
django-elasticsearch-dsl-drf Documentation

Release 0.22.5

Artur Barseghyan <artur.barseghyan@gmail.com>

Dec 27, 2022

CONTENTS

1	Documentation	3
2	Prerequisites	5
3	Main features and highlights	7
4	Demos	9
4.1	Online demo	9
4.2	Local 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	27
12.2.1	Docker	27
12.2.1.1	6.x	27
12.2.1.2	7.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	31
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	33
12.3.2.2	Admin classes	34
12.3.2.3	Create database tables	35

12.3.2.4	Fill in some data	35
12.3.2.5	Sample document	35
12.3.2.5.1	Required imports	35
12.3.2.5.2	Index definition	36
12.3.2.5.2.1	Settings	36
12.3.2.5.2.2	Document index	36
12.3.2.5.3	Custom analyzers	37
12.3.2.5.4	Document definition	37
12.3.2.6	Syncing Django's database with Elasticsearch indexes	38
12.3.2.6.1	Full database sync	39
12.3.2.6.2	Sample partial sync (using custom signals)	39
12.3.2.6.2.1	Required imports	39
12.3.2.6.2.2	Update book index on related model change	39
12.3.2.6.2.3	Update book index on related model removal	40
12.3.2.7	Sample serializer	41
12.3.2.7.1	Required imports	41
12.3.2.7.2	Serializer definition	41
12.3.2.8	ViewSet definition	43
12.3.2.8.1	Required imports	43
12.3.2.8.2	ViewSet definition	43
12.3.2.9	URLs	46
12.3.2.9.1	Required imports	46
12.3.2.9.2	Router definition	46
12.3.2.9.3	URL patterns	46
12.3.2.10	Check what you've done so far	46
12.3.2.10.1	URLs	46
12.3.2.10.2	Test in browser	47
12.3.3	Development and debugging	47
12.3.3.1	Profiling tools	47
12.3.3.1.1	Installation	47
12.3.3.1.2	Configuration	47
12.3.3.2	Debugging	48
12.4	Filter usage examples	48
12.4.1	Search	49
12.4.1.1	Search in all fields	50
12.4.1.2	Search a single term on specific field	50
12.4.1.3	Search for multiple terms	50
12.4.1.4	Search for multiple terms in specific fields	50
12.4.2	Filtering	50
12.4.2.1	Supported lookups	50
12.4.2.1.1	Native	50
12.4.2.1.1.1	term	51
12.4.2.1.1.2	terms	51
12.4.2.1.1.3	range	51
12.4.2.1.1.4	exists	51
12.4.2.1.1.5	prefix	51
12.4.2.1.1.6	wildcard	51
12.4.2.1.1.7	ids	52
12.4.2.1.2	Functional	52
12.4.2.1.2.1	contains	52
12.4.2.1.2.2	in	52
12.4.2.1.2.3	gt	52
12.4.2.1.2.4	gte	53
12.4.2.1.2.5	lt	53

	12.4.2.1.2.6	lte	53
	12.4.2.1.2.7	startswith	53
	12.4.2.1.2.8	endswith	53
	12.4.2.1.2.9	isnull	53
	12.4.2.1.2.10	exclude	54
	12.4.3	Usage examples	54
12.5	Search backends		54
	12.5.1	Compound search filter backend	54
	12.5.1.1	Multi-match	54
	12.5.1.1.1	Sample view	54
	12.5.1.1.2	Sample request	55
	12.5.1.1.3	Generated query	55
	12.5.1.2	Fuzzy search	56
	12.5.1.2.1	Sample view	56
	12.5.1.2.2	Sample request	56
	12.5.1.2.3	Generated query	56
	12.5.2	Multi match search filter backend	57
	12.5.2.1	Sample view	57
	12.5.2.2	Sample request	58
	12.5.2.3	Generated query	58
	12.5.2.4	Options	59
	12.5.2.4.1	Type options	59
	12.5.2.4.2	Operator options	59
	12.5.3	Simple query string filter backend	59
	12.5.3.1	Sample view	60
	12.5.3.2	Sample request 1	60
	12.5.3.3	Generated query 1	61
	12.5.3.4	Sample request 2	61
	12.5.3.5	Generated query 2	61
	12.5.3.6	Sample request 3	62
	12.5.3.7	Generated query 3	62
	12.5.3.8	Options	63
	12.5.3.8.1	Default Operator options	63
12.6	Basic usage examples		63
	12.6.1	Example app	64
	12.6.1.1	Sample models	64
	12.6.1.2	Sample document	65
	12.6.1.3	Sample serializer	66
	12.6.1.4	Sample view	67
	12.6.1.5	Usage example	68
	12.6.1.5.1	Sample queries	68
	12.6.1.5.1.1	Search	68
	12.6.1.5.1.2	Filtering	69
	12.6.1.5.1.3	Ordering	70
12.7	Advanced usage examples		70
	12.7.1	Example app	72
	12.7.1.1	Sample models	72
	12.7.1.2	Sample document	75
	12.7.1.2.1	Index definition	75
	12.7.1.2.1.1	Settings	75
	12.7.1.2.1.2	Document index	75
	12.7.1.3	Sample serializer	77
	12.7.1.4	Sample view	79
	12.7.1.5	Usage example	80

12.7.1.5.1	Search	80
12.7.1.5.2	Filtering	81
12.7.1.5.3	Ordering	82
12.7.1.6	Ids filter	83
12.7.1.7	Faceted search	83
12.7.1.8	Faceted and filtered search	84
12.7.1.9	Post-filter	85
12.7.1.9.1	Sample view	85
12.7.1.9.2	Sample queries	87
12.7.1.10	Geo-spatial features	88
12.7.1.10.1	Filtering	88
12.7.1.10.2	Ordering	88
12.7.1.10.3	Geo-shape	88
12.7.1.11	Suggestions	90
12.7.1.11.1	Completion suggesters	90
12.7.1.11.1.1	Document definition	90
12.7.1.11.1.2	Serializer definition	92
12.7.1.11.1.3	ViewSet definition	92
12.7.1.11.1.4	Sample requests/responses	94
12.7.1.11.1.5	Suggestions on Array/List field	96
12.7.1.11.1.6	Sample requests/responses	96
12.7.1.11.1.7	Context suggesters	97
12.7.1.11.1.8	<i>category</i> context	97
12.7.1.11.1.9	<i>geo</i> context	98
12.7.1.11.2	Term and Phrase suggestions	100
12.7.1.11.2.1	Document definition	100
12.7.1.11.2.2	ViewSet definition	102
12.7.1.11.2.3	Sample requests/responses	106
12.7.1.11.2.4	Completion	106
12.7.1.11.2.5	Term	108
12.7.1.11.2.6	Phrase	109
12.7.1.12	Functional suggestions	109
12.7.1.12.1	Document definition	109
12.7.1.12.2	ViewSet definition	111
12.7.1.13	Highlighting	113
12.7.1.14	Pagination	115
12.7.1.14.1	Page number pagination	115
12.7.1.14.2	Limit/offset pagination	115
12.7.1.14.3	Customisations	116
12.8	Nested fields usage examples	116
12.8.1	Example app	117
12.8.1.1	Sample models	117
12.8.1.2	Sample document	122
12.8.1.2.1	Index definition	122
12.8.1.2.1.1	Settings	122
12.8.1.2.1.2	Document index	123
12.8.1.3	Sample serializer	126
12.8.1.4	Sample view	126
12.8.1.5	Usage example	129
12.8.1.5.1	Sample queries	129
12.8.1.5.1.1	Search	129
12.8.1.5.1.2	Nested filtering	129
12.8.1.5.1.3	Nested search	129
12.8.1.5.1.4	Sample models	130

	12.8.1.5.1.5	Sample document	130
	12.8.1.5.1.6	Sample view	132
	12.8.1.5.1.7	Sample request	134
	12.8.1.5.1.8	Filtering	134
	12.8.1.5.1.9	Ordering	135
	12.8.1.6	Suggestions	135
	12.8.1.7	Nested aggregations/facets	135
12.9	More like this		139
12.9.1	Usage example		139
12.9.1.1	Sample document		139
12.9.1.2	Sample view		141
12.9.1.3	Sample request		142
12.9.1.4	Generated query		142
12.9.1.5	Options		142
12.10	Global aggregations		142
12.10.1	Sample view		143
12.10.2	Sample request		143
12.10.3	Generated query		143
12.10.4	Sample response		144
12.11	Configuration tweaks		146
12.11.1	Ignore certain Elasticsearch exceptions		146
12.12	Source filtering backend		146
12.13	Pagination		147
12.13.1	Page number pagination		147
12.13.2	Query friendly page number pagination		147
12.13.3	Limit/offset pagination		148
12.13.3.1	Customisations		148
12.14	Indexing troubleshooting		149
12.14.1	Timeout		149
12.14.2	Chunk size		149
12.14.3	Use parallel indexing		150
12.14.4	Limit the number of items indexed at once		150
12.15	FAQ		151
12.15.1	Questions and answers		151
12.16	Demo		152
12.16.1	Run demo locally		152
12.16.2	Prerequisites		153
12.17	frontend demo for django-elasticsearch-dsl-drf		153
12.17.1	Quick start		153
12.17.1.1	Install the django requirements		153
12.17.1.2	Install Elasticsearch requirements		153
12.17.1.3	Run Elasticsearch		154
12.17.1.4	Build Elasticsearch index		154
12.17.1.5	Install React requirements		154
12.17.1.6	Run Django		154
12.17.1.7	Run React demo app		154
12.18	Release history and notes		155
12.18.1	0.22.5		155
12.18.2	0.22.4		155
12.18.3	0.22.3		155
12.18.4	0.22.2		155
12.18.5	0.22.1		155
12.18.6	0.22		156
12.18.7	0.21		156

12.18.8 0.20.9	156
12.18.9 0.20.8	157
12.18.100.20.7	157
12.18.110.20.6	157
12.18.120.20.5	157
12.18.130.20.4	157
12.18.140.20.3	157
12.18.150.20.2	157
12.18.160.20.1	158
12.18.170.20	158
12.18.180.19	158
12.18.190.18	158
12.18.200.17.7	158
12.18.210.17.6	159
12.18.220.17.5	159
12.18.230.17.4	159
12.18.240.17.3	159
12.18.250.17.2	159
12.18.260.17.1	159
12.18.270.17	160
12.18.280.16.3	160
12.18.290.16.2	160
12.18.300.16.1	160
12.18.310.16	160
12.18.320.15.1	161
12.18.330.15	161
12.18.340.14	161
12.18.350.13.2	161
12.18.360.13.1	162
12.18.370.13	162
12.18.380.12	162
12.18.390.11	162
12.18.400.10	163
12.18.410.9	164
12.18.420.8.4	164
12.18.430.8.3	164
12.18.440.8.2	164
12.18.450.8.1	164
12.18.460.8	164
12.18.470.7.2	165
12.18.480.7.1	165
12.18.490.7	165
12.18.500.6.4	166
12.18.510.6.3	166
12.18.520.6.2	166
12.18.530.6.1	166
12.18.540.6	166
12.18.550.5.1	166
12.18.560.5	166
12.18.570.4.4	167
12.18.580.4.3	167
12.18.590.4.2	167
12.18.600.4.1	167
12.18.610.4	167

12.18.620.3.12	168
12.18.630.3.11	168
12.18.640.3.10	168
12.18.650.3.9	168
12.18.660.3.8	168
12.18.670.3.7	168
12.18.680.3.6	169
12.18.690.3.5	169
12.18.700.3.4	169
12.18.710.3.3	169
12.18.720.3.2	169
12.18.730.3.1	169
12.18.740.3	169
12.18.750.2.6	170
12.18.760.2.5	170
12.18.770.2.4	170
12.18.780.2.3	170
12.18.790.2.2	170
12.18.800.2.1	170
12.18.810.2	170
12.18.820.1.8	171
12.18.830.1.7	171
12.18.840.1.6	171
12.18.850.1.5	171
12.18.860.1.4	171
12.18.870.1.3	172
12.18.880.1.2	172
12.18.890.1.1	172
12.18.900.1	172
12.19 django_elasticsearch_dsl_drf package	172
12.19.1 Subpackages	172
12.19.1.1 django_elasticsearch_dsl_drf.fields package	172
12.19.1.1.1 Submodules	172
12.19.1.1.2 django_elasticsearch_dsl_drf.fields.common module	172
12.19.1.1.3 django_elasticsearch_dsl_drf.fields.helpers module	173
12.19.1.1.4 django_elasticsearch_dsl_drf.fields.nested_fields module	174
12.19.1.1.5 Module contents	174
12.19.1.2 django_elasticsearch_dsl_drf.filter_backends package	177
12.19.1.2.1 Subpackages	177
12.19.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package	177
12.19.1.2.1.2 Submodules	177
12.19.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module	177
12.19.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module	177
12.19.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module	177
12.19.1.2.1.6 Module contents	177
12.19.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package	177
12.19.1.2.1.8 Submodules	177
12.19.1.2.1.9 django_elasticsearch_dsl_drf.filter_backends.filtering.common module	177
12.19.1.2.1.10 django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial module	185

12.19.1.2.1.11	django_elasticsearch_dsl_drf.filter_backends.filtering.ids module	190
12.19.1.2.1.12	django_elasticsearch_dsl_drf.filter_backends.filtering.nested module	192
12.19.1.2.1.13	django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter module	194
12.19.1.2.1.14	Module contents	195
12.19.1.2.1.15	django_elasticsearch_dsl_drf.filter_backends.ordering package	211
12.19.1.2.1.16	Submodules	211
12.19.1.2.1.17	django_elasticsearch_dsl_drf.filter_backends.ordering.common module	211
12.19.1.2.1.18	django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial module	214
12.19.1.2.1.19	Module contents	216
12.19.1.2.1.20	django_elasticsearch_dsl_drf.filter_backends.search package	220
12.19.1.2.1.21	Subpackages	220
12.19.1.2.1.22	django_elasticsearch_dsl_drf.filter_backends.search.query_backends package	220
12.19.1.2.1.23	Submodules	220
12.19.1.2.1.24	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base module	220
12.19.1.2.1.25	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match module	221
12.19.1.2.1.26	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase module	221
12.19.1.2.1.27	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix module	222
12.19.1.2.1.28	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match module	222
12.19.1.2.1.29	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested module	223
12.19.1.2.1.30	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string module	224
12.19.1.2.1.31	Module contents	225
12.19.1.2.1.32	Submodules	229
12.19.1.2.1.33	django_elasticsearch_dsl_drf.filter_backends.search.base module	229
12.19.1.2.1.34	django_elasticsearch_dsl_drf.filter_backends.search.compound module	230
12.19.1.2.1.35	django_elasticsearch_dsl_drf.filter_backends.search.historical module	230
12.19.1.2.1.36	django_elasticsearch_dsl_drf.filter_backends.search.multi_match module	232
12.19.1.2.1.37	django_elasticsearch_dsl_drf.filter_backends.search.query_string module	233
12.19.1.2.1.38	django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string module	233
12.19.1.2.1.39	Module contents	233
12.19.1.2.1.40	django_elasticsearch_dsl_drf.filter_backends.suggester package	237
12.19.1.2.1.41	Submodules	237
12.19.1.2.1.42	django_elasticsearch_dsl_drf.filter_backends.suggester.functional module	237
12.19.1.2.1.43	django_elasticsearch_dsl_drf.filter_backends.suggester.native module	241
12.19.1.2.1.44	Module contents	246
12.19.1.2.2	Submodules	253
12.19.1.2.3	django_elasticsearch_dsl_drf.filter_backends.faceted_search module	253

12.19.1.2.4	django_elasticsearch_dsl_drf.filter_backends.highlight module	256
12.19.1.2.5	django_elasticsearch_dsl_drf.filter_backends.mixins module	258
12.19.1.2.6	django_elasticsearch_dsl_drf.filter_backends.source module	259
12.19.1.2.7	Module contents	260
12.19.1.3	django_elasticsearch_dsl_drf.management package	260
12.19.1.3.1	Subpackages	260
12.19.1.3.1.1	django_elasticsearch_dsl_drf.management.commands package	260
12.19.1.3.1.2	Submodules	260
12.19.1.3.1.3	django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes module	260
12.19.1.3.1.4	Module contents	262
12.19.1.3.2	Module contents	262
12.19.1.4	django_elasticsearch_dsl_drf.tests package	262
12.19.1.4.1	Submodules	262
12.19.1.4.2	django_elasticsearch_dsl_drf.tests.base module	262
12.19.1.4.3	django_elasticsearch_dsl_drf.tests.data_mixins module	262
12.19.1.4.4	django_elasticsearch_dsl_drf.tests.test_faceted_search module	262
12.19.1.4.5	django_elasticsearch_dsl_drf.tests.test_filtering_common module	262
12.19.1.4.6	django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module	262
12.19.1.4.7	django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module	262
12.19.1.4.8	django_elasticsearch_dsl_drf.tests.test_filtering_nested module	262
12.19.1.4.9	django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module	262
12.19.1.4.10	django_elasticsearch_dsl_drf.tests.test_functional_suggesters module	262
12.19.1.4.11	django_elasticsearch_dsl_drf.tests.test_helpers module	262
12.19.1.4.12	django_elasticsearch_dsl_drf.tests.test_highlight module	262
12.19.1.4.13	django_elasticsearch_dsl_drf.tests.test_more_like_this module	262
12.19.1.4.14	django_elasticsearch_dsl_drf.tests.test_ordering_common module	262
12.19.1.4.15	django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial module	262
12.19.1.4.16	django_elasticsearch_dsl_drf.tests.test_pagination module	262
12.19.1.4.17	django_elasticsearch_dsl_drf.tests.test_pip_helpers module	262
12.19.1.4.18	django_elasticsearch_dsl_drf.tests.test_query_friendly_pagination module	262
12.19.1.4.19	django_elasticsearch_dsl_drf.tests.test_search module	262
12.19.1.4.20	django_elasticsearch_dsl_drf.tests.test_search_multi_match module	262
12.19.1.4.21	django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module	262
12.19.1.4.22	django_elasticsearch_dsl_drf.tests.test_serializers module	262
12.19.1.4.23	django_elasticsearch_dsl_drf.tests.test_source module	262
12.19.1.4.24	django_elasticsearch_dsl_drf.tests.test_suggesters module	262
12.19.1.4.25	django_elasticsearch_dsl_drf.tests.test_versions module	262
12.19.1.4.26	django_elasticsearch_dsl_drf.tests.test_views module	263
12.19.1.4.27	django_elasticsearch_dsl_drf.tests.test_wrappers module	263
12.19.1.4.28	Module contents	263
12.19.2	Submodules	263
12.19.3	django_elasticsearch_dsl_drf.analyzers module	263
12.19.4	django_elasticsearch_dsl_drf.apps module	263
12.19.5	django_elasticsearch_dsl_drf.compat module	263
12.19.6	django_elasticsearch_dsl_drf.constants module	264
12.19.7	django_elasticsearch_dsl_drf.elasticsearch_helpers module	264
12.19.8	django_elasticsearch_dsl_drf.helpers module	264
12.19.9	django_elasticsearch_dsl_drf.pagination module	265
12.19.10	django_elasticsearch_dsl_drf.pip_helpers module	267
12.19.11	django_elasticsearch_dsl_drf.serializers module	268
12.19.12	django_elasticsearch_dsl_drf.utils module	269
12.19.13	django_elasticsearch_dsl_drf.versions module	269

12.19.14	django_elasticsearch_dsl_drf.viewsets module	269
12.19.15	django_elasticsearch_dsl_drf.wrappers module	270
12.19.16	Module contents	271
13	Indices and tables	273
	Python Module Index	275
	Index	277

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.

DOCUMENTATION

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

Make sure to read [FAQ](#).

PREREQUISITES

- Django 2.2, 3.1 and 3.2.
- Python 3.6, 3.7, 3.8 and 3.9.
- 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.*

Do you need a similar tool for GraphQL? Check [graphene-elastic](#).

4.1 Online demo

Check the [live demo app](#) (Django 3.1 + Django REST Framework 3.12 + Elasticsearch 7.5) hosted on Heroku and [bonsai.io](#).

4.2 Local 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
```

- (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',  
    # ...  
)
```


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.

TESTING

Project is covered with tests.

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

```
tox -e py38-django30
```

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  
*****
```


LICENSE

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

CHAPTER
TEN

SUPPORT

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

CHAPTER
ELEVEN

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

PROJECT DOCUMENTATION

Contents:

Table of Contents

- *django-elasticsearch-dsl-drf*
 - *Documentation*
 - *Prerequisites*
 - *Main features and highlights*
 - *Demos*
 - * *Online demo*
 - * *Local 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
0.18.x	2.x, 5.x, 6.x
0.19.x	6.x
0.20.x	6.x, 7.x

django-elasticsearch-dsl

You are advised to use the latest version of [django-elasticsearch-dsl](#).

The following versions have been tested and work well together:

elasticsearch	elasticsearch-dsl	django-elasticsearch-dsl
2.4.1	2.2.0	0.5.1
5.4.0	5.3.0	0.5.1
6.3.0	6.1.0	0.5.1
6.3.0	6.4.0	6.4.2
7.0.2	7.0.0	7.0.0

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 [djangorestframework](#) 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
3.0	3.11.0

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

Using docker-compose (persistent):

```
docker-compose up elasticsearch
```

12.2.1.1 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
```

12.2.1.2 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
```

12.3 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
```

(continues on next page)

(continued from previous page)

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

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 gettext, gettext_lazy as _
```

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

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

        ordering = ["id"]

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

(continues on next page)

(continued from previous page)

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

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

12.3.2.1.4 Author model

books/models/author.py

```
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:
        """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:
        """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

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

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

    class Meta:
        """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.
```

(continues on next page)

(continued from previous page)

```
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,
```

(continues on next page)

(continued from previous page)

```
    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,
        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'),
        }
    )
```

(continues on next page)

(continued from previous page)

```
)

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 Django(object):
    """Inner nested class Django."""

    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':
```

(continues on next page)

(continued from previous page)

```

        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.
        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:

```

(continues on next page)

(continued from previous page)

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

        # Specify the correspondent document class
        document = BookDocument

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

(continues on next page)

(continued from previous page)

```
        'tags',  
    )
```

However, if dynamic serializer doesn't work for you 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:  
        """Meta options."""  
  
        # List the serializer fields. Note, that the order of the fields  
        # is preserved in the ViewSet.  
        fields = (  
            'id',  
            'title',  
            'description',  
            'summary',  
            'publisher',  
            'publication_date',  
            'state',  
            'isbn',  
            'price',  
            'pages',  
            'stock_count',  
            'tags',  
        )  
  
    def get_tags(self, obj):  
        """Get tags."""  
        if obj.tags:  
            return list(obj.tags)  
        else:  
            return []
```

12.3.2.8 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,
        IdsFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
```

(continues on next page)

(continued from previous page)

```

]
# 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': {

```

(continues on next page)

(continued from previous page)

```

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

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

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

```

12.3.2.9 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, re_path
from search_indexes import urls as search_index_urls

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

(continues on next page)

(continued from previous page)

```
# ...
]
```

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 = {
```

(continues on next page)

(continued from previous page)

```

    'INTERCEPT_REDIRECTS': False,
}

DEBUG_TOOLBAR_PANELS = (
    # Defaults
    'debug_toolbar.panels.versions.VersionsPanel',
    'debug_toolbar.panels.timer.TimerPanel',
    'debug_toolbar.panels.settings.SettingsPanel',
    'debug_toolbar.panels.headers.HeadersPanel',
    'debug_toolbar.panels.request.RequestPanel',
    'debug_toolbar.panels.sql.SQLPanel',
    'debug_toolbar.panels.staticfiles.StaticFilesPanel',
    'debug_toolbar.panels.templates.TemplatesPanel',
    'debug_toolbar.panels.cache.CachePanel',
    'debug_toolbar.panels.signals.SignalsPanel',
    'debug_toolbar.panels.logging.LoggingPanel',
    'debug_toolbar.panels.redirects.RedirectsPanel',
    # Additional
    'elastic_panel.panel.ElasticDebugPanel',
)

```

12.3.3.2 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 Multi-match

12.5.1.1.1 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'
```

(continues on next page)

(continued from previous page)

```

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

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

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

```

12.5.1.1.2 Sample request

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

12.5.1.1.3 Generated query

```

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

```

(continues on next page)

(continued from previous page)

```
{
    "match": {
        "summary": {
            "query": "enim"
        }
    }
}
```

12.5.1.2 Fuzzy search

12.5.1.2.1 Sample view

```
class BookCompoundFuzzySearchBackendDocumentViewSet(DocumentViewSet):
    # ...
    filter_backends = [
        # ...
        CompoundSearchFilterBackend,
        # ...
    ]

    search_fields = {
        'title': {'fuzziness': 'AUTO'},
        'description': None,
        'summary': None,
    }
```

12.5.1.2.2 Sample request

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

12.5.1.2.3 Generated query

```
{
  "from": 0,
  "size": 1,
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "title": {
              "fuzziness": "AUTO",
              "query": "Performance"
            }
          }
        }
      ]
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

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

```

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 = [
        # ...
        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",
```

(continues on next page)

(continued from previous page)

```

    "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",
        "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:
        """Meta options."""

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

12.6.1.2 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(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
    country = fields.StringField(
        fields={
            'raw': fields.StringField(analyzer='keyword'),
        }
    )
```

(continues on next page)

(continued from previous page)

```
website = fields.StringField()

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

class Meta:
    """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:
        """Meta options."""

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

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

12.6.1.4 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*
- * *Faceted and filtered search*
- * *Post-filter*
 - *Sample view*
 - *Sample queries*
- * *Geo-spatial features*
 - *Filtering*
 - *Ordering*
 - *Geo-shape*
- * *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 gettext, gettext_lazy as _

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

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,
```

(continues on next page)

(continued from previous page)

```

max_digits=19,
default=0)

class Meta:
    """Meta options."""

    ordering = ["id"]

def __str__(self):
    return self.name

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

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

```

books/models/author.py

```

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:
        """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:
        """Meta options."""

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

```

(continues on next page)

(continued from previous page)

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

books/models/book.py

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

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

    class Meta:
        """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
)
```

(continues on next page)

(continued from previous page)

```

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'),
        }
    )

    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'),
        }
    )

```

(continues on next page)

(continued from previous page)

```

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:
    """Meta options."""

    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:
        """Meta options."""

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

```

search_indexes/serializers/book.py

```

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

```

(continues on next page)

(continued from previous page)

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

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

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

12.7.1.4 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',
    )
    # Define filtering fields
    filter_fields = {
        'id': {
            'field': '_id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
            ],
        },
        'publisher': 'publisher.raw',
    }
```

(continues on next page)

(continued from previous page)

```

'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,
        },
        'publication_date': {
```

(continues on next page)

(continued from previous page)

```

        '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 Faceted and filtered search

It is sometimes useful to be able to facet and filter on the same field. For example, this would allow the user to apply a filter such as `size=medium` and a `size` facet, and get back a `size` facet that has all sizes, not just `medium`. This is similar to `elasticsearch-dsl's FacetedSearch class` https://elasticsearch-dsl.readthedocs.io/en/latest/faceted_search.html.

To do this, use `FacetedFilterSearchFilterBackend` instead of both `FacetedSearchFilterBackend` and `FilteringFilterBackend`. It will apply both the filters and the facets. It uses exactly the same configuration as the two backends it replaces.

`search_indexes/viewsets/book.py`

```

# ...

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

# ...

from elasticsearch_dsl import (
    DateHistogramFacet,

```

(continues on next page)

(continued from previous page)

```

    RangeFacet,
    TermsFacet,
)

# ...

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

    # ...

    filter_backends = [
        # ...
        FacetedFilterSearchFilterBackend,
    ]

    # ...

    filter_fields = {
        'state': 'state.raw', # The filter and facet fields MUST both use the same ↵
                             ↵elasticsearch field
    }

    faceted_search_fields = {
        'state': 'state.raw', # The filter and facet fields MUST both use the same ↵
                             ↵elasticsearch field
    }

    # ...

```

In the example below, we show results with faceted *state* and the filter *state=published*.

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

12.7.1.9 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.

12.7.1.9.1 Sample view

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

search_indexes/viewsets/book.py

```

# ...

from django_elasticsearch_dsl_drf.filter_backends import (
    # ...

```

(continues on next page)

(continued from previous page)

```

        PostFilterFilteringFilterBackend,
    )

    # ...

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

    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,
            ],
        },
    }

```

(continues on next page)

(continued from previous page)

```

        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',
        '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',)

```

12.7.1.9.2 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
```

12.7.1.10 Geo-spatial features

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

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

12.7.1.10.1 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
```

12.7.1.10.2 Ordering

Geo-distance ordering

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

12.7.1.10.3 Geo-shape

Setup

In order to be able to do all geo-shape queries, you need a GeoShapeField with 'recursive' strategy. Details about spatial strategies here : <https://www.elastic.co/guide/en/elasticsearch/reference/master/geo-shape.html#spatial-strategy>

```
# ...

@INDEX.doc_type
class PublisherDocument(Document):

# ...

    location_circle = fields.GeoShapeField(strategy='recursive',
                                           attr='location_circle_indexing')

# ...
```

(continues on next page)

(continued from previous page)

```

class Publisher(models.Model):
    # ...

    @property
    def location_circle_indexing(self):
        """
        Indexing circle geo_shape with 10km radius.
        Used in Elasticsearch indexing/tests of `geo_shape` native filter.
        """
        return {
            'type': 'circle',
            'coordinates': [self.latitude, self.longitude],
            'radius': '10km',
        }

```

You need to use `GeoSpatialFilteringFilterBackend` and set the `LOOKUP_FILTER_GEO_SHAPE` to the `geo_spatial_filter_field`. (This takes place in `ViewSet`)

```

# ...
class PublisherDocumentViewSet(DocumentViewSet):
    # ...
    filter_backends = [
        # ...
        GeoSpatialFilteringFilterBackend,
        # ...
    ]
    # ...
    geo_spatial_filter_fields = {
        # ...
        'location_circle': {
            'lookups': [
                LOOKUP_FILTER_GEO_SHAPE,
            ]
        },
        # ...
    }
    # ...

```

Supported shapes & queries

With this setup, we can do several types of Geo-shape queries.

Supported and tested shapes types are : point, circle, envelope

Potentially supported but untested shapes are : multipoint and linestring

Supported and tested queries are : INTERSECTS, DISJOINT, WITHIN, CONTAINS

Shapes intersects

Interesting queries are shape intersects : this gives you all documents whose shape intersects with the shape given in query. (Should be 2 with the actual test dataset)

```
http://localhost:8000/search/publishers/?location_circle__geo_shape=49.119696,6.176355__
↪radius,15km__relation,intersects__type,circle
```

This request give you all publishers having a location_circle intersecting with the one in the query.

12.7.1.11 Suggestions

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

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

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

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

12.7.1.11.1 Completion suggesters

12.7.1.11.1.1 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,
    number_of_replicas=1
)

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

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

(continues on next page)

(continued from previous page)

```

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:
    """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.

12.7.1.11.1.2 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:
        """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',
        )
```

12.7.1.11.1.3 ViewSet definition

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

Note: You should inherit from *DocumentViewSet* instead of *BaseDocumentViewSet*.

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."""
```

(continues on next page)

(continued from previous page)

```

document = PublisherDocument

# ...

filter_backends = [
    # ...
    SuggesterFilterBackend,
]

# ...

# Suggester fields
suggester_fields = {
    'name_suggest': {
        'field': 'name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
        'options': {
            'size': 20, # Override default number of suggestions
            'skip_duplicates': True, # Whether duplicate suggestions should be
↪ filtered out.
        },
    },
    '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,
        ],
    },
}

```

(continues on next page)

(continued from previous page)

```
}
```

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

12.7.1.11.1.4 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,
```

(continues on next page)

(continued from previous page)

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

12.7.1.11.1.5 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*)

12.7.1.11.1.6 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
  },
  "tag_suggest__completion": [
    {
      "options": [
        {
          "score": 1.0,
          "text": "Biography"
        },
        {
          "score": 1.0,
          "text": "Biology"
        }
      ],
      "offset": 0,
      "length": 2,
      "text": "bio"
    }
  ]
}
```

12.7.1.11.1.7 Context suggesters

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

12.7.1.11.1.8 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",
                        # 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 be 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.
                'skip_duplicates': True, # Whether duplicate suggestions should be_
    filtered out.
            },
        },
    }

    # ...
```

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
```

12.7.1.11.1.9 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(
```

(continues on next page)

(continued from previous page)

```

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',
            '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.
                'skip_duplicates': True, # Whether duplicate suggestions should be
    ↪ filtered out.

```

(continues on next page)

(continued from previous page)

```

        },
    },
}

# ...

```

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
```

12.7.1.11.2 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.

12.7.1.11.2.1 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,
    number_of_replicas=1
)

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

    title = StringField(

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

    # Publication date
    publication_date = fields.DateField()

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

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

    # Price
    price = fields.FloatField()

```

(continues on next page)

(continued from previous page)

```

# 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:
    """Meta options."""

    model = Book # The model associate with this Document

```

12.7.1.11.2.2 ViewSet definition

Note: The suggerer 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 suggerer:

```
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 suggerer 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 suggerer:

```
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': {
```

(continues on next page)

(continued from previous page)

```

        'field': 'id',
        'lookups': [
            LOOKUP_FILTER_RANGE,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_GT,
            LOOKUP_QUERY_GTE,
            LOOKUP_QUERY_LT,
            LOOKUP_QUERY_LTE,
            LOOKUP_FILTER_TERMS,
        ],
    },
    'title': 'title.raw',
    'publisher': 'publisher.raw',
    'publication_date': 'publication_date',
    'state': 'state.raw',
    'isbn': 'isbn.raw',
    'price': {
        'field': 'price.raw',
        'lookups': [
            LOOKUP_FILTER_RANGE,
        ],
    },
    'pages': {
        'field': 'pages',
        'lookups': [
            LOOKUP_FILTER_RANGE,
            LOOKUP_QUERY_GT,
            LOOKUP_QUERY_GTE,
            LOOKUP_QUERY_LT,
            LOOKUP_QUERY_LTE,
        ],
    },
    'stock_count': {
        # 'field': 'stock_count',
        'lookups': [
            LOOKUP_FILTER_RANGE,
            LOOKUP_QUERY_GT,
            LOOKUP_QUERY_GTE,
            LOOKUP_QUERY_LT,
            LOOKUP_QUERY_LTE,
        ],
    },
    'tags': {
        'field': 'tags',
        'lookups': [
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
            LOOKUP_QUERY_ISNULL,
        ],
    },

```

(continues on next page)

(continued from previous page)

```

    },
    'tags.raw': {
        'field': 'tags.raw',
        'lookups': [
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
        ],
    },
    # This has been added to test `exists` filter.
    'non_existent_field': 'non_existent_field',
    # This has been added to test `isnull` filter.
    'null_field': 'null_field',
}
# Define ordering fields
ordering_fields = {
    'id': 'id',
    'title': 'title.raw',
    'price': 'price.raw',
    'state': 'state.raw',
    'publication_date': 'publication_date',
}
# Specify default ordering
ordering = ('id', 'title', 'price',)

# Suggester fields
suggester_fields = {
    'title_suggest': {
        'field': 'title.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
            SUGGESTER_TERM,
            SUGGESTER_PHRASE,
        ]
        'default_suggester': SUGGESTER_COMPLETION,
        'options': {
            'size': 10, # Number of suggestions to retrieve.
            'skip_duplicates': True, # Whether duplicate suggestions should be
→ filtered out.
        },
    },
    '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.

12.7.1.11.2.3 Sample requests/responses

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

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

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

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

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

12.7.1.11.2.4 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",

```

(continues on next page)

(continued from previous page)

```

        "authors": [
            "Jayden van Luysse",
            "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,
            "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."
        }
    }
}

```

(continues on next page)

(continued from previous page)

```
    ],
    "offset": 0
  }
]
```

12.7.1.11.2.5 Term

Request

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

Response

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


12.7.1.11.2.6 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
  }
}
```

12.7.1.12 Functional suggestions

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

Configuration is very similar to native suggesters.

12.7.1.12.1 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
```

(continues on next page)

(continued from previous page)

```

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')

    # *****
    # ***** 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:

```

(continues on next page)

(continued from previous page)

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

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

12.7.1.12.2 ViewSet definition

Note: The suggerter 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 suggerter 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 suggerter 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` suggerter:

```
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` suggerter 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` suggerter:

```
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` suggerter 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,
)
```

(continues on next page)

(continued from previous page)

```
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.

12.7.1.13 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)

(continued from previous 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>"
        ]
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
        "null_field": null
    }
]
}
```

12.7.1.14 Pagination

12.7.1.14.1 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
```

12.7.1.14.2 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
```

12.7.1.14.3 Customisations

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

```
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination

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

    def get_paginated_response_context(self, data):
        __data = super(
            CustomPageNumberPagination,
            self
        ).get_paginated_response_context(data)
        __data.append(
            ('current_page', int(self.request.query_params.get('page', 1)))
        )
        __data.append(
            ('page_size', self.get_page_size(self.request))
        )

        return sorted(__data)
```

Same applies to the customisations of the `LimitOffsetPagination`.

12.8 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

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
    )
```

(continues on next page)

(continued from previous page)

```

class Meta:
    """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/country.py

```

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:
        """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/city.py

```

class City(models.Model):
    """City."""

    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:
        """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/address.py

```

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

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,
        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:
        """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.

```

(continues on next page)

(continued from previous page)

```

"""
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,
        '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:
    """

```

(continues on next page)

(continued from previous page)

```
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(),
            'suggest': fields.CompletionField(),
        }
    )

    house_number = StringField(analyzer=html_strip)

    appendix = StringField(analyzer=html_strip)

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

        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:
    """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:
        """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
```

(continues on next page)

(continued from previous page)

```

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,
        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,
            ],
        },
    },

```

(continues on next page)

(continued from previous page)

```

    },
}
# Define ordering fields
ordering_fields = {
    'id': None,
    'street': None,
    'city': 'city.name.raw',
    'country': 'city.country.name.raw',
    'zip_code': None,
}
# Define ordering fields
geo_spatial_ordering_fields = {
    'location': None,
}
# Specify default ordering
ordering = (
    'id',
    'street.raw',
    'city.name.raw',
)
# Suggester fields
suggester_fields = {
    'street_suggest': {
        'field': 'street.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'city_suggest': {
        'field': 'city.name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'country_suggest': {
        'field': 'city.country.name.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
}
}

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

```

(continues on next page)

(continued from previous page)

```
}
```

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 "Picadilly".

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

Search a single term on specific field

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

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

12.8.1.5.1.2 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

class City(models.Model):
    """City."""

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

books/models/country.py

```
from django.db import models

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
```

(continues on next page)

(continued from previous page)

```

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.

    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),

```

(continues on next page)

(continued from previous page)

```

        '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')
)

class Meta:
    """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

```

(continues on next page)

(continued from previous page)

```

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 = {
        '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,
    }

```

(continues on next page)

(continued from previous page)

```

        '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 are suggest action/route are restricted.

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

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

Suggest completion for field country.

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

Suggest completion for field city.

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

12.8.1.7 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.
```

(continues on next page)

(continued from previous page)

```

: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',
                    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,
    ]
    pagination_class = LimitOffsetPagination
    # Define search fields
    search_fields = (
        'street',
        'zip_code',
        'city.name',
        'city.country.name',
    )
    # Define filtering fields
    filter_fields = {
        'id': None,

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

        '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(
    number_of_shards=1,
    number_of_replicas=1,

```

(continues on next page)

(continued from previous page)

```

        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:
        """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,
    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": [
        "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'

    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": [

```

(continues on next page)

(continued from previous page)

```

        "rejected"
    ]
}
},
"size": 25,
"aggs": {
    "_filter_state_global": {
        "aggs": {
            "state_global": {
                "terms": {
                    "field": "state.raw"
                }
            }
        },
        "global": {
        }
    }
},
"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"
                    }
                ]
            }
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        },
        {
            "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",
            "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  
  
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 Pagination

Check the view definitions from the [advanced usage examples](#).

12.13.1 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
```

12.13.2 Query friendly page number pagination

Works exactly as `PageNumberPagination` but fires (mostly) a single query to Elasticsearch, instead of 2.

search_indexes/viewsets/book.py

```
# ...

from django_elasticsearch_dsl_drf.pagination import QueryFriendlyPageNumberPagination

# ...

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

    # ...

    pagination_class = QueryFriendlyPageNumberPagination

    # ...
```

12.13.3 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
```

12.13.3.1 Customisations

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

```
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination

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

    def get_paginated_response_context(self, data):
        __data = super(
            CustomPageNumberPagination,
            self
        ).get_paginated_response_context(data)
        __data.append(
            ('current_page', int(self.request.query_params.get('page', 1)))
        )
        __data.append(
            ('page_size', self.get_page_size(self.request))
        )

        return sorted(__data)
```

Same applies to the customisations of the LimitOffsetPagination.

12.14 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.

12.14.1 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
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).

12.14.2 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
```

12.14.3 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:  
        """Meta options."""  
  
        model = Location  
        parallel_indexing = True
```

12.14.4 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:  
        """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:
        """Meta options."""

        model = Location
        queryset_pagination = settings.ELASTICSEARCH_DSL_QUERYSET_PAGINATION
```

12.15 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](#).

12.15.1 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](#).

Question

I keep getting `[FORBIDDEN/12/index read-only / allow delete (api)]` error when saving models despite having `blocks={'read_only_allow_delete': None}`, in settings.

Answer

Once of the possible reasons for the mentioned symptom might be [low disk space](#).

12.16 Demo

12.16.1 Run demo locally

In order to be able to quickly evaluate the `django-elasticsearch-dsl-drf`, a demo app (with a quick installer) has been created (works on Ubuntu/Debian, may work on other Linux systems as well, although not guaranteed). Follow the instructions below for having the demo running within a minute.

12.16.2 Prerequisites

- Python 3
- Docker

Grab and run Elasticsearch:

```
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
```

Grab and run the latest `django_elasticsearch_dsl_drf_demo_installer.sh`:

```
wget -O - https://raw.githubusercontent.com/barseghyanartur/django-elasticsearch-dsl-drf/master/
↪ examples/django_elasticsearch_dsl_drf_demo_installer.sh | bash
```

Open your browser and test the app.

- URL: <http://127.0.0.1:8001/search/>

12.17 frontend demo for django-elasticsearch-dsl-drf

Frontend demo for django-elasticsearch-dsl-drf

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

12.17.1 Quick start

From the project root directory.

12.17.1.1 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
```

12.17.1.2 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
```

12.17.1.3 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
```

12.17.1.4 Build Elasticsearch index

First, create some test data:

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

Then build Elasticsearch index:

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

12.17.1.5 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
```

12.17.1.6 Run Django

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

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

12.17.1.7 Run React demo app

Finally, run the React demo app:

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

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

12.18 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

<code>major.minor[.revision]</code>

- 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.

12.18.1 0.22.5

2022-07-04

- Return 0 for count when using EmptySearch.

12.18.2 0.22.4

2022-01-24

- Skip some parts of the ViewSet class definition on empty *document*.

12.18.3 0.22.3

2022-01-24

- Make running of ViewSet checks flexible (to allow dynamic document definition).
- Added the *FacetedFilterSearchFilterBackend*.

12.18.4 0.22.2

2021-08-29

- Tested against Django 3.2.

12.18.5 0.22.1

2021-05-01

- *DictionaryProxy* gets an optional *meta* argument, which will hold meta information of the hit.

12.18.6 0.22

2021-03-26

Note: Release dedicated to my dear son, Raffi, who turns 16 at the end of the month. Happy birthday, dear Raffi.

- Make it easier to override the *DictionaryProxy* by moving it to the scope of the *BaseDocumentViewSet*.
- Add *tzinfo* check for better date vs datetime determination.
- Broader *EmptySearch* compatibility with *elasticsearch_dsl.Search*.
- Remove old Django code from docs.
- Allow more specific targeting for nested sort fields.
- Add more tests.

12.18.7 0.21

2021-02-04

Note: Release dedicated to defenders of Armenia and Artsakh (Nagorno Karabakh) and all the victims of Turkish and Azerbaijani aggression.

- Drop support for Python 2.7 and 3.5.
- Drop support for Django 1.11, 2.0 and 2.1.
- Tested against Django 3.1 and Python 3.9.
- Take options (ex: boost, fuzziness) into consideration in *NestedQueryBackend*.
- Added an example of a proper date formatting.
- Added an example of alternative document ID.
- Added experimental *QueryFriendlyPageNumberPagination* which fires just a single query instead of used two used by *PageNumberPagination*.

12.18.8 0.20.9

2020-10-16

Note: Help to rebuild the Armenian homeland. Please, [make a donation](#). All donations received are solely for humanitarian purposes.

- Implemented *geo_shape* filter.

12.18.9 0.20.8

2020-04-10

- Fixes in `skip_duplicates` option support for native suggester.

12.18.10 0.20.7

2020-04-10

- Make `skip_duplicates` available for native suggester.

12.18.11 0.20.6

2020-04-04

- Basic implementation of RegExp field lookup.

12.18.12 0.20.5

2019-12-30

- Minor fixes.

12.18.13 0.20.4

2019-12-25

- Tested against Django 3.0.
- Tested against Python 3.8.
- Tested against Django REST Framework 3.11.
- Minor fixes.
- Test optimisations.

12.18.14 0.20.3

2019-09-20

- Testing the auxiliary versions module.

12.18.15 0.20.2

2019-08-30

- Minor improvements in test coverage.

12.18.16 0.20.1

2019-08-18

- Minor Elasticsearch 7.x compatibility fixes.

12.18.17 0.20

2019-08-17

- Adding Elasticsearch 7.x support.

12.18.18 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.

12.18.19 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.

12.18.20 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.

12.18.21 0.17.6

2019-04-08

- Minor fixes.
- Additions to the docs.

12.18.22 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.

12.18.23 0.17.4

2019-03-13

- Source backend.

12.18.24 0.17.3

2019-02-08

- Obey object permissions.

12.18.25 0.17.2

2019-01-07

- Add nested ordering.

12.18.26 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.

12.18.27 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.

12.18.28 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.

12.18.29 0.16.2

2018-09-21

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

12.18.30 0.16.1

2018-09-18

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

12.18.31 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.

12.18.32 0.15.1

2018-08-22

- More tests.
- Fixes in docs.

12.18.33 0.15

2018-08-10

- Global aggregations.

12.18.34 0.14

2018-08-06

- More like this support through detail action.

12.18.35 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).

12.18.36 0.13.1

2018-07-26

- Minor fix in suggesters on Elasticsearch 6.x.

12.18.37 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.

12.18.38 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.

12.18.39 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.

12.18.40 0.10

2018-07-06

- Elasticsearch 6.x support.
- Minor fixes.

12.18.41 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`.

12.18.42 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.

12.18.43 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`.

12.18.44 0.8.2

2018-06-05

- Minor fixes.

12.18.45 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.

12.18.46 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.

12.18.47 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.

12.18.48 0.7.1

2018-04-04

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

- Add query *boost* support for search fields.

12.18.49 0.7

2018-03-08

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

- `CoreAPI`/`CoreSchema` support.

12.18.50 0.6.4

2018-03-05

- Minor fix: explicitly use DocType in the ViewSets.

12.18.51 0.6.3

2018-01-03

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

12.18.52 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.

12.18.53 0.6.1

2017-11-28

- Documentation fixes.

12.18.54 0.6

2017-11-28

- Added highlight backend.
- Added nested search functionality.

12.18.55 0.5.1

2017-10-18

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

12.18.56 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.

12.18.57 0.4.4

2017-10-02

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

12.18.58 0.4.3

2017-09-28

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

12.18.59 0.4.2

2017-09-28

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

12.18.60 0.4.1

2017-09-26

- Fixes in docs.

12.18.61 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).

12.18.62 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.

12.18.63 0.3.11

2017-09-21

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

12.18.64 0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

12.18.65 0.3.9

2017-09-12

- Python 2.x compatibility fix.

12.18.66 0.3.8

2017-09-12

- Fixes tests on some environments.

12.18.67 0.3.7

2017-09-07

- Docs fixes.

12.18.68 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.

12.18.69 0.3.5

2017-08-24

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

12.18.70 0.3.4

2017-08-23

- Minor fixes in the ordering backend.

12.18.71 0.3.3

2017-07-13

- Minor fixes and improvements.

12.18.72 0.3.2

2017-07-12

- Minor fixes and improvements.

12.18.73 0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

12.18.74 0.3

2017-07-11

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

12.18.75 0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

12.18.76 0.2.5

2017-07-11

- Fixes in documentation.

12.18.77 0.2.4

2017-07-11

- Fixes in documentation.

12.18.78 0.2.3

2017-07-11

- Fixes in documentation.

12.18.79 0.2.2

2017-07-11

- Fixes in documentation.

12.18.80 0.2.1

2017-07-11

- Fixes in documentation.

12.18.81 0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

12.18.82 0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

12.18.83 0.1.7

2017-06-25

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

12.18.84 0.1.6

2017-06-23

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

12.18.85 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.

12.18.86 0.1.4

2017-06-22

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

12.18.87 0.1.3

2017-06-21

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

12.18.88 0.1.2

2017-06-20

- Minor fixes in tests.

12.18.89 0.1.1

2017-06-20

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

12.18.90 0.1

2017-06-19

- Initial beta release.

12.19 django_elasticsearch_dsl_drf package

12.19.1 Subpackages

12.19.1.1 django_elasticsearch_dsl_drf.fields package

12.19.1.1.1 Submodules

12.19.1.1.2 django_elasticsearch_dsl_drf.fields.common module

Common fields.

```
class django_elasticsearch_dsl_drf.fields.common.BooleanField(*args, **kwargs)
```

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(*args, **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**(*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**(*args, **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.**IPAddressField**(*args, **kwargs)

Bases: rest_framework.fields.IPAddressField

Object field.

get_value(*dictionary*)

Get value.

to_representation(*value*)

To representation.

class django_elasticsearch_dsl_drf.fields.common.**IntegerField**(*args, **kwargs)

Bases: rest_framework.fields.IntegerField

Object field.

get_value(*dictionary*)

Get value.

to_representation(*value*)

To representation.

12.19.1.1.3 django_elasticsearch_dsl_drf.fields.helpers module

Helpers.

django_elasticsearch_dsl_drf.fields.helpers.**to_representation**(*value*)

To representation.

12.19.1.1.4 django_elasticsearch_dsl_drf.fields.nested_fields module

Nested fields.

class django_elasticsearch_dsl_drf.fields.nested_fields.**GeoPointField**(*args, **kwargs)
 Bases: [django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField](#)

Geo point field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**GeoShapeField**(*args, **kwargs)
 Bases: [django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField](#)

Geo shape field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**ListField**(*args, **kwargs)
 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.nested_fields.**NestedField**(*args, **kwargs)
 Bases: [django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField](#)

Nested field.

class django_elasticsearch_dsl_drf.fields.nested_fields.**ObjectField**(*args, **kwargs)
 Bases: [rest_framework.fields.Field](#)

Object field.

get_value(dictionary)
 Get value.

to_internal_value(data)
 To internal value.

to_representation(value)
 To representation.

12.19.1.1.5 Module contents

Fields.

class django_elasticsearch_dsl_drf.fields.**BooleanField**(*args, **kwargs)
 Bases: [rest_framework.fields.BooleanField](#)

Object field.

get_value(dictionary)
 Get value.

to_representation(value)
 To representation.

```
class django_elasticsearch_dsl_drf.fields.CharField(*args, **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(*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(*args, **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(*args, **kwargs)
    Bases: django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField

    Geo point field.

class django_elasticsearch_dsl_drf.fields.GeoShapeField(*args, **kwargs)
    Bases: django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField

    Geo shape field.

class django_elasticsearch_dsl_drf.fields.IPAddressField(*args, **kwargs)
    Bases: rest_framework.fields.IPAddressField

    Object field.

    get_value(dictionary)
        Get value.

    to_representation(value)
        To representation.

class django_elasticsearch_dsl_drf.fields.IntegerField(*args, **kwargs)
    Bases: rest_framework.fields.IntegerField

    Object field.

    get_value(dictionary)
        Get value.

    to_representation(value)
        To representation.
```

```
class django_elasticsearch_dsl_drf.fields.ListField(*args, **kwargs)
    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(*args, **kwargs)
    Bases: django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField

    Nested field.

class django_elasticsearch_dsl_drf.fields.ObjectField(*args, **kwargs)
    Bases: rest_framework.fields.Field

    Object field.

    get_value(dictionary)
        Get value.

    to_internal_value(data)
        To internal value.

    to_representation(value)
        To representation.
```

12.19.1.2 django_elasticsearch_dsl_drf.filter_backends package

12.19.1.2.1 Subpackages

12.19.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package

12.19.1.2.1.2 Submodules

12.19.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module

12.19.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module

12.19.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module

12.19.1.2.1.6 Module contents

12.19.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package

12.19.1.2.1.8 Submodules

12.19.1.2.1.9 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
```

(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 = [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_regexp**(*queryset, options, value*)

Apply *regexp* filter.

Syntax:

`/endpoint/?field_name__regexp={regexp}`

Example:

`http://localhost:8000/api/users/?age__regexp=1{[6-9]}` `http://localhost:8000/api/users/?age__regexp=2.*`

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.

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__lt=1__2.0 http://localhost:8000/api/articles/?id__lt=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 } /end-
point/?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`

12.19.1.2.1.10 `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.`

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 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*

classmethod `apply_query_geo_shape(queryset, options, value)`

Apply *geo_shape* 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 `get_geo_shape_params(value, field)`

Get params for *geo_shape* query.

Example:

```
/search/publishers/?location__geo_shape=48.9864453,6.37977 __relation,intersects
__type,circle __radius,20km
```

Example:

```
/search/publishers/?location__geo_shape=48.906254,6.378593 __48.985850,6.479359
__relation,within __type,envelope
```

Elasticsearch:

```
{
    "query": {
        "bool": [{
            "must": [{ "match_all": {} }],
            "filter": {
                "geo_shape": [{
                    "location": [{
                        "shape": { "type": "circle", "coordinates": [48.9864453,
                            6.37977], "radius": "20km"
                        }, "relation": "intersects"
                    }
                ]
            }
        ]
    }
}
```

```

        }
    }
}
}
}

```

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_shape* query.

Return type dict

classmethod `prepare_filter_fields(view)`

Prepare filter fields.

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

Returns Filtering options.

Return type dict

12.19.1.2.1.11 `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'

12.19.1.2.1.12 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

12.19.1.2.1.13 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.

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

12.19.1.2.1.14 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
```

(continues on next page)

(continued from previous page)

```
>>> 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_regexp(queryset, options, value)`

Apply *regexp* filter.

Syntax:

`/endpoint/?field_name__regexp={regexp}`

Example:

`http://localhost:8000/api/users/?age__regexp=1 [{]6-9]` `http://localhost:8000/api/users/?age__regexp=2.*`

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.

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__lt=1__2.0 http://localhost:8000/api/articles/?id__lt=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 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*

classmethod `apply_query_geo_shape(queryset, options, value)`

Apply *geo_shape* 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 `get_geo_shape_params(value, field)`

Get params for *geo_shape* query.

Example:

`/search/publishers/?location__geo_shape=48.9864453,6.37977__relation,intersects__type,circle__radius,20km`

Example:

`/search/publishers/?location__geo_shape=48.906254,6.378593__48.985850,6.479359__relation,within__type,envelope`

Elasticsearch:

```
{
  "query": {
```

```
        "bool" [{}
            "must" [{} "match_all": {}
            ], "filter": {
                "geo_shape" [{}
                    "location" [{}
                        "shape": { "type": "circle", "coordinates": [48.9864453,
                            6.37977], "radius": "20km"
                        }, "relation": "intersects"
                    }
                ]
            }
        ]
    }
}
```

Parameters

- **value** (*str*) –
- **field** –

Returns Params to be used in *geo_shape* 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):
```

(continues on next page)

(continued from previous page)

```
>>>
>>> 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
>>> )
```

(continues on next page)

(continued from previous page)

```
>>> 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

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):
>>>
```

(continues on next page)

(continued from previous page)

```
>>> 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

12.19.1.2.1.15 django_elasticsearch_dsl_drf.filter_backends.ordering package

12.19.1.2.1.16 Submodules

12.19.1.2.1.17 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
>>>
>>> # 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': {

```

(continues on next page)

(continued from previous page)

```
>>>         '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
```

(continues on next page)

(continued from previous page)

```

>>>
>>> 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'

12.19.1.2.1.18 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'

12.19.1.2.1.19 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',
>>>         }
>>>         '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
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialOrderingFilterBackend,]
```

(continues on next page)

(continued from previous page)

```
>>> 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',
>>>         '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 = ('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'

12.19.1.2.1.20 `django_elasticsearch_dsl_drf.filter_backends.search` package

12.19.1.2.1.21 Subpackages

12.19.1.2.1.22 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends` package

12.19.1.2.1.23 Submodules

12.19.1.2.1.24 `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

12.19.1.2.1.25 `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'`

12.19.1.2.1.26 `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'`

12.19.1.2.1.27 `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.MatchPhrasePrefixQueryBackend`

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'`

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

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

`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'`

12.19.1.2.1.29 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested` module

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

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'

12.19.1.2.1.30 `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.SimpleQueryStringQueryBackend`

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'`

12.19.1.2.1.31 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.`

MatchPhrasePrefixQueryBackend

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.`

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.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.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'reily
```

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.`

SimpleQueryStringQueryBackend

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'
```

12.19.1.2.1.32 Submodules

12.19.1.2.1.33 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'
```

12.19.1.2.1.34 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_backends.match.MatchQueryBackend'>, <class
'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.
NestedQueryBackend'>]
```

12.19.1.2.1.35 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'

12.19.1.2.1.36 `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.`

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_backends.multi_match.MultiMatchQueryBackend'>`]

search_param = 'search_multi_match'

12.19.1.2.1.37 `django_elasticsearch_dsl_drf.filter_backends.search.query_string` module

12.19.1.2.1.38 `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.`

SimpleQueryStringSearchFilterBackend

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_backends.simple_query_string.SimpleQueryStringQueryBackend'>`]

search_param = 'search_simple_query_string'

12.19.1.2.1.39 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_backends.match.MatchQueryBackend'>, <class
'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.
NestedQueryBackend'>]
```

```
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_backends.multi_match.MultiMatchQueryBackend'>]
```

```
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'],
```

(continues on next page)

(continued from previous page)

```
>>> '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'

class

django_elasticsearch_dsl_drf.filter_backends.search.SimpleQueryStringSearchFilterBackend

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_backends.simple_query_string.SimpleQueryStringQueryBackend'>*](#)]

search_param = 'search_simple_query_string'

12.19.1.2.1.40 django_elasticsearch_dsl_drf.filter_backends.suggester package

12.19.1.2.1.41 Submodules

12.19.1.2.1.42 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(),
>>>         }
>>>     )
>>>
>>>

```

(continues on next page)

(continued from previous page)

```

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

```

class `django_elasticsearch_dsl_drf.filter_backends.suggester.functional.`

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
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
>>>

```

(continues on next page)

(continued from previous page)

```

>>> 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

12.19.1.2.1.43 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={
```

(continues on next page)

(continued from previous page)

```
>>>         '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:
>>>     "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
```

(continues on next page)

(continued from previous page)

```

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

```

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

Apply completion suggester.

Parameters

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

- **value** (*str*) – value to filter on.

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

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

Apply *phrase* suggester.

Parameters

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

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

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

Apply *term* suggester.

Parameters

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

Returns Modified queryset.

Return type `elasticsearch_dsl.search.Search`

filter_queryset(request, queryset, view)

Filter the queryset.

Parameters

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

Returns Updated queryset.

Return type `elasticsearch_dsl.search.Search`

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': {
```

(continues on next page)

(continued from previous page)

```

>>>         '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', 'Documentary__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

12.19.1.2.1.44 Module contents

Suggester filtering backends.

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
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
```

(continues on next page)

(continued from previous page)

```

>>>
>>> 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

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(BaseDocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         SuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     suggester_fields = {
>>>         'name_suggest': {
```

(continues on next page)

(continued from previous page)

```

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

```

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

Apply *completion* suggester.

Parameters

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

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

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

Apply *phrase* suggester.

Parameters

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

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

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

Apply *term* suggester.

Parameters

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

Returns Modified queryset.

Return type *elasticsearch_dsl.search.Search*

filter_queryset (*request, queryset, view*)

Filter the queryset.

Parameters

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

Returns Updated queryset.

Return type *elasticsearch_dsl.search.Search*

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'],
```

(continues on next page)

(continued from previous page)

```
>>>     }
>>> }
```

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', 'Documentary__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

12.19.1.2.2 Submodules

12.19.1.2.3 `django_elasticsearch_dsl_drf.filter_backends.faceted_search` module

Faceted search backend.

class `django_elasticsearch_dsl_drf.filter_backends.faceted_search.`

FacetedFilterSearchFilterBackend

Bases: `django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend`

Combined faceting and filtering backend similar to elasticsearch-dsl's `FacetedSearch` class. It combines the functionality of `FilteringFilterBackend` and `FacetedSearchFilterBackend` to take filters into account when creating facets.

This backend uses the same configuration fields as `FilteringFilterBackend` and `FacetedSearchFilterBackend`. This backend replaces their functionality and should not be used together with either of those backends.

Note that to work correctly, the actual elasticsearch field must be the same for a facet and its matching filter. For example, if a facet will aggregate on field `state.raw`, then the filter must also filter on `state.raw`, and not just `state`.

When creating a facet, filters for faceted fields other than for the current facet are applied. Filters for faceted fields are then applied as `post_filters`. Filters on non-faceted fields are applied as normal filters.

aggregate(*request, queryset, view*)

Aggregate.

Parameters

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

Returns

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

Apply filter.

Parameters

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

Returns

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`

classdjango_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
>>>     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

12.19.1.2.4 `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 = {
```

(continues on next page)

(continued from previous page)

```

>>>         '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': {

```

(continues on next page)

(continued from previous page)

```
>>>         'options': {
>>>             'pre_tags' : ["<em>"],
>>>             'post_tags' : ["</em>"]
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

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

Returns Highlight fields options.

Return type dict

12.19.1.2.5 `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

12.19.1.2.6 django_elasticsearch_dsl_drf.filter_backends.source module

Source backend.

class `django_elasticsearch_dsl_drf.filter_backends.source.SourceBackend`

Bases: `rest_framework.filters.BaseFilterBackend`

Static source backend.

Example 1 (simple):

```
>>> 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
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [SourceBackend,]
>>>     source = ["title"]
```

Example 2 (complex):

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

Example 3 (even more complex):

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

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

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*

12.19.1.2.7 Module contents

All filter backends.

12.19.1.3 django_elasticsearch_dsl_drf.management package

12.19.1.3.1 Subpackages

12.19.1.3.1.1 django_elasticsearch_dsl_drf.management.commands package

12.19.1.3.1.2 Submodules

12.19.1.3.1.3 django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes module

```
class django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command(stdout=None,
                                                                                          stderr=None,
                                                                                          no_color=False,
                                                                                          force_color=False)
```

Bases: *django.core.management.base.BaseCommand*

add_arguments(*parser*)

Entry point for subclassed commands to add custom arguments.

handle(*args, **options)

The actual logic of the command. Subclasses must implement this method.

help = 'Remove all indexes from Elasticsearch'

12.19.1.3.1.4 Module contents

12.19.1.3.2 Module contents

12.19.1.4 django_elasticsearch_dsl_drf.tests package

12.19.1.4.1 Submodules

12.19.1.4.2 django_elasticsearch_dsl_drf.tests.base module

12.19.1.4.3 django_elasticsearch_dsl_drf.tests.data_mixins module

12.19.1.4.4 django_elasticsearch_dsl_drf.tests.test_faceted_search module

12.19.1.4.5 django_elasticsearch_dsl_drf.tests.test_filtering_common module

12.19.1.4.6 django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module

12.19.1.4.7 django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module

12.19.1.4.8 django_elasticsearch_dsl_drf.tests.test_filtering_nested module

12.19.1.4.9 django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module

12.19.1.4.10 django_elasticsearch_dsl_drf.tests.test_functional_suggesters module

12.19.1.4.11 django_elasticsearch_dsl_drf.tests.test_helpers module

12.19.1.4.12 django_elasticsearch_dsl_drf.tests.test_highlight module

12.19.1.4.13 django_elasticsearch_dsl_drf.tests.test_more_like_this module

12.19.1.4.14 django_elasticsearch_dsl_drf.tests.test_ordering_common module

12.19.1.4.15 django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial module

12.19.1.4.16 django_elasticsearch_dsl_drf.tests.test_pagination module

12.19.1.4.17 django_elasticsearch_dsl_drf.tests.test_pip_helpers module

12.19.1.4.18 django_elasticsearch_dsl_drf.tests.test_query_friendly_pagination module

12.19.1.4.19 django_elasticsearch_dsl_drf.tests.test_search module

12.19.1.4.20 django_elasticsearch_dsl_drf.tests.test_search_multi_match module

12.19.1.4.21 django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module

12.19.1.4.22 django_elasticsearch_dsl_drf.tests.test_serializers module

12.19.1.4.23 django_elasticsearch_dsl_drf.tests.test_source module

Tests of `django_elasticsearch_dsl_drf.versions` module.

setUp()

Hook method for setting up the test fixture before exercising it.

test_elasticsearch_dsl_6_3_0()

Tests as if we were using `elasticsearch_dsl==6.3.0`.

test_elasticsearch_dsl_7_0_0()

Tests as if we were using `elasticsearch_dsl==7.0.0`.

12.19.1.4.26 `django_elasticsearch_dsl_drf.tests.test_views` module

12.19.1.4.27 `django_elasticsearch_dsl_drf.tests.test_wrappers` module

12.19.1.4.28 Module contents

12.19.2 Submodules

12.19.3 `django_elasticsearch_dsl_drf.analyzers` module

Analyzers.

12.19.4 `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'`

12.19.5 `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.

class `django_elasticsearch_dsl_drf.compat.KeywordField(attr=None, **kwargs)`

Bases: `django_elasticsearch_dsl.fields.DEDField, elasticsearch_dsl.field.Keyword`

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

String field.

Parameters `kwargs` –

Returns

12.19.6 django_elasticsearch_dsl_drf.constants module

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

12.19.7 django_elasticsearch_dsl_drf.elasticsearch_helpers module

`django_elasticsearch_dsl_drf.elasticsearch_helpers.delete_all_indices(with_protected=False)`

Delete all indices. Args:
with_protected (bool):

Returns: tuple: Tuple of two lists with removed and errored indices.

`django_elasticsearch_dsl_drf.elasticsearch_helpers.get_all_indices(with_protected=False)`

Get all indices. Args:
with_protected (bool):

Returns: list: List of indices.

12.19.8 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*

12.19.9 django_elasticsearch_dsl_drf.pagination module

Pagination.

class `django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination(*args, **kwargs)`
 Bases: `rest_framework.pagination.LimitOffsetPagination`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

A limit/offset pagination.

Example:

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

get_facets(*facets=None*)

Get facets.

Parameters *facets* –

Returns

get_paginated_response(*data*)

Get paginated response.

Parameters *data* –

Returns

get_paginated_response_context(*data*)

Get paginated response data.

Parameters *data* –

Returns

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

class `django_elasticsearch_dsl_drf.pagination.Page(object_list, number, paginator, facets)`
 Bases: `django.core.paginator.Page`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

Page for Elasticsearch.

class `django_elasticsearch_dsl_drf.pagination.PageNumberPagination(*args, **kwargs)`
 Bases: `rest_framework.pagination.PageNumberPagination`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

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 `django_elasticsearch_dsl_drf.pagination.Paginator`

get_facets(*page=None*)

Get facets.

Parameters *page* –

Returns

get_paginated_response(*data*)

Get paginated response.

Parameters *data* –

Returns

get_paginated_response_context(*data*)

Get paginated response data.

Parameters *data* –

Returns

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

Paginate a queryset.

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

Parameters

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

Returns

class `django_elasticsearch_dsl_drf.pagination.Paginator`(*object_list, per_page, orphans=0, allow_empty_first_page=True*)

Bases: `django.core.paginator.Paginator`

Paginator for Elasticsearch.

page(*number*)

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

Parameters *number* –

Returns

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

Bases: `django_elasticsearch_dsl_drf.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 `django_elasticsearch_dsl_drf.pagination.QueryFriendlyPaginator`

`orphans_query_param = 'orphans'`

`page_size_query_param = 'page_size'`

`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.QueryFriendlyPaginator`(*object_list*, *per_page*,
orphans=0, *al-*
low_empty_first_page=True)

Bases: `django_elasticsearch_dsl_drf.pagination.Paginator`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

Paginator for Elasticsearch.

page(*number*)

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

Parameters **number** –

Returns

12.19.10 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`

12.19.11 `django_elasticsearch_dsl_drf.serializers` module

Serializers.

```
class django_elasticsearch_dsl_drf.serializers.DocumentSerializer(*args, **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(name, bases, attrs)
    Bases: rest_framework.serializers.SerializerMetaclass

    Metaclass for the DocumentSerializer.

    Ensures that all declared subclasses implemented a Meta.

class django_elasticsearch_dsl_drf.serializers.Meta(name, bases, attrs)
    Bases: type

    Template for the DocumentSerializerMeta.Meta class.

    exclude = ()
    field_aliases = {}
    field_options = {}
    fields = ()
    ignore_fields = ()
    index_aliases = {}
    index_classes = ()
    search_fields = ()
    serializers = ()
```


12.19.12 django_elasticsearch_dsl_drf.utils module

Utils.

```
class django_elasticsearch_dsl_drf.utils.DictionaryProxy(mapping, meta=None)
    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)
    property hits
    sort(*args, **kwargs)
    to_dict(*args, **kwargs)
```

12.19.13 django_elasticsearch_dsl_drf.versions module

Contains information about the current Elasticsearch version in use, including (LTE and GTE).

```
django_elasticsearch_dsl_drf.versions.get_elasticsearch_version(default=(2, 0, 0))
    Get Elasticsearch version.
        Parameters default (tuple) – Default value. Mainly added for building the docs when Elastic-
            search is not running.
        Returns
        Return type list
```

12.19.14 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.

    dictionary_proxy
        alias of django_elasticsearch_dsl_drf.utils.DictionaryProxy

    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*

run_checks()

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.

12.19.15 django_elasticsearch_dsl_drf.wrappers module

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"

property *as_dict*

As dict.

Returns

Return type dict

property `as_json`

As JSON.

Returns

Return type `str`

`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`

12.19.16 Module contents

Integrate Elasticsearch DSL with Django REST framework.

INDICES AND TABLES

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

PYTHON MODULE INDEX

d

`django_elasticsearch_dsl_drf`, 271

`django_elasticsearch_dsl_drf.analyzers`, 263

`django_elasticsearch_dsl_drf.apps`, 263

`django_elasticsearch_dsl_drf.compat`, 263

`django_elasticsearch_dsl_drf.constants`, 264

`django_elasticsearch_dsl_drf.elasticsearch_helpers`, 264

`django_elasticsearch_dsl_drf.fields`, 174

`django_elasticsearch_dsl_drf.fields.common`, 172

`django_elasticsearch_dsl_drf.fields.helpers`, 173

`django_elasticsearch_dsl_drf.fields.nested_fields`, 174

`django_elasticsearch_dsl_drf.filter_backends`, 260

`django_elasticsearch_dsl_drf.filter_backends.aggregations`, 177

`django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations`, 177

`django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations`, 177

`django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations`, 177

`django_elasticsearch_dsl_drf.filter_backends.faceted_search`, 253

`django_elasticsearch_dsl_drf.filter_backends.filtering`, 195

`django_elasticsearch_dsl_drf.filter_backends.filtering.common`, 177

`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial`, 185

`django_elasticsearch_dsl_drf.filter_backends.filtering.qs`, 190

`django_elasticsearch_dsl_drf.filter_backends.filtering.nested`, 192

`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter`, 194

`django_elasticsearch_dsl_drf.filter_backends.highlight`, 256

`django_elasticsearch_dsl_drf.filter_backends.mixins`, 258

`django_elasticsearch_dsl_drf.filter_backends.ordering`, 216

`django_elasticsearch_dsl_drf.filter_backends.ordering.common`, 211

`django_elasticsearch_dsl_drf.filter_backends.ordering.geo`, 214

`django_elasticsearch_dsl_drf.filter_backends.search`, 233

`django_elasticsearch_dsl_drf.filter_backends.search.base`, 229

`django_elasticsearch_dsl_drf.filter_backends.search.compound`, 230

`django_elasticsearch_dsl_drf.filter_backends.search.history`, 230

`django_elasticsearch_dsl_drf.filter_backends.search.multisearch`, 232

`django_elasticsearch_dsl_drf.filter_backends.search.query`, 225

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations`, 220

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 221

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 221

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.bucket_aggregations`, 222

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 222

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 223

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.bucket_aggregations`, 224

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 233

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 233

`django_elasticsearch_dsl_drf.filter_backends.source`, 259

`django_elasticsearch_dsl_drf.filter_backends.suggester`, 246

`django_elasticsearch_dsl_drf.filter_backends.suggester.fuzzy`, 246

[237](#)
django_elasticsearch_dsl_drf.filter_backends.suggester.native,
[241](#)
django_elasticsearch_dsl_drf.helpers, [264](#)
django_elasticsearch_dsl_drf.management, [262](#)
django_elasticsearch_dsl_drf.management.commands,
[262](#)
django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes,
[260](#)
django_elasticsearch_dsl_drf.pagination, [265](#)
django_elasticsearch_dsl_drf.pip_helpers, [267](#)
django_elasticsearch_dsl_drf.serializers, [268](#)
django_elasticsearch_dsl_drf.tests, [263](#)
django_elasticsearch_dsl_drf.tests.test_versions,
[262](#)
django_elasticsearch_dsl_drf.utils, [269](#)
django_elasticsearch_dsl_drf.versions, [269](#)
django_elasticsearch_dsl_drf.viewsets, [269](#)
django_elasticsearch_dsl_drf.wrappers, [270](#)

A

A

- `add_arguments()` (`django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command` method), 260
- `aggregate()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedFilterSearchFilterBackend` class method), 253
- `aggregate()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend` class method), 254
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedFilterSearchFilterBackend` class method), 253
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 192
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 208
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filters.PostFilterFilteringFilterBackend` class method), 208
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filters.PostFilterFilteringFilterBackend` class method), 195
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filters.PostFilterFilteringFilterBackend` class method), 210
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filters.PostFilterFilteringFilterBackend` class method), 210
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin` class method), 258
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 178
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 197
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 178
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_filter_regexp()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 179
- `apply_filter_regexp()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 179
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 198
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 181
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 198
- `apply_query_geo_bounding_box()`

(`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filter_backend`), 186

`apply_query_geo_bounding_box()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 202

`apply_query_geo_distance()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filter_backend` class method), 186

`apply_query_geo_distance()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 203

`apply_query_geo_polygon()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filter_backend` class method), 186

`apply_query_geo_polygon()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 203

`apply_query_geo_shape()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filter_backend` class method), 187

`apply_query_geo_shape()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 203

`apply_query_gt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 181

`apply_query_gt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 199

`apply_query_gte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 182

`apply_query_gte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 199

`apply_query_in()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 182

`apply_query_in()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 199

`apply_query_isnull()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilterBackend` class method), 182

`apply_query_isnull()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 199

`apply_query_lt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 183

`apply_query_lt()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 200

`apply_query_lte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 183

`apply_query_lte()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 200

`apply_query_size()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend` class method), 239

`apply_query_size()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend` class method), 247

`apply_query_wildcard()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilterBackend` class method), 183

`apply_query_wildcard()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 200

`apply_suggester_completion()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterFilterBackend` class method), 243

`apply_suggester_completion()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 250

`apply_suggester_completion_match()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend` class method), 239

`apply_suggester_completion_match()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend` class method), 247

`apply_suggester_completion_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend` class method), 240

`apply_suggester_completion_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend` class method), 248

`apply_suggester_phrase()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 250

`apply_suggester_term()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend` class method), 250

`as_dict` (`django_elasticsearch_dsl_drf.wrappers.Wrapper` property), 270

`as_json` (`django_elasticsearch_dsl_drf.wrappers.Wrapper` property), 270

B

`BaseDocumentReviewsSet` (`django_elasticsearch_dsl_drf.viewsets` class), 269

`BaseSearchFilterBackend` (`django_elasticsearch_dsl_drf.filter_backends.search` class), 229

`BaseSearchFilterBackend` (`django_elasticsearch_dsl_drf.filter_backends.search.base` class), 229

`BaseSearchQueryBackend` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backend` class), 229

220
 BooleanField (class in `django_elasticsearch_dsl_drf.fields`), 174
 BooleanField (class in `django_elasticsearch_dsl_drf.fields.common`), 172
C
 CharField (class in `django_elasticsearch_dsl_drf.fields`), 174
 CharField (class in `django_elasticsearch_dsl_drf.fields.common`), 172
 check_if_installed() (in module `django_elasticsearch_dsl_drf.pip_helpers`), 267
 clean_queryset() (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend` method), 240
 clean_queryset() (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend` method), 248
 Command (class in `django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes`), 260
 CompoundSearchFilterBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.compound`), 234
 CompoundSearchFilterBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.compound`), 230
 Config (class in `django_elasticsearch_dsl_drf.apps`), 263
 construct_facets() (`django_elasticsearch_dsl_drf.filter_backends.filtered_search.FacetedSearchFilterBackend` method), 255
 construct_nested_search() (`django_elasticsearch_dsl_drf.filter_backends.search.historical.SearchFilterBackend` method), 230
 construct_nested_search() (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend` method), 235
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.search.historical.SearchFilterBackend` method), 231
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.base.BaseSearchQueryBackend` class method), 220
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.BaseSearchQueryBackend` class method), 225
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.match.MatchQueryBackend` class method), 221
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.match_phrase.MatchPhraseQueryBackend` class method), 221
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.match_phrase_prefix.MatchPhrasePrefixQueryBackend` class method), 222
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhrasePrefixQueryBackend` class method), 225
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 225
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 226
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 227
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 224
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 228
 construct_search() (`django_elasticsearch_dsl_drf.filter_backends.query_backends.MatchPhraseQueryBackend` class method), 225
 create() (`django_elasticsearch_dsl_drf.serializers.DocumentSerializer` method), 268
D
 DateField (class in `django_elasticsearch_dsl_drf.fields`), 175
 DateField (class in `django_elasticsearch_dsl_drf.fields.common`), 173
 DefaultOrderingFilterBackend (class in `django_elasticsearch_dsl_drf.filter_backends.ordering`), 216
 DefaultOrderingFilterBackend (class in `django_elasticsearch_dsl_drf.filter_backends.ordering.common`), 216
 delete_all_indices() (in module `django_elasticsearch_dsl_drf.elasticsearch_helpers`), 264
 dict_to_obj() (in module `django_elasticsearch_dsl_drf.wrappers`), 264
 dictionary_proxy (`django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet` attribute), 269
 DictionaryProxy (class in `django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`), 269
 django_elasticsearch_dsl_drf
 module, 269
 django_elasticsearch_dsl_drf.analyzers
 module, 273
 django_elasticsearch_dsl_drf.apps
 module, 273
 django_elasticsearch_dsl_drf.compat
 module, 273
 django_elasticsearch_dsl_drf.constants
 module, 273
 django_elasticsearch_dsl_drf.elasticsearch_helpers
 module, 274
 django_elasticsearch_dsl_drf.fields
 module, 174

django_elasticsearch_dsl_drf.fields.common module, 172	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 221
django_elasticsearch_dsl_drf.fields.helpers module, 173	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 221
django_elasticsearch_dsl_drf.fields.nested_fields module, 174	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 222
django_elasticsearch_dsl_drf.filter_backends module, 260	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 222
django_elasticsearch_dsl_drf.filter_backends.aggregate_base module, 177	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 223
django_elasticsearch_dsl_drf.filter_backends.aggregate_base.bucket_aggregations module, 177	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 224
django_elasticsearch_dsl_drf.filter_backends.aggregate_base.search_aggregations module, 177	django_elasticsearch_dsl_drf.filter_backends.search.query. module, 233
django_elasticsearch_dsl_drf.filter_backends.aggregate_base.pipeline_aggregations module, 177	django_elasticsearch_dsl_drf.filter_backends.search.simple module, 233
django_elasticsearch_dsl_drf.filter_backends.factoriesearch module, 253	django_elasticsearch_dsl_drf.filter_backends.source module, 259
django_elasticsearch_dsl_drf.filter_backends.filtering module, 195	django_elasticsearch_dsl_drf.filter_backends.suggester module, 246
django_elasticsearch_dsl_drf.filter_backends.filtering.elasticsearch module, 177	django_elasticsearch_dsl_drf.filter_backends.suggester.fur module, 237
django_elasticsearch_dsl_drf.filter_backends.filtering.elasticsearch.spatial module, 185	django_elasticsearch_dsl_drf.filter_backends.suggester.nat module, 241
django_elasticsearch_dsl_drf.filter_backends.filtering.elasticsearch.helpers module, 190	django_elasticsearch_dsl_drf.helpers module, 264
django_elasticsearch_dsl_drf.filter_backends.filtering.elasticsearch.management module, 192	django_elasticsearch_dsl_drf.management module, 262
django_elasticsearch_dsl_drf.filter_backends.filtering.elasticsearch.management.commands module, 194	django_elasticsearch_dsl_drf.management.commands module, 262
django_elasticsearch_dsl_drf.filter_backends.highlight module, 256	django_elasticsearch_dsl_drf.management.commands.elasticse module, 260
django_elasticsearch_dsl_drf.filter_backends.mappings module, 258	django_elasticsearch_dsl_drf.pagination module, 265
django_elasticsearch_dsl_drf.filter_backends.ordering module, 216	django_elasticsearch_dsl_drf.pip_helpers module, 267
django_elasticsearch_dsl_drf.filter_backends.ordering.common module, 211	django_elasticsearch_dsl_drf.serializers module, 268
django_elasticsearch_dsl_drf.filter_backends.ordering.elasticsearch module, 214	django_elasticsearch_dsl_drf.tests module, 263
django_elasticsearch_dsl_drf.filter_backends.seperators module, 233	django_elasticsearch_dsl_drf.tests.test_versions module, 262
django_elasticsearch_dsl_drf.filter_backends.seperators.base module, 229	django_elasticsearch_dsl_drf.utils module, 269
django_elasticsearch_dsl_drf.filter_backends.seperators.compound module, 230	django_elasticsearch_dsl_drf.versions module, 269
django_elasticsearch_dsl_drf.filter_backends.seperators.historical module, 230	django_elasticsearch_dsl_drf.viewsets module, 269
django_elasticsearch_dsl_drf.filter_backends.seperators.match module, 232	django_elasticsearch_dsl_drf.wrappers module, 270
django_elasticsearch_dsl_drf.filter_backends.seperators.paginator_base module, 225	django_elasticsearch_dsl_drf.paginator_base (django_elasticsearch_dsl_drf.pagination.PageNumberPagination)
django_elasticsearch_dsl_drf.filter_backends.search.query_backend module, 220	django_elasticsearch_dsl_drf.filter_backends.base module, 220

(`django_elasticsearch_dsl_drf.viewsets.FunctionalSuggesterMixin`), 210
 method), 270
`FunctionalSuggesterFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend`), 246
 (`django_elasticsearch_dsl_drf.filter_backends.suggester`), method), 229
`FunctionalSuggesterFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend`), 238
 (`django_elasticsearch_dsl_drf.filter_backends.suggester.functional`), method), 233
`FunctionalSuggestMixin` (class in (`django_elasticsearch_dsl_drf.filter_backends.search.historical.BaseSearchFilterBackend`), 270
 (`django_elasticsearch_dsl_drf.viewsets`), method), 232
G
`GeoPointField` (class in (`django_elasticsearch_dsl_drf.fields`), 175
 (`django_elasticsearch_dsl_drf.fields`), method), 236
`GeoPointField` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.CommonOrderingFilterBackend`), 174
 (`django_elasticsearch_dsl_drf.fields.nested_fields`), class method), 212
`GeoShapeField` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 175
 (`django_elasticsearch_dsl_drf.fields`), class method), 217
`GeoShapeField` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 174
 (`django_elasticsearch_dsl_drf.fields.nested_fields`), class method), 217
`GeoSpatialFilteringFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 202
 (`django_elasticsearch_dsl_drf.filter_backends.filtering`), method), 255
`GeoSpatialFilteringFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend`), 185
 (`django_elasticsearch_dsl_drf.filter_backends.filtering`), method), 265
`GeoSpatialOrderingFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 217
 (`django_elasticsearch_dsl_drf.filter_backends.ordering`), method), 266
`GeoSpatialOrderingFilterBackend` (class in (`django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend`), 214
 (`django_elasticsearch_dsl_drf.filter_backends.ordering`), class method), 223
`get_all_indices()` (in module (`django_elasticsearch_dsl_drf.elasticsearch_helpers`), 264
 (`django_elasticsearch_dsl_drf.elasticsearch_helpers`), class method), 224
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.CommonFilteringFilterBackend`), 184
 method), 184
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.CommonFilteringFilterBackend`), 201
 method), 201
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`), 193
 method), 193
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`), 208
 method), 208
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.CommonFilteringFilterBackend`), 184
 method), 184
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.CommonFilteringFilterBackend`), 195
 method), 195
`get_coreschema_field()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend`), 201
 method), 201

method), 187

get_filter_query_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
method), 203

get_filter_query_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend
method), 193

get_filter_query_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend
method), 209

get_geo_bounding_box_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
class method), 187

get_geo_bounding_box_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
class method), 204

get_geo_distance_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilteringFilterBackend
class method), 188

get_geo_distance_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
class method), 204

get_geo_distance_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.GeoSpatialOrderingFilterBackend
class method), 214

get_geo_distance_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend
class method), 218

get_geo_polygon_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilteringFilterBackend
class method), 188

get_geo_polygon_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
class method), 205

get_geo_shape_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilteringFilterBackend
class method), 189

get_geo_shape_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend
class method), 205

get_geo_spatial_field_name()
(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.GeoSpatialOrderingFilterBackend
method), 215

get_geo_spatial_field_name()
(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend
method), 218

get_gte_lte_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend
class method), 184

get_gte_lte_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend
class method), 201

get_highlight_query_params()
(django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightFilterBackend
method), 257

get_ids_query_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend
method), 191

get_ids_query_params()
(django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend
method), 207

get_ids_values()
(django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend
method), 207

get_index_and_mapping_for_model() (in module
django_elasticsearch_dsl_drf.helpers), 264

get_installed_packages() (in module
django_elasticsearch_dsl_drf.pip_helpers),
267

get_object() (django_elasticsearch_dsl_drf.viewsets.BaseDocumentView
method), 265

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.common.FilteringFilterBackend
method), 207

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.common.FilteringFilterBackend
method), 207

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend
method), 218

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.GeoSpatialFilteringFilterBackend
method), 205

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend
method), 215

get_ordering_query_params()
(django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend
method), 225

get_paginated_response()
(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination
method), 267

get_paginated_response()
(django_elasticsearch_dsl_drf.pagination.PageNumberPagination
method), 267

get_paginated_response_context()
(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination
method), 267

get_paginated_response_context()
(django_elasticsearch_dsl_drf.pagination.PageNumberPagination
method), 267

get_query_backends()
(django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchBackend
method), 201

get_query_backends()
(django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend
method), 201

get_query_options() (django_elasticsearch_dsl_dr.filter_backends.search.query. class method), 223	get_search_query_params() (django_elasticsearch_dsl_dr.filter_backends.search. method), 229
get_query_options() (django_elasticsearch_dsl_dr.filter_backends.search.query. class method), 227	get_search_query_params() (django_elasticsearch_dsl_dr.filter_backends.search. method), 233
get_query_options() (django_elasticsearch_dsl_dr.filter_backends.search.query. class method), 225	get_search_query_params() (django_elasticsearch_dsl_dr.filter_backends.search. method), 232
get_query_options() (django_elasticsearch_dsl_dr.filter_backends.search.query. class method), 229	get_search_query_params() (django_elasticsearch_dsl_dr.filter_backends.search. method), 236
get_queryset() (django_elasticsearch_dsl_dr.viewsets. method), 269	get_suggester_context() (django_elasticsearch_dsl_dr.filter_backends.suggester.native. method), 248
get_range_params() (django_elasticsearch_dsl_dr.filter_backends. class method), 184	get_suggester_context() (django_elasticsearch_dsl_dr.filter_backends.suggester. class method), 251
get_range_params() (django_elasticsearch_dsl_dr.filter_backends. class method), 201	get_suggester_query_params() (django_elasticsearch_dsl_dr.filter_backends.suggester. method), 240
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.com. method), 185	get_suggester_query_params() (django_elasticsearch_dsl_dr.filter_backends.suggester. method), 248
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.Filter. method), 202	get_suggester_query_params() (django_elasticsearch_dsl_dr.filter_backends.suggester. method), 246
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.nested. method), 193	get_suggester_query_params() (django_elasticsearch_dsl_dr.filter_backends.suggester. method), 252
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.Nested. method), 209	get_value() (django_elasticsearch_dsl_dr.fields.BooleanField method), 195
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.post. method), 195	get_value() (django_elasticsearch_dsl_dr.fields.CharField method), 175
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.filtering.post. method), 210	get_value() (django_elasticsearch_dsl_dr.fields.common. method), 172
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.ordering.com. method), 213	get_value() (django_elasticsearch_dsl_dr.fields.common. method), 173
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.ordering.Filter. method), 220	get_value() (django_elasticsearch_dsl_dr.fields.common. method), 173
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.search.base. method), 229	get_value() (django_elasticsearch_dsl_dr.fields.common. method), 173
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.search. method), 233	get_value() (django_elasticsearch_dsl_dr.fields. method), 175
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.search. method), 232	get_value() (django_elasticsearch_dsl_dr.fields. method), 175
get_schema_fields() (django_elasticsearch_dsl_dr.filter_backends.search. method), 236	get_value() (django_elasticsearch_dsl_dr.fields. method), 175

[get_value\(\)](#) (`django_elasticsearch_dsl_drf.fields.ListField` **K**
 method), 176 [KeywordField](#) (class in
[get_value\(\)](#) (`django_elasticsearch_dsl_drf.fields.nested_fields.ListField`
 method), 174 [django_elasticsearch_dsl_drf.compat](#)), 263
[get_value\(\)](#) (`django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField`
 method), 174
[get_value\(\)](#) (`django_elasticsearch_dsl_drf.fields.ObjectField`
 method), 176 [label](#) (`django_elasticsearch_dsl_drf.apps.Config`
 attribute), 263
[LimitOffsetPagination](#) (class in
[django_elasticsearch_dsl_drf.pagination](#)),
 265
H
[handle\(\)](#) (`django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command`
 method), 260 [ListField](#) (class in `django_elasticsearch_dsl_drf.fields`),
 175
[help](#) (`django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command`
 attribute), 260 [ListField](#) (class in `django_elasticsearch_dsl_drf.fields.nested_fields`),
 174
[highlight\(\)](#) (`django_elasticsearch_dsl_drf.utils.EmptySearch`
 method), 269 **M**
[highlight_param](#) (`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend`
 attribute), 257 [matching](#) (`django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearcher`
 attribute), 229
[HighlightBackend](#) (class in `matching` (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearcher`
`django_elasticsearch_dsl_drf.filter_backends.highlight`), attribute), 233
 256 [matching](#) (`django_elasticsearch_dsl_drf.filter_backends.search.multi_matching`
 attribute), 232
[hits](#) (`django_elasticsearch_dsl_drf.utils.EmptySearch`
 property), 269 [matching](#) (`django_elasticsearch_dsl_drf.filter_backends.search.MultiMatching`
 attribute), 234
I
[ids_query_param](#) (`django_elasticsearch_dsl_drf.filter_backends.filter_ids.IdsFilterBackend`
 attribute), 191 [matching](#) (`django_elasticsearch_dsl_drf.filter_backends.search.SimpleQueryBackend`
 attribute), 226
[ids_query_param](#) (`django_elasticsearch_dsl_drf.filter_backends.filter_ids.IdsFilterBackend`
 attribute), 207 [MatchPhrasePrefixQueryBackend](#) (class in
[IdsFilterBackend](#) (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
`django_elasticsearch_dsl_drf.filter_backends.filtering`), 225
 206 [MatchPhrasePrefixQueryBackend](#) (class in
[IdsFilterBackend](#) (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
`django_elasticsearch_dsl_drf.filter_backends.filtering.ids`), 222
 190 [MatchPhraseQueryBackend](#) (class in
[ignore](#) (`django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`
 attribute), 269 `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
 225
[ignore_fields](#) (`django_elasticsearch_dsl_drf.serializers.MatchPhraseQueryBackend` (class in
 attribute), 268 `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
 221
[index_aliases](#) (`django_elasticsearch_dsl_drf.serializers.Meta` 221
 attribute), 268 [MatchQueryBackend](#) (class in
[index_classes](#) (`django_elasticsearch_dsl_drf.serializers.Meta` `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
 attribute), 268 226
[IntegerField](#) (class in `MatchQueryBackend` (class in
`django_elasticsearch_dsl_drf.fields`), 175 `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`
 221
[IntegerField](#) (class in 221
`django_elasticsearch_dsl_drf.fields.common`), [Meta](#) (class in `django_elasticsearch_dsl_drf.serializers`),
 173 268
[IPAddressField](#) (class in module
`django_elasticsearch_dsl_drf.fields`), 175 `django_elasticsearch_dsl_drf`, 271
[IPAddressField](#) (class in `django_elasticsearch_dsl_drf.analyzers`,
`django_elasticsearch_dsl_drf.fields.common`), 263
 173 `django_elasticsearch_dsl_drf.apps`, 263
`django_elasticsearch_dsl_drf.compat`, 263

[django_elasticsearch_dsl_drf.constants](#), 232
[264](#)
[django_elasticsearch_dsl_drf.elasticsearch_helpers](#), 225
[264](#)
[django_elasticsearch_dsl_drf.fields](#), 174, 220
[django_elasticsearch_dsl_drf.fields.common](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[172](#), 221
[django_elasticsearch_dsl_drf.fields.helpers](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[173](#), 221
[django_elasticsearch_dsl_drf.fields.nested_fields](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[174](#), 222
[django_elasticsearch_dsl_drf.filter_backends](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[260](#), 222
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[177](#), 223
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.boost_aggregations](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[177](#), 224
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.intersect_aggregations](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[177](#), 233
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.pipeline_aggregations](#), [django_elasticsearch_dsl_drf.filter_backends.search.query](#)
[177](#), 233
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.source](#), [django_elasticsearch_dsl_drf.filter_backends.source](#)
[253](#), 259
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.suggester](#), [django_elasticsearch_dsl_drf.filter_backends.suggester](#)
[195](#), 246
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.suggester](#), [django_elasticsearch_dsl_drf.filter_backends.suggester](#)
[177](#), 237
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.spatial](#), [django_elasticsearch_dsl_drf.filter_backends.suggester](#)
[185](#), 241
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.lists](#), [django_elasticsearch_dsl_drf.helpers](#), 264
[190](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.nested](#), [django_elasticsearch_dsl_drf.management](#)
[192](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.post_filter](#), [django_elasticsearch_dsl_drf.management.commands](#)
[194](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.highlight](#), [django_elasticsearch_dsl_drf.management.commands.elastic](#)
[256](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.pagination](#), [django_elasticsearch_dsl_drf.pagination](#)
[258](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.mixins](#), 255
[258](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.pip_helpers](#), [django_elasticsearch_dsl_drf.pip_helpers](#)
[216](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.serializers](#), [django_elasticsearch_dsl_drf.serializers](#)
[211](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.common](#), [django_elasticsearch_dsl_drf.tests](#), 263
[214](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.test_versions](#), [django_elasticsearch_dsl_drf.tests.test_versions](#)
[233](#), 262
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.utils](#), [django_elasticsearch_dsl_drf.utils](#), 269
[229](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.viewsets](#), [django_elasticsearch_dsl_drf.viewsets](#)
[230](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.compound](#), [django_elasticsearch_dsl_drf.compound](#)
[230](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.historical](#), [django_elasticsearch_dsl_drf.historical](#)
[230](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.more_like_this](#), [more_like_this\(\) \(django_elasticsearch_dsl_drf.viewsets.MoreLikeThis\)](#)
[230](#)
[django_elasticsearch_dsl_drf.filter_backends.adjustable_base.match](#), [django_elasticsearch_dsl_drf.match](#)

`more_like_this()` (in module `django_elasticsearch_dsl_drf.helpers`), 264

`MoreLikeThisMixin` (class in `django_elasticsearch_dsl_drf.viewsets`), 270

`MultiMatchQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`), 226

`MultiMatchQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend.multi_match`), 222

`MultiMatchSearchFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search`), 234

`MultiMatchSearchFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.multi_match`), 232

N

`name` (`django_elasticsearch_dsl_drf.apps.Config` attribute), 263

`NestedField` (class in `django_elasticsearch_dsl_drf.fields`), 176

`NestedField` (class in `django_elasticsearch_dsl_drf.fields.nested_fields`), 174

`NestedFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering.nested`), 207

`NestedFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering.nested`), 192

`NestedQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`), 227

`NestedQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend`), 223

O

`obj_to_dict()` (in module `django_elasticsearch_dsl_drf.wrappers`), 271

`ObjectField` (class in `django_elasticsearch_dsl_drf.fields`), 176

`ObjectField` (class in `django_elasticsearch_dsl_drf.fields.nested_fields`), 174

`ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend` attribute), 212

`ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend` attribute), 213

`ordering_param` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingFilterBackend` attribute), 217

P

`Page` (class in `django_elasticsearch_dsl_drf.pagination`), 265

`page()` (`django_elasticsearch_dsl_drf.pagination.Paginator` method), 266

`page()` (`django_elasticsearch_dsl_drf.pagination.QueryFriendlyPaginator` method), 267

`page_size_query_param` (`django_elasticsearch_dsl_drf.pagination.QueryFriendlyPageNumberPagination` attribute), 267

`PageNumberPagination` (class in `django_elasticsearch_dsl_drf.pagination`), 265

`paginate_queryset()` (`django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination` method), 265

`paginate_queryset()` (`django_elasticsearch_dsl_drf.pagination.PageNumberPagination` method), 266

`paginate_queryset()` (`django_elasticsearch_dsl_drf.pagination.QueryFriendlyPageNumberPagination` method), 267

`pagination_class` (`django_elasticsearch_dsl_drf.viewsets.BaseDocumentView` attribute), 270

`Paginator` (class in `django_elasticsearch_dsl_drf.pagination`), 266

`PostFilterFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering`), 209

`PostFilterFilteringFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter`), 209

`prepare_faceted_search_fields()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearch` class method), 255

`prepare_faceted_search_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FacetedSearch` class method), 185

[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend](#) class method), 202
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 190
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 206
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 193
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 209
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 195
[prepare_filter_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilterBackend](#) class method), 210
[prepare_highlight_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightFilterBackend](#) class method), 257
[prepare_suggester_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend](#) class method), 240
[prepare_suggester_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend](#) class method), 248
[prepare_suggester_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend](#) class method), 246
[prepare_suggester_fields\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend](#) class method), 252

Q

[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 229
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 233
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 230
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 232
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 233
[query_backends](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 236

[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 221
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 222
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 226
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 226
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 227
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 224
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 228
[query_type](#) ([django_elasticsearch_dsl_drf.filter_backends.search.query_base.BaseSearchFilterBackend](#) attribute), 225

R

[raw_checks](#) ([django_elasticsearch_dsl_drf.viewsets.BaseDocumentView](#) method), 270

S

[search_fields](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#) attribute), 268
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234
[search_param](#) ([django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend](#) attribute), 234

[SearchFilterBackend](#) (class in [SuggesterFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), [234](#))
[SearchFilterBackend](#) (class in [SuggestMixin](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search.historical](#)), [230](#))
[SearchFilterBackend](#) (class in [SuggesterFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.suggester.native](#)), [242](#))
[SearchFilterBackend](#) (class in [SuggestMixin](#) (class in [django_elasticsearch_dsl_drf.viewsets](#)), [270](#))
[serialize_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend](#) method), [241](#))
[serialize_queryset\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.suggester.functional.SuggesterFilterBackend](#) method), [249](#))
[serializers](#) ([django_elasticsearch_dsl_drf.serializers.Meta](#) attribute), [268](#))
[setUp\(\)](#) ([django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest](#) method), [263](#))
[SimpleQueryStringQueryBackend](#) (class in [SimpleQueryStringSearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), [228](#))
[SimpleQueryStringQueryBackend](#) (class in [SimpleQueryStringSearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), [224](#))
[SimpleQueryStringSearchFilterBackend](#) (class in [SimpleQueryStringSearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), [236](#))
[SimpleQueryStringSearchFilterBackend](#) (class in [SimpleQueryStringSearchFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.search](#)), [233](#))
[sort\(\)](#) ([django_elasticsearch_dsl_drf.utils.EmptySearch](#) method), [269](#))
[sort_by_list\(\)](#) (in [django_elasticsearch_dsl_drf.helpers](#) module), [265](#))
[SourceBackend](#) (class in [SourceBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.source](#)), [259](#))
[split_lookup_complex_multiple_value\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin](#) class method), [258](#))
[split_lookup_complex_value\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin](#) class method), [258](#))
[split_lookup_filter\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin](#) class method), [259](#))
[split_lookup_name\(\)](#) ([django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin](#) class method), [259](#))
[StringField\(\)](#) (in [django_elasticsearch_dsl_drf.compat](#) module), [263](#))
[suggest\(\)](#) ([django_elasticsearch_dsl_drf.viewsets.SuggestMixin](#) method), [270](#))
[SuggesterFilterBackend](#) (class in [SuggesterFilterBackend](#) (class in [django_elasticsearch_dsl_drf.filter_backends.suggester](#)), [249](#))

T

[test_elasticsearch_dsl_6_8_0\(\)](#) ([django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest](#) method), [263](#))
[test_elasticsearch_dsl_6_8_0\(\)](#) ([django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest](#) method), [263](#))
[to_dict\(\)](#) ([django_elasticsearch_dsl_drf.utils.DictionaryProxy](#) method), [269](#))
[to_dict\(\)](#) ([django_elasticsearch_dsl_drf.utils.EmptySearch](#) method), [269](#))
[to_internal_value\(\)](#) ([django_elasticsearch_dsl_drf.fields.ListField](#) method), [176](#))
[to_internal_value\(\)](#) ([django_elasticsearch_dsl_drf.fields.nested_fields.ListField](#) method), [174](#))
[to_internal_value\(\)](#) ([django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField](#) method), [174](#))
[to_internal_value\(\)](#) ([django_elasticsearch_dsl_drf.fields.ObjectField](#) method), [176](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.BooleanField](#) method), [174](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.CharField](#) method), [175](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.BooleanField](#) method), [172](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.CharField](#) method), [173](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.DateField](#) method), [173](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.FloatField](#) method), [173](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.IntegerField](#) method), [173](#))
[to_representation\(\)](#) ([django_elasticsearch_dsl_drf.fields.common.IPAddressField](#) method), [173](#))

*(django_elasticsearch_dsl_drf.fields.DateField
method), 175*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.FloatField
method), 175*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.IntegerField
method), 175*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.IPAddressField
method), 175*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.ListField
method), 176*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.nested_fields.ListField
method), 174*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField
method), 174*
 to_representation()
*(django_elasticsearch_dsl_drf.fields.ObjectField
method), 176*
 to_representation() (in module
*django_elasticsearch_dsl_drf.fields.helpers),
173*

U

update() (*django_elasticsearch_dsl_drf.serializers.DocumentSerializer
method*), 268

V

VersionsTest (class in
django_elasticsearch_dsl_drf.tests.test_versions),
 262

W

Wrapper (class in *django_elasticsearch_dsl_drf.wrappers*),
 270