's documented [here](#).

12.15 frontend demo for django-elasticsearch-dsl-drf

Frontend demo for django-elasticsearch-dsl-drf

Based on `Book` model, `BookDocument` and `BookFrontendDocumentViewSet` viewset.

12.15.1 Quick start

From the project root directory.

12.15.1.1 Install the django requirements

Since project supports Django versions from 1.8 to 2.1, you may install any version you want.

To install latest LTS version, do:

```
pip install -r examples/requirements/django_1_11.txt
```

12.15.1.2 Install Elasticsearch requirements

Since project supports Elasticsearch versions from 2.x to 6.x, you may install any version you want.

To install requirements for 6.x, do:

```
pip install -r examples/requirements/elastic_6x.txt
```

12.15.1.3 Run Elasticsearch

It's really easy using Docker.

To run 6.3.2 using Docker, do:

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:6.3.2
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.
↳enabled=false" docker.elastic.co/elasticsearch/elasticsearch:6.3.2
```

12.15.1.4 Build Elasticsearch index

First, create some test data:

```
./scripts/create_test_data.sh
```

Then build Elasticsearch index:

```
./scripts/rebuild_index.sh
```

12.15.1.5 Install React requirements

Note, that you should be using NodeJS > 7.5.

Typically, you would first do:

```
nvm use 9
```

Then run the installer:

```
./scripts/yarn_install.sh
```

12.15.1.6 Run Django

The following script would run the Django server which is used by the demo app.

```
./scripts/runserver.sh
```

12.15.1.7 Run React demo app

Finally, run the React demo app:

```
./scripts/frontend.sh
```

Open <http://localhost:3000> to view the frontend in the browser.

12.16 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.

12.16.1 0.20

2019-08-17

- Adding Elasticsearch 7.x support.

12.16.2 0.19

2019-08-06

Note: Dropping support for Elasticsearch versions prior 6.x. This is unfortunate, but this project depends on the upstream `django-elasticsearch-dsl` where as of version 6.4.x the support for older Elasticsearch versions was dropped. Use `django-elasticsearch-dsl-drf` version 0.18 if you need to work with 5.x or 2.x.

- Dropping support for Elasticsearch versions prior to 6.x.

12.16.3 0.18

2019-06-26

Note: Support for Django versions prior 1.11 has been dropped. Support for Django REST Framework prior 3.9 has been dropped.

- Dropping support for Django versions prior 1.11.
- Dropping support for Django REST Framework versions prior 3.9.
- Fix Django REST Framework deprecations.

12.16.4 0.17.7

2019-05-30

Note: Support for Django 1.8, 1.9 and 1.10 will be dropped in the next release. As usual, compatibility shims won't be removed directly. The change will affect the test matrix only first.

- Prevent unicode errors in tests on Python 2.7.
- Fixes in occasionally failing search test (`test_search` and `test_filtering_geo_spatial`).
- Working travis.
- Fixed issue with errors on empty `ids` filter.

12.16.5 0.17.6

2019-04-08

- Minor fixes.
- Additions to the docs.

12.16.6 0.17.5

2019-04-03

Note: Dropping support for Python 3.4. As of this version everything works, but no longer tested.

- Minor fixes.

- Dropping Python 3.4 support.
- Django 2.2 support.

12.16.7 0.17.4

2019-03-13

- Source backend.

12.16.8 0.17.3

2019-02-08

- Obey object permissions.

12.16.9 0.17.2

2019-01-07

- Add nested ordering.

12.16.10 0.17.1

2018-12-12

- Skipping the new context suggester tests for Elasticsearch 2.x and a number of other 2.x related fixes in tests.
- A number of 5.x fixes in tests.

12.16.11 0.17

2018-12-12

Note: Release supported by [whythawk](#).

- Added support for context suggesters (*category* and *geo*). Note, that this functionality is available for Elasticsearch 5.x and 6.x (thus, not for Elasticsearch 2.x).
- Added support for *size* attribute on suggesters.

12.16.12 0.16.3

2018-10-31

Note: Release dedicated to Charles Aznavour.

- Make it possible to ignore certain Elastic exceptions by providing the appropriate `ignore` argument (on the view level). Default behaviour is intact. Set it to a list of integers (error codes) if you need it so.

12.16.13 0.16.2

2018-09-21

- Tested yet untested `pip_helpers` module.
- More tests.

12.16.14 0.16.1

2018-09-18

- Make it possible to control the size of the functional suggester queries.

12.16.15 0.16

2018-09-10

Note: This release contains minor backwards incompatible changes. You might need to update your code if you have been making use of nested search.

Old way of declaring nested search fields

```
search_nested_fields = {
    'country': ['name'],
    'country.city': ['name'],
}
```

New way of declaring nested search fields

```
search_nested_fields = {
    'country': {
        'path': 'country',
        'fields': ['name'],
    },
    'city': {
        'path': 'country.city',
        'fields': ['name'],
    },
}
```

- Changes in nested search. This affects usage of both historical `SearchFilterBackend` and `CompoundSearchFilterBackend`. Update your code accordingly.
- Take meta property `using` of the document `Meta` into consideration.

12.16.16 0.15.1

2018-08-22

- More tests.
- Fixes in docs.

12.16.17 0.15

2018-08-10

- Global aggregations.

12.16.18 0.14

2018-08-06

- More like this support through detail action.

12.16.19 0.13.2

2018-08-03

- Successfully tested against Python 3.7 and Django 2.1.
- Unified the base `BaseSearchFilterBackend` class.
- Minor clean up and fixes in docs.
- Upgrading test suite to modern versions (pytest, tox, factory_boy, Faker). Removing unused dependencies from requirements (drf-extensions).
- Fixed missing PDF generation in offline documentation (non ReadTheDocs). The `rst2pdf` package (which does not support Python 3) has been replaced with `rinohype` package (which does support Python 3).

12.16.20 0.13.1

2018-07-26

- Minor fix in suggesters on Elasticsearch 6.x.

12.16.21 0.13

2018-07-23

Note: Release dedicated to Guido van Rossum, the former Python BDFL, who resigned from his BDFL position recently. Guido knew it better than we all do. His charisma, talent and leadership will be certainly missed a lot by the community. Thumbs up again for the best BDFL ever.

- The `SimpleQueryStringSearchFilterBackend` backend has been implemented.
- Minor fixes in the `MultiMatchSearchFilterBackend` backend.

12.16.22 0.12

2018-07-21

- New-style Search Filter Backends. Old style `SearchFilterBackend` is still supported (until at least version 0.16), but is deprecated. Migrate to `CompoundSearchFilterBackend`. `MultiMatchSearchFilterBackend` introduced (the name speaks for itself).

- From now on, your views would also work with model- and object-level permissions of the Django REST Framework (such as `DjangoModelPermissions`, `DjangoModelPermissionsOrAnonReadOnly` and `DjangoObjectPermissions`). Correspondent model or object would be used for that. If you find it incorrect in your case, write custom permissions and declare the explicitly in your view-sets.
- Fixed geo-spatial `geo_distance` ordering for Elastic 5.x. and 6.x.
- Fixes occasionally failing tests.

12.16.23 0.11

2018-07-15

Note: This release contains backwards incompatible changes. You should update your Django code and front-end parts of your applications that were relying on the complex queries using `|` and `:` chars in the GET params.

Note: If you have used custom filter backends using `SEPARATOR_LOOKUP_VALUE`, `SEPARATOR_LOOKUP_COMPLEX_VALUE` or `SEPARATOR_LOOKUP_COMPLEX_MULTIPLE_VALUE` constants or `split_lookup_complex_value` helper method of the `FilterBackendMixin`, you most likely want to run your functional tests to see if everything still works.

Note: Do not keep things as they were in your own fork, since new search backends will use the `|` and `:` symbols differently.

Examples of old API requests vs new API requests

Note: Note, that `|` and `:` chars were mostly replaced with `__` and `,`.

Old API requests

```
http://127.0.0.1:8080/search/publisher/?search=name|reilly&search=city|london
http://127.0.0.1:8000/search/publishers/?location__geo_distance=100000km|12.04|-63.93
http://localhost:8000/api/articles/?id__terms=1|2|3
http://localhost:8000/api/users/?age__range=16|67|2.0
http://localhost:8000/api/articles/?id__in=1|2|3
http://localhost:8000/api/articles/?location__geo_polygon=40,-70|30,-80|20,-90|_
↪name:myname|validation_method:IGNORE_MALFORMED
```

New API requests

```
http://127.0.0.1:8080/search/publisher/?search=name:reilly&search=city:london
http://127.0.0.1:8000/search/publishers/?location__geo_distance=100000km__12.04__-63.
↪93
http://localhost:8000/api/articles/?id__terms=1__2__3
http://localhost:8000/api/users/?age__range=16__67__2.0
http://localhost:8000/api/articles/?id__in=1__2__3
http://localhost:8000/api/articles/?location__geo_polygon=40,-70__30,-80__20,-90__
↪name,myname__validation_method,IGNORE_MALFORMED
```

- `SEPARATOR_LOOKUP_VALUE` has been removed. Use `SEPARATOR_LOOKUP_COMPLEX_VALUE` and `SEPARATOR_LOOKUP_COMPLEX_MULTIPLE_VALUE` instead.

- `SEPARATOR_LOOKUP_NAME` has been added.
- The method `split_lookup_complex_value` has been removed. Use `split_lookup_complex_value` instead.
- Default filter lookup option is added. In past, if no specific lookup was provided and there were multiple values for a single field to filter on, by default `terms` filter was used. The `term` lookup was used by default in similar situation for a single value to filter on. It's now possible to declare default lookup which will be used when no lookup is given.
- Removed deprecated `views` module. Import from `viewsets` instead.
- Removed undocumented `get_count` helper from `helpers` module.

12.16.24 0.10

2018-07-06

- Elasticsearch 6.x support.
- Minor fixes.

12.16.25 0.9

2018-07-04

- Introduced `post_filter` support.
- Generalised the `FilteringFilterBackend` backend. Both `PostFilterFilteringFilterBackend` and `NestedFilteringFilterBackend` backends are now primarily based on it.
- Reduced Elastic queries from 3 to 2 when using `LimitOffsetPagination`.

12.16.26 0.8.4

2018-06-27

Note: Release supported by [Goldmund](#), [Wyldebeast](#) & [Wunderliebe](#).

- Added `NestedFilteringFilterBackend` backend.
- Documentation updated with examples of implementing a nested aggregations/facets.

12.16.27 0.8.3

2018-06-25

- It's possible to retrieve original dictionary from `DictionaryProxy` object.
- Added helper wrappers and helper functions as a temporary fix for issues in the `django-elasticsearch-dsl`.

12.16.28 0.8.2

2018-06-05

- Minor fixes.

12.16.29 0.8.1

2018-06-05

- Fixed wrong filter name in functional suggesters results into an error on Django 1.10 (and prior).
- Documentation improvements.

12.16.30 0.8

2018-06-01

Note: Release supported by [Goldmund](#), [Wyldebeast](#) & [Wunderliebe](#).

Note: This release contain minor backwards incompatible changes. You should update your code.

- (1) `BaseDocumentViewSet` (which from now on does not contain `suggest` functionality) has been renamed to `DocumentViewSet` (which does contain `suggest` functionality).
- (2) You should no longer import from `django_elasticsearch_dsl_drf.views`. Instead, import from `django_elasticsearch_dsl_drf.viewsets`.

-
- Deprecated `django_elasticsearch_dsl_drf.views` in favour of `django_elasticsearch_dsl_drf.viewsets`.
 - `Suggest` action/method has been moved to `SuggestMixin` class.
 - `FunctionalSuggestMixin` class introduced which resembled functionality of the `SuggestMixin` with several improvements/additions, such as advanced filtering and context-aware suggestions.
 - You can now define a default suggester in `suggester_fields` which will be used if you do not provide suffix for the filter name.

12.16.31 0.7.2

2018-05-09

Note: Release dedicated to the Victory Day, the victims of the Second World War and Liberation of Shushi.

- Django REST framework 3.8.x support.

12.16.32 0.7.1

2018-04-04

Note: Release supported by [Goldmund, Wyldebeast & Wunderliebe](#).

- Add query *boost* support for search fields.

12.16.33 0.7

2018-03-08

Note: Dear ladies, congratulations on [International Women's Day](#)

- CoreAPI/CoreSchema support.

12.16.34 0.6.4

2018-03-05

- Minor fix: explicitly use DocType in the ViewSets.

12.16.35 0.6.3

2018-01-03

- Minor fix in the search backend.
- Update the year in the license and code.

12.16.36 0.6.2

2017-12-29

- Update example project (and the tests that are dependant on the example project) to work with Django 2.0.
- Set minimal requirement for `django-elasticsearch-dsl` to 3.0.

12.16.37 0.6.1

2017-11-28

- Documentation fixes.

12.16.38 0.6

2017-11-28

- Added highlight backend.
- Added nested search functionality.

12.16.39 0.5.1

2017-10-18

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

12.16.40 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.

12.16.41 0.4.4

2017-10-02

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

12.16.42 0.4.3

2017-09-28

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

12.16.43 0.4.2

2017-09-28

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

12.16.44 0.4.1

2017-09-26

- Fixes in docs.

12.16.45 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).

12.16.46 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.

12.16.47 0.3.11

2017-09-21

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

12.16.48 0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

12.16.49 0.3.9

2017-09-12

- Python 2.x compatibility fix.

12.16.50 0.3.8

2017-09-12

- Fixes tests on some environments.

12.16.51 0.3.7

2017-09-07

- Docs fixes.

12.16.52 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.

12.16.53 0.3.5

2017-08-24

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

12.16.54 0.3.4

2017-08-23

- Minor fixes in the ordering backend.

12.16.55 0.3.3

2017-07-13

- Minor fixes and improvements.

12.16.56 0.3.2

2017-07-12

- Minor fixes and improvements.

12.16.57 0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

12.16.58 0.3

2017-07-11

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

12.16.59 0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

12.16.60 0.2.5

2017-07-11

- Fixes in documentation.

12.16.61 0.2.4

2017-07-11

- Fixes in documentation.

12.16.62 0.2.3

2017-07-11

- Fixes in documentation.

12.16.63 0.2.2

2017-07-11

- Fixes in documentation.

12.16.64 0.2.1

2017-07-11

- Fixes in documentation.

12.16.65 0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

12.16.66 0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

12.16.67 0.1.7

2017-06-25

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

12.16.68 0.1.6

2017-06-23

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

12.16.69 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.

12.16.70 0.1.4

2017-06-22

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

12.16.71 0.1.3

2017-06-21

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

12.16.72 0.1.2

2017-06-20

- Minor fixes in tests.

12.16.73 0.1.1

2017-06-20

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

12.16.74 0.1

2017-06-19

- Initial beta release.

12.17 django_elasticsearch_dsl_drf package

12.17.1 Subpackages

12.17.1.1 django_elasticsearch_dsl_drf.fields package

12.17.1.1.1 Submodules

12.17.1.1.2 django_elasticsearch_dsl_drf.fields.common module

Common fields.

```
class django_elasticsearch_dsl_drf.fields.common.BooleanField(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,
                                                             valida-
                                                             tors=None, al-
                                                             low_null=False)
```

Bases: `rest_framework.fields.BooleanField`

Object field.

get_value (*dictionary*)
Get value.

to_representation (*value*)

To representation.

class django_elasticsearch_dsl_drf.fields.common.**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.common.**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.common.**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.common.**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.common.**IPAddressField** (*protocol='both',*
***kwargs*)

Bases: rest_framework.fields.IPAddressField

Object field.

get_value (*dictionary*)

Get value.

to_representation (*value*)

To representation.

12.17.1.1.3 `django_elasticsearch_dsl_drf.fields.helpers` module

Helpers.

`django_elasticsearch_dsl_drf.fields.helpers.to_representation(value)`
 To representation.

12.17.1.1.4 `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,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    val-
                                                                    ida-
                                                                    tors=None,
                                                                    al-
                                                                    low_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,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    val-
                                                                    ida-
                                                                    tors=None,
                                                                    al-
                                                                    low_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,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    valida-
                                                                    tors=None,
                                                                    al-
                                                                    low_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,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    valida-
                                                                    tors=None,
                                                                    al-
                                                                    low_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,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    valida-
                                                                    tors=None,
                                                                    al-
                                                                    low_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.
```

12.17.1.1.5 Module contents

Fields.

```
class django_elasticsearch_dsl_drf.fields.BooleanField (read_only=False,
                                                    write_only=False,
                                                    required=None,
                                                    default=<class
                                                    'rest_framework.fields.empty'>,
                                                    initial=<class
                                                    'rest_framework.fields.empty'>,
                                                    source=None,          la-
                                                    bel=None, help_text=None,
                                                    style=None,          er-
                                                    ror_messages=None,
                                                    validators=None,      al-
                                                    low_null=False)
```

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, *er-*
ror_messages=None,
validators=None, *al-*
low_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, *er-*
ror_messages=None,
validators=None, *al-*
low_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, re-
                                                    quired=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, valida-
                                                    tors=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, re-
                                                       quired=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, al-
                                                       low_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,      re-
                                                    quired=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,      al-
                                                    low_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.

12.17.1.2 django_elasticsearch_dsl_drf.filter_backends package

12.17.1.2.1 Subpackages

12.17.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package

12.17.1.2.1.2 Submodules

12.17.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module

12.17.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module

12.17.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module

12.17.1.2.1.6 Module contents

12.17.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package

12.17.1.2.1.8 Submodules

12.17.1.2.1.9 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.viewsets 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,
>>>             ],
>>>         'default_lookup': LOOKUP_FILTER_WILDCARD,
>>>     }
>>> }
```

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

Apply *prefix* filter.

Syntax:

```
/endpoint/?field_name__prefix={value}
```

Example:

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

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.

Syntax:

```
/endpoint/?field_name__range={lower}__{upper}__{boost} /end-
point/?field_name__range={lower}__{upper}
```

Example:

```
http://localhost:8000/api/users/?age__range=16__67__2.0 http://localhost:8000/api/users/?age_
__range=16__67 http://localhost:8000/api/users/?age__range=16
```

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.

Syntax:

```
/endpoint/?field_name={value}
```

Example:

```
http://localhost:8000/api/articles/?tags=children
```

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.

Syntax:

```
/endpoint/?field_name__terms={value1}__{value2} /endpoint/?field_name__terms={value1}
```

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__terms=children__python` `http://localhost:8000/api/articles/?tags__terms=children`

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.

Syntax:

`/endpoint/?field_name__contains={value}`

Example:

`http://localhost:8000/api/articles/?state__contains=lis`

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.

Syntax:

`/endpoint/?field_name__endswith={value}`

Example:

`http://localhost:8000/api/articles/?tags__endswith=dren`

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.

Syntax:

/endpoint/?field_name__isnull={value1}__{value2} /endpoint/?field_name__exclude={value1}

Note, that number of values is not limited.

Example:

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

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.

Syntax:

/endpoint/?field_name__exists=true /endpoint/?field_name__exists=false

Example:

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

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.

Syntax:

/endpoint/?field_name__gt={value}__{boost} /endpoint/?field_name__gt={value}

Example:

`http://localhost:8000/api/articles/?id__gt=1__2.0` `http://localhost:8000/api/articles/?id__gt=1`

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.

Syntax:

`/endpoint/?field_name__gte={value}__{boost} /endpoint/?field_name__gte={value}`

Example:

`http://localhost:8000/api/articles/?id__gte=1__2.0 http://localhost:8000/api/articles/?id__gte=1`

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.

Syntax:

`/endpoint/?field_name__in={value1}__{value2} /endpoint/?field_name__in={value1}`

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__in=children__python`

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.

Syntax:

`/endpoint/?field_name__isnull=true /endpoint/?field_name__isnull=false`

Example:

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

Parameters

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

- **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_coreschema_field (*field*)

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.

Syntax:

`/endpoint/?field_name__gt={lower}__{boost} /endpoint/?field_name__gt={lower}`

Example:

`http://localhost:8000/api/articles/?id__gt=1 http://localhost:8000/api/articles/?id__gt=1__2.0`

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.

Syntax:

```
/endpoint/?field_name__range={lower}__{upper}__{boost}
point/?field_name__range={lower}__{upper}
```

Example:

```
http://localhost:8000/api/users/?age__range=16__67__2.0 http://localhost:8000/api/users/?age__
_range=16__67 http://localhost:8000/api/users/?age__range=16
```

Parameters *value* –

Type *str*

Returns Params to be used in *range* query.

Return type *dict*

get_schema_fields (*view*)

classmethod **prepare_filter_fields** (*view*)

Prepare filter fields.

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

Returns Filtering options.

Return type *dict*

12.17.1.2.1.10 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.GeoSpatialFiltering`

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 (
```

(continues on next page)

(continued from previous page)

```

>>>     GeoSpatialFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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": {
```



```

        }
    }
}
}
}

```

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

12.17.1.2.1.11 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.viewsets 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'

12.17.1.2.1.12 django_elasticsearch_dsl_drf.filter_backends.filtering.nested module

Nested filtering backend.

class django_elasticsearch_dsl_drf.filter_backends.filtering.nested.**NestedFilteringFilterBackend**
 Bases: *django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend*

Nested filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     NestedFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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 = [NestedFilteringFilterBackend,]
>>>     nested_filter_fields = {
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_TERM,
>>>                 LOOKUP_FILTER_TERMS,
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>         }
>>>     }
```

classmethod **apply_filter** (queryset, options=None, args=None, kwargs=None)

Apply filter.

Parameters

- **queryset** –
- **options** –

- **args** –
- **kwargs** –

Returns

classmethod **apply_query** (*queryset, options=None, args=None, kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

get_coreschema_field (*field*)

get_filter_field_nested_path (*filter_fields, field_name*)

Get filter field path to be used in nested query.

Parameters

- **filter_fields** –
- **field_name** –

Returns

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

get_schema_fields (*view*)

classmethod **prepare_filter_fields** (*view*)

Prepare filter fields.

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

Returns Filtering options.

Return type dict

12.17.1.2.1.13 `django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter` module

The `post_filter` filtering backend.

class `django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`

The `post_filter` 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 (
>>>     PostFilterFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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 = [PostFilterFilteringFilterBackend,]
>>>     post_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` (*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply filter.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod `apply_query` (*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

`get_coreschema_field` (*field*)

`get_schema_fields` (*view*)

classmethod `prepare_filter_fields` (*view*)

Prepare filter fields.

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

Returns Filtering options.

Return type dict

12.17.1.2.1.14 Module contents

Term level filtering and `post_filter` 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.viewsets 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,
>>>             ],
>>>     },
```

(continues on next page)

(continued from previous page)

```

>>>         'default_lookup': LOOKUP_FILTER_WILDCARD,
>>>     }
>>> }

```

classmethod `apply_filter_prefix(queryset, options, value)`

Apply *prefix* filter.

Syntax:

`/endpoint/?field_name__prefix={value}`

Example:

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

Parameters

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

Syntax:

`/endpoint/?field_name__range={lower}__{upper}__{boost}` `/end-`
`point/?field_name__range={lower}__{upper}`

Example:

`http://localhost:8000/api/users/?age__range=16__67__2.0` `http://localhost:8000/api/users/?age__range=16__67` `http://localhost:8000/api/users/?age__range=16`

Parameters

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

Syntax:

`/endpoint/?field_name={value}`

Example:

`http://localhost:8000/api/articles/?tags=children`

Parameters

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

Syntax:

`/endpoint/?field_name__terms={ value1 }__ { value2 } /endpoint/?field_name__terms={ value1 }`

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__terms=children__python` `http://localhost:8000/api/articles/?tags__terms=children`

Parameters

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

Syntax:

`/endpoint/?field_name__contains={ value }`

Example:

`http://localhost:8000/api/articles/?state__contains=lis`

Parameters

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

Syntax:

`/endpoint/?field_name__endswith={ value }`

Example:

`http://localhost:8000/api/articles/?tags__endswith=dren`

Parameters

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

Syntax:

/endpoint/?field_name__isnull={ value1 }__{ value2 } /endpoint/?field_name__exclude={ value1 }

Note, that number of values is not limited.

Example:

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

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.

Syntax:

/endpoint/?field_name__exists=true /endpoint/?field_name__exists=false

Example:

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

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.

Syntax:

/endpoint/?field_name__gt={ value }__{ boost } /endpoint/?field_name__gt={ value }

Example:

`http://localhost:8000/api/articles/?id__gt=1__2.0` `http://localhost:8000/api/articles/?id__gt=1`

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.

Syntax:

/endpoint/?field_name__gte={value}__{boost} /endpoint/?field_name__gte={value}

Example:

`http://localhost:8000/api/articles/?id__gte=1__2.0` `http://localhost:8000/api/articles/?id__gte=1`

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.

Syntax:

/endpoint/?field_name__in={value1}__{value2} /endpoint/?field_name__in={value1}

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__in=children__python`

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.

Syntax:

/endpoint/?field_name__isnull=true /endpoint/?field_name__isnull=false

Example:

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

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.

Syntax:

/endpoint/?field_name__lt={value}__{boost} /endpoint/?field_name__lt={value}

Example:

http://localhost:8000/api/articles/?id__lte=1__2.0 http://localhost:8000/api/articles/?id__lte=1

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.

Syntax:

`/endpoint/?field_name__lte={value}__{boost} /endpoint/?field_name__lte={value}`

Example:

http://localhost:8000/api/articles/?id__lte=1__2.0 http://localhost:8000/api/articles/?id__lte=1

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.

Syntax:

`/endpoint/?field_name__wildcard={value}* /endpoint/?field_name__wildcard={value}`
`/endpoint/?field_name__wildcard={value}* /endpoint/?field_name__wildcard={value}`

Example:

http://localhost:8000/api/articles/?tags__wildcard=child*

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_coreschema_field (*field*)

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.

Syntax:

`/endpoint/?field_name__gt={lower}__{boost} /endpoint/?field_name__gt={lower}`

Example:

`http://localhost:8000/api/articles/?id__gt=1 http://localhost:8000/api/articles/?id__gt=1__2.0`

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.

Syntax:

`/endpoint/?field_name__range={lower}__{upper}__{boost} /end-point/?field_name__range={lower}__{upper}`

Example:

`http://localhost:8000/api/users/?age__range=16__67__2.0 http://localhost:8000/api/users/?age__range=16__67 http://localhost:8000/api/users/?age__range=16`

Parameters value –

Type `str`

Returns Params to be used in *range* query.

Return type `dict`

get_schema_fields (*view*)

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.viewsets 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 query-set.
- **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 query-set.
- **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.viewsets 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'

class *django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend*

Bases: *django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend*

Nested filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     NestedFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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
```

(continues on next page)

(continued from previous page)

```
>>> filter_backends = [NestedFilteringFilterBackend,]
>>> nested_filter_fields = {
>>>     'country': {
>>>         'field': 'continent.country.name.raw',
>>>         'path': 'continent.country',
>>>         'lookups': [
>>>             LOOKUP_FILTER_TERM,
>>>             LOOKUP_FILTER_TERMS,
>>>             LOOKUP_FILTER_PREFIX,
>>>             LOOKUP_FILTER_WILDCARD,
>>>             LOOKUP_QUERY_EXCLUDE,
>>>             LOOKUP_QUERY_ISNULL,
>>>         ],
>>>     },
>>> }
```

classmethod `apply_filter` (*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply filter.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod `apply_query` (*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

`get_coreschema_field` (*field*)

`get_filter_field_nested_path` (*filter_fields*, *field_name*)

Get filter field path to be used in nested query.

Parameters

- **filter_fields** –
- **field_name** –

Returns

`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

get_schema_fields (*view*)

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.**PostFilterFilteringFilterBackend**

Bases: *django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend*

The post_filter 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 (
>>>     PostFilterFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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 = [PostFilterFilteringFilterBackend,]
>>>     post_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 (*queryset, options=None, args=None, kwargs=None*)

Apply filter.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod **apply_query** (*queryset, options=None, args=None, kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

get_coreschema_field (*field*)

get_schema_fields (*view*)

classmethod **prepare_filter_fields** (*view*)

Prepare filter fields.

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

Returns Filtering options.

Return type dict

12.17.1.2.1.15 django_elasticsearch_dsl_drf.filter_backends.ordering package

12.17.1.2.1.16 Submodules

12.17.1.2.1.17 django_elasticsearch_dsl_drf.filter_backends.ordering.common module

Ordering backend.

class `django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingMixin`

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.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
```

(continues on next page)

(continued from previous page)

```

>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [
>>>         DefaultOrderingFilterBackend,
>>>         OrderingFilterBackend,
>>>     ]
>>>     ordering_fields = {
>>>         'id': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = 'city'

```

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`, `django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingMixin`

Ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = ('id', 'title',)
```

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

get_schema_fields (*view*)

ordering_param = 'ordering'

12.17.1.2.1.18 `django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial` module

Geo-spatial ordering backend.

class `django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.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.viewsets 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_geo_spatial_field_name (*request, view, name*)

Get geo-spatial field name.

We have to deal with a couple of situations here:

Example 1:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': None,
>>> }
```

Example 2:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': 'location',
>>> }
```

Example 3:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': {
>>>         'field': 'location'
>>>     },
>>> }
```

Parameters

- **request** –
- **view** –
- **name** –

Returns

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'
```

12.17.1.2.1.19 Module contents

Ordering backends.

```
class django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend
    Bases: rest_framework.filters.BaseFilterBackend, django_elasticsearch_dsl_drf.
    filter_backends.ordering.common.OrderingMixin
```

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.viewsets 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': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>     }
```

(continues on next page)

(continued from previous page)

```
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = 'city'
```

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.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
```

(continues on next page)

(continued from previous page)

```
>>>
>>> # 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_geo_spatial_field_name (*request, view, name*)

Get geo-spatial field name.

We have to deal with a couple of situations here:

Example 1:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': None,
>>> }
```

Example 2:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': 'location',
>>> }
```

Example 3:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': {
>>>         'field': 'location'
>>>     },
>>> }
```

Parameters

- **request** –
- **view** –
- **name** –

Returns

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`, `django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingMixin`

Ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
```

(continues on next page)

(continued from previous page)

```

>>>     'continent': {
>>>         'field': 'continent.name.raw',
>>>         'path': 'continent',
>>>     }
>>>     'country': {
>>>         'field': 'continent.country.name.raw',
>>>         'path': 'continent.country',
>>>     }
>>>     'city': {
>>>         'field': 'continent.country.city.name.raw',
>>>         'path': 'continent.country.city',
>>>     }
>>> }
>>> ordering = ('id', 'title',)

```

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*

get_schema_fields (*view*)

ordering_param = 'ordering'

12.17.1.2.1.20 `django_elasticsearch_dsl_drf.filter_backends.search` package

12.17.1.2.1.21 Subpackages

12.17.1.2.1.22 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends` package

12.17.1.2.1.23 Submodules

12.17.1.2.1.24 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Bases: `object`

Search query backend.

classmethod `construct_search(request, view, search_backend)`

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

12.17.1.2.1.25 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match.MatchQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match query backend.

classmethod `construct_search(request, view, search_backend)`

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

`query_type = 'match'`

12.17.1.2.1.26 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase.MatchPhraseQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match phrase query backend.

classmethod **construct_search** (*request, view, search_backend*)

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

query_type = 'match_phrase'

12.17.1.2.1.27 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match phrase prefix query backend.

classmethod **construct_search** (*request, view, search_backend*)

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

query_type = 'match_phrase_prefix'

12.17.1.2.1.28 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match.MultiMatch`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Multi match query backend.

classmethod **construct_search** (*request, view, search_backend*)

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_multi_match=lorem ipsum /search/books/?search_multi_match=title,summary:lorem ipsum
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_multi_match=title,summary:lorem ipsum
&search_multi_match=author,publisher=o'reilly
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
multi_match_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'summary':
    { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
multi_match_search_fields = ( 'title', 'summary', 'description',
)
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

classmethod **get_field** (*field, options*)

Get field.

Parameters

- **field** –
- **options** –

Returns

classmethod **get_query_options** (*request, view, search_backend*)

query_type = 'multi_match'

12.17.1.2.1.29 django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.NestedQuery`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Nested query backend.

classmethod **construct_search** (*request, view, search_backend*)

Construct search.

Dictionary key is the GET param name. The path option stands for the path in Elasticsearch.

Type 1:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': ['name'],
    }, 'city': {
        'path': 'country.city', 'fields': ['name'],
    },
}
```

Type 2:

```
search_nested_fields = {
```



```

        'country': { 'path': 'country', 'fields': [{ 'name': { 'boost': 2 } }]
    }, 'city': {
        'path': 'country.city', 'fields': [{ 'name': { 'boost': 2 } }]
    },
}

```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

query_type = 'nested'

12.17.1.2.1.30 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string` module

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Simple query string query backend.

classmethod **construct_search** (*request, view, search_backend*)

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```

/search/books/?search_simple_query_string= "fried eggs" %2B(eggplant | potato) -
frittata
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata

```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```

/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata &search_simple_query_string= author,publisher="fried eggs" +(egg-
plant | potato) -frittata

```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```

simple_query_string_search_fields = { 'title': { 'field': 'title.english', 'boost': 4 }, 'sum-
mary': { 'boost': 2 }, 'description': None,
}

```

Example 2 (simple list):

```

simple_query_string_search_fields = ( 'title', 'summary', 'description',
)

```

Query examples:

```
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22 -considering
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
classmethod get_field(field, options)
    Get field.
```

Parameters

- **field** –
- **options** –

Returns

```
classmethod get_query_options(request, view, search_backend)
query_type = 'simple_query_string'
```

12.17.1.2.1.31 Module contents

Search query backends.

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.BaseSearchQueryBackend
    Bases: object
```

Search query backend.

```
classmethod construct_search(request, view, search_backend)
    Construct search.
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchQueryBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
base.BaseSearchQueryBackend
```

Match query backend.

```
classmethod construct_search(request, view, search_backend)
    Construct search.
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
query_type = 'match'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchPhraseQueryBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
base.BaseSearchQueryBackend
```

Match phrase query backend.

```
classmethod construct_search (request, view, search_backend)
```

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
query_type = 'match_phrase'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchPhrasePrefixBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
base.BaseSearchQueryBackend
```

Match phrase prefix query backend.

```
classmethod construct_search (request, view, search_backend)
```

Construct search.

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
query_type = 'match_phrase_prefix'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MultiMatchQueryBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
base.BaseSearchQueryBackend
```

Multi match query backend.

```
classmethod construct_search (request, view, search_backend)
```

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_multi_match=lorem ipsum /search/books/?search_multi_match=title,summary:lorem ipsum
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_multi_match=title,summary:lorem ipsum
&search_multi_match=author,publisher=o'reilly
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
multi_match_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'summary':
    { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
multi_match_search_fields = ( 'title', 'summary', 'description',
)
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

classmethod `get_field` (*field*, *options*)

Get field.

Parameters

- **field** –
- **options** –

Returns

classmethod `get_query_options` (*request*, *view*, *search_backend*)

query_type = 'multi_match'

class `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.NestedQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Nested query backend.

classmethod `construct_search` (*request*, *view*, *search_backend*)

Construct search.

Dictionary key is the GET param name. The path option stands for the path in Elasticsearch.

Type 1:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': ['name'],
    }, 'city': {
        'path': 'country.city', 'fields': ['name'],
    },
}
```

Type 2:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': [{ 'name': { 'boost': 2} }}
    }, 'city': {
        'path': 'country.city', 'fields': [{ 'name': { 'boost': 2} }}
    },
}
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

query_type = 'nested'

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.SimpleQueryString
Bases: django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
base.BaseSearchQueryBackend
```

Simple query string query backend.

```
classmethod construct_search(request, view, search_backend)
Construct search.
```

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_simple_query_string= "fried eggs" %2B(eggplant | potato) -
frittata
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata &search_simple_query_string= author,publisher="fried eggs" +(egg-
plant | potato) -frittata
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
simple_query_string_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'sum-
mary': { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
simple_query_string_search_fields = ( 'title', 'summary', 'description',
)
```

Query examples:

```
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22 -considering
```

Parameters

- **request** –
- **view** –
- **search_backend** –

Returns

```
classmethod get_field(field, options)
```

Get field.

Parameters

- **field** –
- **options** –

Returns

```
classmethod get_query_options(request, view, search_backend)
```

```
query_type = 'simple_query_string'
```

12.17.1.2.1.32 Submodules

12.17.1.2.1.33 `django_elasticsearch_dsl_drf.filter_backends.search.base` module

Base search backend.

```
class django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend
```

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

Base search filter backend.

```
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_coreschema_field(field)
```

```
get_query_backends(request, view)
```

Get query backends.

Returns

```
get_schema_fields(view)
```

```
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

```
matching = 'should'
```

```
query_backends = []
```

```
search_param = 'search'
```

12.17.1.2.1.34 django_elasticsearch_dsl_drf.filter_backends.search.compound module

Compound search backend.

```
class django_elasticsearch_dsl_drf.filter_backends.search.compound.CompoundSearchFilterBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend
```

Compound search backend.

```
query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_base.QueryBackend']
```

12.17.1.2.1.35 django_elasticsearch_dsl_drf.filter_backends.search.historical module

Search backend. Most likely to be deprecated soon.

```
class django_elasticsearch_dsl_drf.filter_backends.search.historical.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.viewsets 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',
>>>     )
>>>     search_nested_fields = {
>>>         'state': ['name'],
>>>         'documents.author': ['title', 'description'],
>>>     }
```

construct_nested_search (*request, view*)

Construct nested search.

We have to deal with two types of structures:

Type 1:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': ['name'],
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': ['name'],
>>>     },
>>> }
```

Type 2:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': [{'name': {'boost': 2}}]
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': [{'name': {'boost': 2}}]
>>>     },
>>> }
```

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*

construct_search (*request, view*)

Construct search.

We have to deal with two types of structures:

Type 1:

```
>>> search_fields = (
>>>     'title',
>>>     'description',
>>>     'summary',
>>> )
```

Type 2:

```
>>> search_fields = {
>>>     'title': {'boost': 2},
>>>     'description': None,
>>>     'summary': None,
>>> }
```

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_coreschema_field (*field*)

get_schema_fields (*view*)

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'

12.17.1.2.1.36 `django_elasticsearch_dsl_drf.filter_backends.search.multi_match` module

Multi match search filter backend.

class `django_elasticsearch_dsl_drf.filter_backends.search.multi_match.MultiMatchSearchFilter`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend`

Multi match search filter backend.

matching = 'must'

query_backends = [`<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_base.QueryBaseSearchFilterBackend'>`]

search_param = 'search_multi_match'

12.17.1.2.1.37 `django_elasticsearch_dsl_drf.filter_backends.search.query_string` module

12.17.1.2.1.38 `django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string` module

Simple query string search filter backend.

```
class django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.SimpleQueryST
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.base.
BaseSearchFilterBackend

    Simple query string search filter backend.

    matching = 'must'

    query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_ba
    search_param = 'search_simple_query_string'
```

12.17.1.2.1.39 Module contents

Search filter backends.

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

    Base search filter backend.

    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_coreschema_field (field)

    get_query_backends (request, view)
        Get query backends.
        Returns

    get_schema_fields (view)

    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

    matching = 'should'

    query_backends = []

    search_param = 'search'

class django_elasticsearch_dsl_drf.filter_backends.search.CompoundSearchFilterBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.base.
BaseSearchFilterBackend

    Compound search backend.
```

```

query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_ba

class django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.base.
           BaseSearchFilterBackend
    Multi match search filter backend.

    matching = 'must'

    query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_ba

    search_param = 'search_multi_match'

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.viewsets 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',
>>>     )
>>>     search_nested_fields = {
>>>         'state': ['name'],
>>>         'documents.author': ['title', 'description'],
>>>     }

```

construct_nested_search(*request, view*)

Construct nested search.

We have to deal with two types of structures:

Type 1:

```

>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': ['name'],
>>>     },

```

(continues on next page)

(continued from previous page)

```
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': ['name'],
>>>     },
>>> }
```

Type 2:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': [{'name': {'boost': 2}}]
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': [{'name': {'boost': 2}}]
>>>     },
>>> }
```

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***construct_search** (*request, view*)

Construct search.

We have to deal with two types of structures:

Type 1:

```
>>> search_fields = (
>>>     'title',
>>>     'description',
>>>     'summary',
>>> )
```

Type 2:

```
>>> search_fields = {
>>>     'title': {'boost': 2},
>>>     'description': None,
>>>     'summary': None,
>>> }
```

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_coreschema_field (*field*)

get_schema_fields (*view*)

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'

class *django_elasticsearch_dsl_drf.filter_backends.search.SimpleQueryStringSearchFilterBackend*

Bases: *django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend*

BaseSearchFilterBackend

Simple query string search filter backend.

matching = 'must'

query_backends = [*<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_base.QueryBase'>*]

search_param = 'search_simple_query_string'

12.17.1.2.1.40 *django_elasticsearch_dsl_drf.filter_backends.suggester* package

12.17.1.2.1.41 Submodules

12.17.1.2.1.42 *django_elasticsearch_dsl_drf.filter_backends.suggester.functional* module

Functional 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 Document, 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(Document):
>>>     "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):
```

(continues on next page)

(continued from previous page)

```
>>>         "Meta options."
>>>
>>>         model = Publisher # The model associate with this Document
```

class django_elasticsearch_dsl_drf.filter_backends.suggester.functional.**FunctionalSuggester**

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 `FunctionalSuggesterFilterBackend`, the latter will transform your current search query into another search query (altered). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_TERM_MATCH,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FunctionalSuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         FunctionalSuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     functional_suggester_fields = {
>>>         'name_suggest': {
>>>             'field': 'name.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>         'city_suggest': {
>>>             'field': 'city.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>     }
```

(continues on next page)

(continued from previous page)

```

>>>         'state_province_suggest': {
>>>             'field': 'state_province.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>             ],
>>>         },
>>>         'country_suggest': {
>>>             'field': 'country.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>     }

```

classmethod `apply_query_size(queryset, options)`

Apply query size.

Parameters

- **queryset** –
- **options** –

Returns

classmethod `apply_suggester_completion_match(suggester_name, queryset, options, value)`

Apply *completion* suggester match.

This is effective when used with Ngram fields.

Parameters

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

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

classmethod `apply_suggester_completion_prefix(suggester_name, queryset, options, value)`

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

Parameters

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

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

clean_queryset(queryset)

Clean the queryset.

- Remove aggregations.

- Remove highlight.
- Remove sorting options.

Parameters `queryset` –

Returns

extract_field_name (*field_name*)

Extract field name.

For instance, “name.suggest” or “name.raw” becomes “name”.

Parameters `field_name` –

Returns

Return type `str`

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`

serialize_queryset (*queryset, suggester_name, value, serializer_field*)

Serialize queryset.

This shall be done here, since we don’t want to delegate it to pagination.

Parameters

- **queryset** –
- **suggester_name** –
- **value** –
- **serializer_field** –

Returns

12.17.1.2.1.43 django_elasticsearch_dsl_drf.filter_backends.suggester.native 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 Document, 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(Document):
>>>     "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={
```

(continues on next page)

(continued from previous page)

```
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> website = fields.StringField()
>>>
>>> class Meta(object):
>>>     "Meta options."
>>>
>>> model = Publisher # The model associate with this Document
```

class django_elasticsearch_dsl_drf.filter_backends.suggester.native.**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.viewsets import DocumentViewSet
>>>
>>> # 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,
>>>             ],
```

(continues on next page)

(continued from previous page)

```
>>> },
>>> '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 query-set.
- **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 query-set.
- **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 query-set.
- **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`

classmethod **get_suggester_context** (*field, suggester_name, request, view*)

Get suggester context.

Given the following definition (in ViewSets):

```
>>> # Suggester fields
>>> suggester_fields = {
>>>     'title_suggest': {
>>>         'field': 'title.suggest',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>     },
>>>     'title_suggest_context': {
>>>         'field': 'title.suggest_context',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>         'completion_options': {
>>>             'filters': {
>>>                 'title_suggest_tag': 'tag',
>>>                 'title_suggest_state': 'state',
>>>                 'title_suggest_publisher': 'publisher',
>>>             },
>>>             'size': 10,
>>>         }
>>>     },
>>> }
```

http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M

When talking about the queries made, we have the following. Multiple values per field are combined with OR:

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': ['History', 'Drama'],
>>>     }
>>> }
```

The following works with OR as well, so it seems we have OR only. However, the following construction is more handy, since it allows us to play with boosting nicely. Also, it allows to provide *prefix* param (which in case of the example given below means that suggestions shall match both categories with “Child”, “Children”, “Childrend’s”). Simply put it’s treated as *prefix*, rather than *term*.

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': [
>>>             {'context': 'History'},
>>>             {'context': 'Drama', 'boost': 2},
>>>             {'context': 'Children', 'prefix': True},
>>>         ],
>>>     },
>>> }
```

Sample query for *category* filter:

```
/search/books-frontend/suggest/?title_suggest_context=M &title_suggest_tag=Art__2.0
&title_suggest_tag=Documentary__2.0__prefix &title_suggest_publisher=Apress
```

The query params would be:

```
query_params: <QueryDict: {
    'title_suggest_context': ['M'], 'title_suggest_tag': ['Art__2.0', 'Documen-
    tary__2.0__prefix'], 'title_suggest_publisher': ['Apress']
}>
```

Sample query for *geo* filter:

```
/search/address/suggest/?street_suggest_context=M &street_suggest_loc=43.66_-
79.22__2.0__10000km
```

The query params would be:

```
query_params: <QueryDict: {
    'street_suggest_context': ['M'], 'street_suggest_loc': ['Art_43.66_-
    79.22__2.0__10000km'],
}>
```

Returns

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

12.17.1.2.1.44 Module contents

Suggester filtering backends.

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.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
>>>
>>>     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,
>>>             ],
>>>         },
>>>     }
```

(continues on next page)

(continued from previous page)

```
>>>         },
>>>     }
```

classmethod `apply_suggester_completion(suggester_name, queryset, options, value)`

Apply *completion* suggester.

Parameters

- **suggester_name** (*str*) –
- **queryset** (*elasticsearch_dsl.search.Search*) – Original query-set.
- **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 query-set.
- **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 query-set.
- **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*

classmethod `get_suggester_context` (*field, suggester_name, request, view*)

Get suggester context.

Given the following definition (in ViewSets):

```
>>> # Suggester fields
>>> suggester_fields = {
>>>     'title_suggest': {
>>>         'field': 'title.suggest',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>     },
>>>     'title_suggest_context': {
>>>         'field': 'title.suggest_context',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>         'completion_options': {
>>>             'filters': {
>>>                 'title_suggest_tag': 'tag',
>>>                 'title_suggest_state': 'state',
>>>                 'title_suggest_publisher': 'publisher',
>>>             },
>>>             'size': 10,
>>>         },
>>>     },
>>> }
```

http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M

When talking about the queries made, we have the following. Multiple values per field are combined with OR:

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': ['History', 'Drama'],
>>>     }
>>> }
```

The following works with OR as well, so it seems we have OR only. However, the following construction is more handy, since it allows us to play with boosting nicely. Also, it allows to provide *prefix* param (which in case of the example given below means that suggestions shall match both categories with “Child”, “Children”, “Childrend’s”). Simply put it’s treated as *prefix*, rather than *term*.

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': [
>>>             {'context': 'History'},
>>>             {'context': 'Drama', 'boost': 2},
>>>             {'context': 'Children', 'prefix': True},
>>>         ],
>>>     },
>>> }
```

Sample query for *category* filter:

```
/search/books-frontend/suggest/ ?title_suggest_context=M &title_suggest_tag=Art__2.0
&title_suggest_tag=Documentary__2.0__prefix &title_suggest_publisher=Apress
```

The query params would be:

```
query_params: <QueryDict: {
  'title_suggest_context': ['M'], 'title_suggest_tag': ['Art__2.0', 'Documen-
    tary__2.0__prefix'], 'title_suggest_publisher': ['Apress']
}>
```

Sample query for *geo* filter:

```
/search/address/suggest/?street_suggest_context=M &street_suggest_loc=43.66__-
79.22__2.0__10000km
```

The query params would be:

```
query_params: <QueryDict: {
  'street_suggest_context': ['M'], 'street_suggest_loc': ['Art__43.66__-
    79.22__2.0__10000km'],
}>
```

Returns

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

class `django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend`

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 `FunctionalSuggesterFilterBackend`, the latter will transform your current search query into another search query (altered). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_TERM_MATCH,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FunctionalSuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
```

(continues on next page)

(continued from previous page)

```

>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         FunctionalSuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     functional_suggester_fields = {
>>>         'name_suggest': {
>>>             'field': 'name.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>         'city_suggest': {
>>>             'field': 'city.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>         'state_province_suggest': {
>>>             'field': 'state_province.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>             ],
>>>         },
>>>         'country_suggest': {
>>>             'field': 'country.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>     }
>>> }

```

classmethod `apply_query_size` (*queryset*, *options*)

Apply query size.

Parameters

- **queryset** –
- **options** –

Returns

classmethod `apply_suggester_completion_match` (*suggester_name*, *queryset*, *options*, *value*)

Apply *completion* suggester match.

This is effective when used with Ngram fields.

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_completion_prefix** (*suggester_name, queryset, options, value*)

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

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*

clean_queryset (*queryset*)

Clean the queryset.

- Remove aggregations.
- Remove highlight.
- Remove sorting options.

Parameters **queryset** –

Returns

extract_field_name (*field_name*)

Extract field name.

For instance, “name.suggest” or “name.raw” becomes “name”.

Parameters **field_name** –

Returns

Return type *str*

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

serialize_queryset (*queryset, suggester_name, value, serializer_field*)

Serialize queryset.

This shall be done here, since we don't want to delegate it to pagination.

Parameters

- **queryset** –
- **suggester_name** –
- **value** –
- **serializer_field** –

Returns

12.17.1.2.2 Submodules

12.17.1.2.3 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.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
```

(continues on next page)

(continued from previous page)

```

>>>     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

12.17.1.2.4 django_elasticsearch_dsl_drf.filter_backends.highlight module

Highlight backend.

class django_elasticsearch_dsl_drf.filter_backends.highlight.**HighlightBackend**
Bases: rest_framework.filters.BaseFilterBackend

Highlight backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     HighlightBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets 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 = [HighlightBackend,]
>>>     highlight_fields = {
>>>         'author.name': {
>>>             'enabled': False,
>>>             'options': {
>>>                 'fragment_size': 150,
>>>                 'number_of_fragments': 3
>>>             }
>>>         }
>>>         'title': {
>>>             'options': {
>>>                 'pre_tags' : ["<em>"],
>>>                 'post_tags' : ["</em>"]
>>>             },
>>>             'enabled': True,
>>>         },
>>>     }
```

Highlight make queries to be more heavy. That's why by default all highlights are disabled and enabled only explicitly either in the filter options (*enabled* set to True) or via query params *?highlight=author.name&highlight=title*.

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_highlight_query_params (*request*)
 Get highlight query params.
Parameters **request** (*rest_framework.request.Request*) – Django REST framework request.
Returns List of search query params.
Return type list

highlight_param = 'highlight'

classmethod prepare_highlight_fields (*view*)
 Prepare faceted search fields.
 Prepares the following structure:

```
>>> {
>>>     'author.name': {
>>>         'enabled': False,
>>>         'options': {
>>>             'fragment_size': 150,
>>>             'number_of_fragments': 3
>>>         }
>>>     }
>>>     'title': {
>>>         'options': {
>>>             'pre_tags' : ["<em>"],
>>>             'post_tags' : ["</em>"]
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

Parameters **view** (*rest_framework.viewsets.ReadOnlyModelViewSet*) –
Returns Highlight fields options.
Return type dict

12.17.1.2.5 django_elasticsearch_dsl_drf.filter_backends.mixins module

Mixins.

class `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`
 Bases: object

Filter backend mixin.

classmethod **apply_filter** (*queryset, options=None, args=None, kwargs=None*)
 Apply filter.
Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod `apply_query` (*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod `split_lookup_complex_multiple_value` (*value*, *maxsplit=-1*)

Split lookup complex multiple 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_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_name` (*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

12.17.1.2.6 Module contents

All filter backends.

12.17.1.3 django_elasticsearch_dsl_drf.tests package

12.17.1.3.1 Submodules

12.17.1.3.2 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 = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass ()
        Set up class.

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

    Base test case.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass ()
        Set up class.
```

12.17.1.3.3 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
```

12.17.1.3.4 django_elasticsearch_dsl_drf.tests.test_faceted_search module

Test faceted search backend.

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

    pytestmark = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUp ()
        Hook method for setting up the test fixture before exercising it.

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

12.17.1.3.5 django_elasticsearch_dsl_drf.tests.test_filtering_common module

Test filtering backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon (methodName='runTest')
    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 = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUpClass ()
        Set up.

    test_default_filter_lookup ()
        Test default filter lookup.

        Example:
        http://localhost:8000/search/books-default-filter-lookup/ ?authors=Robin&authors=Luc

    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_empty_filter()

Test ids filter with empty value. This should not fail.

Example:

<http://localhost:8000/api/articles/?ids=>

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_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_various_complex_fields()
Test various complex fields.
Returns

12.17.1.3.6 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**
Bases: *django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase*

Test filtering geo-spatial.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

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

Returns

test_field_filter_geo_distance_distance_type_arc()
Field filter geo-distance.

Example:

http://localhost:8000/api/publisher/?location__geo_distance=1km__48.8549__2.3000__arc

Returns

test_field_filter_geo_distance_not_enough_args_fail()
Field filter geo-distance. Fail test on not enough args.

Example:

http://localhost:8000/api/publisher/?location__geo_distance=1km__48.8549

Returns

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
```

12.17.1.3.7 django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module

Test filtering *post_filter* backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations.TestFilteringG
    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 with global aggregations.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up.

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

12.17.1.3.8 django_elasticsearch_dsl_drf.tests.test_filtering_nested module

Test nested filtering backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested(methodNan
    Bases:      django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
               django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin

    Test filtering nested.

    base_url

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up.

    test_field_filter_contains()
        Test filter contains.

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

    test_field_filter_endswith()
        Test filter endswith.

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

    test_field_filter_exclude()
        Test filter exclude.

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


test_field_filter_exists_false()
Test filter exists.
Example:
http://localhost:8000/api/articles/?non_existent__exists=false

test_field_filter_exists_true()
Test filter exists true.
Example:
http://localhost:8000/api/articles/?tags__exists=true

test_field_filter_in()
Test filter in.
Example:
http://localhost:8000/api/articles/?id__in=1__2__3

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_schema_field_not_required()
Test schema fields always not required

test_schema_fields_with_filter_fields_list()
Test schema field generator

12.17.1.3.9 django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module

Test filtering *post_filter* backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter:
    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 *post_filter*.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
    classmethod setUpClass()
```

Set up.

```
    test_field_filter_contains()
```

Test filter contains.

Example:

```
    http://localhost:8000/api/articles/?state__contains=lished
```

```
    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_list_results_with_facets()

Test list results with facets.

test_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_various_complex_fields()

Test various complex fields.

Returns

12.17.1.3.10 django_elasticsearch_dsl_drf.tests.test_functional_suggesters module

Test functional suggestions backend.

class django_elasticsearch_dsl_drf.tests.test_functional_suggesters.**TestFunctionalSuggesters**

Bases: [django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase](#),
[django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin](#)

Test functional suggesters.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpClass()

Set up class.

test_suggesters_completion()

Test suggesters completion.

test_suggesters_completion_no_args_provided()

Test suggesters completion with no args provided.

12.17.1.3.11 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 = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUpClass ()
        Set up class.

    test_filter_by_field ()
        Filter by field.
```

12.17.1.3.12 django_elasticsearch_dsl_drf.tests.test_highlight module

Test highlight backend.

```
class django_elasticsearch_dsl_drf.tests.test_highlight.TestHighlight (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

    Test highlight.

    pytestmark = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUp ()
        Hook method for setting up the test fixture before exercising it.

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

12.17.1.3.13 django_elasticsearch_dsl_drf.tests.test_more_like_this module

Test more-like-this functionality.

```
class django_elasticsearch_dsl_drf.tests.test_more_like_this.TestMoreLikeThis (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

    Test suggesters.

    pytestmark = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUpClass ()
        Set up class.

    test_more_like_this ()
        Test more-like-this.
```

12.17.1.3.14 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.
```

static deep_get (*dic*, *keys*, *default=None*)

Returns value at period separated keys in dictionary or default

From <https://stackoverflow.com/a/46890853>

Parameters

- **dic** (*dict*) – Dictionary to retrieve value from.
- **keys** (*str*) – Period separated path of keys
- **default** (*str*) – Default value to return if keys not found

Returns Value at keys or default

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpClass ()

Set up class.

test_address_default_nested_order_by ()

test_address_default_order_by ()

Test if default ordering on addresses is correct (asc).

test_address_order_by_nested_field_continent_ascending ()

Order by field *continent.country.name.raw* ascending.

test_address_order_by_nested_field_continent_descending ()

Order by field *continent.country.name.raw* descending.

test_address_order_by_nested_field_country_ascending ()

Order by field *continent.country.name.raw* ascending.

test_address_order_by_nested_field_country_descending ()

Order by field *continent.country.name.raw* descending.

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.

test_schema_field_not_required()
    Test schema fields always not required

test_schema_fields_with_filter_fields_list()
    Test schema field generator
```

12.17.1.3.15 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 (
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test ordering geo-spatial.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    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
```

12.17.1.3.16 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 = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_pagination()
        Test pagination.
```

12.17.1.3.17 django_elasticsearch_dsl_drf.tests.test_pip_helpers module

Test pip_helpers.

```
class django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers (methodName='runTest')
    Bases: unittest.case.TestCase
    Test pip_helpers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={})]

    classmethod setUpClass()
        Hook method for setting up class fixture before running tests in the class.
```

```
test_check_if_installed()
    Test check_if_installed.
    Returns

test_get_installed_packages()
    Test get_installed_packages.
    Returns

test_get_installed_packages_with_versions()
    Test get_installed_packages.
    Returns
```

12.17.1.3.18 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 = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUp()
        Hook method for setting up the test fixture before exercising it.

    test_compound_search_boost_by_field()

    test_compound_search_by_field()

    test_compound_search_by_field_multi_terms()

    test_compound_search_by_nested_field()

    test_schema_field_not_required()
        Test schema fields always not required

    test_schema_fields_with_filter_fields_list()
        Test schema field generator

    test_search_boost(url=None, search_field='search')
        Search boost.
        Returns

    test_search_boost_compound(search_field='search')

    test_search_by_field(url=None, search_field='search')
        Search by field.

    test_search_by_field_multi_terms(url=None, search_field='search')
        Search by field, multiple terms.

    test_search_by_nested_field(url=None)
        Search by field.
```

12.17.1.3.19 django_elasticsearch_dsl_drf.tests.test_search_multi_match module

Test multi match search filter backend.


```
class django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch (method
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

    Test multi match search.

    pytestmark = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUp ()
        Hook method for setting up the test fixture before exercising it.

    test_schema_field_not_required ()
        Test schema fields always not required

    test_schema_fields_with_filter_fields_list ()
        Test schema field generator

    test_search (url=None)
        Search.

    test_search_boost (url=None)
        Search boost.
        Returns

    test_search_boost_selected_fields (url=None)
        Search boost.
        Returns

    test_search_selected_fields (url=None)
        Search boost.
        Returns
```

12.17.1.3.20 django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module

Test multi match search filter backend.

```
class django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQuerySt
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase

    Test simple query string search.

    pytestmark = [Mark (name='django_db', args=(), kwargs={}), Mark (name='django_db', args=
    classmethod setUp ()
        Hook method for setting up the test fixture before exercising it.

    test_schema_field_not_required ()
        Test schema fields always not required

    test_schema_fields_with_filter_fields_list ()
        Test schema field generator

    test_search_boost_selected_fields (url=None)
        Search boost.
        Returns

    test_search_selected_fields (url=None)
        Search boost.
        Returns

    test_search_with_quotes (url=None)
        Search with quotes.
```

```

test_search_with_quotes_alternative()
    Test search by field.
    Parameters url -
    Returns

test_search_with_quotes_boost(url=None)
    Search boost.
    Returns

test_search_with_quotes_boost_alternative()
    Search boost.
    Returns

test_search_without_quotes(url=None)
    Test search without quotes. This does not work on Elasticsearch 6.x.
    Parameters url -
    Returns

test_search_without_quotes_boost(url=None)
    Search boost without quotes. Does not work on Elasticsearch 6.x.
    Returns

```

12.17.1.3.21 django_elasticsearch_dsl_drf.tests.test_serializers module

Test serializers.

```

class django_elasticsearch_dsl_drf.tests.test_serializers.TestSerializers(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test serializers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    test_serializer_document_equals_to_none()
        Test serializer no document specified.

    test_serializer_fields_and_exclude()
        Test serializer fields and exclude.

    test_serializer_meta_del_attr()
        Test serializer set attr.

    test_serializer_meta_set_attr()
        Test serializer set attr.

    test_serializer_no_document_specified()
        Test serializer no document specified.

```

12.17.1.3.22 django_elasticsearch_dsl_drf.tests.test_suggesters module

Test suggestions backend.

```

class django_elasticsearch_dsl_drf.tests.test_suggesters.TestContextSuggesters(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Test context suggesters.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

```

```
classmethod setUpClass()
    Set up class.

test_suggesters_completion_context()
    Test suggesters completion context.

class django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Test suggesters.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass()
        Set up class.

    test_nested_fields_suggesters_completion()
        Test suggesters completion for nested fields.

    test_suggesters_completion()
        Test suggesters completion.

    test_suggesters_completion_no_args_provided()
        Test suggesters completion with no args provided.

    test_suggesters_phrase()
        Test suggesters phrase.

    test_suggesters_term()
        Test suggesters term.

class django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggestersEmptyIndex (methodName=
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Test suggesters on empty index.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass()
        Set up class.

    test_suggesters_on_empty_index()
        Test suggesters phrase.
```

12.17.1.3.23 `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 = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass()
        Set up class.

    test_detail_view()
        Test detail view.
```

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

12.17.1.3.24 django_elasticsearch_dsl_drf.tests.test_wrappers module

Test wrappers.

```
class django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers (methodName='runTest')
    Bases: unittest.case.TestCase

    Test wrappers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={})]

    classmethod setUpClass()
        Hook method for setting up class fixture before running tests in the class.

    test_dict_to_obj()
        Test dict_to_obj.
        Returns

    test_obj_to_dict()
        Test obj_to_dict.
        Returns

    test_wrapper_as_json()
        Test :Wrapper:'as_json' property.
```

12.17.1.3.25 Module contents

Tests.

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

    Test faceted search.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUp()
        Hook method for setting up the test fixture before exercising it.

    test_list_results_with_facets()
        Test list results with facets.

class django_elasticsearch_dsl_drf.tests.TestFilteringCommon (methodName='runTest')
    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 = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

    classmethod setUpClass()
        Set up.

    test_default_filter_lookup()
        Test default filter lookup.

        Example:
```

[http://localhost:8000/search/books-default-filter-lookup/ ?authors=Robin&authors=Luc](http://localhost:8000/search/books-default-filter-lookup/?authors=Robin&authors=Luc)

test_field_filter_contains()

Test filter contains.

Example:

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

test_field_filter_endswith()

Test filter ends with.

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_empty_filter()

Test ids filter with empty value. This should not fail.

Example:

<http://localhost:8000/api/articles/?ids=>

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_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_various_complex_fields()

Test various complex fields.

Returns

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test filtering geo-spatial.

pytestmark = [`Mark`(name='django_db', args=(), kwargs={}), `Mark`(name='django_db', args=

`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

Returns

test_field_filter_geo_distance_distance_type_arc()

Field filter geo-distance.

Example:

http://localhost:8000/api/publisher/?location__geo_distance=1km__48.8549__2.3000__arc

Returns

test_field_filter_geo_distance_not_enough_args_fail()
Field filter geo-distance. Fail test on not enough args.

Example:

http://localhost:8000/api/publisher/?location__geo_distance=1km__48.8549

Returns

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

```
class django_elasticsearch_dsl_drf.tests.TestFilteringGlobalAggregations (methodName='runTest')
    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 with global aggregations.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

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

test_list_results_with_facets()
Test list results with facets.

```
class django_elasticsearch_dsl_drf.tests.TestFilteringNested (methodName='runTest')
    Bases: django\_elasticsearch\_dsl\_drf.tests.base.BaseRestFrameworkTestCase,
           django\_elasticsearch\_dsl\_drf.tests.data\_mixins.AddressesMixin
```

Test filtering nested.

base_url

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

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

test_field_filter_contains()
Test filter contains.

Example:

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

test_field_filter_endswith()
Test filter endswith.

Example:

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

test_field_filter_exclude()

Test filter exclude.

Example:

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

test_field_filter_exists_false()

Test filter exists.

Example:

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

test_field_filter_exists_true()

Test filter exists true.

Example:

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

test_field_filter_in()

Test filter in.

Example:

`http://localhost:8000/api/articles/?id__in=1__2__3`

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_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

class django_elasticsearch_dsl_drf.tests.**TestFilteringPostFilter** (*methodName='runTest'*)

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 *post_filter*.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod **setUpClass()**

Set up.

test_field_filter_contains()

Test filter contains.

Example:

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

test_field_filter_endswith()

Test filter endswith.

Example:

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

test_field_filter_exclude()

Test filter exclude.

Example:

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

test_field_filter_exists_false()

Test filter exists.

Example:

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

test_field_filter_exists_true()

Test filter exists true.

Example:

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

test_field_filter_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_list_results_with_facets()

Test list results with facets.

test_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_various_complex_fields()

Test various complex fields.

Returns

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`,
`django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin`

Test functional suggesters.

pytestmark = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

`classmethod setUpClass()`

Set up class.

test_suggesters_completion()

Test suggesters completion.

test_suggesters_completion_no_args_provided()

Test suggesters completion with no args provided.

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseTestCase`

Test helpers.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUpClass()
```

Set up class.

```
test_filter_by_field()
```

Filter by field.

```
class django_elasticsearch_dsl_drf.tests.TestHighlight(methodName='runTest')
```

Bases: *django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase*

Test highlight.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUp()
```

Hook method for setting up the test fixture before exercising it.

```
test_list_results_with_highlights()
```

Test list results with facets.

```
class django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch(methodName='runTest')
```

Bases: *django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase*

Test multi match search.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUp()
```

Hook method for setting up the test fixture before exercising it.

```
test_schema_field_not_required()
```

Test schema fields always not required

```
test_schema_fields_with_filter_fields_list()
```

Test schema field generator

```
test_search(url=None)
```

Search.

```
test_search_boost(url=None)
```

Search boost.

Returns

```
test_search_boost_selected_fields(url=None)
```

Search boost.

Returns

```
test_search_selected_fields(url=None)
```

Search boost.

Returns

```
class django_elasticsearch_dsl_drf.tests.TestSimpleQueryStringSearch(methodName='runTest')
```

Bases: *django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase*

Test simple query string search.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUp()
```

Hook method for setting up the test fixture before exercising it.

test_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_search_boost_selected_fields(url=None)

Search boost.

Returns

test_search_selected_fields(url=None)

Search boost.

Returns

test_search_with_quotes(url=None)

Search with quotes.

test_search_with_quotes_alternative()

Test search by field.

Parameters *url* –

Returns

test_search_with_quotes_boost(url=None)

Search boost.

Returns

test_search_with_quotes_boost_alternative()

Search boost.

Returns

test_search_without_quotes(url=None)

Test search without quotes. This does not work on Elasticsearch 6.x.

Parameters *url* –

Returns

test_search_without_quotes_boost(url=None)

Search boost without quotes. Does not work on Elasticsearch 6.x.

Returns

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test ordering.

static `deep_get(dic, keys, default=None)`

Returns value at period separated keys in dictionary or default

From <https://stackoverflow.com/a/46890853>

Parameters

- **dic** (*dict*) – Dictionary to retrieve value from.
- **keys** (*str*) – Period separated path of keys
- **default** (*str*) – Default value to return if keys not found

Returns Value at keys or default

`pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=`

`classmethod setUpClass()`

Set up class.

test_address_default_nested_order_by()

```
test_address_default_order_by()
    Test if default ordering on addresses is correct (asc).

test_address_order_by_nested_field_continent_ascending()
    Order by field continent.country.name.raw ascending.

test_address_order_by_nested_field_continent_descending()
    Order by field continent.country.name.raw descending.

test_address_order_by_nested_field_country_ascending()
    Order by field continent.country.name.raw ascending.

test_address_order_by_nested_field_country_descending()
    Order by field continent.country.name.raw descending.

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.

test_schema_field_not_required()
    Test schema fields always not required

test_schema_fields_with_filter_fields_list()
    Test schema field generator

class django_elasticsearch_dsl_drf.tests.TestOrderingGeoSpatial(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test ordering geo-spatial.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    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>

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

Test pagination.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUpClass()

Set up class.

test_pagination()

Test pagination.

class django_elasticsearch_dsl_drf.tests.**TestPipHelpers** (*methodName='runTest'*)
 Bases: *unittest.case.TestCase*

Test pip_helpers.

pytestmark = [Mark(name='django_db', args=(), kwargs={})]

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

test_check_if_installed()

Test *check_if_installed*.

Returns

test_get_installed_packages()

Test *get_installed_packages*.

Returns

test_get_installed_packages_with_versions()

Test *get_installed_packages*.

Returns

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

Test search.

pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

classmethod setUp()

Hook method for setting up the test fixture before exercising it.

test_compound_search_boost_by_field()

test_compound_search_by_field()

test_compound_search_by_field_multi_terms()

test_compound_search_by_nested_field()

test_schema_field_not_required()

Test schema fields always not required

test_schema_fields_with_filter_fields_list()

Test schema field generator

test_search_boost (*url=None, search_field='search'*)

Search boost.

Returns

test_search_boost_compound (*search_field='search'*)

test_search_by_field (*url=None, search_field='search'*)

Search by field.

test_search_by_field_multi_terms (*url=None, search_field='search'*)

Search by field, multiple terms.

test_search_by_nested_field (*url=None*)

Search by field.

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test serializers.

pytestmark = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

test_serializer_document_equals_to_none ()

Test serializer no document specified.

test_serializer_fields_and_exclude ()

Test serializer fields and exclude.

test_serializer_meta_del_attr ()

Test serializer set attr.

test_serializer_meta_set_attr ()

Test serializer set attr.

test_serializer_no_document_specified ()

Test serializer no document specified.

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

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`,

`django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin`

Test suggesters.

pytestmark = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

classmethod `setUpClass` ()

Set up class.

test_nested_fields_suggesters_completion ()

Test suggesters completion for nested fields.

test_suggesters_completion ()

Test suggesters completion.

test_suggesters_completion_no_args_provided ()

Test suggesters completion with no args provided.

test_suggesters_phrase ()

Test suggesters phrase.

test_suggesters_term ()

Test suggesters term.

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

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_detail_view()
        Test detail view.

    test_listing_view()
        Test listing view.

class django_elasticsearch_dsl_drf.tests.TestWrappers (methodName='runTest')
    Bases: unittest.case.TestCase
    Test wrappers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={})]
    classmethod setUpClass()
        Hook method for setting up class fixture before running tests in the class.

    test_dict_to_obj()
        Test dict_to_obj.
        Returns

    test_obj_to_dict()
        Test obj_to_dict.
        Returns

    test_wrapper_as_json()
        Test :Wrapper:'as_json' property.
```

12.17.2 Submodules

12.17.3 django_elasticsearch_dsl_drf.analyzers module

Analyzers.

12.17.4 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'
```

12.17.5 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

12.17.6 django_elasticsearch_dsl_drf.constants module

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

12.17.7 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.Document`.

`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`

12.17.8 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_count (*es_response*)

Determine an object count, supporting either queriesets or regular lists.

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_count (*es_response*)

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, al-
                                                         low_empty_first_page=True)
```

Bases: `django.core.paginator.Paginator`

Paginator for Elasticsearch.

page (*number*)

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

Parameters *number* –

Returns

12.17.9 django_elasticsearch_dsl_drf.pip_helpers module

Pip helpers module.

`django_elasticsearch_dsl_drf.pip_helpers.check_if_installed(package, installed_packages=None)`

Check if package is installed.

Parameters

- **package** (*str*) –
- **installed_packages** (*iterable*) –

Returns

Return type bool

`django_elasticsearch_dsl_drf.pip_helpers.get_installed_packages(with_versions=False)`

Get installed packages.

Parameters **with_versions** (*bool*) – If set to True, returned with versions.

Returns

Return type list

12.17.10 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 = ()
```

12.17.11 django_elasticsearch_dsl_drf.utils module

Utils.

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

Bases: object

Dictionary proxy.

```
to_dict()
```

To dict.

Returns

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

Bases: object

Empty Search.

```
execute(*args, **kwargs)
```

```
highlight(*args, **kwargs)
```

```
sort(*args, **kwargs)
```

```
to_dict(*args, **kwargs)
```

12.17.12 django_elasticsearch_dsl_drf.versions module

Contains information about the current Elasticsearch version in use, including (LTE and GTE).

12.17.13 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.

ignore = []

pagination_class
alias of `django_elasticsearch_dsl_drf.pagination.PageNumberPagination`

class `django_elasticsearch_dsl_drf.viewsets.DocumentViewSet(*args, **kwargs)`
Bases: `django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`,
`django_elasticsearch_dsl_drf.viewsets.SuggestMixin`, `django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin`

DocumentViewSet with suggest and functional-suggest mix-ins.

class `django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin`
Bases: `object`

Functional suggest mixin.

functional_suggest(request)
Functional suggest functionality.
Parameters `request` –
Returns

class `django_elasticsearch_dsl_drf.viewsets.MoreLikeThisMixin`
Bases: `object`

More-like-this mixin.

more_like_this(request, pk=None, id=None)
More-like-this functionality detail view.
Parameters `request` –
Returns

class `django_elasticsearch_dsl_drf.viewsets.SuggestMixin`
Bases: `object`

Suggest mixin.

suggest(request)
Suggest functionality.

12.17.14 `django_elasticsearch_dsl_drf.wrappers` module

`django_elasticsearch_dsl_drf.wrappers.dict_to_obj(mapping)`
dict to obj mapping.

Parameters `mapping(dict)` –
Returns
Return type `Wrapper`

`django_elasticsearch_dsl_drf.wrappers.obj_to_dict(obj)`
Wrapper to dict.

Parameters `obj('obj':Wrapper)` –
Returns
Return type `dict`

class django_elasticsearch_dsl_drf.wrappers.**Wrapper**

Bases: object

Wrapper.

Example: >>> from django_elasticsearch_dsl_drf.wrappers import dict_to_obj >>> >>> mapping = { >>> 'country': { >>> 'name': 'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': { >>> 'name': 'Amsterdam', >>> } >>> } >>> } >>> } >>> >>> wrapper = dict_to_obj(mapping) >>> wrapper.country.name >>> "Netherlands" >>> wrapper.country.province.name >>> "North Holland" >>> wrapper.country.province.city.name >>> "Amsterdam" >>> wrapper.as_dict >>> { >>> 'country': { >>> 'name': 'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': { >>> 'name': 'Amsterdam', >>> } >>> } >>> } >>> } >>> str(wrapper) >>> "Netherlands"

as_dict

As dict.

Returns

Return type dict

as_json

As JSON.

Returns

Return type str

12.17.15 Module contents

Integrate Elasticsearch DSL with Django REST framework.

CHAPTER 13

Indices and tables

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

d

`django_elasticsearch_dsl_drf`, 277

`django_elasticsearch_dsl_drf.analyzers`, 270

`django_elasticsearch_dsl_drf.apps`, 270

`django_elasticsearch_dsl_drf.compat`, 271

`django_elasticsearch_dsl_drf.constants`, 271

`django_elasticsearch_dsl_drf.fields`, 158

`django_elasticsearch_dsl_drf.fields.common`, 153

`django_elasticsearch_dsl_drf.fields.helpers`, 155

`django_elasticsearch_dsl_drf.fields.nested_fields`, 155

`django_elasticsearch_dsl_drf.filter_backends`, 238

`django_elasticsearch_dsl_drf.filter_backends.aggregations`, 161

`django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations`, 161

`django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations`, 161

`django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations`, 161

`django_elasticsearch_dsl_drf.filter_backends.faceted_search`, 233

`django_elasticsearch_dsl_drf.filter_backends.filtering`, 178

`django_elasticsearch_dsl_drf.filter_backends.filtering.common`, 161

`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial`, 169

`django_elasticsearch_dsl_drf.filter_backends.filtering.ids`, 173

`django_elasticsearch_dsl_drf.filter_backends.filtering.nested`, 175

`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter`, 176

`django_elasticsearch_dsl_drf.filter_backends.highlights`, 236

`django_elasticsearch_dsl_drf.filter_backends.mixins`, 237

`django_elasticsearch_dsl_drf.filter_backends.ordering`, 197

`django_elasticsearch_dsl_drf.filter_backends.ordering_fields`, 192

`django_elasticsearch_dsl_drf.filter_backends.ordering_filters`, 195

`django_elasticsearch_dsl_drf.filter_backends.search_filters`, 214

`django_elasticsearch_dsl_drf.filter_backends.search_filters.aggregations`, 210

`django_elasticsearch_dsl_drf.filter_backends.search_filters.aggregations.bucket_aggregations`, 213

`django_elasticsearch_dsl_drf.filter_backends.search_filters.aggregations.metrics_aggregations`, 206

`django_elasticsearch_dsl_drf.filter_backends.search_filters.aggregations.pipeline_aggregations`, 202

`django_elasticsearch_dsl_drf.filter_backends.search_filters.faceted_search`, 202

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering`, 202

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering.common`, 203

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering.geo_spatial`, 203

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering.ids`, 204

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering.nested`, 205

`django_elasticsearch_dsl_drf.filter_backends.search_filters.filtering.post_filter`, 213

[django_elasticsearch_dsl_drf.filter_backends.suggester](#),
[226](#) [django_elasticsearch_dsl_drf.tests.test_suggesters](#),
[django_elasticsearch_dsl_drf.filter_backends.suggester.functional](#),
[217](#) [django_elasticsearch_dsl_drf.tests.test_views](#),
[django_elasticsearch_dsl_drf.filter_backends.suggester.native](#),
[222](#) [django_elasticsearch_dsl_drf.tests.test_wrappers](#),
[django_elasticsearch_dsl_drf.helpers](#), [256](#)
[271](#) [django_elasticsearch_dsl_drf.utils](#), [275](#)
[django_elasticsearch_dsl_drf.pagination](#), [django_elasticsearch_dsl_drf.versions](#),
[272](#) [275](#)
[django_elasticsearch_dsl_drf.pip_helpers](#) [django_elasticsearch_dsl_drf.viewsets](#),
[274](#) [275](#)
[django_elasticsearch_dsl_drf.serializers](#) [django_elasticsearch_dsl_drf.wrappers](#),
[274](#) [276](#)
[django_elasticsearch_dsl_drf.tests](#), [256](#)
[django_elasticsearch_dsl_drf.tests.base](#),
[239](#)
[django_elasticsearch_dsl_drf.tests.data_mixins](#),
[239](#)
[django_elasticsearch_dsl_drf.tests.test_faceted_search](#),
[239](#)
[django_elasticsearch_dsl_drf.tests.test_filtering_common](#),
[240](#)
[django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial](#),
[243](#)
[django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations](#),
[244](#)
[django_elasticsearch_dsl_drf.tests.test_filtering_nested](#),
[244](#)
[django_elasticsearch_dsl_drf.tests.test_filtering_post_filter](#),
[246](#)
[django_elasticsearch_dsl_drf.tests.test_functional_suggesters](#),
[248](#)
[django_elasticsearch_dsl_drf.tests.test_helpers](#),
[249](#)
[django_elasticsearch_dsl_drf.tests.test_highlight](#),
[249](#)
[django_elasticsearch_dsl_drf.tests.test_more_like_this](#),
[249](#)
[django_elasticsearch_dsl_drf.tests.test_ordering_common](#),
[249](#)
[django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial](#),
[251](#)
[django_elasticsearch_dsl_drf.tests.test_pagination](#),
[251](#)
[django_elasticsearch_dsl_drf.tests.test_pip_helpers](#),
[251](#)
[django_elasticsearch_dsl_drf.tests.test_search](#),
[252](#)
[django_elasticsearch_dsl_drf.tests.test_search_multi_match](#),
[252](#)
[django_elasticsearch_dsl_drf.tests.test_search_simple_query_string](#),
[253](#)
[django_elasticsearch_dsl_drf.tests.test_serializers](#),

A

AddressesMixin (class in `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering`),
 `django_elasticsearch_dsl_drf.tests.data_mixins`), class method), 176
 239
 `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering`),
 aggregate() (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend`
 method), 234
 `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering`),
 apply_filter() (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend`
 class method), 175
 `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering`),
 apply_filter() (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend`
 class method), 190
 `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.mixins.F`
 apply_filter() (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`
 class method), 177
 `apply_query_contains()`
 apply_filter() (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`
 class method), 191
 `apply_query_contains()`
 apply_filter() (`django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`
 class method), 237
 `django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringF`
 apply_filter_prefix() (class method), 180
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`
 class method), 162
 `apply_query_endswith()`
 apply_filter_prefix() (class method), 164
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`
 class method), 179
 `apply_query_endswith()`
 apply_filter_range() (class method), 180
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`
 class method), 163
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.F`
 apply_filter_range() (class method), 164
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`
 class method), 179
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringF`
 apply_filter_term() (class method), 180
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`
 class method), 163
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.F`
 apply_filter_term() (class method), 165
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringF`
 class method), 179
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringF`
 apply_filter_terms() (class method), 181
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`
 class method), 163
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.F`
 apply_filter_terms() (class method), 170
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`
 class method), 170
 (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatial`

BaseSearchQueryBackend (class in construct_search()
 django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 206
 class method), 206

BaseSearchQueryBackend (class in construct_search()
 django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 202
 class method), 202

BaseTestCase (class in construct_search()
 django_elasticsearch_dsl_drf.tests.base), (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 239
 class method), 202

BooksMixin (class in construct_search()
 django_elasticsearch_dsl_drf.tests.data_mixins), (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 239
 class method), 203

BooleanField (class in construct_search()
 django_elasticsearch_dsl_drf.fields), 158 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 207
 class method), 207

BooleanField (class in construct_search()
 django_elasticsearch_dsl_drf.fields.common), construct_search()
 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 153
 class method), 207

C

CharField (class in construct_search()
 django_elasticsearch_dsl_drf.fields), 158 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 206
 class method), 206

CharField (class in construct_search()
 django_elasticsearch_dsl_drf.fields.common), (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 154
 class method), 203

check_if_installed() (in module construct_search()
 django_elasticsearch_dsl_drf.pip_helpers), (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 274
 class method), 207

clean_queryset() (django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend
 method), 220 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 204
 class method), 204

clean_queryset() (django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend
 method), 232 construct_search()
 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 214
 class method), 208

CompoundSearchFilterBackend (class in construct_search()
 django_elasticsearch_dsl_drf.filter_backends.search), class method), 208
 214 construct_search()
 (django_elasticsearch_dsl_drf.filter_backends.search.compound), 211
 class method), 205

Config (class in django_elasticsearch_dsl_drf.apps), (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 270
 class method), 209

construct_facets() construct_search()
 (django_elasticsearch_dsl_drf.filter_backends.faceted_search.HierarchicalFacetBackend), 234
 method), 216

construct_nested_search() create() (django_elasticsearch_dsl_drf.serializers.DocumentSerializer
 (django_elasticsearch_dsl_drf.filter_backends.search.historical.SearchFilterBackend
 method), 211 create_books() (django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin
 class method), 239
 (django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend)
 method), 215 (django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
 class method), 239

construct_search() class method), 239
 (django_elasticsearch_dsl_drf.filter_backends.search.historical.SearchFilterBackend
 method), 212

D

construct_search() DateField (class in
 (django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 202
 class method), 202 DateField (class in

<code>django_elasticsearch_dsl_drf.fields.common</code>), 154	<code>django_elasticsearch_dsl_drf.filter_backends.filter</code> (module), 175
<code>deep_get()</code> (<code>django_elasticsearch_dsl_drf.tests.test_ordering</code>), static method), 249	<code>django_elasticsearch_dsl_drf.filter_backends.filter</code> (module), 176
<code>deep_get()</code> (<code>django_elasticsearch_dsl_drf.tests.TestOrdering</code>), static method), 266	<code>django_elasticsearch_dsl_drf.filter_backends.highl</code> (module), 236
<code>DefaultOrderingFilterBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.mixins</code>), <code>django_elasticsearch_dsl_drf.filter_backends.ordering</code>), 197	<code>django_elasticsearch_dsl_drf.filter_backends.ordering</code> (module), 197
<code>DefaultOrderingFilterBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.ordering.common</code>), 192	<code>django_elasticsearch_dsl_drf.filter_backends.ordering</code> (module), 192
<code>dict_to_obj()</code> (in module <code>django_elasticsearch_dsl_drf.wrappers</code>), 276	<code>django_elasticsearch_dsl_drf.filter_backends.ordering</code> (module), 195
<code>DictionaryProxy</code> (class in <code>django_elasticsearch_dsl_drf.utils</code>), 275	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 214
<code>django_elasticsearch_dsl_drf</code> (module), 277	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 210
<code>django_elasticsearch_dsl_drf.analyzers</code> (module), 270	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 211
<code>django_elasticsearch_dsl_drf.apps</code> (module), 270	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 211
<code>django_elasticsearch_dsl_drf.compat</code> (module), 271	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 213
<code>django_elasticsearch_dsl_drf.constants</code> (module), 271	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 206
<code>django_elasticsearch_dsl_drf.fields</code> (module), 158	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 202
<code>django_elasticsearch_dsl_drf.fields.common</code> (module), 153	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 202
<code>django_elasticsearch_dsl_drf.fields.helpers</code> (module), 155	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 202
<code>django_elasticsearch_dsl_drf.fields.nested</code> (module), 155	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 203
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 238	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 203
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 161	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 204
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 161	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 205
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 161	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 213
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 161	<code>django_elasticsearch_dsl_drf.filter_backends.search</code> (module), 213
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 233	<code>django_elasticsearch_dsl_drf.filter_backends.suggest</code> (module), 226
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 178	<code>django_elasticsearch_dsl_drf.filter_backends.suggest</code> (module), 217
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 161	<code>django_elasticsearch_dsl_drf.filter_backends.suggest</code> (module), 222
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 169	<code>django_elasticsearch_dsl_drf.helpers</code> (module), 271
<code>django_elasticsearch_dsl_drf.filter_backend</code> (module), 173	<code>django_elasticsearch_dsl_drf.pagination</code> (module), 272

[django_elasticsearch_dsl_drf.pip_helpers](#) [django_elasticsearch_dsl_drf.versions](#)
 (module), 274 (module), 275
[django_elasticsearch_dsl_drf.serializers](#) [django_elasticsearch_dsl_drf.viewsets](#)
 (module), 274 (module), 275
[django_elasticsearch_dsl_drf.tests](#) (module), 256 [django_elasticsearch_dsl_drf.wrappers](#)
 (module), 276
[django_elasticsearch_dsl_drf.tests.base](#) [django.paginator_class](#)
 (module), 239 (django_elasticsearch_dsl_drf.pagination.PageNumberPagination
 attribute), 273
[django_elasticsearch_dsl_drf.tests.data_mixins](#) (module), 239 document (django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_faceted](#) (module), 239 document_uid_field
 (module), 239 (django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_filtering](#) (module), 240 (class in
 django_elasticsearch_dsl_drf.serializers),
 (module), 243 274 global aggregations
 (module), 244 DocumentSerializerMeta (class in
 django_elasticsearch_dsl_drf.serializers),
 (module), 244 274
[django_elasticsearch_dsl_drf.tests.test_document_viewset_filter](#) (class in
 django_elasticsearch_dsl_drf.viewsets), 276
 (module), 246
[django_elasticsearch_dsl_drf.tests.test_functional_suggesters](#)
 (module), 248
E
[django_elasticsearch_dsl_drf.tests.test_helpers](#) (class in
 django_elasticsearch_dsl_drf.utils), 275
 (module), 249 (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_highlight](#) (django_elasticsearch_dsl_drf.utils.EmptySearch
 method), 275
 (module), 249
[django_elasticsearch_dsl_drf.tests.test_execute_like](#) (method), 275
 (module), 249
[django_elasticsearch_dsl_drf.tests.test_ordering_common](#) (method), 275
 (module), 249 (django_elasticsearch_dsl_drf.filter_backends.suggester.function
 method), 275
[django_elasticsearch_dsl_drf.tests.test_ordering_spatial](#)
 (module), 251 extract_field_name()
 (module), 251 (django_elasticsearch_dsl_drf.filter_backends.suggester.Function
 method), 232
[django_elasticsearch_dsl_drf.tests.test_pagination](#)
 (module), 251
[django_elasticsearch_dsl_drf.tests.test_pip_helpers](#)
 (module), 251
F
[django_elasticsearch_dsl_drf.tests.test_search](#) (module), 252 faceted_search_param
 (module), 252 (django_elasticsearch_dsl_drf.filter_backends.faceted_search.Fac
 attribute), 235
 (module), 252 FacetedSearchFilterBackend (class in
 django_elasticsearch_dsl_drf.filter_backends.faceted_search),
 (module), 253 233
[django_elasticsearch_dsl_drf.tests.test_serializers](#) (module), 254 field_aliases (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_suggesters](#) (module), 254 field_options (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_views](#) (module), 255 (django_elasticsearch_dsl_drf.serializers.Meta
 attribute), 275
[django_elasticsearch_dsl_drf.tests.test_wrappers](#) (module), 256 filter_queryset()
 (module), 256 (django_elasticsearch_dsl_drf.filter_backends.faceted_search.Fac
 method), 235
[django_elasticsearch_dsl_drf.utils](#) (module), 275

[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.CommonFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 168)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 183)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 171)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackendMixin](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 186)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.ids.FilteringFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 174)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.filtering](#)), 188)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend](#) (class in [django_elasticsearch_dsl_drf.fields](#)), 236)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 193)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 194)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 198)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.GeoSpatialOrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 195)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 199)
G
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.ordering](#)), 201)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), 210)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.search.base.SearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), 214)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.search.geospatial.GeoSpatialFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), 213)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.search.geospatial.FilteringFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), 217)
[filter_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.suggester](#)), 221)

198
GeoSpatialOrderingFilterBackend (class in `django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial`), 204
195
`get_coreschema_field()` (class method), 208
(`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`), 206
method), 168
`get_coreschema_field()` (class method), 206
(`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`), 200
method), 184
`get_coreschema_field()` (class method), 274
(`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend`), 176
method), 176
`get_coreschema_field()` (class method), 176
(`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`), 190
method), 190
`get_coreschema_field()` (class method), 190
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 178
method), 178
`get_coreschema_field()` (class method), 168
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 192
method), 192
`get_coreschema_field()` (class method), 184
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 210
method), 210
`get_coreschema_field()` (class method), 171
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 214
method), 214
`get_coreschema_field()` (class method), 186
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 213
method), 213
`get_coreschema_field()` (class method), 176
(`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend`), 217
method), 217
`get_count()` (`django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination`), 190
method), 272
`get_count()` (`django_elasticsearch_dsl_drf.pagination.PageNumberPagination`), 190
method), 273
`get_default_ordering_params()` (class method), 193
(`django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend`), 186
class method), 186
`get_default_ordering_params()` (class method), 172
(`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 198
class method), 198
`get_document_for_model()` (in module `django_elasticsearch_dsl_drf.helpers`), 271
`get_elasticsearch_version()` (in module `django_elasticsearch_dsl_drf.compat`), 271
`get_faceted_search_query_params()` (class method), 235
(`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend`), 235
method), 235
`get_facets()` (`django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination`), 199
method), 272
`get_facets()` (`django_elasticsearch_dsl_drf.pagination.PageNumberPagination`), 199
method), 272

[method\), 178](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.filtering.OrderingFilterBackend class method\), 192](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.ordering.CommonOrderingFilterBackend class method\), 195](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend class method\), 201](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend class method\), 210](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.base.SearchFilterBackend class method\), 214](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.historical.SubchFilterBackend class method\), 213](#)
[get_schema_fields\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend class method\), 217](#)
[get_search_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend class method\), 210](#)
[get_search_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.base.SearchFilterBackend class method\), 214](#)
[get_search_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.historical.SubchFilterBackend class method\), 213](#)
[get_search_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend class method\), 217](#)
[get_suggester_context\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterFilterBackend class method\), 225](#)
[get_suggester_context\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend class method\), 228](#)
[get_suggester_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend class method\), 221](#)
[get_suggester_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend class method\), 232](#)
[get_suggester_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterFilterBackend class method\), 226](#)
[get_suggester_query_params\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend class method\), 230](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.BooleanField attribute\), 158](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.CharField attribute\), 158](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.BooleanField attribute\), 153](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.CharField attribute\), 154](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.DateField attribute\), 154](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.FloatField attribute\), 154](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.IntegerField attribute\), 154](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.common.IPAddressField attribute\), 154](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.DateField attribute\), 158](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.FloatField attribute\), 158](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.IntegerField attribute\), 159](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.IPAddressField attribute\), 160](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.ListField attribute\), 160](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.nested_fields.ListField attribute\), 158](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField attribute\), 157](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.fields.ObjectField attribute\), 157](#)
[get_value\(\) \(django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend class attribute\), 275](#)
[highlight_param\(\) \(django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend class method\), 275](#)
[HighlightBackend \(class in django_elasticsearch_dsl_drf.filter_backends.highlight\), 275](#)
[ids_query_param\(\) \(django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend class attribute\), 189](#)
[IdsFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.filtering\), 188](#)
[IdsFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.filtering.ids\), 188](#)
[ignore\(\) \(django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet attribute\), 276](#)

H

I

<code>ignore_fields</code> (<code>django_elasticsearch_dsl_drf.serializers.Meta</code> attribute), 275	<code>MatchPhraseQueryBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.search.query_backends</code>), 207
<code>index_aliases</code> (<code>django_elasticsearch_dsl_drf.serializers.Meta</code> attribute), 275	<code>MatchPhraseQueryBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.search.query_backends</code>), 202
<code>index_classes</code> (<code>django_elasticsearch_dsl_drf.serializers.Meta</code> attribute), 275	<code>MatchQueryBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.search.query_backends</code>), 206
<code>IntegerField</code> (class in <code>django_elasticsearch_dsl_drf.fields</code>), 159	<code>MatchQueryBackend</code> (class in <code>django_elasticsearch_dsl_drf.filter_backends.search.query_backends</code>), 202
<code>IntegerField</code> (class in <code>django_elasticsearch_dsl_drf.fields.common</code>), 154	<code>Meta</code> (class in <code>django_elasticsearch_dsl_drf.serializers</code>), 274
<code>IPAddressField</code> (class in <code>django_elasticsearch_dsl_drf.fields</code>), 159	<code>more_like_this()</code> (<code>django_elasticsearch_dsl_drf.viewsets.MoreLikeThis</code> method), 276
<code>IPAddressField</code> (class in <code>django_elasticsearch_dsl_drf.fields.common</code>), 154	

K

KeywordField()	(in	module	MoreLikeThisMixin	(class	in
django_elasticsearch_dsl_drf.compat),	271		django_elasticsearch_dsl_drf.viewsets),	276	
			MultiMatchQueryBackend	(class	in
			django_elasticsearch_dsl_drf.filter_backends.search.query_backe		
			207		
label	(django_elasticsearch_dsl_drf.apps.Config	at-	MultiMatchQueryBackend	(class	in
tribute),	270		django_elasticsearch_dsl_drf.filter_backends.search.query_backe		
LimitOffsetPagination	(class	in	203		
django_elasticsearch_dsl_drf.pagination),			MultiMatchSearchFilterBackend	(class	in
272			django_elasticsearch_dsl_drf.filter_backends.search),		
ListField	(class	in	215		
django_elasticsearch_dsl_drf.fields),	160		MultiMatchSearchFilterBackend	(class	in
ListField	(class	in	django_elasticsearch_dsl_drf.filter_backends.search.multi_match		
django_elasticsearch_dsl_drf.fields.nested_fields),			213		
157					

M

matching (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend attribute), 210	name (django_elasticsearch_dsl_drf.apps.Config attribute), 270
matching (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend attribute), 214	NestedField (class in django_elasticsearch_dsl_drf.fields), 160
matching (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend attribute), 213	NestedField (class in django_elasticsearch_dsl_drf.fields.nested_fields), 159
matching (django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend attribute), 215	NestedFilteringFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.filtering), 189
matching (django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.SimpleQueryStringFilterBackend attribute), 214	NestedSimpleQueryStringFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.filtering.nested), 175
matching (django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.SimpleQueryStringFilterBackend attribute), 217	NestedQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 207
MatchPhrasePrefixQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 207	NestedQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 208
MatchPhrasePrefixQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 203	NestedQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 204

N

matching (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.base), 210	attribute), 210
matching (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.base), 214	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.base), 214
matching (django_elasticsearch_dsl_drf.filter_backends.search.multi_match.MultiMatchSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.multi_match), 213	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.multi_match), 213
matching (django_elasticsearch_dsl_drf.filter_backends.search.multi_match.MultiMatchSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.multi_match), 215	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.multi_match), 215
matching (django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.SimpleQueryStringSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string), 214	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string), 214
matching (django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.SimpleQueryStringSearchFilterBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string), 217	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string), 217
MatchPhrasePrefixQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend, 207	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 207
MatchPhrasePrefixQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend, 208	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 208
MatchPhrasePrefixQueryBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend, 203	NestedFieldBackend (class in django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 203

O

O

- `obj_to_dict()` (in `module django_elasticsearch_dsl_drf.wrappers`), 276
- `ObjectField` (class in `django_elasticsearch_dsl_drf.fields`), 160
- `ObjectField` (class in `django_elasticsearch_dsl_drf.fields.nested_fields`), 156
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend` attribute), 194
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingFilterBackend` attribute), 195
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend` attribute), 198
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.GeoSpatialOrderingFilterBackend` attribute), 197
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend` attribute), 200
- `ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend` attribute), 201
- `OrderingFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.ordering`), 200
- `OrderingFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.ordering.common`), 194

P

- `Page` (class in `django_elasticsearch_dsl_drf.pagination`), 272
- `page()` (`django_elasticsearch_dsl_drf.pagination.Paginator` method), 273
- `PageNumberPagination` (class in `django_elasticsearch_dsl_drf.pagination`), 273
- `paginate_queryset()` (`django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination` method), 272
- `paginate_queryset()` (`django_elasticsearch_dsl_drf.pagination.PageNumberPagination` method), 273
- `pagination_class` (`django_elasticsearch_dsl_drf.views.BaseDocumentViewSet` attribute), 276
- `Paginator` (class in `django_elasticsearch_dsl_drf.pagination`), 273
- `PostFilterFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering`), 191
- `PostFilterFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter`), 176

prepare_faceted_search_fields() (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearch` class method), 235

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.Filtering` class method), 169

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 184

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.DefaultOrderingFilterBackend` class method), 173

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialOrderingFilterBackend` class method), 188

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilterBackend` class method), 176

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend` class method), 191

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterBackend` class method), 178

prepare_filter_fields() (`django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilterBackend` class method), 192

prepare_highlight_fields() (`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightFilterBackend` class method), 237

prepare_suggester_fields() (`django_elasticsearch_dsl_drf.filter_backends.suggester.function.FunctionalSuggester` class method), 221

prepare_suggester_fields() (`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggester` class method), 233

prepare_suggester_fields() (`django_elasticsearch_dsl_drf.filter_backends.suggester.native.NativeSuggester` class method), 226

prepare_suggester_fields() (`django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester` class method), 230

pytestmark (`django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTest` attribute), 239

pytestmark (`django_elasticsearch_dsl_drf.tests.base.BaseTestCase` attribute), 239

pytestmark (`django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch` attribute), 240

pytestmark (`django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon` attribute), 240

pytestmark (`django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial` attribute), 243

pytestmark (`django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregation.TestFilteringGlobalAggregation` attribute), 244

[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.QueryBackend.query_type\(attribute\), 204](#)
[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchQueryBackend.query_type\(attribute\), 208](#)
[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.NestedQueryBackend.query_type\(attribute\), 205](#)
[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.NestedQueryBackend.TestMultiMatchSearchBackend.query_type\(attribute\), 209](#)
[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.QueryStringQueryBackend.query_type\(attribute\), 206](#)
[query_type\(django_elasticsearch_dsl_drf.filter_backends.search\)\(django_elasticsearch_dsl_drf.filter_backends.search.QueryStringSearchBackend.query_type\(attribute\), 210](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTest case method\), 239](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.base.BaseTestCase case method\), 239](#)
[search_fields\(django_elasticsearch_dsl_drf.serializers.Meta attribute\), 275](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend.search_param\(attribute\), 210](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend.search_param\(attribute\), 214](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend.search_param\(attribute\), 213](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend.search_param\(attribute\), 213](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend.search_param\(attribute\), 215](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend.search_param\(attribute\), 217](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.QueryStringSearchFilterBackend.search_param\(attribute\), 214](#)
[search_param\(django_elasticsearch_dsl_drf.filter_backends.search.QueryStringSearchFilterBackend.search_param\(attribute\), 217](#)
[SearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search\), class method\), 250](#)
[SearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search.history\), class method\), 251](#)
[SearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search.history\), class method\), 251](#)
[serialize_queryset\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend method\), 221](#)
[serialize_queryset\(\) \(django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend method\), 233](#)
[serializers\(django_elasticsearch_dsl_drf.serializers.Meta attribute\), 275](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch class method\), 240](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_highlight.TestHighlight class method\), 249](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_search.TestSearch class method\), 252](#)
[setUp\(\) \(django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch class method\), 253](#)

[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringStubAlgorithms class method\), 260](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringNestedFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.suggester\), class method\), 260](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter SuggesterFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.suggester.native\), class method\), 262](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFunctionalSuggester class method\), 264](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestHelpersTestMixin \(class in django_elasticsearch_dsl_drf.viewsets\), class method\), 265](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 266](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestOrderingGeoSpatial class method\), 267](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestPagination method\), 250](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestPipHelpers\(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 268](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestSuggesters test_address_default_order_by\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO class method\), 269](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestViews method\), 250](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestWrappers \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 270](#)
[SimpleQueryStringQueryBackend \(class in test_address_order_by_nested_field_continent_ascen django_elasticsearch_dsl_drf.filter_backends.search.query_backend\), 209](#)
[SimpleQueryStringQueryBackend \(class in test_address_order_by_nested_field_continent_ascen django_elasticsearch_dsl_drf.filter_backends.search.query_backend.simple_query_string\), 205](#)
[SimpleQueryStringSearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search\) test_address_order_by_nested_field_continent_descen \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO class method\), 217](#)
[SimpleQueryStringSearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search\) test_address_order_by_nested_field_continent_descen \(simple_query_string\) \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 213](#)
[sort\(\) \(django_elasticsearch_dsl_drf.utils.EmptySearch test_address_order_by_nested_field_country_ascendi method\), 275](#)
[sort_by_list\(\) \(in module \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO method\), 250](#)
[split_lookup_complex_multiple_value\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_complex_value\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_filter\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_name\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[StringField\(\) \(in module \(django_elasticsearch_dsl_drf.tests.TestOrdering method\), 267](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringStubAlgorithms class method\), 260](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringNestedFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.suggester\), class method\), 260](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter SuggesterFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.suggester.native\), class method\), 262](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestFunctionalSuggester class method\), 264](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestHelpersTestMixin \(class in django_elasticsearch_dsl_drf.viewsets\), class method\), 265](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 266](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestOrderingGeoSpatial class method\), 267](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestPagination method\), 250](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestPipHelpers\(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 268](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestSuggesters test_address_default_order_by\(\) \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO class method\), 269](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestViews method\), 250](#)
[setUpClass\(\) \(django_elasticsearch_dsl_drf.tests.TestWrappers \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 270](#)
[SimpleQueryStringQueryBackend \(class in test_address_order_by_nested_field_continent_ascen django_elasticsearch_dsl_drf.filter_backends.search.query_backend\), 209](#)
[SimpleQueryStringQueryBackend \(class in test_address_order_by_nested_field_continent_ascen django_elasticsearch_dsl_drf.filter_backends.search.query_backend.simple_query_string\), 205](#)
[SimpleQueryStringSearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search\) test_address_order_by_nested_field_continent_descen \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO class method\), 217](#)
[SimpleQueryStringSearchFilterBackend \(class in django_elasticsearch_dsl_drf.filter_backends.search\) test_address_order_by_nested_field_continent_descen \(simple_query_string\) \(django_elasticsearch_dsl_drf.tests.TestOrdering class method\), 213](#)
[sort\(\) \(django_elasticsearch_dsl_drf.utils.EmptySearch test_address_order_by_nested_field_country_ascendi method\), 275](#)
[sort_by_list\(\) \(in module \(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO method\), 250](#)
[split_lookup_complex_multiple_value\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_complex_value\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_filter\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[split_lookup_name\(\) \(django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin method\), 238](#)
[StringField\(\) \(in module \(django_elasticsearch_dsl_drf.tests.TestOrdering method\), 267](#)

test_author_order_by_field_id_ascending()	test_book_order_by_non_existent_field()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)
test_author_order_by_field_id_ascending()	test_book_order_by_non_existent_field()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)
test_author_order_by_field_id_descending()	test_check_if_installed()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers, 251)
test_author_order_by_field_id_descending()	test_check_if_installed()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestPipHelpers, 268)
test_author_order_by_field_name_ascending()	test_compound_search_boost_by_field()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_search.TestSearch, 252)
test_author_order_by_field_name_ascending()	test_compound_search_boost_by_field()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestSearch, 268)
test_author_order_by_field_name_descending()	test_compound_search_by_field()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_search.TestSearch, 252)
test_author_order_by_field_name_descending()	test_compound_search_by_field()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestSearch, 268)
test_book_default_order_by()	test_compound_search_by_field_multi_terms()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_search.TestSearch, 252)
test_book_default_order_by()	test_compound_search_by_field_multi_terms()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestSearch, 268)
test_book_order_by_field_id_ascending()	test_compound_search_by_nested_field()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_search.TestSearch, 252)
test_book_order_by_field_id_ascending()	test_compound_search_by_nested_field()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestSearch, 268)
test_book_order_by_field_id_descending()	test_default_filter_lookup()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon, 240)
test_book_order_by_field_id_descending()	test_default_filter_lookup()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestFilteringCommon, 256)
test_book_order_by_field_title_ascending()	test_detail_view()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_views.TestViews, 255)
test_book_order_by_field_title_ascending()	test_detail_view()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestViews, 270)
test_book_order_by_field_title_descending()	test_dict_to_obj()
(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon, 250)	(django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers, 256)
test_book_order_by_field_title_descending()	test_dict_to_obj()
(django_elasticsearch_dsl_drf.tests.TestOrdering, 267)	(django_elasticsearch_dsl_drf.tests.TestWrappers, 270)

test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 240	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 240
test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 244	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 244
test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 246	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 246
test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 257	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 257
test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 260	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261
test_field_filter_contains() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 262	test_field_filter_exists_false() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 262
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 240	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 240
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 244	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 245
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 246	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 246
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 257	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 257
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 260	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261
test_field_filter_endswith() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 262	test_field_filter_exists_true() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 262
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 240	test_field_filter_geo_bounding_box() (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial method), 243
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 244	test_field_filter_geo_bounding_box() (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial method), 259
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 246	test_field_filter_geo_bounding_box_fail_test() (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial method), 243
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 257	test_field_filter_geo_bounding_box_fail_test() (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial method), 259
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261	test_field_filter_geo_distance() (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial method), 243
test_field_filter_exclude() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 262	test_field_filter_geo_distance() (django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial.TestFilteringGeoSpatial method), 251

```

test_field_filter_geo_distance()          test_field_filter_gt_with_boost()
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon
    method), 259)                                method), 241
test_field_filter_geo_distance()          test_field_filter_gt_with_boost()
    (django_elasticsearch_dsl_drf.tests.TestOrderingGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter
    method), 267)                                method), 246
test_field_filter_geo_distance_distance_type_field_filter_gt_with_boost()
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringCommon
    method), 243)                                method), 257
test_field_filter_geo_distance_distance_type_field_filter_gt_with_boost()
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter
    method), 259)                                method), 262
test_field_filter_geo_distance_not_enough_args_field_filter_gte()
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon
    method), 243)                                method), 241
test_field_filter_geo_distance_not_enough_args_field_filter_gte()
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter
    method), 260)                                method), 246
test_field_filter_geo_polygon()           test_field_filter_gte()
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringCommon
    method), 243)                                method), 257
test_field_filter_geo_polygon()           test_field_filter_gte()
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter
    method), 260)                                method), 262
test_field_filter_geo_polygon_fail_test(test_field_filter_in())
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon
    method), 243)                                method), 241
test_field_filter_geo_polygon_fail_test(test_field_filter_in())
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested
    method), 260)                                method), 245
test_field_filter_geo_polygon_string_options(field_filter_in())
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter
    method), 243)                                method), 247
test_field_filter_geo_polygon_string_options(field_filter_in())
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringCommon
    method), 260)                                method), 257
test_field_filter_geo_polygon_string_options_field_filter_in()
    (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringNested
    method), 243)                                method), 261
test_field_filter_geo_polygon_string_options_field_filter_in()
    (django_elasticsearch_dsl_drf.tests.TestFilteringGeoSpatial(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter
    method), 260)                                method), 263
test_field_filter_gt()                   test_field_filter_isnull_false()
    (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon
    method), 241)                                method), 241
test_field_filter_gt()                   test_field_filter_isnull_false()
    (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter
    method), 246)                                method), 247
test_field_filter_gt()                   test_field_filter_isnull_false()
    (django_elasticsearch_dsl_drf.tests.TestFilteringCommon(django_elasticsearch_dsl_drf.tests.TestFilteringCommon
    method), 257)                                method), 257
test_field_filter_gt()                   test_field_filter_isnull_false()
    (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter(django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter
    method), 262)                                method), 263

```

test_field_filter_isnull_true() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 241	test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247
test_field_filter_isnull_true() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247	test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258
test_field_filter_isnull_true() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258	test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests. TestFilteringNested method), 261
test_field_filter_isnull_true() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263	test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263
test_field_filter_lt() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 241	test_field_filter_range() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 242
test_field_filter_lt() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247	test_field_filter_range() (django_elasticsearch_dsl_drf.tests.test_filtering_nested. TestFilteringNested method), 245
test_field_filter_lt() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258	test_field_filter_range() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247
test_field_filter_lt() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263	test_field_filter_range() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258
test_field_filter_lt_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 241	test_field_filter_range() (django_elasticsearch_dsl_drf.tests. TestFilteringNested method), 261
test_field_filter_lt_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247	test_field_filter_range() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263
test_field_filter_lt_with_boost() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 242
test_field_filter_lt_with_boost() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering_nested. TestFilteringNested method), 245
test_field_filter_lte() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 241	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247
test_field_filter_lte() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. TestFilteringPostFilter method), 247	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258
test_field_filter_lte() (django_elasticsearch_dsl_drf.tests. TestFilteringCommon method), 258	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests. TestFilteringNested method), 261
test_field_filter_lte() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263	test_field_filter_range_with_boost() (django_elasticsearch_dsl_drf.tests. TestFilteringPostFilter method), 263
test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 242	test_field_filter_term() (django_elasticsearch_dsl_drf.tests.test_filtering_common. TestFilteringCommon method), 242
test_field_filter_prefix() (django_elasticsearch_dsl_drf.tests.test_filtering_nested. TestFilteringNested method), 245	test_field_filter_term() (django_elasticsearch_dsl_drf.tests.test_filtering_nested. TestFilteringNested method), 245

test_field_filter_term() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 247	test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 248
test_field_filter_term() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 258	test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 258
test_field_filter_term() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261	test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261
test_field_filter_term() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264	test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 242	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 242
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 245	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 245
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 248	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 248
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 258	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 258
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261
test_field_filter_term_explicit() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264	test_field_filter_wildcard() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 242	test_filter_by_field() (django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers method), 249
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 245	test_filter_by_field() (django_elasticsearch_dsl_drf.tests.TestHelpers method), 265
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter method), 248	test_get_installed_packages() (django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers method), 252
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 258	test_get_installed_packages() (django_elasticsearch_dsl_drf.tests.TestPipHelpers method), 268
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261	test_get_installed_packages_with_versions() (django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers method), 252
test_field_filter_terms_list() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264	test_get_installed_packages_with_versions() (django_elasticsearch_dsl_drf.tests.TestPipHelpers method), 268
test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 242	test_ids_empty_filter() (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon method), 242
test_field_filter_terms_string() (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested method), 245	test_ids_empty_filter() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 259

test_ids_filter() (django_elasticsearch_dsl_drf.tests.test_filtering_common. method), 242	test_nested_fields_suggesters_completion() (django_elasticsearch_dsl_drf.tests.TestSuggesters method), 269
test_ids_filter() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. method), 248	test_obj_to_dict() (django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers method), 256
test_ids_filter() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 259	test_obj_to_dict() (django_elasticsearch_dsl_drf.tests.TestWrappers method), 270
test_ids_filter() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264	test_pagination() (django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination method), 251
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch method), 240	test_pagination() (django_elasticsearch_dsl_drf.tests.TestPagination method), 268
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregation.TestFilteringGlobalAggregation method), 244	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregation.TestFilteringGlobalAggregation method), 242
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. method), 248	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter. method), 245
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.TestFacetedSearch method), 256	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestF method), 248
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.TestFilteringGlobalAgg method), 260	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO method), 251
test_list_results_with_facets() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_search.TestSearch method), 252
test_list_results_with_highlights() (django_elasticsearch_dsl_drf.tests.test_highlight.TestHighl method), 249	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_search_multi_match.Test method), 253
test_list_results_with_highlights() (django_elasticsearch_dsl_drf.tests.TestHighlight method), 265	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.test_search_simple_query_str method), 253
test_listing_view() (django_elasticsearch_dsl_drf.tests.test_views.TestViews method), 255	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 259
test_listing_view() (django_elasticsearch_dsl_drf.tests.TestViews method), 270	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestFilteringNested method), 261
test_more_like_this() (django_elasticsearch_dsl_drf.tests.test_more_like_this.TestM method), 249	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter method), 264
test_nested_field_filter_term() (django_elasticsearch_dsl_drf.tests.test_filtering_common. method), 242	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch method), 265
test_nested_field_filter_term() (django_elasticsearch_dsl_drf.tests.TestFilteringCommon method), 259	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestOrdering method), 267
test_nested_fields_suggesters_completion() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSugg method), 255	test_schema_field_not_required() (django_elasticsearch_dsl_drf.tests.TestSearch method), 268

test_schema_field_not_required() *method*), 253
 (django_elasticsearch_dsl_drf.tests.TestSimpleQueryStringSearch_boost()
method), 265 (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch
 test_schema_fields_with_filter_fields_list() *method*), 265
 (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon
method), 242 (django_elasticsearch_dsl_drf.tests.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 268
 (django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested_compound()
method), 245 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 252
 (django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter_compound()
method), 248 (django_elasticsearch_dsl_drf.tests.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 269
 (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrdering_common_selected_fields()
method), 251 (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestSearch_multi_match_selected_fields()
 test_schema_fields_with_filter_fields_list() *method*), 253
 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch_search_boost_selected_fields()
method), 252 (django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSearch_simple_query_string_selected_fields()
 test_schema_fields_with_filter_fields_list() *method*), 253
 (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch_selected_fields()
method), 253 (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch
 test_schema_fields_with_filter_fields_list() *method*), 265
 (django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch()
method), 253 (django_elasticsearch_dsl_drf.tests.TestSimpleQueryStringSearch
 test_schema_fields_with_filter_fields_list() *method*), 266
 (django_elasticsearch_dsl_drf.tests.TestFilteringCommon_search_by_field()
method), 259 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 252
 (django_elasticsearch_dsl_drf.tests.TestFilteringNested_search_by_field()
method), 262 (django_elasticsearch_dsl_drf.tests.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 269
 (django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter_search_by_field_multi_terms()
method), 264 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 252
 (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch_search_by_field_multi_terms()
method), 265 (django_elasticsearch_dsl_drf.tests.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 269
 (django_elasticsearch_dsl_drf.tests.TestOrdering_test_search_by_nested_field()
method), 267 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 252
 (django_elasticsearch_dsl_drf.tests.TestSearch_test_search_by_nested_field()
method), 268 (django_elasticsearch_dsl_drf.tests.TestSearch
 test_schema_fields_with_filter_fields_list() *method*), 269
 (django_elasticsearch_dsl_drf.tests.TestSimpleQueryStringSearch_selected_fields()
method), 266 (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestSearch_multi_match_selected_fields()
 test_search() (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch
method), 253 test_search_selected_fields()
 test_search() (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch_elasticsearch_dsl_drf.tests.test_search_simple_query_string
method), 265 *method*), 253
 test_search_boost() test_search_selected_fields()
 (django_elasticsearch_dsl_drf.tests.test_search.TestSearch (django_elasticsearch_dsl_drf.tests.TestMultiMatchSearch
method), 252 *method*), 265
 test_search_boost() test_search_selected_fields()
 (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch_elasticsearch_dsl_drf.tests.TestSimpleQueryStringSearch
method), 253 *method*), 253

method), 269

test_various_complex_fields() (class in `TestFunctionalSuggesters`), 248

(`django_elasticsearch_dsl_drf.tests.test_filtering_common`), 248

method), 242

test_various_complex_fields() (class in `TestHelpers`), 264

(`django_elasticsearch_dsl_drf.tests.test_filtering_post_filter`), 248

method), 248

test_various_complex_fields() (class in `TestHighlight`), 249

(`django_elasticsearch_dsl_drf.tests.TestFilteringCommon`), 265

method), 259

test_various_complex_fields() (class in `TestHighlight`), 249

(`django_elasticsearch_dsl_drf.tests.TestFilteringPostFilter`), 249

method), 264

test_wrapper_as_json() (class in `TestMoreLikeThis`), 249

(`django_elasticsearch_dsl_drf.tests.test_wrappers`), 249

method), 256

test_wrapper_as_json() (class in `TestMultiMatchSearch`), 265

(`django_elasticsearch_dsl_drf.tests.TestWrappers`), 270

method), 270

TestContextSuggesters (class in `TestMultiMatchSearch`), 252

(`django_elasticsearch_dsl_drf.tests.test_suggesters`), 254

TestFacetedSearch (class in `TestOrdering`), 266

(`django_elasticsearch_dsl_drf.tests`), 256

TestFacetedSearch (class in `TestOrdering`), 249

(`django_elasticsearch_dsl_drf.tests.test_faceted_search`), 239

TestFilteringCommon (class in `TestOrderingGeoSpatial`), 267

(`django_elasticsearch_dsl_drf.tests`), 256

TestFilteringCommon (class in `TestOrderingGeoSpatial`), 251

(`django_elasticsearch_dsl_drf.tests.test_filtering_common`), 240

TestFilteringGeoSpatial (class in `TestPagination`), 268

(`django_elasticsearch_dsl_drf.tests`), 259

TestFilteringGeoSpatial (class in `TestPagination`), 251

(`django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial`), 243

TestFilteringGlobalAggregations (class in `TestPipHelpers`), 268

(`django_elasticsearch_dsl_drf.tests`), 260

TestFilteringGlobalAggregations (class in `TestPipHelpers`), 264

(`django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations`), 244

TestFilteringNested (class in `TestSearch`), 268

(`django_elasticsearch_dsl_drf.tests`), 260

TestFilteringNested (class in `TestSearch`), 252

(`django_elasticsearch_dsl_drf.tests.test_filtering_nested`), 244

TestFilteringPostFilter (class in `TestSerializers`), 269

(`django_elasticsearch_dsl_drf.tests`), 262

TestFilteringPostFilter (class in `TestSerializers`), 254

(`django_elasticsearch_dsl_drf.tests.test_filtering_post_filter`), 246

TestFunctionalSuggesters (class in `TestSimpleQueryStringSearch`), 265

(`django_elasticsearch_dsl_drf.tests`), 264

`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string().to_representation()`
253 `(django_elasticsearch_dsl_drf.fields.common.IntegerField`
TestSuggesters `(class in method), 154`
`django_elasticsearch_dsl_drf.tests), 269` `to_representation()`
TestSuggesters `(class in (django_elasticsearch_dsl_drf.fields.common.IPAddressField`
`django_elasticsearch_dsl_drf.tests.test_suggesters), method), 154`
255 `to_representation()`
TestSuggestersEmptyIndex `(class in (django_elasticsearch_dsl_drf.fields.DateField`
`django_elasticsearch_dsl_drf.tests.test_suggesters), method), 158`
255 `to_representation()`
TestViews `(class in (django_elasticsearch_dsl_drf.fields.FloatField`
`django_elasticsearch_dsl_drf.tests), 269` `method), 159`
TestViews `(class in to_representation()`
`django_elasticsearch_dsl_drf.tests.test_views), (django_elasticsearch_dsl_drf.fields.IntegerField`
255 `method), 159`
TestWrappers `(class in to_representation()`
`django_elasticsearch_dsl_drf.tests), 270` `(django_elasticsearch_dsl_drf.fields.IPAddressField`
TestWrappers `(class in method), 160`
`django_elasticsearch_dsl_drf.tests.test_wrappers)to_representation()`
256 `(django_elasticsearch_dsl_drf.fields.ListField`
`to_dict() (django_elasticsearch_dsl_drf.utils.DictionaryProxy method), 160`
`method), 275` `to_representation()`
`to_dict() (django_elasticsearch_dsl_drf.utils.EmptySearch (django_elasticsearch_dsl_drf.fields.nested_fields.ListField`
`method), 275` `method), 158`
`to_internal_value() to_representation()`
`(django_elasticsearch_dsl_drf.fields.ListField (django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField`
`method), 160` `method), 157`
`to_internal_value() to_representation()`
`(django_elasticsearch_dsl_drf.fields.nested_fields.ListField (django_elasticsearch_dsl_drf.fields.ObjectField`
`method), 158` `method), 161`
`to_internal_value() to_representation() (in module`
`(django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField), 155`
`method), 157` `django_elasticsearch_dsl_drf.fields.helpers),`
`to_internal_value() to_representation() (in module`
`(django_elasticsearch_dsl_drf.fields.ObjectField method), 161`
`method), 161` **U**
`update() (django_elasticsearch_dsl_drf.serializers.DocumentSerializer`
`to_representation() method), 274`
`(django_elasticsearch_dsl_drf.fields.BooleanField`
`method), 158` **W**
`to_representation() Wrapper (class in django_elasticsearch_dsl_drf.wrappers),`
`(django_elasticsearch_dsl_drf.fields.CharField 276`
`method), 158`
`to_representation()`
`(django_elasticsearch_dsl_drf.fields.common.BooleanField`
`method), 153`
`to_representation()`
`(django_elasticsearch_dsl_drf.fields.common.CharField`
`method), 154`
`to_representation()`
`(django_elasticsearch_dsl_drf.fields.common.DateField`
`method), 154`
`to_representation()`
`(django_elasticsearch_dsl_drf.fields.common.FloatField`
`method), 154`