

---

# **django-elasticsearch-dsl-drf Documentation**

*Release 0.21*

**Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>**

**Feb 03, 2021**



---

## Contents

---

<b>1</b>	<b>Documentation</b>	<b>3</b>
<b>2</b>	<b>Prerequisites</b>	<b>5</b>
<b>3</b>	<b>Main features and highlights</b>	<b>7</b>
<b>4</b>	<b>Demo</b>	<b>9</b>
<b>5</b>	<b>Installation</b>	<b>11</b>
<b>6</b>	<b>Quick start</b>	<b>13</b>
<b>7</b>	<b>Testing</b>	<b>15</b>
<b>8</b>	<b>Writing documentation</b>	<b>17</b>
<b>9</b>	<b>License</b>	<b>19</b>
<b>10</b>	<b>Support</b>	<b>21</b>
<b>11</b>	<b>Author</b>	<b>23</b>
<b>12</b>	<b>Project documentation</b>	<b>25</b>
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 . . . . .	30
12.3.2.1.3	Publisher model . . . . .	31
12.3.2.1.4	Author model . . . . .	32
12.3.2.1.5	Tag model . . . . .	32
12.3.2.1.6	Book model . . . . .	32

12.3.2.2	Admin classes	33
12.3.2.3	Create database tables	34
12.3.2.4	Fill in some data	34
12.3.2.5	Sample document	35
12.3.2.5.1	Required imports	35
12.3.2.5.2	Index definition	35
12.3.2.5.2.1	Settings	35
12.3.2.5.2.2	Document index	36
12.3.2.5.3	Custom analyzers	36
12.3.2.5.4	Document definition	36
12.3.2.6	Syncing Django’s database with Elasticsearch indexes	38
12.3.2.6.1	Full database sync	38
12.3.2.6.2	Sample partial sync (using custom signals)	38
12.3.2.6.2.1	Required imports	38
12.3.2.6.2.2	Update book index on related model change	38
12.3.2.6.2.3	Update book index on related model removal	39
12.3.2.7	Sample serializer	40
12.3.2.7.1	Required imports	40
12.3.2.7.2	Serializer definition	40
12.3.2.8	ViewSet definition	42
12.3.2.8.1	Required imports	42
12.3.2.8.2	ViewSet definition	42
12.3.2.9	URLs	44
12.3.2.9.1	Required imports	45
12.3.2.9.2	Router definition	45
12.3.2.9.3	URL patterns	45
12.3.2.10	Check what you’ve done so far	45
12.3.2.10.1	URLs	45
12.3.2.10.2	Test in browser	45
12.3.3	Development and debugging	46
12.3.3.1	Profiling tools	46
12.3.3.1.1	Installation	46
12.3.3.1.2	Configuration	46
12.3.3.2	Debugging	47
12.4	Filter usage examples	47
12.4.1	Search	48
12.4.1.1	Search in all fields	48
12.4.1.2	Search a single term on specific field	48
12.4.1.3	Search for multiple terms	48
12.4.1.4	Search for multiple terms in specific fields	48
12.4.2	Filtering	49
12.4.2.1	Supported lookups	49
12.4.2.1.1	Native	49
12.4.2.1.1.1	term	49
12.4.2.1.1.2	terms	49
12.4.2.1.1.3	range	49
12.4.2.1.1.4	exists	50
12.4.2.1.1.5	prefix	50
12.4.2.1.1.6	wildcard	50
12.4.2.1.1.7	ids	50
12.4.2.1.2	Functional	50
12.4.2.1.2.1	contains	51
12.4.2.1.2.2	in	51
12.4.2.1.2.3	gt	51

		12.4.2.1.2.4	gte	51		
		12.4.2.1.2.5	lt	51		
		12.4.2.1.2.6	lte	51		
		12.4.2.1.2.7	startswith	51		
		12.4.2.1.2.8	endswith	52		
		12.4.2.1.2.9	isnull	52		
		12.4.2.1.2.10	exclude	52		
	12.4.3	Usage examples		52		
12.5	Search backends			52		
	12.5.1	Compound search filter backend		52		
		12.5.1.1	Multi-match	53		
			12.5.1.1.1	Sample view	53	
			12.5.1.1.2	Sample request	53	
			12.5.1.1.3	Generated query	53	
		12.5.1.2	Fuzzy search	54		
			12.5.1.2.1	Sample view	54	
			12.5.1.2.2	Sample request	54	
			12.5.1.2.3	Generated query	55	
	12.5.2	Multi match search filter backend		55		
		12.5.2.1	Sample view	55		
		12.5.2.2	Sample request	56		
		12.5.2.3	Generated query	56		
		12.5.2.4	Options	57		
			12.5.2.4.1	Type options	57	
			12.5.2.4.2	Operator options	57	
	12.5.3	Simple query string filter backend		57		
		12.5.3.1	Sample view	57		
		12.5.3.2	Sample request 1	58		
		12.5.3.3	Generated query 1	58		
		12.5.3.4	Sample request 2	59		
		12.5.3.5	Generated query 2	59		
		12.5.3.6	Sample request 3	59		
		12.5.3.7	Generated query 3	60		
		12.5.3.8	Options	60		
			12.5.3.8.1	Default Operator options	60	
12.6	Basic usage examples			60		
	12.6.1	Example app		61		
		12.6.1.1	Sample models	61		
		12.6.1.2	Sample document	62		
		12.6.1.3	Sample serializer	63		
		12.6.1.4	Sample view	64		
		12.6.1.5	Usage example	65		
			12.6.1.5.1	Sample queries	65	
				12.6.1.5.1.1	Search	65
				12.6.1.5.1.2	Filtering	66
				12.6.1.5.1.3	Ordering	67
12.7	Advanced usage examples			67		
	12.7.1	Example app		69		
		12.7.1.1	Sample models	69		
		12.7.1.2	Sample document	71		
			12.7.1.2.1	Index definition	71	
				12.7.1.2.1.1	Settings	71
				12.7.1.2.1.2	Document index	72
		12.7.1.3	Sample serializer	74		

12.7.1.4	Sample view . . . . .	75
12.7.1.5	Usage example . . . . .	76
12.7.1.5.1	Search . . . . .	77
12.7.1.5.2	Filtering . . . . .	78
12.7.1.5.3	Ordering . . . . .	78
12.7.1.6	Ids filter . . . . .	79
12.7.1.7	Faceted search . . . . .	79
12.7.1.8	Post-filter . . . . .	80
12.7.1.8.1	Sample view . . . . .	80
12.7.1.8.2	Sample queries . . . . .	82
12.7.1.9	Geo-spatial features . . . . .	82
12.7.1.9.1	Filtering . . . . .	82
12.7.1.9.2	Ordering . . . . .	83
12.7.1.9.3	Geo-shape . . . . .	83
12.7.1.10	Suggestions . . . . .	84
12.7.1.10.1	Completion suggesters . . . . .	85
12.7.1.10.1.1	Document definition . . . . .	85
12.7.1.10.1.2	Serializer definition . . . . .	86
12.7.1.10.1.3	ViewSet definition . . . . .	87
12.7.1.10.1.4	Sample requests/responses . . . . .	88
12.7.1.10.1.5	Suggestions on Array/List field . . . . .	90
12.7.1.10.1.6	Sample requests/responses . . . . .	90
12.7.1.10.1.7	Context suggesters . . . . .	91
12.7.1.10.1.8	<i>category</i> context . . . . .	91
12.7.1.10.1.9	<i>geo</i> context . . . . .	92
12.7.1.10.2	Term and Phrase suggestions . . . . .	94
12.7.1.10.2.1	Document definition . . . . .	94
12.7.1.10.2.2	ViewSet definition . . . . .	96
12.7.1.10.2.3	Sample requests/responses . . . . .	99
12.7.1.10.2.4	Completion . . . . .	100
12.7.1.10.2.5	Term . . . . .	101
12.7.1.10.2.6	Phrase . . . . .	102
12.7.1.11	Functional suggestions . . . . .	102
12.7.1.11.1	Document definition . . . . .	103
12.7.1.11.2	ViewSet definition . . . . .	104
12.7.1.12	Highlighting . . . . .	106
12.7.1.13	Pagination . . . . .	108
12.7.1.13.1	Page number pagination . . . . .	108
12.7.1.13.2	Limit/offset pagination . . . . .	108
12.7.1.13.3	Customisations . . . . .	108
12.8	Nested fields usage examples . . . . .	109
12.8.1	Example app . . . . .	110
12.8.1.1	Sample models . . . . .	110
12.8.1.2	Sample document . . . . .	114
12.8.1.2.1	Index definition . . . . .	114
12.8.1.2.1.1	Settings . . . . .	114
12.8.1.2.1.2	Document index . . . . .	115
12.8.1.3	Sample serializer . . . . .	118
12.8.1.4	Sample view . . . . .	118
12.8.1.5	Usage example . . . . .	120
12.8.1.5.1	Sample queries . . . . .	121
12.8.1.5.1.1	Search . . . . .	121
12.8.1.5.1.2	Nested filtering . . . . .	121
12.8.1.5.1.3	Nested search . . . . .	121

	12.8.1.5.1.4	Sample models	121
	12.8.1.5.1.5	Sample document	122
	12.8.1.5.1.6	Sample view	124
	12.8.1.5.1.7	Sample request	126
	12.8.1.5.1.8	Filtering	126
	12.8.1.5.1.9	Ordering	126
	12.8.1.6	Suggestions	126
	12.8.1.7	Nested aggregations/facets	127
12.9	More like this		130
12.9.1	Usage example		130
12.9.1.1	Sample document		130
12.9.1.2	Sample view		131
12.9.1.3	Sample request		132
12.9.1.4	Generated query		132
12.9.1.5	Options		133
12.10	Global aggregations		133
12.10.1	Sample view		133
12.10.2	Sample request		134
12.10.3	Generated query		134
12.10.4	Sample response		135
12.11	Configuration tweaks		136
12.11.1	Ignore certain Elasticsearch exceptions		136
12.12	Source filtering backend		136
12.13	Pagination		137
12.13.1	Page number pagination		137
12.13.2	Query friendly page number pagination		137
12.13.3	Limit/offset pagination		138
12.13.3.1	Customisations		138
12.14	Indexing troubleshooting		139
12.14.1	Timeout		139
12.14.2	Chunk size		139
12.14.3	Use parallel indexing		140
12.14.4	Limit the number of items indexed at once		140
12.15	FAQ		141
12.15.1	Questions and answers		141
12.16	Demo		142
12.16.1	Run demo locally		142
12.16.2	Prerequisites		143
12.17	frontend demo for django-elasticsearch-dsl-drf		143
12.17.1	Quick start		143
12.17.1.1	Install the django requirements		143
12.17.1.2	Install Elasticsearch requirements		143
12.17.1.3	Run Elasticsearch		143
12.17.1.4	Build Elasticsearch index		144
12.17.1.5	Install React requirements		144
12.17.1.6	Run Django		144
12.17.1.7	Run React demo app		144
12.18	Release history and notes		144
12.18.1	0.21		145
12.18.2	0.20.9		145
12.18.3	0.20.8		145
12.18.4	0.20.7		145
12.18.5	0.20.6		145
12.18.6	0.20.5		146

12.18.7 0.20.4 . . . . .	146
12.18.8 0.20.3 . . . . .	146
12.18.9 0.20.2 . . . . .	146
12.18.100.20.1 . . . . .	146
12.18.110.20 . . . . .	146
12.18.120.19 . . . . .	146
12.18.130.18 . . . . .	147
12.18.140.17.7 . . . . .	147
12.18.150.17.6 . . . . .	147
12.18.160.17.5 . . . . .	147
12.18.170.17.4 . . . . .	147
12.18.180.17.3 . . . . .	148
12.18.190.17.2 . . . . .	148
12.18.200.17.1 . . . . .	148
12.18.210.17 . . . . .	148
12.18.220.16.3 . . . . .	148
12.18.230.16.2 . . . . .	148
12.18.240.16.1 . . . . .	149
12.18.250.16 . . . . .	149
12.18.260.15.1 . . . . .	149
12.18.270.15 . . . . .	149
12.18.280.14 . . . . .	150
12.18.290.13.2 . . . . .	150
12.18.300.13.1 . . . . .	150
12.18.310.13 . . . . .	150
12.18.320.12 . . . . .	150
12.18.330.11 . . . . .	151
12.18.340.10 . . . . .	152
12.18.350.9 . . . . .	152
12.18.360.8.4 . . . . .	152
12.18.370.8.3 . . . . .	152
12.18.380.8.2 . . . . .	152
12.18.390.8.1 . . . . .	153
12.18.400.8 . . . . .	153
12.18.410.7.2 . . . . .	153
12.18.420.7.1 . . . . .	153
12.18.430.7 . . . . .	154
12.18.440.6.4 . . . . .	154
12.18.450.6.3 . . . . .	154
12.18.460.6.2 . . . . .	154
12.18.470.6.1 . . . . .	154
12.18.480.6 . . . . .	154
12.18.490.5.1 . . . . .	154
12.18.500.5 . . . . .	155
12.18.510.4.4 . . . . .	155
12.18.520.4.3 . . . . .	155
12.18.530.4.2 . . . . .	155
12.18.540.4.1 . . . . .	155
12.18.550.4 . . . . .	156
12.18.560.3.12 . . . . .	156
12.18.570.3.11 . . . . .	156
12.18.580.3.10 . . . . .	156
12.18.590.3.9 . . . . .	156
12.18.600.3.8 . . . . .	156



12.18.610.3.7	157
12.18.620.3.6	157
12.18.630.3.5	157
12.18.640.3.4	157
12.18.650.3.3	157
12.18.660.3.2	157
12.18.670.3.1	157
12.18.680.3	157
12.18.690.2.6	158
12.18.700.2.5	158
12.18.710.2.4	158
12.18.720.2.3	158
12.18.730.2.2	158
12.18.740.2.1	158
12.18.750.2	158
12.18.760.1.8	158
12.18.770.1.7	159
12.18.780.1.6	159
12.18.790.1.5	159
12.18.800.1.4	159
12.18.810.1.3	159
12.18.820.1.2	159
12.18.830.1.1	160
12.18.840.1	160
12.19 django_elasticsearch_dsl_drf package	160
12.19.1 Subpackages	160
12.19.1.1 django_elasticsearch_dsl_drf.fields package	160
12.19.1.1.1 Submodules	160
12.19.1.1.2 django_elasticsearch_dsl_drf.fields.common module	160
12.19.1.1.3 django_elasticsearch_dsl_drf.fields.helpers module	161
12.19.1.1.4 django_elasticsearch_dsl_drf.fields.nested_fields module	162
12.19.1.1.5 Module contents	164
12.19.1.2 django_elasticsearch_dsl_drf.filter_backends package	168
12.19.1.2.1 Subpackages	168
12.19.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package	168
12.19.1.2.1.2 Submodules	168
12.19.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module	168
12.19.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module	168
12.19.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module	168
12.19.1.2.1.6 Module contents	168
12.19.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package	168
12.19.1.2.1.8 Submodules	168
12.19.1.2.1.9 django_elasticsearch_dsl_drf.filter_backends.filtering.common module	168
12.19.1.2.1.10 django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial module	176
12.19.1.2.1.11 django_elasticsearch_dsl_drf.filter_backends.filtering.ids module	181
12.19.1.2.1.12 django_elasticsearch_dsl_drf.filter_backends.filtering.nested module	182
12.19.1.2.1.13 django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter module	184

12.19.1.2.1.14	Module contents . . . . .	186
12.19.1.2.1.15	django_elasticsearch_dsl_drf.filter_backends.ordering package . .	201
12.19.1.2.1.16	Submodules . . . . .	201
12.19.1.2.1.17	django_elasticsearch_dsl_drf.filter_backends.ordering.common module . . . . .	201
12.19.1.2.1.18	django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial module . . . . .	204
12.19.1.2.1.19	Module contents . . . . .	206
12.19.1.2.1.20	django_elasticsearch_dsl_drf.filter_backends.search package . . .	211
12.19.1.2.1.21	Subpackages . . . . .	211
12.19.1.2.1.22	django_elasticsearch_dsl_drf.filter_backends.search.query_backends package . . . . .	211
12.19.1.2.1.23	Submodules . . . . .	211
12.19.1.2.1.24	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base module . . . . .	211
12.19.1.2.1.25	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match module . . . . .	211
12.19.1.2.1.26	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase module . . . . .	211
12.19.1.2.1.27	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix module . . . . .	212
12.19.1.2.1.28	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match module . . . . .	212
12.19.1.2.1.29	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested module . . . . .	213
12.19.1.2.1.30	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string module . . . . .	214
12.19.1.2.1.31	Module contents . . . . .	215
12.19.1.2.1.32	Submodules . . . . .	219
12.19.1.2.1.33	django_elasticsearch_dsl_drf.filter_backends.search.base module .	219
12.19.1.2.1.34	django_elasticsearch_dsl_drf.filter_backends.search.compound module . . . . .	220
12.19.1.2.1.35	django_elasticsearch_dsl_drf.filter_backends.search.historical module . . . . .	220
12.19.1.2.1.36	django_elasticsearch_dsl_drf.filter_backends.search.multi_match module . . . . .	222
12.19.1.2.1.37	django_elasticsearch_dsl_drf.filter_backends.search.query_string module . . . . .	222
12.19.1.2.1.38	django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string module . . . . .	222
12.19.1.2.1.39	Module contents . . . . .	223
12.19.1.2.1.40	django_elasticsearch_dsl_drf.filter_backends.suggester package .	226
12.19.1.2.1.41	Submodules . . . . .	226
12.19.1.2.1.42	django_elasticsearch_dsl_drf.filter_backends.suggester.functional module . . . . .	226
12.19.1.2.1.43	django_elasticsearch_dsl_drf.filter_backends.suggester.native module . . . . .	231
12.19.1.2.1.44	Module contents . . . . .	235
12.19.1.2.2	Submodules . . . . .	242
12.19.1.2.3	django_elasticsearch_dsl_drf.filter_backends.faceted_search module . . .	242
12.19.1.2.4	django_elasticsearch_dsl_drf.filter_backends.highlight module . . . . .	245
12.19.1.2.5	django_elasticsearch_dsl_drf.filter_backends.mixins module . . . . .	246
12.19.1.2.6	django_elasticsearch_dsl_drf.filter_backends.source module . . . . .	247
12.19.1.2.7	Module contents . . . . .	248

12.19.1.3	django_elasticsearch_dsl_drf.management package	249
12.19.1.3.1	Subpackages	249
12.19.1.3.1.1	django_elasticsearch_dsl_drf.management.commands package	249
12.19.1.3.1.2	Submodules	249
12.19.1.3.1.3	django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes module	249
12.19.1.3.1.4	Module contents	249
12.19.1.3.2	Module contents	249
12.19.1.4	django_elasticsearch_dsl_drf.tests package	249
12.19.1.4.1	Submodules	249
12.19.1.4.2	django_elasticsearch_dsl_drf.tests.base module	249
12.19.1.4.3	django_elasticsearch_dsl_drf.tests.data_mixins module	250
12.19.1.4.4	django_elasticsearch_dsl_drf.tests.test_faceted_search module	250
12.19.1.4.5	django_elasticsearch_dsl_drf.tests.test_filtering_common module	250
12.19.1.4.6	django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module	253
12.19.1.4.7	django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module	255
12.19.1.4.8	django_elasticsearch_dsl_drf.tests.test_filtering_nested module	255
12.19.1.4.9	django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module	257
12.19.1.4.10	django_elasticsearch_dsl_drf.tests.test_functional_suggesters module	259
12.19.1.4.11	django_elasticsearch_dsl_drf.tests.test_helpers module	260
12.19.1.4.12	django_elasticsearch_dsl_drf.tests.test_highlight module	260
12.19.1.4.13	django_elasticsearch_dsl_drf.tests.test_more_like_this module	260
12.19.1.4.14	django_elasticsearch_dsl_drf.tests.test_ordering_common module	260
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	263
12.19.1.4.19	django_elasticsearch_dsl_drf.tests.test_search module	263
12.19.1.4.20	django_elasticsearch_dsl_drf.tests.test_search_multi_match module	264
12.19.1.4.21	django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module	264
12.19.1.4.22	django_elasticsearch_dsl_drf.tests.test_serializers module	265
12.19.1.4.23	django_elasticsearch_dsl_drf.tests.test_source module	266
12.19.1.4.24	django_elasticsearch_dsl_drf.tests.test_suggesters module	266
12.19.1.4.25	django_elasticsearch_dsl_drf.tests.test_versions module	267
12.19.1.4.26	django_elasticsearch_dsl_drf.tests.test_views module	267
12.19.1.4.27	django_elasticsearch_dsl_drf.tests.test_wrappers module	268
12.19.1.4.28	Module contents	268
12.19.2	Submodules	268
12.19.3	django_elasticsearch_dsl_drf.analyzers module	268
12.19.4	django_elasticsearch_dsl_drf.apps module	268
12.19.5	django_elasticsearch_dsl_drf.compat module	269
12.19.6	django_elasticsearch_dsl_drf.constants module	269
12.19.7	django_elasticsearch_dsl_drf.elasticsearch_helpers module	269
12.19.8	django_elasticsearch_dsl_drf.helpers module	269
12.19.9	django_elasticsearch_dsl_drf.pagination module	270
12.19.10	django_elasticsearch_dsl_drf.pip_helpers module	272
12.19.11	django_elasticsearch_dsl_drf.serializers module	273
12.19.12	django_elasticsearch_dsl_drf.utils module	274
12.19.13	django_elasticsearch_dsl_drf.versions module	274
12.19.14	django_elasticsearch_dsl_drf.viewsets module	274
12.19.15	django_elasticsearch_dsl_drf.wrappers module	275
12.19.16	Module contents	276

<b>13 Indices and tables</b>	<b>277</b>
<b>Python Module Index</b>	<b>279</b>
<b>Index</b>	<b>281</b>

Integrate [Elasticsearch DSL](#) with [Django REST framework](#) in the shortest way possible, with least efforts possible.

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

You are expected to use [django-elasticsearch-dsl](#) for defining your Elasticsearch documents.



# CHAPTER 1

---

## Documentation

---

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

Make sure to read [FAQ](#).





## CHAPTER 2

---

### Prerequisites

---

- Django 2.2, 3.0 and 3.1.
- 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](#).



## CHAPTER 4

---

### Demo

---

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

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

```
docker-compose up
```

It will set up:

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



---

## Installation

---

- (1) Install latest stable version from PyPI:

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

or latest stable version from GitHub:

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

- (2) Add `rest_framework`, `django_elasticsearch_dsl` and `django_elasticsearch_dsl_drf` to `INSTALLED_APPS`:

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





## CHAPTER 6

---

### Quick start

---

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

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



## CHAPTER 7

---

### Testing

---

Project is covered with tests.

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

```
tox -e 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  
*****
```



## CHAPTER 9

---

### License

---

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





## CHAPTER 10

---

### Support

---

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



## CHAPTER 11

---

Author

---

Artur Barseghyan <[artur.barseghyan@gmail.com](mailto:artur.barseghyan@gmail.com)>



# CHAPTER 12

---

## Project documentation

---

Contents:

### Table of Contents

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

## 12.1 Dependencies

elasticsearch and elasticsearch-dsl

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

Current compatibility matrix is:

This package	Elasticsearch
0.18.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 `django-rest-framework` 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
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'localhost:9200'
```

(continues on next page)

(continued from previous page)

```
} ,  
}
```

## 12.3.2 Example app

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

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

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

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

### 12.3.2.1 Sample models

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

#### 12.3.2.1.1 Required imports

Imports required for model definition.

*books/models/book.py*

```
import json  
  
from django.conf import settings  
from django.db import models  
from django.utils.translation import ugettext, ugettext_lazy as _  
  
from six import python_2_unicode_compatible
```

#### 12.3.2.1.2 Book statuses

*books/models/book.py*

```

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

```

### 12.3.2.1.3 Publisher model

*books/models/book.py*

```

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

    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)

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

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

```

(continues on next page)

(continued from previous page)

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

#### 12.3.2.1.4 Author model

*books/models/author.py*

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

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

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

        ordering = ["id"]

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

#### 12.3.2.1.5 Tag model

*books/models/tag.py*

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

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

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

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

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

#### 12.3.2.1.6 Book model

*books/models/book.py*

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

(continues on next page)

(continued from previous page)

```

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

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

    ordering = ["isbn"]

def __str__(self):
    return self.title

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

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

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

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

```

### 12.3.2.2 Admin classes

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

*books/admin.py*

```
from django.contrib import admin

from .models import *

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

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

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

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

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

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

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

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

### 12.3.2.3 Create database tables

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

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

### 12.3.2.4 Fill in some data

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

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

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

### 12.3.2.5 Sample document

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

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

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

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

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

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

#### 12.3.2.5.1 Required imports

*search\_indexes/documents/book.py*

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

from books.models import Book
```

#### 12.3.2.5.2 Index definition

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

##### 12.3.2.5.2.1 Settings

---

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

---

*settings/base.py*

---

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

---

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

*settings/testing.py*

---

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

---

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

*settings/production.py*

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

#### 12.3.2.5.2.2 Document index

*search\_indexes/documents/book.py*

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

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

#### 12.3.2.5.3 Custom analyzers

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

#### 12.3.2.5.4 Document definition

*search\_indexes/documents/book.py*

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

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

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

(continues on next page)



(continued from previous page)

```

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

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

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

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

    publication_date = fields.DateField()

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

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

    price = fields.FloatField()

    pages = fields.IntegerField()

    stock_count = fields.IntegerField()

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

```

(continues on next page)

(continued from previous page)

```
)

class 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':
            instances = instance.books.all()
            for _instance in instances:
                registry.update(_instance)

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

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

```

### 12.3.2.6.2.3 Update book index on related model removal

*search\_indexes/signals.py*

```

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

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

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

        # If it is `books.Author` that is being updated.

```

(continues on next page)

(continued from previous page)

```
if model_name == 'author':
    instances = instance.books.all()
    for _instance in instances:
        registry.update(_instance)
        # registry.delete(_instance, raise_on_error=False)

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

### 12.3.2.7 Sample serializer

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

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

#### 12.3.2.7.1 Required imports

*search\_indexes/serializers/book.py*

```
import json

from rest_framework import serializers
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from .documents import BookDocument
```

#### 12.3.2.7.2 Serializer definition

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

*search\_indexes/serializers/book.py*

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

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

        # Specify the correspondent document class
        document = BookDocument

        # List the serializer fields. Note, that the order of the fields
        # is preserved in the ViewSet.
        fields = (
            'id',
            'title',
            'description',
```

(continues on next page)

(continued from previous page)

```

        'summary',
        'publisher',
        'publication_date',
        'state',
        'isbn',
        'price',
        'pages',
        'stock_count',
        'tags',
    )

```

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

*search\_indexes/serializers/book.py*

```

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

    id = serializers.IntegerField(read_only=True)

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

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

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

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

    def get_tags(self, obj):
        """Get tags."""
        if obj.tags:

```

(continues on next page)

(continued from previous page)

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

### 12.3.2.8 ViewSet definition

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

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

#### 12.3.2.8.1 Required imports

*search\_indexes/viewsets/book.py*

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

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

#### 12.3.2.8.2 ViewSet definition

*search\_indexes/viewsets/book.py*

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

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

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

### 12.3.2.9 URLs

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

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



#### 12.3.2.9.1 Required imports

*search\_indexes/urls.py*

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

from .views import BookDocumentView
```

#### 12.3.2.9.2 Router definition

*search\_indexes/urls.py*

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

#### 12.3.2.9.3 URL patterns

*search\_indexes/urls.py*

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

#### 12.3.2.10 Check what you've done so far

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

##### 12.3.2.10.1 URLs

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

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

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

##### 12.3.2.10.2 Test in browser

Open the following URL in your browser.

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

Perform a number of lookups:

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

## 12.3.3 Development and debugging

### 12.3.3.1 Profiling tools

Looking for profiling tools for Elasticsearch?

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

#### 12.3.3.1.1 Installation

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

#### 12.3.3.1.2 Configuration

Change your development settings in the following way:

*settings/dev.py*

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

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

DEBUG_TOOLBAR_CONFIG = {
    'INTERCEPT_REDIRECTS': False,
}

DEBUG_TOOLBAR_PANELS = (
    # Defaults
    'debug_toolbar.panels.versions.VersionsPanel',
    'debug_toolbar.panels.timer.TimerPanel',
    'debug_toolbar.panels.settings.SettingsPanel',
    'debug_toolbar.panels.headers.HeadersPanel',
    'debug_toolbar.panels.request.RequestPanel',
    'debug_toolbar.panels.sql.SQLPanel',
)
```

(continues on next page)

(continued from previous page)

```

'debug_toolbar.panels.staticfiles.StaticFilesPanel',
'debug_toolbar.panels.templates.TemplatesPanel',
'debug_toolbar.panels.cache.CachePanel',
'debug_toolbar.panels.signals.SignalsPanel',
'debug_toolbar.panels.logging.LoggingPanel',
'debug_toolbar.panels.redirects.RedirectsPanel',
# Additional
'elastic_panel.panel.ElasticDebugPanel',
)

```

### 12.3.3.2 Debugging

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

## 12.4 Filter usage examples

Example usage of filtering backends.

Contents:

### Table of Contents

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

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

## 12.4.1 Search

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

Multiple search terms are joined with OR.

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

### 12.4.1.1 Search in all fields

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

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

### 12.4.1.2 Search a single term on specific field

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

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

### 12.4.1.3 Search for multiple terms

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

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

### 12.4.1.4 Search for multiple terms in specific fields

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

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

## 12.4.2 Filtering

### 12.4.2.1 Supported lookups

#### 12.4.2.1.1 Native

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

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

##### 12.4.2.1.1.1 term

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

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

##### 12.4.2.1.1.2 terms

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

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

##### 12.4.2.1.1.3 range

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

**From, to**

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

**From, to, boost**

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

#### **12.4.2.1.1.4 exists**

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

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

#### **12.4.2.1.1.5 prefix**

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

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

#### **12.4.2.1.1.6 wildcard**

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

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

Should match: *delusional insanity*.

#### **12.4.2.1.1.7 ids**

Find documents with the specified type and IDs.

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

#### **12.4.2.1.2 Functional**

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

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

#### 12.4.2.1.2.1 contains

Case-insensitive containment test.

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

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

#### 12.4.2.1.2.2 in

In a given list.

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

#### 12.4.2.1.2.3 gt

Greater than.

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

#### 12.4.2.1.2.4 gte

Greater than or equal to.

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

#### 12.4.2.1.2.5 lt

Less than.

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

#### 12.4.2.1.2.6 lte

Less than or equal to.

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

#### 12.4.2.1.2.7 startswith

Case-sensitive starts-with.

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

Should match: *biography*, *bio mechanics*

#### 12.4.2.1.2.8 endswith

Case-sensitive ends-with.

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

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

#### 12.4.2.1.2.9 isnull

Takes either True or False.

**True**

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

**False**

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

#### 12.4.2.1.2.10 exclude

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

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

### 12.4.3 Usage examples

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

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

Additionally, see:

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

## 12.5 Search backends

### 12.5.1 Compound search filter backend

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



### 12.5.1.1 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'

    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": {
```

(continues on next page)

(continued from previous page)

```
"should": [  
    {  
        "match": {  
            "title": {  
                "query": "enim"  
            }  
        }  
    },  
    {  
        "match": {  
            "description": {  
                "query": "enim"  
            }  
        }  
    },  
    {  
        "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"
            }
          }
        },
        {
          "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):
```

(continues on next page)

(continued from previous page)

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

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

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
```

(continues on next page)

(continued from previous page)

```

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",

```

(continues on next page)

(continued from previous page)

```

    "price"
  ],
  "from": 0,
  "size": 1
}

```

#### 12.5.3.4 Sample request 2

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

```

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

```

#### 12.5.3.5 Generated query 2

```

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

```

#### 12.5.3.6 Sample request 3

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

```

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

```

### 12.5.3.7 Generated query 3

```
{
  "query": {
    "simple_query_string": {
      "query": "\"Pool of Tears\" -considering",
      "default_operator": "and",
      "fields": [
        "title",
        "description",
        "summary"
      ]
    }
  },
  "sort": [
    "_score",
    "id",
    "title",
    "price"
  ],
  "from": 0,
  "size": 1
}
```

### 12.5.3.8 Options

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

Selective list of available options:

- `default_operator`

#### 12.5.3.8.1 Default Operator options

Can be either `and` or `or`.

#### Example

```
class BookSimpleQueryStringSearchFilterBackendDocumentViewSet(DocumentViewSet):

    # ...

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

## 12.6 Basic usage examples

Basic Django REST framework integration example

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

- [models](#)



- documents
- serializers
- views

Contents:

## Table of Contents

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

## 12.6.1 Example app

### 12.6.1.1 Sample models

*books/models/publisher.py*

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

    name = models.CharField(max_length=30)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)

    class Meta(object):
```

(continues on next page)

(continued from previous page)

```

        """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'),

```

(continues on next page)

(continued from previous page)

```

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

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

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

    model = Publisher # The model associate with this Document

```

### 12.6.1.3 Sample serializer

*search\_indexes/serializers/book.py*

```

import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

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

    location = serializers.SerializerMethodField()

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

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

```

(continues on next page)

(continued from previous page)

```

        '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',
    }

```

(continues on next page)

(continued from previous page)

```

        'country': 'country.raw',
    }
    # Define ordering fields
    ordering_fields = {
        'id': None,
        'name': None,
        'city': None,
        'country': None,
    }
    # Specify default ordering
    ordering = ('id', 'name',)
    # Define geo-spatial filtering fields
    geo_spatial_filter_fields = {
        'location': {
            'lookups': [
                LOOKUP_FILTER_GEO_DISTANCE,
            ],
        },
    }
}

```

### 12.6.1.5 Usage example

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

#### 12.6.1.5.1 Sample queries

##### 12.6.1.5.1.1 Search

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

Multiple search terms are joined with OR.

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

#### Search in all fields

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

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

#### Search a single term on specific field

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

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

#### Search for multiple terms

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

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

#### Search for multiple terms in specific fields

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

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

### 12.6.1.5.1.2 Filtering

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

Multiple filter terms are joined with AND.

#### Filter documents by single field

Filter documents by field (`city`) “yerevan”.

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

#### Filter documents by multiple fields

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

```
http://127.0.0.1:8080/search/publisher/?city__in=yerevan__groningen
```

#### Filter document by a single field

Filter documents by (field `country`) “Armenia”.

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

#### Filter documents by multiple fields

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

```
http://127.0.0.1:8080/search/publisher/?city__in=yerevan__amsterdam
```

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

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

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

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

#### Filter documents by a word part of a single field

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

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

#### Geo-distance filtering

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

```
http://127.0.0.1:8000/search/publishers/?location__geo_distance=100000km__12.04__-63.  
→ 93
```

### 12.6.1.5.1.3 Ordering

The `-` prefix means ordering should be descending.

#### Order documents by field (ascending)

Filter documents by field `city` (ascending).

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

#### Order documents by field (descending)

Filter documents by field `country` (descending).

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

#### Order documents by multiple fields

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

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

## 12.7 Advanced usage examples

Advanced Django REST framework integration examples.

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

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

Contents:

### Table of Contents

- *Advanced usage examples*
  - *Example app*
    - \* *Sample models*
    - \* *Sample document*
      - *Index definition*
      - *Settings*
      - *Document index*
    - \* *Sample serializer*
    - \* *Sample view*
    - \* *Usage example*

- *Search*
- *Filtering*
- *Ordering*
- \* *Ids filter*
- \* *Faceted search*
- \* *Post-filter*
  - *Sample view*
  - *Sample queries*
- \* *Geo-spatial features*
  - *Filtering*
  - *Ordering*
  - *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 ugettext, ugettext_lazy as _

from six import python_2_unicode_compatible

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

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

    name = models.CharField(max_length=30)
    info = models.TextField(null=True, blank=True)
    address = models.CharField(max_length=50)
    city = models.CharField(max_length=60)
    state_province = models.CharField(max_length=30)
    country = models.CharField(max_length=50)
    website = models.URLField()
    latitude = models.DecimalField(null=True,
                                   blank=True,
                                   decimal_places=15,
                                   max_digits=19,
                                   default=0)
    longitude = models.DecimalField(null=True,
                                    blank=True,
                                    decimal_places=15,
                                    max_digits=19,
                                    default=0)
```

(continues on next page)

(continued from previous page)

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

    ordering = ["id"]

    def __str__(self):
        return self.name

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

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

*books/models/author.py*

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

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

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

        ordering = ["id"]

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

*books/models/tag.py*

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

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

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

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

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

*books/models/book.py*

```

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

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

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

        ordering = ["isbn"]

    def __str__(self):
        return self.title

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

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

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

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

```

### 12.7.1.2 Sample document

#### 12.7.1.2.1 Index definition

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

##### 12.7.1.2.1.1 Settings

*settings/base.py*

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

*settings/testing.py*

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

*settings/production.py*

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

#### 12.7.1.2.1.2 Document index

*search\_indexes/documents/book.py*

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

from books.models import Book

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

publication_date = fields.DateField()

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

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

price = fields.FloatField()

pages = fields.IntegerField()

stock_count = fields.IntegerField()

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

```

(continues on next page)

(continued from previous page)

```

    },
    multi=True
)

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

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

```

*search\_indexes/serializers/book.py*

```

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

    id = serializers.SerializerMethodField()

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

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

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

        fields = (
            'id',
            'title',
            'description',

```

(continues on next page)

(continued from previous page)

```

        '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,
    ]

```

(continues on next page)

(continued from previous page)

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

    # ...
```

(continues on next page)

(continued from previous page)

```

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

# ...

```

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

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

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

### 12.7.1.8 Post-filter

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

#### 12.7.1.8.1 Sample view

---

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

---

*search\_indexes/viewsets/book.py*

```

# ...

from django_elasticsearch_dsl_drf.filter_backends import (
    # ...

```

(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,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
        },
    }

```

(continues on next page)

(continued from previous page)

```
# 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',)
```

### 12.7.1.8.2 Sample queries

#### Filter documents by field

Filter documents by field (state) “published”.

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

#### Filter documents by multiple fields

Filter documents by field (states) “published” and “in\_progress”.

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

### 12.7.1.9 Geo-spatial features

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

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

#### 12.7.1.9.1 Filtering

##### Geo-distance filtering

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

```
http://localhost:8000/search/publishers/?location__geo_distance=100000km__12.04__-63.
↪ 93
```

### Geo-polygon filtering

Filter documents that are located in the given polygon.

```
http://localhost:8000/search/publishers/?location__geo_polygon=40,-70__30,-80__20,-90
```

### Geo-bounding-box filtering

Filter documents that are located in the given bounding box.

```
http://localhost:8000/search/publishers/?location__geo_bounding_box=44.87,40.07__43.
↪ 87,41.11
```

## 12.7.1.9.2 Ordering

### Geo-distance ordering

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

## 12.7.1.9.3 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')

# ...

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],
```

(continues on next page)

(continued from previous page)

```
        '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.

### 12.7.1.10 Suggestions

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

---

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

---

### 12.7.1.10.1 Completion suggesters

#### 12.7.1.10.1.1 Document definition

To make use of suggestions, you should properly index relevant fields of your documents using `fields.CompletionField`.

*search\_indexes/documents/publisher.py*

```
from django.conf import settings

from django_elasticsearch_dsl import Document, Index, fields

from books.models import Publisher

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

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

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

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

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

    info = fields.StringField()

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

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

(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()

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

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

    model = Publisher # The model associate with this Document

```

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

#### 12.7.1.10.1.2 Serializer definition

This is how publisher serializer would look like.

*search\_indexes/serializers/publisher.py*

```

import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

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

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

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

```

(continues on next page)

(continued from previous page)

```

        'city',
        'state_province',
        'country',
        'website',
    )

```

#### 12.7.1.10.1.3 ViewSet definition

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

---

**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."""

    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',

```

(continues on next page)

(continued from previous page)

```

        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'state_province_suggest': {
        'field': 'state_province.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
    'country_suggest': {
        'field': 'country.suggest',
        'suggesters': [
            SUGGESTER_COMPLETION,
        ],
    },
}

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

```

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

#### 12.7.1.10.1.4 Sample requests/responses

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

##### Request

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

##### Response

```

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

```

(continues on next page)

(continued from previous page)

```

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

```

You can also have multiple suggesters per request.

### Request

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

### Response

```

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

```

(continues on next page)

(continued from previous page)

```

        },
        {
            "score": 1.0,
            "text": "Borman, Post and Wendt"
        },
        {
            "score": 1.0,
            "text": "Book League of America"
        }
    ],
    "offset": 0,
    "length": 1
}
]
}

```

#### 12.7.1.10.1.5 Suggestions on Array/List field

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

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

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

#### 12.7.1.10.1.6 Sample requests/responses

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

##### Request

```
GET http://127.0.0.1:8000/search/books/suggest/?tag_suggest__completion=bio
```

##### Response

```

{
  "_shards": {
    "failed": 0,
    "successful": 1,
    "total": 1
  },
  "tag_suggest__completion": [
    {
      "options": [
        {
          "score": 1.0,
          "text": "Biography"
        },
        {
          "score": 1.0,

```

(continues on next page)

(continued from previous page)

```

        "text": "Biology"
    }
],
"offset": 0,
"length": 2,
"text": "bio"
}
]
}

```

#### 12.7.1.10.1.7 Context suggesters

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

#### 12.7.1.10.1.8 category context

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

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

##### Document definition

```

class BookDocument(Document):

    # ...

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

```

(continues on next page)

(continued from previous page)

```
        '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
```

#### 12.7.1.10.1.9 geo context

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

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



## Document definition

```

class AddressDocument(Document):

    # ...

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

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

    # ...

```

ViewSet should altered as follows:

## ViewSet definition

```

class BookFrontendDocumentViewSet(DocumentViewSet):

    # ...

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

```

(continues on next page)

(continued from previous page)

```

        '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 within 8000km distance from geo-point (-30, -100).

```

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

```

Same query with boosting (boost value 2.0):

```

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

```

## 12.7.1.10.2 Term and Phrase suggestions

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

### 12.7.1.10.2.1 Document definition

*search\_indexes/documents/book.py*

```

from django.conf import settings

from django_elasticsearch_dsl import Document, Index, fields

from books.models import Book

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

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

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

    title = StringField(
        analyzer=html_strip,

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

    # Publication date
    publication_date = fields.DateField()

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

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

    # Price
    price = fields.FloatField()

    # Pages
    pages = fields.IntegerField()

    # Stock count
    stock_count = fields.IntegerField()

```

(continues on next page)

(continued from previous page)

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

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

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

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

### 12.7.1.10.2.2 ViewSet definition

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

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

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

URL shall be constructed in the following way:

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

Example for completion suggerster:

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

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

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

Example for term suggerster:

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

Example for phrase suggerster:

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

*search\_indexes/viewsets/book.py*

```

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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.

#### 12.7.1.10.2.3 Sample requests/responses

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

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

```

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

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

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

```

#### 12.7.1.10.2.4 Completion

##### Request

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

##### Response

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

(continues on next page)



(continued from previous page)

```

    },
    },
    {
        "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": "canelled",
            "authors": [
                "Juliette Estey",
                "Keano de Keijzer",
                "Koen Scheffers",
                "Florian van 't Erve",
                "Tara Oversteeg",
                "Mats van Nimwegen"
            ],
            "title": "Temporibus exercitationem minus expedita."
        }
    }
],
"offset": 0
}
]
}

```

#### 12.7.1.10.2.5 Term

##### Request

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

##### Response

```

{
  "_shards": {
    "failed": 0,
    "total": 1,
    "successful": 1
  },
  "summary_suggest__term": [
    {
      "text": "tovs",
      "offset": 0,
      "options": [
        {
          "text": "tove",
          "score": 0.75,
          "freq": 1
        }
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```
        },
        {
            "text": "took",
            "score": 0.5,
            "freq": 1
        },
        {
            "text": "twas",
            "score": 0.5,
            "freq": 1
        }
    ],
    "length": 5
}
]
```

#### 12.7.1.10.2.6 Phrase

##### Request

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

##### Response

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

#### 12.7.1.11 Functional suggestions

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

Configuration is very similar to native suggesters.

### 12.7.1.11.1 Document definition

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

The following example indicates Ngram analyzer/filter usage.

*search\_indexes/documents/book.py*

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

from elasticsearch_dsl import analyzer
from elasticsearch_dsl.analysis import token_filter

from books.models import Book

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

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

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

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

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

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

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

    title = StringField(
        analyzer=html_strip,
```

(continues on next page)

(continued from previous page)

```

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

    # ...

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

    model = Book  # The model associate with this Document

```

### 12.7.1.11.2 ViewSet definition

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

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

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

URL shall be constructed in the following way:

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

Example for `completion_prefix` suggerer:

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

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

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

Example for `completion_match` suggerer:

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

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

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

*search\_indexes/viewsets/book.py*

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

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

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

    # ...

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

---

**Note:** Note, that in `functional_suggester_fields['title_suggest']['options']` there are two params: `size` and `from`. They control the query size and the offset of the generated functional suggest query.

---

### 12.7.1.12 Highlighting

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

#### ViewSet definition

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

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

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

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

    # ...

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

    # ...
```

#### Request

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

#### Response

```

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

```

### 12.7.1.13 Pagination

#### 12.7.1.13.1 Page number pagination

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

Example:

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

#### 12.7.1.13.2 Limit/offset pagination

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

*search\_indexes/viewsets/book.py*

```
# ...

from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination

# ...

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

    # ...

    pagination_class = LimitOffsetPagination

    # ...
```

Example:

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

#### 12.7.1.13.3 Customisations

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

```
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination

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

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

(continues on next page)



(continued from previous page)

```

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

return sorted(__data)

```

Same applies to the customisations of the `LimitOffsetPagination`.

## 12.8 Nested fields usage examples

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

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

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

Contents:

### Table of Contents

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

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

## 12.8.1 Example app

### 12.8.1.1 Sample models

*books/models/continent.py*

```
from django.db import models

from six import python_2_unicode_compatible

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

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

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

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

(continues on next page)

(continued from previous page)

```

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

```

*books/models/country.py*

```

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

    name = models.CharField(max_length=255)
    info = models.TextField(null=True, blank=True)
    continent = models.ForeignKey(
        'books.Continent',
        on_delete=models.CASCADE
    )
    latitude = models.DecimalField(
        null=True,
        blank=True,
        decimal_places=15,
        max_digits=19,
        default=0
    )
    longitude = models.DecimalField(
        null=True,
        blank=True,
        decimal_places=15,
        max_digits=19,
        default=0
    )

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

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

```

*books/models/city.py*

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

    ordering = ["id"]

    def __str__(self):
        return self.name

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

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

```

*books/models/address.py*

```

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

from six import python_2_unicode_compatible

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

        ordering = ["id"]

    def __str__(self):
        return "{} {} {} {}".format(
            self.street,
            self.house_number,
            self.appendix,
            self.zip_code
        )

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

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

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

        Example:

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

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

```

(continues on next page)

(continued from previous page)

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

    return wrapper

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

    Example:

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

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

    return wrapper
```

### 12.8.1.2 Sample document

#### 12.8.1.2.1 Index definition

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

##### 12.8.1.2.1.1 Settings

*settings/base.py*

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

*settings/testing.py*

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

*settings/production.py*

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

### 12.8.1.2.1.2 Document index

*search\_indexes/documents/address.py*

```
from django.conf import settings

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

from books.models import Address

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

@INDEX.doc_type
class AddressDocument(Document):
    """Address Elasticsearch document."""

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

house_number = StringField(analyzer=html_strip)

appendix = StringField(analyzer=html_strip)

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

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

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

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

```

(continues on next page)



(continued from previous page)

```

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

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

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

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

    model = Address # The model associate with this Document

```

### 12.8.1.3 Sample serializer

*search\_indexes/serializers/address.py*

```
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from ..documents import AddressDocument


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

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

        document = AddressDocument
        fields = (
            'id',
            'street',
            'house_number',
            'appendix',
            'zip_code',
            'city',
            'country',
            'continent',
            'location',
        )
```

### 12.8.1.4 Sample view

*search\_indexes/viewsets/address.py*

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
    LOOKUP_FILTER_GEO_POLYGON,
    LOOKUP_FILTER_GEO_BOUNDING_BOX,
    SUGGESTER_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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


class AddressDocumentViewSet(DocumentViewSet):
```

(continues on next page)

(continued from previous page)

```

"""The AddressDocument view."""

document = AddressDocument
serializer_class = AddressDocumentSerializer
lookup_field = 'id'
filter_backends = [
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    DefaultOrderingFilterBackend,
    SuggesterFilterBackend,
]
pagination_class = LimitOffsetPagination
# Define search fields
search_fields = (
    'street',
    'zip_code',
    'city.name',
    'city.country.name',
)
# Define filtering fields
filter_fields = {
    'id': None,
    'city': 'city.name.raw',
}
# Nested filtering fields
nested_filter_fields = {
    'continent_country': {
        'field': 'continent.country.name.raw',
        'path': 'continent.country',
    },
    'continent_country_city': {
        'field': 'continent.country.city.name.raw',
        'path': 'continent.country.city',
    },
}
# Define geo-spatial filtering fields
geo_spatial_filter_fields = {
    'location': {
        'lookups': [
            LOOKUP_FILTER_GEO_BOUNDING_BOX,
            LOOKUP_FILTER_GEO_DISTANCE,
            LOOKUP_FILTER_GEO_POLYGON,
        ],
    },
}
# Define ordering fields
ordering_fields = {
    'id': None,
    'street': None,
    'city': 'city.name.raw',
    'country': 'city.country.name.raw',
}

```

(continues on next page)

(continued from previous page)

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

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

#### 12.8.1.5 Usage example

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

### 12.8.1.5.1 Sample queries

#### 12.8.1.5.1.1 Search

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

##### Search in all fields

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

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

##### Search a single term on specific field

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

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

#### 12.8.1.5.1.2 Nested filtering

##### Filter documents by nested field

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

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

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

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

#### 12.8.1.5.1.3 Nested search

For nested search, let's have another example.

#### 12.8.1.5.1.4 Sample models

*books/models/city.py*

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

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

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

(continues on next page)

(continued from previous page)

```

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

```

*books/models/country.py*

```

from django.db import models
from six import python_2_unicode_compatible

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

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

```

### 12.8.1.5.1.5 Sample document

*documents/city.py*

```

from django.conf import settings

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

from books.models import City

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

@INDEX.doc_type
class CityDocument(Document):
    """City Elasticsearch document.

```

(continues on next page)

(continued from previous page)

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

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

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

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

info = StringField(analyzer=html_strip)

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

# City object
country = fields.NestedField(
    properties={
        'name': StringField(
            analyzer=html_strip,
            fields={
                'raw': KeywordField(),
                'suggest': fields.CompletionField(),
            }
        ),
        'info': StringField(analyzer=html_strip),
        'location': fields.GeoPointField(attr='location_field_indexing'),
    }
)

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
)

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

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

### 12.8.1.5.1.6 Sample view

*viewsets/city.py*

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
    LOOKUP_FILTER_GEO_POLYGON,
    LOOKUP_FILTER_GEO_BOUNDING_BOX,
    SUGGESTER_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    DefaultOrderingFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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

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

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

    search_nested_fields = {
```

(continues on next page)



(continued from previous page)

```

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

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

    # Define geo-spatial filtering fields
    geo_spatial_filter_fields = {
        'location': {
            'lookups': [
                LOOKUP_FILTER_GEO_BOUNDING_BOX,
                LOOKUP_FILTER_GEO_DISTANCE,
                LOOKUP_FILTER_GEO_POLYGON,
            ],
        },
    }

    # Define ordering fields
    ordering_fields = {
        'id': None,
        'name': None,
        'country': 'country.name.raw',
    }

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

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

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

```

#### 12.8.1.5.1.7 Sample request

##### Request

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

#### 12.8.1.5.1.8 Filtering

##### Filter documents by field

Filter documents by field (`city`) “Dublin”.

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

##### Filter documents by multiple fields

Filter documents by field (`states`) “published” and “in\_progress”.

```
http://127.0.0.1:8000/search/addresses/?city__in=Yerevan__Dublin
```

#### 12.8.1.5.1.9 Ordering

The `-` prefix means ordering should be descending.

##### Order documents by field (descending)

Order documents by field `country` (ascending).

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

#### 12.8.1.6 Suggestions

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

---

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

---

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

---

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

---

Suggest completion for field `country`.

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

Suggest completion for field `city`.

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

### 12.8.1.7 Nested aggregations/facets

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

*search\_indexes/backends/nested\_continents.py*

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

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

    faceted_search_param = 'nested_facet'

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

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

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

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

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

(continues on next page)

(continued from previous page)

```

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

    return queryset

```

The view will look as follows:

*search\_indexes/viewsets/address.py*

```

from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
    LOOKUP_FILTER_GEO_POLYGON,
    LOOKUP_FILTER_GEO_BOUNDING_BOX,
    SUGGESTER_COMPLETION,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    DefaultOrderingFilterBackend,
    FacetedSearchFilterBackend,
    FilteringFilterBackend,
    GeoSpatialFilteringFilterBackend,
    GeoSpatialOrderingFilterBackend,
    NestedFilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
    SuggesterFilterBackend,
)
from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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

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

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

```

(continues on next page)

(continued from previous page)

```

pagination_class = LimitOffsetPagination
# Define search fields
search_fields = (
    'street',
    'zip_code',
    'city.name',
    'city.country.name',
)
# Define filtering fields
filter_fields = {
    'id': None,
    'city': 'city.name.raw',
}
# Nested filtering fields
nested_filter_fields = {
    'continent_country': {
        'field': 'continent.country.name.raw',
        'path': 'continent.country',
    },
    'continent_country_city': {
        'field': 'continent.country.city.name.raw',
        'path': 'continent.country.city',
    },
}
# Define geo-spatial filtering fields
geo_spatial_filter_fields = {
    'location': {
        'lookups': [
            LOOKUP_FILTER_GEO_BOUNDING_BOX,
            LOOKUP_FILTER_GEO_DISTANCE,
            LOOKUP_FILTER_GEO_POLYGON,
        ],
    },
}
# Define ordering fields
ordering_fields = {
    'id': None,
    'street': None,
    'city': 'city.name.raw',
    'country': 'city.country.name.raw',
    'zip_code': None,
}
# Define ordering fields
geo_spatial_ordering_fields = {
    'location': None,
}
# Specify default ordering
ordering = (
    'id',
    'street.raw',
    'city.name.raw',
)
# Suggester fields
suggester_fields = {
    'street_suggest': {
        'field': 'street.suggest',

```

(continues on next page)

(continued from previous page)

```

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

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

```

## 12.9 More like this

More like this functionality.

### 12.9.1 Usage example

#### 12.9.1.1 Sample document

```

from django.conf import settings

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

from books.models import Book

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

```

(continues on next page)

(continued from previous page)

```

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

@INDEX.doc_type
class BookDocument(Document):

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

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

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

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

    # ...

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

        model = Book  # The model associate with this Document

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

```

### 12.9.1.2 Sample view

```

from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,

```

(continues on next page)

(continued from previous page)

```

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

    from .serializers import BookDocumentSerializer

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

        # ...

        document = BookDocument
        lookup_field = 'id'
        serializer_class = BookDocumentSerializer

        # ...

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

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

```

### 12.9.1.3 Sample request

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

### 12.9.1.4 Generated query

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

```

(continues on next page)



(continued from previous page)

```

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

```

### 12.9.1.5 Options

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

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

## 12.10 Global aggregations

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

### 12.10.1 Sample view

```

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

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookCompoundSearchBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

```

(continues on next page)

(continued from previous page)

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

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

### 12.10.2 Sample request

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

### 12.10.3 Generated query

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

```

(continues on next page)

(continued from previous page)

```

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

```

### 12.10.4 Sample response

```

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

```

(continues on next page)

(continued from previous page)

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

## 12.11 Configuration tweaks

### 12.11.1 Ignore certain Elasticsearch exceptions

```
class BookIgnoreIndexErrorsDocumentViewSet(DocumentViewSet):

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

## 12.12 Source filtering backend

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

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

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

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

# Local article document definition
from .documents import ArticleDocument

# Local article document serializer
from .serializers import ArticleDocumentSerializer
```

(continues on next page)

(continued from previous page)

```
class ArticleDocumentView(BaseDocumentViewSet):

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

To disable `_source` retrieval set to False:

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

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

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

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

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

---

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

---

## 12.13 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):
```

(continues on next page)

(continued from previous page)

```

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

(continues on next page)



(continued from previous page)

```
model = Location
queryset_pagination = 50
```

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

*settings/base.py*

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

*settings/indexing.py*

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

*yourapp/documents.py*

```
from django.conf import settings

# ...

class LocationDocument(Document):

    # ...

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

        model = Location
        queryset_pagination = settings.ELASTICSEARCH_DSL_QUERYSET_PAGINATION
```

## 12.15 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/stable/
↳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):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).

- All backwards incompatible changes are mentioned in this document.

## 12.18.1 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*.

## 12.18.2 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.

## 12.18.3 0.20.8

2020-04-10

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

## 12.18.4 0.20.7

2020-04-10

- Make `skip_duplicates` available for native suggester.

## 12.18.5 0.20.6

2020-04-04

- Basic implementation of RegExp field lookup.

### **12.18.6 0.20.5**

2019-12-30

- Minor fixes.

### **12.18.7 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.

### **12.18.8 0.20.3**

2019-09-20

- Testing the auxiliary versions module.

### **12.18.9 0.20.2**

2019-08-30

- Minor improvements in test coverage.

### **12.18.10 0.20.1**

2019-08-18

- Minor Elasticsearch 7.x compatibility fixes.

### **12.18.11 0.20**

2019-08-17

- Adding Elasticsearch 7.x support.

### **12.18.12 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.

### 12.18.13 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.

### 12.18.14 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.

### 12.18.15 0.17.6

2019-04-08

- Minor fixes.
- Additions to the docs.

### 12.18.16 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.

### 12.18.17 0.17.4

2019-03-13

- Source backend.

### **12.18.18 0.17.3**

2019-02-08

- Obey object permissions.

### **12.18.19 0.17.2**

2019-01-07

- Add nested ordering.

### **12.18.20 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.

### **12.18.21 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.

### **12.18.22 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.

### **12.18.23 0.16.2**

2018-09-21

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



### 12.18.24 0.16.1

2018-09-18

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

### 12.18.25 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.

### 12.18.26 0.15.1

2018-08-22

- More tests.
- Fixes in docs.

### 12.18.27 0.15

2018-08-10

- Global aggregations.

### 12.18.28 0.14

2018-08-06

- More like this support through detail action.

### 12.18.29 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).

### 12.18.30 0.13.1

2018-07-26

- Minor fix in suggesters on Elasticsearch 6.x.

### 12.18.31 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.

### 12.18.32 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.

### 12.18.33 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.

### 12.18.34 0.10

2018-07-06

- Elasticsearch 6.x support.
- Minor fixes.

### 12.18.35 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`.

### 12.18.36 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.

### 12.18.37 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`.

### 12.18.38 0.8.2

2018-06-05

- Minor fixes.

### 12.18.39 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.

### 12.18.40 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 re-named 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.

### 12.18.41 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.

### 12.18.42 0.7.1

2018-04-04

---

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

---

- Add query *boost* support for search fields.

### **12.18.43 0.7**

2018-03-08

---

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

---

- CoreAPI/CoreSchema support.

### **12.18.44 0.6.4**

2018-03-05

- Minor fix: explicitly use DocType in the ViewSets.

### **12.18.45 0.6.3**

2018-01-03

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

### **12.18.46 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.

### **12.18.47 0.6.1**

2017-11-28

- Documentation fixes.

### **12.18.48 0.6**

2017-11-28

- Added highlight backend.
- Added nested search functionality.

### **12.18.49 0.5.1**

2017-10-18

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

### 12.18.50 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.

### 12.18.51 0.4.4

2017-10-02

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

### 12.18.52 0.4.3

2017-09-28

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

### 12.18.53 0.4.2

2017-09-28

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

### 12.18.54 0.4.1

2017-09-26

- Fixes in docs.

## 12.18.55 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).

## 12.18.56 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.

## 12.18.57 0.3.11

2017-09-21

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

## 12.18.58 0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

## 12.18.59 0.3.9

2017-09-12

- Python 2.x compatibility fix.

## 12.18.60 0.3.8

2017-09-12

- Fixes tests on some environments.



### 12.18.61 0.3.7

2017-09-07

- Docs fixes.

### 12.18.62 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.

### 12.18.63 0.3.5

2017-08-24

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

### 12.18.64 0.3.4

2017-08-23

- Minor fixes in the ordering backend.

### 12.18.65 0.3.3

2017-07-13

- Minor fixes and improvements.

### 12.18.66 0.3.2

2017-07-12

- Minor fixes and improvements.

### 12.18.67 0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

### 12.18.68 0.3

2017-07-11

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

### 12.18.69 0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

### 12.18.70 0.2.5

2017-07-11

- Fixes in documentation.

### 12.18.71 0.2.4

2017-07-11

- Fixes in documentation.

### 12.18.72 0.2.3

2017-07-11

- Fixes in documentation.

### 12.18.73 0.2.2

2017-07-11

- Fixes in documentation.

### 12.18.74 0.2.1

2017-07-11

- Fixes in documentation.

### 12.18.75 0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

### 12.18.76 0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

### 12.18.77 0.1.7

2017-06-25

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

### 12.18.78 0.1.6

2017-06-23

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

### 12.18.79 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.

### 12.18.80 0.1.4

2017-06-22

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

### 12.18.81 0.1.3

2017-06-21

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

### 12.18.82 0.1.2

2017-06-20

- Minor fixes in tests.

### 12.18.83 0.1.1

2017-06-20

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

### 12.18.84 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(read_only=False,  
                                                             write_only=False,  
                                                             required=None,  
                                                             default=<class  
                                                             'rest_framework.fields.empty'>,  
                                                             initial=<class  
                                                             'rest_framework.fields.empty'>,  
                                                             source=None,  
                                                             label=None,  
                                                             help_text=None,  
                                                             style=None, error  
                                                             messages=None,  
                                                             valida-  
                                                             tors=None, al-  
                                                             low_null=False)
```

Bases: `rest_framework.fields.BooleanField`

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

```
class django_elasticsearch_dsl_drf.fields.common.CharField(**kwargs)
```

Bases: `rest_framework.fields.CharField`

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

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

Bases: `rest_framework.fields.DateField`

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

```
class django_elasticsearch_dsl_drf.fields.common.FloatField (**kwargs)
    Bases: rest_framework.fields.FloatField
```

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

```
class django_elasticsearch_dsl_drf.fields.common.IntegerField (**kwargs)
    Bases: rest_framework.fields.IntegerField
```

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

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

Bases: `rest_framework.fields.IPAddressField`

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

### 12.19.1.1.3 `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(read_only=False,
                                                                    write_only=False,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    val-
                                                                    ida-
                                                                    tors=None,
                                                                    al-
                                                                    low_null=False)
```

Bases: *django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*

Geo point field.

```
class django_elasticsearch_dsl_drf.fields.nested_fields.GeoShapeField(read_only=False,
                                                                    write_only=False,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    val-
                                                                    ida-
                                                                    tors=None,
                                                                    al-
                                                                    low_null=False)
```

Bases: *django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*

Geo shape field.

```
class django_elasticsearch_dsl_drf.fields.nested_fields.NestedField(read_only=False,
                                                                    write_only=False,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    valida-
                                                                    tors=None,
                                                                    al-
                                                                    low_null=False)
```

Bases: `django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField`

Nested field.

```
class django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField(read_only=False,
                                                                    write_only=False,
                                                                    re-
                                                                    quired=None,
                                                                    de-
                                                                    fault=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    ini-
                                                                    tial=<class
                                                                    'rest_framework.fields.empty'>,
                                                                    source=None,
                                                                    la-
                                                                    bel=None,
                                                                    help_text=None,
                                                                    style=None,
                                                                    er-
                                                                    ror_messages=None,
                                                                    valida-
                                                                    tors=None,
                                                                    al-
                                                                    low_null=False)
```

Bases: `rest_framework.fields.Field`

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_internal\_value** (*data*)  
To internal value.

**to\_representation** (*value*)

To representation.

```
class django_elasticsearch_dsl_drf.fields.nested_fields.ListField(read_only=False,
                                                                write_only=False,
                                                                re-
                                                                quired=None,
                                                                de-
                                                                fault=<class
                                                                'rest_framework.fields.empty'>,
                                                                ini-
                                                                tial=<class
                                                                'rest_framework.fields.empty'>,
                                                                source=None,
                                                                la-
                                                                bel=None,
                                                                help_text=None,
                                                                style=None,
                                                                er-
                                                                ror_messages=None,
                                                                valida-
                                                                tors=None,
                                                                al-
                                                                low_null=False)
```

Bases: `rest_framework.fields.Field`

List field.

**get\_value** (*dictionary*)  
Get value.

**to\_internal\_value** (*data*)  
To internal value.

**to\_representation** (*value*)  
To representation.

### 12.19.1.1.5 Module contents

Fields.

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

Bases: `rest_framework.fields.BooleanField`

Object field.



**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**CharField** (\*\**kwargs*)

Bases: rest\_framework.fields.CharField

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**DateField** (*format=<class*  
*'rest\_framework.fields.empty'>*,  
*input\_formats=None*, *\*args*,  
*\*\*kwargs*)

Bases: rest\_framework.fields.DateField

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**FloatField** (\*\**kwargs*)

Bases: rest\_framework.fields.FloatField

Object field.

**get\_value** (*dictionary*)  
Get value.

**to\_representation** (*value*)  
To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**GeoPointField** (*read\_only=False*,  
*write\_only=False*,  
*required=None*,  
*default=<class*  
*'rest\_framework.fields.empty'>*,  
*initial=<class*  
*'rest\_framework.fields.empty'>*,  
*source=None*,  
*label=None*,  
*help\_text=None*,  
*style=None*, *er-*  
*ror\_messages=None*,  
*validators=None*, *al-*  
*low\_null=False*)

Bases: *django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*

Geo point field.

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

Bases: django_elasticsearch_dsl_drf.fields.nested_fields.ObjectField
```

Geo shape field.

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

Object field.

```
get_value (dictionary)
    Get value.
```

```
to_representation (value)
    To representation.
```

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

Bases: rest\_framework.fields.IPAddressField

Object field.

```
get_value (dictionary)
    Get value.
```

```
to_representation (value)
    To representation.
```

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

Bases: rest\_framework.fields.Field

List field.

```
get_value (dictionary)
    Get value.
```

```
to_internal_value (data)
    To internal value.
```

**to\_representation** (*value*)

To representation.

```
class django_elasticsearch_dsl_drf.fields.NestedField(read_only=False,
                                                    write_only=False,           re-
                                                    quired=None, default=<class
                                                    'rest_framework.fields.empty'>,
                                                    initial=<class
                                                    'rest_framework.fields.empty'>,
                                                    source=None,   label=None,
                                                    help_text=None, style=None,
                                                    error_messages=None,
                                                    validators=None,           al-
                                                    low_null=False)
```

Bases: *django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*

Nested field.

```
class django_elasticsearch_dsl_drf.fields.ObjectField(read_only=False,
                                                    write_only=False,           re-
                                                    quired=None, default=<class
                                                    'rest_framework.fields.empty'>,
                                                    initial=<class
                                                    'rest_framework.fields.empty'>,
                                                    source=None,   label=None,
                                                    help_text=None, style=None,
                                                    error_messages=None,
                                                    validators=None,           al-
                                                    low_null=False)
```

Bases: *rest\_framework.fields.Field*

Object field.

**get\_value** (*dictionary*)

Get value.

**to\_internal\_value** (*data*)

To internal value.

**to\_representation** (*value*)

To representation.

## 12.19.1.2 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
>>>
>>> # 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 = [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}* /endpoint/?field_name__wildcard={value}
```

Example:

[http://localhost:8000/api/articles/?tags\\_\\_wildcard=child\\*](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}
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.**GeoSpatialFiltering**

Bases: *rest\_framework.filters.BaseFilterBackend*, *django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin*

Geo-spatial filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_GEO_DISTANCE,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
```

(continues on next page)

(continued from previous page)

```
>>> GeoSpatialFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialFilteringFilterBackend,]
>>>     geo_spatial_filter_fields = {
>>>         'loc': 'location',
>>>         'location': {
>>>             'field': 'location',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_GEO_DISTANCE,
>>>             ],
>>>         }
>>> }
```

**classmethod `apply_query_geo_bounding_box`** (*queryset, options, value*)  
Apply *geo\_bounding\_box* query.

#### Parameters

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

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod `apply_query_geo_distance`** (*queryset, options, value*)  
Apply *geo\_distance* query.

#### Parameters

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

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod `apply_query_geo_polygon`** (*queryset, options, value*)  
Apply *geo\_polygon* query.

#### Parameters

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.

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

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**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:

```
{
```



```

        "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__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 (

```

(continues on next page)

(continued from previous page)

```
>>> 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.PostFilterFilteringBackend`

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

The `post_filter` filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
```

(continues on next page)

(continued from previous page)

```
>>> )
>>> 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
>>> 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](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__2.0) [http://localhost:8000/api/users/?age\\_\\_range=16\\_\\_67](http://localhost:8000/api/users/?age__range=16__67) [http://localhost:8000/api/users/?age\\_\\_range=16](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=1[{}6-9]) [http://localhost:8000/api/users/?age\\_\\_regexp=2.\\*](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 query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** **apply\_filter\_terms** (*queryset, options, value*)

Apply *terms* filter.

Syntax:

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

Note, that number of values is not limited.

Example:

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

#### Parameters

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

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** **apply\_query\_contains** (*queryset, options, value*)

Apply *contains* filter.

Syntax:

`/endpoint/?field_name__contains={value}`

Example:

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

#### Parameters

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

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** **apply\_query\_endswith** (*queryset, options, value*)

Apply *endswith* filter.

Syntax:

`/endpoint/?field_name__endswith={value}`

Example:

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

#### Parameters



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

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** **apply\_query\_exclude** (*queryset, options, value*)

Apply *exclude* functional query.

Syntax:

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

Note, that number of values is not limited.

Example:

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

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **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 query-set.
- **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 query-set.
- **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 query-set.
- **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 query-set.
- **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 query-set.

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

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

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

Apply *geo\_distance* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original 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):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [IdsFilterBackend]
    
```

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ids\_query\_params** (*request*)

Get search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**get\_ids\_values** (*request, view*)

Get ids values for query.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**ids\_query\_param** = 'ids'

**class** *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.NestedFilteringFilterBackend*

Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.*

*FilteringFilterBackend*

Nested filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     NestedFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
```

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

```

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

Apply filter.

#### Parameters

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

#### Returns

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

Apply query.

#### Parameters

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

#### Returns

**get\_coreschema\_field** (*field*)

**get\_filter\_field\_nested\_path** (*filter\_fields, field\_name*)

Get filter field path to be used in nested query.

#### Parameters

- **filter\_fields** –
- **field\_name** –

#### Returns

**get\_filter\_query\_params** (*request*, *view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**get\_schema\_fields** (*view*)

**classmethod prepare\_filter\_fields** (*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**class** `django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilterFilteringFilterBackend`

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

The post\_filter filtering filter backend for Elasticsearch.

Example:

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

(continues on next page)

(continued from previous page)

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

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod get\_default\_ordering\_params** (*view*)

Get the default ordering params for the view.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**get\_ordering\_query\_params** (*request*, *view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**ordering\_param** = 'ordering'

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.**OrderingFilterBackend**

Bases: *rest\_framework.filters.BaseFilterBackend*, *django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.OrderingMixin*

Ordering filter backend for Elasticsearch.

Example:

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

(continues on next page)

(continued from previous page)

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

(continues on next page)



(continued from previous page)

```
>>> 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):
>>>
```

(continues on next page)

(continued from previous page)

```

>>> 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,]
>>>     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
```

(continues on next page)

(continued from previous page)

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

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ordering\_query\_params** (*request, view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** *list*

**get\_schema\_fields** (*view*)

**ordering\_param** = 'ordering'

#### 12.19.1.2.1.20 `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`

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.MultiMatch`

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

Multi match query backend.

**classmethod** **construct\_search** (*request, view, search\_backend*)

Construct search.

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

Example:

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

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

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



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

Example 1 (complex):

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

Example 2 (simple list):

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

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**classmethod** **get\_field** (*field, options*)

Get field.

#### Parameters

- **field** –
- **options** –

#### Returns

**classmethod** **get\_query\_options** (*request, view, search\_backend*)

**query\_type** = 'multi\_match'

### 12.19.1.2.1.29 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backends.nested module

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

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

Nested query backend.

**classmethod** **construct\_search** (*request, view, search\_backend*)

Construct search.

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

Type 1:

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

Type 2:

```
search_nested_fields = {
```

```

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

```

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**query\_type** = 'nested'

### 12.19.1.2.1.30 `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`

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

Match query backend.

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

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

```
query_type = 'match'
```

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

Match phrase query backend.

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

Construct search.

**Parameters**

- **request** –
- **view** –
- **search\_backend** –

**Returns**

```
query_type = 'match_phrase'
```

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

Match phrase prefix query backend.

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

Construct search.

**Parameters**

- **request** –
- **view** –
- **search\_backend** –

**Returns**

```
query_type = 'match_phrase_prefix'
```

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

Multi match query backend.

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

Construct search.

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

Example:

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

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

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

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

Example 1 (complex):

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

Example 2 (simple list):

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

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

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

Get field.

#### Parameters

- **field** –
- **options** –

#### Returns

**classmethod** `get_query_options` (*request*, *view*, *search\_backend*)

**query\_type** = 'multi\_match'

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

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

Nested query backend.

**classmethod** `construct_search` (*request*, *view*, *search\_backend*)

Construct search.

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

Type 1:

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

Type 2:

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

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**query\_type** = 'nested'

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.SimpleQueryString`

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

Simple query string query backend.

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

Construct search.

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

Example:

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

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

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

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

Example 1 (complex):

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

Example 2 (simple list):

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

Query examples:

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

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

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

Get field.

**Parameters**

- **field** –
- **options** –

**Returns**

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

`query_type = 'simple_query_string'`

### 12.19.1.2.1.32 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_base.QueryBackend']
```

#### 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.MultiMatchSearchFilter*

Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchFilterBackend*

Multi match search filter backend.

**matching** = 'must'

**query\_backends** = [*<class 'django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_base.QueryBaseSearchFilterBackend'>*]

**search\_param** = 'search\_multi\_match'

#### 12.19.1.2.1.37 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.SimpleQueryST
    Bases: django_elasticsearch_dsl_drf.filter_backends.search.base.
           BaseSearchFilterBackend

    Simple query string search filter backend.

    matching = 'must'

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

#### 12.19.1.2.1.39 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_ba

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

matching = 'must'

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

search_param = 'search_multi_match'

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

```

Search filter backend for Elasticsearch.

Example:

```

>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SearchFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [SearchFilterBackend,]
>>>     search_fields = (
>>>         'title',
>>>         'content',
>>>     )
>>>     search_nested_fields = {
>>>         'state': ['name'],
>>>         'documents.author': ['title', 'description'],
>>>     }

```

**construct\_nested\_search**(*request, view*)

Construct nested search.

We have to deal with two types of structures:

Type 1:

```

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

```

(continues on next page)

(continued from previous page)

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

Type 2:

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

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.**Return type** *elasticsearch\_dsl.search.Search***construct\_search** (*request, view*)

Construct search.

We have to deal with two types of structures:

Type 1:

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

Type 2:

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

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.

- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.  
**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset** (*request, queryset, view*)  
 Filter the queryset.  
**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.  
**Return type** *elasticsearch\_dsl.search.Search*

**get\_coreschema\_field** (*field*)

**get\_schema\_fields** (*view*)

**get\_search\_query\_params** (*request*)  
 Get search query params.  
**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.  
**Returns** List of search query params.  
**Return type** list

**search\_param** = 'search'

**class** *django\_elasticsearch\_dsl\_drf.filter\_backends.search.SimpleQueryStringSearchFilterBackend*  
 Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchFilterBackend*  
 Simple query string search filter backend.

**matching** = 'must'

**query\_backends** = [*<class 'django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_base.QueryBackend'>*]

**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(),
>>>         }
>>>     )
>>>
>>>     state_province = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     country = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     website = fields.StringField()
>>>
>>>     class Meta(object):

```

(continues on next page)

(continued from previous page)

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

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional.**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):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         FunctionalSuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     functional_suggester_fields = {
>>>         'name_suggest': {
>>>             'field': 'name.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>         'city_suggest': {
>>>             'field': 'city.suggest',
>>>             'suggesters': [
>>>                 FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>             ],
>>>         },
>>>     }
```

(continues on next page)



(continued from previous page)

```

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

```

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

Apply query size.

**Parameters**

- **queryset** –
- **options** –

**Returns**

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

Apply *completion* suggester match.

This is effective when used with Ngram fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

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

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**clean\_queryset(queryset)**

Clean the queryset.

- Remove aggregations.

- Remove highlight.
- Remove sorting options.

**Parameters** `queryset` –

**Returns**

**extract\_field\_name** (*field\_name*)

Extract field name.

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

**Parameters** `field_name` –

**Returns**

**Return type** `str`

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.

- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.

- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**get\_suggester\_query\_params** (*request, view*)

Get query params to be for suggestions.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.

- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** `dict`

**classmethod prepare\_suggester\_fields** (*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** `dict`

**serialize\_queryset** (*queryset, suggester\_name, value, serializer\_field*)

Serialize queryset.

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

**Parameters**

- **queryset** –

- **suggester\_name** –

- **value** –

- **serializer\_field** –

**Returns**

#### 12.19.1.2.1.43 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={
>>>             'raw': fields.StringField(analyzer='keyword')
>>>         }
>>>     )
>>>
>>>     city = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     state_province = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     country = fields.StringField(
>>>         fields={
```

(continues on next page)

(continued from previous page)

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

```
class django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterFilterBackend
Bases: rest_framework.filters.BaseFilterBackend, django_elasticsearch_dsl_drf.
filter_backends.mixins.FilterBackendMixin
```

Suggester filter backend for Elasticsearch.

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

Example:

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

(continues on next page)

(continued from previous page)

```

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

```

**classmethod `apply_suggester_completion`** (*suggester\_name, queryset, options, value*)

Apply *completion* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod `apply_suggester_phrase`** (*suggester\_name, queryset, options, value*)

Apply *phrase* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod `apply_suggester_term`** (*suggester\_name, queryset, options, value*)

Apply *term* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.

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

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

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

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** **get\_suggester\_context** (*field, suggester\_name, request, view*)

Get suggester context.

Given the following definition (in ViewSets):

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

[http://localhost:8000/search/books-frontend/suggest/?title\\_suggest\\_context=M](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”, “Children’s”). Simply put it’s treated as *prefix*, rather than *term*.

```

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

```

Sample query for *category* filter:

```

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

```

The query params would be:

```

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

```

Sample query for *geo* filter:

```

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

```

The query params would be:

```

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

```

### Returns

**get\_suggester\_query\_params** (*request*, *view*)

Get query params to be for suggestions.

### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod prepare\_suggester\_fields** (*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

## 12.19.1.2.1.44 Module contents

Suggester filtering backends.

```

class django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend
    Bases: rest_framework.filters.BaseFilterBackend, django_elasticsearch_dsl_drf.
    filter_backends.mixins.FilterBackendMixin

```

Suggester filter backend for Elasticsearch.

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

Example:

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

(continues on next page)



(continued from previous page)

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

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

Apply *completion* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

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

Apply *phrase* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

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

Apply *term* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original query-set.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `get_suggester_context` (*field, suggester\_name, request, view*)

Get suggester context.

Given the following definition (in ViewSets):

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

[http://localhost:8000/search/books-frontend/suggest/?title\\_suggest\\_context=M](http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M)

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

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

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

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

Sample query for *category* filter:

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

The query params would be:

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

Sample query for *geo* filter:

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

The query params would be:

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

### Returns

**get\_suggester\_query\_params** (*request*, *view*)

Get query params to be for suggestions.

### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod prepare\_suggester\_fields** (*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**class** `django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend`

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

Suggester filter backend for Elasticsearch.

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

Example:

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

(continues on next page)

(continued from previous page)

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

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

Apply query size.

**Parameters**

- **queryset** –
- **options** –

**Returns**

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

Apply *completion* suggester match.

This is effective when used with Ngram fields.

**Parameters**

- **suggester\_name** (*str*) –

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

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** **apply\_suggester\_completion\_prefix** (*suggester\_name, queryset, options, value*)

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**clean\_queryset** (*queryset*)

Clean the queryset.

- Remove aggregations.
- Remove highlight.
- Remove sorting options.

**Parameters** **queryset** –

**Returns**

**extract\_field\_name** (*field\_name*)

Extract field name.

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

**Parameters** **field\_name** –

**Returns**

**Return type** *str*

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_suggester\_query\_params** (*request, view*)

Get query params to be for suggestions.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod prepare\_suggester\_fields** (*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**serialize\_queryset** (*queryset, suggester\_name, value, serializer\_field*)

Serialize queryset.

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

**Parameters**

- **queryset** –
- **suggester\_name** –
- **value** –
- **serializer\_field** –

**Returns**

### 12.19.1.2.2 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.FacetedSearchFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`

Faceted search backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FacetedSearchFilterBackend
>>> )
>>> from elasticsearch_dsl import TermsFacet, DateHistogramFacet
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
```

(continues on next page)

(continued from previous page)

```

>>> serializer_class = ArticleDocumentSerializer
>>> filter_backends = [FacetedSearchFilterBackend,]
>>> faceted_search_fields = {
>>>     'title': 'title.raw', # Uses `TermsFacet` by default
>>>     'state': {
>>>         'field': 'state.raw',
>>>         'facet': TermsFacet,
>>>     },
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
>>>         'enabled': False,
>>>     },
>>>     'date_published': {
>>>         'field': 'date_published.raw',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }

```

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

**aggregate** (*request, queryset, view*)

Aggregate.

#### Parameters

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

#### Returns

**construct\_facets** (*request, view*)

Construct facets.

Turns the following structure:

```

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

```

Into the following structure:

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

**faceted\_search\_param** = 'facet'

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_faceted\_search\_query\_params** (*request*)

Get faceted search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**classmethod prepare\_faceted\_search\_fields** (*view*)

Prepare faceted search fields.

Prepares the following structure:

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

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Faceted search fields options.

**Return type** dict



#### 12.19.1.2.4 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 = {
>>>         'author.name': {
>>>             'enabled': False,
>>>             'options': {
>>>                 'fragment_size': 150,
>>>                 'number_of_fragments': 3
>>>             }
>>>         }
>>>         'title': {
>>>             'options': {
>>>                 'pre_tags' : ["<em>"],
>>>                 'post_tags' : ["</em>"]
>>>             },
>>>             'enabled': True,
>>>         },
>>>     }
```

Highlight make queries to be more heavy. That's why by default all highlights are disabled and enabled only explicitly either in the filter options (*enabled* set to True) or via query params *?highlight=author.name&highlight=title*.

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.

- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.  
**Return type** *elasticsearch\_dsl.search.Search*

**get\_highlight\_query\_params** (*request*)  
 Get highlight query params.  
**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.  
**Returns** List of search query params.  
**Return type** list

**highlight\_param** = 'highlight'

**classmethod prepare\_highlight\_fields** (*view*)  
 Prepare faceted search fields.  
 Prepares the following structure:

```
>>> {
>>>     'author.name': {
>>>         'enabled': False,
>>>         'options': {
>>>             'fragment_size': 150,
>>>             'number_of_fragments': 3
>>>         }
>>>     }
>>>     'title': {
>>>         'options': {
>>>             'pre_tags' : ["<em>"],
>>>             'post_tags' : ["</em>"]
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –  
**Returns** Highlight fields options.  
**Return type** dict

#### 12.19.1.2.5 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

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

Base tests.

**class** django\_elasticsearch\_dsl\_drf.tests.base.BaseRestFrameworkTestCase (*methodName='runTest'*)

Bases: django.test.testcases.TransactionTestCase, *django\_elasticsearch\_dsl\_drf.tests.base.SleepMixin*

Base REST framework test case.

**authenticate** ()

Helper for logging in Genre Coordinator user.

**Returns**

**pytestmark** = [Mark (name='django\_db', args=(), kwargs={}), Mark (name='django\_db', args=

**classmethod setUpClass** ()

Set up class.

**class** django\_elasticsearch\_dsl\_drf.tests.base.BaseTestCase (*methodName='runTest'*)

Bases: django.test.testcases.TransactionTestCase, *django\_elasticsearch\_dsl\_drf.tests.base.SleepMixin*

Base test case.

```

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

class django_elasticsearch_dsl_drf.tests.base.SleepMixin
    Bases: object

    Sleep mix-in.

    classmethod sleep(wait_for_index=2)
        Sleep for N seconds.
        Parameters wait_for_index –
        Returns

```

#### 12.19.1.4.3 django\_elasticsearch\_dsl\_drf.tests.data\_mixins module

Data mixins.

```

class django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Bases: object

    Addresses mixin.

    classmethod created_addresses()
        Create addresses.
        Returns

class django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin
    Bases: object

    Books mixin.

    classmethod create_books()
        Create books.
        Returns

```

#### 12.19.1.4.4 django\_elasticsearch\_dsl\_drf.tests.test\_faceted\_search module

Test faceted search backend.

```

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

    Test faceted search.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_list_results_with_facets()
        Test list results with facets.

```

#### 12.19.1.4.5 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common module

Test filtering backend.

```

class django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon (methodNames)
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
            django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin,
            django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin

    Test filtering common.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=())]

    classmethod setUpClass ()
        Set up.

    test_default_filter_lookup ()
        Test default filter lookup.

        Example:
        http://localhost:8000/search/books-default-filter-lookup/?authors=Robin&authors=Luc

    test_field_filter_contains ()
        Test filter contains.

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

    test_field_filter_endswith ()
        Test filter endswith.

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

    test_field_filter_exclude ()
        Test filter exclude.

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

    test_field_filter_exists_false ()
        Test filter exists.

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

    test_field_filter_exists_true ()
        Test filter exists true.

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

    test_field_filter_gt ()
        Field filter gt.

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

    Returns

    test_field_filter_gt_with_boost ()
        Field filter gt with boost.

        Example:
        http://localhost:8000/api/users/?id__gt=10__2.0

    Returns

```

**test\_field\_filter\_gte()**

Field filter gte.

Example:

[http://localhost:8000/api/users/?id\\_\\_gte=10](http://localhost:8000/api/users/?id__gte=10)

**Returns**

**test\_field\_filter\_in()**

Test filter in.

Example:

[http://localhost:8000/api/articles/?id\\_\\_in=1\\_\\_2\\_\\_3](http://localhost:8000/api/articles/?id__in=1__2__3)

**test\_field\_filter\_isnull\_false()**

Test filter isnull true.

Example:

[http://localhost:8000/api/articles/?tags\\_\\_isnull=false](http://localhost:8000/api/articles/?tags__isnull=false)

**test\_field\_filter\_isnull\_true()**

Test filter isnull true.

Example:

[http://localhost:8000/api/articles/?null\\_field\\_\\_isnull=true](http://localhost:8000/api/articles/?null_field__isnull=true)

**test\_field\_filter\_lt()**

Field filter lt.

Example:

[http://localhost:8000/api/users/?id\\_\\_lt=10](http://localhost:8000/api/users/?id__lt=10)

**Returns**

**test\_field\_filter\_lt\_with\_boost()**

Field filter lt with boost.

Example:

[http://localhost:8000/api/users/?id\\_\\_lt=10\\_\\_2.0](http://localhost:8000/api/users/?id__lt=10__2.0)

**Returns**

**test\_field\_filter\_lte()**

Field filter lte.

Example:

[http://localhost:8000/api/users/?id\\_\\_lte=10](http://localhost:8000/api/users/?id__lte=10)

**Returns**

**test\_field\_filter\_prefix()**

Test filter prefix.

Example:

[http://localhost:8000/api/articles/?tags\\_\\_prefix=bio](http://localhost:8000/api/articles/?tags__prefix=bio)

**test\_field\_filter\_range()**

Field filter range.

Example:

[http://localhost:8000/api/users/?age\\_\\_range=16\\_\\_67](http://localhost:8000/api/users/?age__range=16__67)

**test\_field\_filter\_range\_with\_boost()**

Field filter range.



Example:

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

**test\_field\_filter\_regexp()**

Field filter regexp.

Example:

`http://localhost:8000/api/users/?title__regexp=De.{8}Ins.*`

**test\_field\_filter\_term()**

Field filter term.

**test\_field\_filter\_term\_explicit()**

Field filter term.

**test\_field\_filter\_terms\_list()**

Test filter terms.

**test\_field\_filter\_terms\_string()**

Test filter terms.

Example:

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

**test\_field\_filter\_wildcard()**

Test filter wildcard.

Example:

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

**test\_ids\_empty\_filter()**

Test ids filter with empty value. This should not fail.

Example:

`http://localhost:8000/api/articles/?ids=`

**test\_ids\_filter()**

Test ids filter.

Example:

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

**test\_nested\_field\_filter\_term()**

Nested field filter term.

**test\_schema\_field\_not\_required()**

Test schema fields always not required

**test\_schema\_fields\_with\_filter\_fields\_list()**

Test schema field generator

**test\_various\_complex\_fields()**

Test various complex fields.

**Returns**

#### 12.19.1.4.6 `django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial` module

Test geo-spatial filtering backend.

**class** `django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test filtering geo-spatial.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUpClass())
```

Set up.

```
test_field_filter_geo_bounding_box()
```

Test field filter geo-bounding-box.

**Returns**

```
test_field_filter_geo_bounding_box_fail_test()
```

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

**Returns**

```
test_field_filter_geo_distance()
```

Field filter geo-distance.

Example:

```
http://localhost:8000/api/publisher/?location\_\_geo\_distance=1km\_\_48.8549\_\_2.3000
```

**Returns**

```
test_field_filter_geo_distance_distance_type_arc()
```

Field filter geo-distance.

Example:

```
http://localhost:8000/api/publisher/?location\_\_geo\_distance=1km\_\_48.8549\_\_2.3000\_\_arc
```

**Returns**

```
test_field_filter_geo_distance_not_enough_args_fail()
```

Field filter geo-distance. Fail test on not enough args.

Example:

```
http://localhost:8000/api/publisher/?location\_\_geo\_distance=1km\_\_48.8549
```

**Returns**

```
test_field_filter_geo_polygon()
```

Test field filter geo-polygon.

**Returns**

```
test_field_filter_geo_polygon_fail_test()
```

Test field filter geo-polygon (fail test).

**Returns**

```
test_field_filter_geo_polygon_string_options()
```

Test field filter geo-polygon.

**Returns**

```
test_field_filter_geo_polygon_string_options_fail_test()
```

Test field filter geo-polygon (fail test).

**Returns**

```
test_field_filter_geo_shape_circle()
```

Test field filter geo-shape.

**Returns**

```
test_field_filter_geo_shape_circle_intersects()
```

Test field filter geo-shape.

**Returns**

**test\_field\_filter\_geo\_shape\_envelope()**  
 Test field filter geo-shape.  
**Returns**

#### 12.19.1.4.7 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_global\_aggregations module

Test filtering *post\_filter* backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations.TestFilteringG
    Bases:      django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
               django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin,
               django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin

    Test filtering with global aggregations.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up.

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

#### 12.19.1.4.8 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested module

Test nested filtering backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested (methodNan
    Bases:      django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
               django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin

    Test filtering nested.

    base_url

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up.

    test_field_filter_contains()
        Test filter contains.

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

    test_field_filter_endswith()
        Test filter endswith.

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

    test_field_filter_exclude()
        Test filter exclude.

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

```

test_field_filter_exists_false()
    Test filter exists.

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

test_field_filter_exists_true()
    Test filter exists true.

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

test_field_filter_in()
    Test filter in.

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

test_field_filter_prefix()
    Test filter prefix.

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

test_field_filter_range()
    Field filter range.

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

test_field_filter_range_with_boost()
    Field filter range.

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

test_field_filter_term()
    Field filter term.

test_field_filter_term_explicit()
    Field filter term.

test_field_filter_terms_list()
    Test filter terms.

test_field_filter_terms_string()
    Test filter terms.

    Example:
        http://localhost:8000/api/articles/?id__terms=1__2__3

test_field_filter_wildcard()
    Test filter wildcard.

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

test_schema_field_not_required()
    Test schema fields always not required

test_schema_fields_with_filter_fields_list()
    Test schema field generator

```

#### 12.19.1.4.9 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter module

Test filtering *post\_filter* backend.

```
class django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter:
    Bases:      django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
               django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin,
               django_elasticsearch_dsl_drf.tests.data_mixins.BooksMixin
```

Test filtering *post\_filter*.

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

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

```
    test_field_filter_contains()
        Test filter contains.
```

Example:

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

```
    test_field_filter_endswith()
        Test filter endswith.
```

Example:

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

```
    test_field_filter_exclude()
        Test filter exclude.
```

Example:

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

```
    test_field_filter_exists_false()
        Test filter exists.
```

Example:

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

```
    test_field_filter_exists_true()
        Test filter exists true.
```

Example:

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

```
    test_field_filter_gt()
        Field filter gt.
```

Example:

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

**Returns**

```
    test_field_filter_gt_with_boost()
        Field filter gt with boost.
```

Example:

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

**Returns**

```
    test_field_filter_gte()
        Field filter gte.
```

Example:

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

#### Returns

**`test_field_filter_in()`**

Test filter in.

Example:

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

**`test_field_filter_isnull_false()`**

Test filter isnull true.

Example:

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

**`test_field_filter_isnull_true()`**

Test filter isnull true.

Example:

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

**`test_field_filter_lt()`**

Field filter lt.

Example:

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

#### Returns

**`test_field_filter_lt_with_boost()`**

Field filter lt with boost.

Example:

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

#### Returns

**`test_field_filter_lte()`**

Field filter lte.

Example:

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

#### Returns

**`test_field_filter_prefix()`**

Test filter prefix.

Example:

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

**`test_field_filter_range()`**

Field filter range.

Example:

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

**`test_field_filter_range_with_boost()`**

Field filter range.

Example:

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

**test\_field\_filter\_term()**

Field filter term.

**test\_field\_filter\_term\_explicit()**

Field filter term.

**test\_field\_filter\_terms\_list()**

Test filter terms.

**test\_field\_filter\_terms\_string()**

Test filter terms.

Example:

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

**test\_field\_filter\_wildcard()**

Test filter wildcard.

Example:

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

**test\_ids\_filter()**

Test ids filter.

Example:

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

**test\_list\_results\_with\_facets()**

Test list results with facets.

**test\_schema\_field\_not\_required()**

Test schema fields always not required

**test\_schema\_fields\_with\_filter\_fields\_list()**

Test schema field generator

**test\_various\_complex\_fields()**

Test various complex fields.

**Returns**

#### 12.19.1.4.10 `django_elasticsearch_dsl_drf.tests.test_functional_suggesters` module

Test functional suggestions backend.

**class** `django_elasticsearch_dsl_drf.tests.test_functional_suggesters.TestFunctionalSuggesters`

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`,  
`django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin`

Test functional suggesters.

**pytestmark** = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

**classmethod** `setUpClass()`

Set up class.

**test\_suggesters\_completion()**

Test suggesters completion.

**test\_suggesters\_completion\_no\_args\_provided()**

Test suggesters completion with no args provided.

#### 12.19.1.4.11 django\_elasticsearch\_dsl\_drf.tests.test\_helpers module

Test helpers.

```
class django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseTestCase  
    Test helpers.  
    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=  
    classmethod setUpClass()  
        Set up class.  
    test_filter_by_field()  
        Filter by field.
```

#### 12.19.1.4.12 django\_elasticsearch\_dsl\_drf.tests.test\_highlight module

Test highlight backend.

```
class django_elasticsearch_dsl_drf.tests.test_highlight.TestHighlight (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase  
    Test highlight.  
    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=  
    classmethod setUpClass()  
        Set up class.  
    test_list_results_with_highlights()  
        Test list results with facets.
```

#### 12.19.1.4.13 django\_elasticsearch\_dsl\_drf.tests.test\_more\_like\_this module

Test more-like-this functionality.

```
class django_elasticsearch_dsl_drf.tests.test_more_like_this.TestMoreLikeThis (methodName='runTest')  
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase  
    Test suggesters.  
    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=  
    classmethod setUpClass()  
        Set up class.  
    test_more_like_this()  
        Test more-like-this.
```

#### 12.19.1.4.14 django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_common module

Test ordering backend.

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



**static** `deep_get (dic, keys, default=None)`

Returns value at period separated keys in dictionary or default

From <https://stackoverflow.com/a/46890853>

**Parameters**

- **dic** (*dict*) – Dictionary to retrieve value from.
- **keys** (*str*) – Period separated path of keys
- **default** (*str*) – Default value to return if keys not found

**Returns** Value at keys or default

```
pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
```

```
classmethod setUpClass ()
```

Set up class.

```
test_address_default_nested_order_by ()
```

```
test_address_default_order_by ()
```

Test if default ordering on addresses is correct (asc).

```
test_address_order_by_nested_field_continent_ascending ()
```

Order by field *continent.country.name.raw* ascending.

```
test_address_order_by_nested_field_continent_descending ()
```

Order by field *continent.country.name.raw* descending.

```
test_address_order_by_nested_field_country_ascending ()
```

Order by field *continent.country.name.raw* ascending.

```
test_address_order_by_nested_field_country_descending ()
```

Order by field *continent.country.name.raw* descending.

```
test_author_default_order_by ()
```

Author order by default.

```
test_author_order_by_field_id_ascending ()
```

Order by field *name* ascending.

```
test_author_order_by_field_id_descending ()
```

Order by field *id* descending.

```
test_author_order_by_field_name_ascending ()
```

Order by field *name* ascending.

```
test_author_order_by_field_name_descending ()
```

Order by field *name* descending.

```
test_book_default_order_by ()
```

Book order by default.

```
test_book_order_by_field_id_ascending ()
```

Order by field *id* ascending.

```
test_book_order_by_field_id_descending ()
```

Order by field *id* descending.

```
test_book_order_by_field_title_ascending ()
```

Order by field *title* ascending.

```
test_book_order_by_field_title_descending ()
```

Order by field *title* descending.

```
test_book_order_by_non_existent_field()
    Order by non-existent field.

test_schema_field_not_required()
    Test schema fields always not required

test_schema_fields_with_filter_fields_list()
    Test schema field generator
```

#### 12.19.1.4.15 django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_geo\_spatial module

Test geo-spatial ordering filter backend.

```
class django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial.TestOrderingGeoSpatial (
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test ordering geo-spatial.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up.

    test_field_filter_geo_distance()
        Field filter geo_distance.

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

#### 12.19.1.4.16 django\_elasticsearch\_dsl\_drf.tests.test\_pagination module

Test pagination.

```
class django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination (methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test pagination.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_pagination()
        Test pagination.
```

#### 12.19.1.4.17 django\_elasticsearch\_dsl\_drf.tests.test\_pip\_helpers module

Test pip\_helpers.

```
class django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers (methodName='runTest')
    Bases: unittest.case.TestCase
    Test pip_helpers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={})]

    classmethod setUpClass()
        Hook method for setting up class fixture before running tests in the class.
```

**test\_check\_if\_installed()**

Test *check\_if\_installed*.

**Returns**

**test\_get\_installed\_packages()**

Test *get\_installed\_packages*.

**Returns**

**test\_get\_installed\_packages\_with\_versions()**

Test *get\_installed\_packages*.

**Returns**

#### 12.19.1.4.18 django\_elasticsearch\_dsl\_drf.tests.test\_query\_friendly\_pagination module

Test pagination.

**class** django\_elasticsearch\_dsl\_drf.tests.test\_query\_friendly\_pagination.**TestQueryFriendlyP**

Bases: *django\_elasticsearch\_dsl\_drf.tests.base.BaseRestFrameworkTestCase*

Test pagination.

**pytestmark** = [Mark(name='django\_db', args=(), kwargs={}), Mark(name='django\_db', args=

**classmethod** setUpClass()

Set up class.

**test\_pagination()**

Test pagination.

#### 12.19.1.4.19 django\_elasticsearch\_dsl\_drf.tests.test\_search module

Test search backend.

**class** django\_elasticsearch\_dsl\_drf.tests.test\_search.**TestSearch** (methodName='runTest')

Bases: *django\_elasticsearch\_dsl\_drf.tests.base.BaseRestFrameworkTestCase*

Test search.

**pytestmark** = [Mark(name='django\_db', args=(), kwargs={}), Mark(name='django\_db', args=

**classmethod** setUpClass()

Set up class.

**test\_compound\_search\_boost\_by\_field()**

**test\_compound\_search\_by\_field()**

**test\_compound\_search\_by\_field\_multi\_terms()**

**test\_compound\_search\_by\_nested\_field()**

**test\_schema\_field\_not\_required()**

Test schema fields always not required

**test\_schema\_fields\_with\_filter\_fields\_list()**

Test schema field generator

**test\_search\_boost** (url=None, search\_field='search')

Search boost.

**Returns**

**test\_search\_boost\_compound** (*search\_field='search'*)

**test\_search\_by\_field** (*url=None, search\_field='search'*)  
Search by field.

**test\_search\_by\_field\_multi\_terms** (*url=None, search\_field='search'*)  
Search by field, multiple terms.

**test\_search\_by\_nested\_field** (*url=None*)  
Search by field.

**class** `django_elasticsearch_dsl_drf.tests.test_search.TestSearchCustomCases` (*methodName='runTest'*)  
Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test search.

**pytestmark** = [**Mark** (name='django\_db', args=(), kwargs={}), **Mark** (name='django\_db', args=

**test\_search\_any\_word\_in\_indexed\_fields** ()

Test search any word in indexed fields.

**Returns**

#### 12.19.1.4.20 `django_elasticsearch_dsl_drf.tests.test_search_multi_match` module

Test multi match search filter backend.

**class** `django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` (*method*)  
Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test multi match search.

**pytestmark** = [**Mark** (name='django\_db', args=(), kwargs={}), **Mark** (name='django\_db', args=

**classmethod** **setUpClass** ()

Set up class.

**test\_schema\_field\_not\_required** ()

Test schema fields always not required

**test\_schema\_fields\_with\_filter\_fields\_list** ()

Test schema field generator

**test\_search** (*url=None*)

Search.

**test\_search\_boost** (*url=None*)

Search boost.

**Returns**

**test\_search\_boost\_selected\_fields** (*url=None*)

Search boost.

**Returns**

**test\_search\_selected\_fields** (*url=None*)

Search boost.

**Returns**

#### 12.19.1.4.21 `django_elasticsearch_dsl_drf.tests.test_search_simple_query_string` module

Test multi match search filter backend.

```

class django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQuerySt
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test simple query string search.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_schema_field_not_required()
        Test schema fields always not required

    test_schema_fields_with_filter_fields_list()
        Test schema field generator

    test_search_boost_selected_fields(url=None)
        Search boost.
        Returns

    test_search_selected_fields(url=None)
        Search boost.
        Returns

    test_search_with_quotes(url=None)
        Search with quotes.

    test_search_with_quotes_alternative()
        Test search by field.
        Parameters url –
        Returns

    test_search_with_quotes_boost(url=None)
        Search boost.
        Returns

    test_search_with_quotes_boost_alternative()
        Search boost.
        Returns

    test_search_without_quotes(url=None)
        Test search without quotes. This does not work on Elasticsearch 6.x.
        Parameters url –
        Returns

    test_search_without_quotes_boost(url=None)
        Search boost without quotes. Does not work on Elasticsearch 6.x.
        Returns

```

#### 12.19.1.4.22 django\_elasticsearch\_dsl\_drf.tests.test\_serializers module

Test serializers.

```

class django_elasticsearch_dsl_drf.tests.test_serializers.TestSerializers(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test serializers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=

```

```

test_serializer_document_equals_to_none()
    Test serializer no document specified.

test_serializer_fields_and_exclude()
    Test serializer fields and exclude.

test_serializer_meta_del_attr()
    Test serializer set attr.

test_serializer_meta_set_attr()
    Test serializer set attr.

test_serializer_no_document_specified()
    Test serializer no document specified.

```

#### 12.19.1.4.23 django\_elasticsearch\_dsl\_drf.tests.test\_source module

Test highlight backend.

```

class django_elasticsearch_dsl_drf.tests.test_source.TestSource(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase
    Test source.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_list_results()
        Test list results.

```

#### 12.19.1.4.24 django\_elasticsearch\_dsl\_drf.tests.test\_suggesters module

Test suggestions backend.

```

class django_elasticsearch_dsl_drf.tests.test_suggesters.TestContextSuggesters(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Test context suggesters.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

    test_suggesters_completion_context()
        Test suggesters completion context.

class django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters(methodName='runTest')
    Bases: django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase,
django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin
    Test suggesters.

    pytestmark = [Mark(name='django_db', args=(), kwargs={}), Mark(name='django_db', args=
    classmethod setUpClass()
        Set up class.

```

**test\_nested\_fields\_suggesters\_completion()**

Test suggesters completion for nested fields.

**test\_suggesters\_completion()**

Test suggesters completion.

**test\_suggesters\_completion\_no\_args\_provided()**

Test suggesters completion with no args provided.

**test\_suggesters\_phrase()**

Test suggesters phrase.

**test\_suggesters\_term()**

Test suggesters term.

**class** `django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggestersEmptyIndex` (*methodName*

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`,  
`django_elasticsearch_dsl_drf.tests.data_mixins.AddressesMixin`

Test suggesters on empty index.

**pytestmark** = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

**classmethod** `setUpClass()`

Set up class.

**test\_suggesters\_on\_empty\_index()**

Test suggesters phrase.

#### 12.19.1.4.25 `django_elasticsearch_dsl_drf.tests.test_versions` module

**class** `django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest` (*methodName='runTest'*)

Bases: `unittest.case.TestCase`

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

Test views.

**class** `django_elasticsearch_dsl_drf.tests.test_views.TestViews` (*methodName='runTest'*)

Bases: `django_elasticsearch_dsl_drf.tests.base.BaseRestFrameworkTestCase`

Test views.

**pytestmark** = [`Mark(name='django_db', args=(), kwargs={})`, `Mark(name='django_db', args=`

**classmethod** `setUpClass()`

Set up class.

**test\_detail\_view()**

Test detail view.

```
test_detail_view_with_custom_lookup()
    Test detail view with a custom lookup field.

test_listing_view()
    Test listing view.
```

#### 12.19.1.4.27 django\_elasticsearch\_dsl\_drf.tests.test\_wrappers module

Test wrappers.

```
class django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers (methodName='runTest')
    Bases: unittest.case.TestCase

    Test wrappers.

    pytestmark = [Mark(name='django_db', args=(), kwargs={})]

    classmethod setUpClass()
        Hook method for setting up class fixture before running tests in the class.

    test_dict_to_obj()
        Test dict_to_obj.
        Returns

    test_obj_to_dict()
        Test obj_to_dict.
        Returns

    test_wrapper_as_json()
        Test :Wrapper:'as_json' property.
```

#### 12.19.1.4.28 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](http://api.example.org/accounts/?page=4&page_size=100)

**django\_paginator\_class**

alias of *Paginator*

**get\_facets** (*page=None*)

Get facets.

**Parameters** *page* –

**Returns**

**get\_paginated\_response** (*data*)

Get paginated response.

**Parameters** *data* –

**Returns**

**get\_paginated\_response\_context** (*data*)

Get paginated response data.

**Parameters** *data* –

**Returns**

**paginate\_queryset** (*queryset, request, view=None*)

Paginate a queryset.

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

**Parameters**

• **queryset** –

• **request** –

• **view** –

**Returns**

**class** django\_elasticsearch\_dsl\_drf.pagination.**Paginator** (*object\_list, per\_page, orphans=0, allow\_empty\_first\_page=True*)

Bases: django.core.paginator.Paginator

Paginator for Elasticsearch.

**page** (*number*)

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

**Parameters** *number* –

**Returns**

**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 `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*,  
*or-*  
*phans=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*, *in-*  
*stalled\_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` (*instance=None*,  
*data=<class 'rest\_framework.fields.empty'>*,  
*\*\*kwargs*)

Bases: `rest_framework.serializers.Serializer`

A dynamic DocumentSerializer class.

**create** (*validated\_data*)

Create.

Do nothing.

**Parameters** **validated\_data** –

**Returns**

**get\_fields** ()

Get the required fields for serializing the result.

**update** (*instance, validated\_data*)

Update.

Do nothing.

**Parameters**

- **instance** –

- **validated\_data** –

**Returns**

**class** `django_elasticsearch_dsl_drf.serializers.DocumentSerializerMeta`

Bases: `rest_framework.serializers.SerializerMetaclass`

Metaclass for the DocumentSerializer.

Ensures that all declared subclasses implemented a Meta.

**class** `django_elasticsearch_dsl_drf.serializers.Meta`

Bases: `type`

Template for the DocumentSerializerMeta.Meta class.

**exclude** = ()

**field\_aliases** = {}

**field\_options** = {}

**fields** = ()

```
ignore_fields = ()
index_aliases = {}
index_classes = ()
search_fields = ()
serializers = ()
```

### 12.19.12 django\_elasticsearch\_dsl\_drf.utils module

Utils.

```
class django_elasticsearch_dsl_drf.utils.DictionaryProxy(mapping)
    Bases: object
    Dictionary proxy.
    to_dict()
        To dict.
        Returns
class django_elasticsearch_dsl_drf.utils.EmptySearch(*args, **kwargs)
    Bases: object
    Empty Search.
    execute(*args, **kwargs)
    highlight(*args, **kwargs)
    sort(*args, **kwargs)
    to_dict(*args, **kwargs)
```

### 12.19.13 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 Elasticsearch 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.
    document = None
    document_uid_field = 'id'
```

```

get_object()
    Get object.

get_queryset()
    Get queryset.

ignore = []

pagination_class
    alias of django_elasticsearch_dsl_drf.pagination.PageNumberPagination

class django_elasticsearch_dsl_drf.viewsets.DocumentViewSet(*args, **kwargs)
    Bases: django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet,
django_elasticsearch_dsl_drf.viewsets.SuggestMixin, django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin

    DocumentViewSet with suggest and functional-suggest mix-ins.

class django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin
    Bases: object

    Functional suggest mixin.

    functional_suggest(request)
        Functional suggest functionality.
        Parameters request –
        Returns

class django_elasticsearch_dsl_drf.viewsets.MoreLikeThisMixin
    Bases: object

    More-like-this mixin.

    more_like_this(request, pk=None, id=None)
        More-like-this functionality detail view.
        Parameters request –
        Returns

class django_elasticsearch_dsl_drf.viewsets.SuggestMixin
    Bases: object

    Suggest mixin.

    suggest(request)
        Suggest functionality.

```

### 12.19.15 django\_elasticsearch\_dsl\_drf.wrappers module

```

django_elasticsearch_dsl_drf.wrappers.dict_to_obj(mapping)
    dict to obj mapping.
    Parameters mapping(dict) –
    Returns
    Return type Wrapper

django_elasticsearch_dsl_drf.wrappers.obj_to_dict(obj)
    Wrapper to dict.
    Parameters obj ('obj':Wrapper) –
    Returns
    Return type dict

```

**class** django\_elasticsearch\_dsl\_drf.wrappers.**Wrapper**

Bases: object

Wrapper.

Example: >>> from django\_elasticsearch\_dsl\_drf.wrappers import dict\_to\_obj >>> >>> mapping = { >>> 'country': { >>> 'name': 'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': { >>> 'name': 'Amsterdam', >>> } >>> } >>> } >>> } >>> >>> wrapper = dict\_to\_obj(mapping) >>> wrapper.country.name >>> "Netherlands" >>> wrapper.country.province.name >>> "North Holland" >>> wrapper.country.province.city.name >>> "Amsterdam" >>> wrapper.as\_dict >>> { >>> 'country': { >>> 'name': 'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': { >>> 'name': 'Amsterdam', >>> } >>> } >>> } >>> } >>> str(wrapper) >>> "Netherlands"

**as\_dict**

As dict.

**Returns**

**Return type** dict

**as\_json**

As JSON.

**Returns**

**Return type** str

## 12.19.16 Module contents

Integrate Elasticsearch DSL with Django REST framework.



## CHAPTER 13

---

### Indices and tables

---

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



### d

`django_elasticsearch_dsl_drf`, 276

`django_elasticsearch_dsl_drf.analyzers`, 268

`django_elasticsearch_dsl_drf.apps`, 268

`django_elasticsearch_dsl_drf.compat`, 269

`django_elasticsearch_dsl_drf.constants`, 269

`django_elasticsearch_dsl_drf.elasticsearch_helpers`, 269

`django_elasticsearch_dsl_drf.fields`, 164

`django_elasticsearch_dsl_drf.fields.common`, 160

`django_elasticsearch_dsl_drf.fields.helpers`, 161

`django_elasticsearch_dsl_drf.fields.nested_fields`, 162

`django_elasticsearch_dsl_drf.filter_backends`, 248

`django_elasticsearch_dsl_drf.filter_backends.aggregations`, 168

`django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations`, 168

`django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations`, 168

`django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations`, 168

`django_elasticsearch_dsl_drf.filter_backends.faceted_search`, 242

`django_elasticsearch_dsl_drf.filter_backends.filtering`, 186

`django_elasticsearch_dsl_drf.filter_backends.filtering.common`, 168

`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial`, 176

`django_elasticsearch_dsl_drf.filter_backends.filtering.ids`, 181

`django_elasticsearch_dsl_drf.filter_backends.filtering.nested`, 182

`django_elasticsearch_dsl_drf.filter_backends.filter`, 184

`django_elasticsearch_dsl_drf.filter_backends.highlights`, 245

`django_elasticsearch_dsl_drf.filter_backends.mixins`, 246

`django_elasticsearch_dsl_drf.filter_backends.ordering`, 206

`django_elasticsearch_dsl_drf.filter_backends.ordering.aggregations`, 201

`django_elasticsearch_dsl_drf.filter_backends.ordering.aggregations.bucket_aggregations`, 204

`django_elasticsearch_dsl_drf.filter_backends.ordering.aggregations.metrics_aggregations`, 223

`django_elasticsearch_dsl_drf.filter_backends.ordering.aggregations.pipeline_aggregations`, 219

`django_elasticsearch_dsl_drf.filter_backends.ordering.faceted_search`, 220

`django_elasticsearch_dsl_drf.filter_backends.ordering.filtering`, 220

`django_elasticsearch_dsl_drf.filter_backends.ordering.filtering.common`, 222

`django_elasticsearch_dsl_drf.filter_backends.ordering.filtering.geo_spatial`, 215

`django_elasticsearch_dsl_drf.filter_backends.ordering.filtering.ids`, 211

`django_elasticsearch_dsl_drf.filter_backends.ordering.filtering.nested`, 211

[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search.simple\\_query\\_string,](#)  
[222](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_pip\\_helpers,](#)  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search,](#)  
[247](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_query\\_friender,](#)  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester,](#)  
[235](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_search,](#)  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester.functional,](#)  
[226](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_search\\_multisearch,](#)  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester.native,](#)  
[231](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_search\\_simple,](#)  
[django\\_elasticsearch\\_dsl\\_drf.helpers,](#) [264](#)  
[269](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_serializers,](#)  
[django\\_elasticsearch\\_dsl\\_drf.management,](#) [265](#)  
[249](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_source,](#)  
[django\\_elasticsearch\\_dsl\\_drf.management.commands,](#)  
[249](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_suggesters,](#)  
[django\\_elasticsearch\\_dsl\\_drf.management.commands.elasticsearch\\_remove\\_indexes,](#)  
[249](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_versions,](#)  
[django\\_elasticsearch\\_dsl\\_drf.pagination,](#) [267](#)  
[270](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_views,](#)  
[django\\_elasticsearch\\_dsl\\_drf.pip\\_helpers,](#) [267](#)  
[272](#) [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_wrappers,](#)  
[django\\_elasticsearch\\_dsl\\_drf.serializers,](#) [268](#)  
[273](#) [django\\_elasticsearch\\_dsl\\_drf.utils,](#) [274](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests,](#) [268](#) [django\\_elasticsearch\\_dsl\\_drf.versions,](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.base,](#) [274](#)  
[249](#) [django\\_elasticsearch\\_dsl\\_drf.viewsets,](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.data\\_mixins,](#) [274](#)  
[250](#) [django\\_elasticsearch\\_dsl\\_drf.wrappers,](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_faceted\\_search,](#)  
[250](#) [275](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_filtering\\_common,](#)  
[250](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_filtering\\_geo\\_spatial,](#)  
[253](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_filtering\\_global\\_aggregations,](#)  
[255](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_filtering\\_nested,](#)  
[255](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_filtering\\_post\\_filter,](#)  
[257](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_functional\\_suggesters,](#)  
[259](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_helpers,](#)  
[260](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_highlight,](#)  
[260](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_more\\_like\\_this,](#)  
[260](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_ordering\\_common,](#)  
[260](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_ordering\\_geo\\_spatial,](#)  
[262](#)  
[django\\_elasticsearch\\_dsl\\_drf.tests.test\\_pagination,](#)

## A

add\_arguments() (django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch\_remove\_indexes.Command  
     method), 249  
 AddressesMixin (class in apply\_filter\_terms()  
     django\_elasticsearch\_dsl\_drf.tests.data\_mixins),  
     250  
 aggregate() (django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search.FacetedSearchFilterBackend  
     method), 243  
 apply\_filter() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.NestedFilteringFilterBackend  
     class method), 183  
 apply\_filter() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.NestedFilteringFilterBackend  
     class method), 199  
 apply\_filter() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.PostFilterFilteringFilterBackend  
     class method), 185  
 apply\_filter() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.PostFilterFilteringFilterBackend  
     class method), 201  
 apply\_filter() (django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin  
     class method), 246  
 apply\_filter\_prefix() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 169  
 apply\_filter\_prefix() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 186  
 apply\_filter\_range() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 169  
 apply\_filter\_range() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 187  
 apply\_filter\_regexp() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 170  
 apply\_filter\_regexp() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 187  
 apply\_filter\_term() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 170  
 apply\_filter\_term() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend  
     class method), 172

<code>apply_query_exists()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 189	<code>apply_query_lte()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 191
<code>apply_query_geo_bounding_box()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.FilteringFilterBackend</code> class method), 177	<code>apply_query_size()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.functionalsuggester.FunctionalSuggester</code> class method), 229
<code>apply_query_geo_bounding_box()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend</code> class method), 193	<code>apply_query_size()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggester</code> class method), 240
<code>apply_query_geo_distance()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.FilteringFilterBackend</code> class method), 177	<code>apply_query_wildcard()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 174
<code>apply_query_geo_distance()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend</code> class method), 193	<code>apply_query_wildcard()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 191
<code>apply_query_geo_polygon()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.FilteringFilterBackend</code> class method), 177	<code>apply_suggester_completion()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.native.Suggester</code> class method), 233
<code>apply_query_geo_polygon()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend</code> class method), 194	<code>apply_suggester_completion()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_geo_shape()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.FilteringFilterBackend</code> class method), 178	<code>apply_suggester_completion_match()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.functionalsuggester.FunctionalSuggester</code> class method), 229
<code>apply_query_geo_shape()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend</code> class method), 194	<code>apply_suggester_completion_match()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggester</code> class method), 240
<code>apply_query_gt()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 172	<code>apply_suggester_completion_prefix()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.functionalsuggester.FunctionalSuggester</code> class method), 229
<code>apply_query_gt()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 189	<code>apply_suggester_completion_prefix()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggester</code> class method), 240
<code>apply_query_gte()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 172	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.native.Suggester</code> class method), 233
<code>apply_query_gte()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 190	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_in()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 173	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.native.Suggester</code> class method), 233
<code>apply_query_in()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 190	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_isnull()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 173	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_isnull()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 190	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_lt()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 174	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_lt()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend</code> class method), 191	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237
<code>apply_query_lte()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend</code> class method), 174	<code>apply_suggester_phrase()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.suggester.Suggester</code> class method), 237

<i>attribute</i> ), 255		( <i>django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchBackend</i> ), 243
<i>BaseDocumentViewSet</i> (class in <i>django_elasticsearch_dsl_drf.viewsets</i> ), 274	<i>construct_nested_search()</i>	
<i>BaseRestFrameworkTestCase</i> (class in <i>django_elasticsearch_dsl_drf.tests.base</i> ), 249	( <i>django_elasticsearch_dsl_drf.filter_backends.search.historical_search.HistoricalSearchBackend</i> ), 220	<i>construct_nested_search()</i>
<i>BaseSearchFilterBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search</i> ), 223	( <i>django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend</i> ), 224	<i>construct_search()</i>
<i>BaseSearchFilterBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search.base</i> ), 219	( <i>django_elasticsearch_dsl_drf.filter_backends.search.historical_search.HistoricalSearchBackend</i> ), 221	<i>construct_search()</i>
<i>BaseSearchQueryBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 215	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 211	<i>construct_search()</i>
<i>BaseSearchQueryBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 211	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 215	<i>construct_search()</i>
<i>BaseTestCase</i> (class in <i>django_elasticsearch_dsl_drf.tests.base</i> ), 249	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 211	<i>construct_search()</i>
<i>BooksMixin</i> (class in <i>django_elasticsearch_dsl_drf.tests.data_mixins</i> ), 250	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 211	<i>construct_search()</i>
<i>BooleanField</i> (class in <i>django_elasticsearch_dsl_drf.fields</i> ), 164	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 212	
<i>BooleanField</i> (class in <i>django_elasticsearch_dsl_drf.fields.common</i> ), 160	<i>construct_search()</i>	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 216
<b>C</b>	<i>construct_search()</i>	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 216
<i>CharField</i> (class in <i>django_elasticsearch_dsl_drf.fields</i> ), 165	<i>construct_search()</i>	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 215
<i>CharField</i> (class in <i>django_elasticsearch_dsl_drf.fields.common</i> ), 160	<i>construct_search()</i>	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 212
<i>check_if_installed()</i> (in module <i>django_elasticsearch_dsl_drf.pip_helpers</i> ), 272	<i>construct_search()</i>	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 216
<i>clean_queryset()</i> ( <i>django_elasticsearch_dsl_drf.filter_backends.suggester.FacetedSuggesterFilterBackend</i> ), 229	( <i>django_elasticsearch_dsl_drf.filter_backends.suggester.FacetedSuggesterFilterBackend</i> ), 216	
<i>clean_queryset()</i> ( <i>django_elasticsearch_dsl_drf.filter_backends.suggester.FacetedSuggesterFilterBackend</i> ), 241	( <i>django_elasticsearch_dsl_drf.filter_backends.suggester.FacetedSuggesterFilterBackend</i> ), 216	
<i>Command</i> (class in <i>django_elasticsearch_dsl_drf.management.commands.elasticsearch_dsl_remove_indexes</i> ), 249	<i>construct_search()</i>	
<i>CompoundSearchFilterBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search</i> ), 223	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 217	<i>construct_search()</i>
<i>CompoundSearchFilterBackend</i> (class in <i>django_elasticsearch_dsl_drf.filter_backends.search.compound_search</i> ), 220	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 214	<i>construct_search()</i>
<i>Config</i> (class in <i>django_elasticsearch_dsl_drf.apps</i> ), 268	( <i>django_elasticsearch_dsl_drf.filter_backends.search.query_backend_base</i> ), 218	
<i>construct_facets()</i>	<i>construct_search()</i>	



[\(django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.SearchFilterBackend method\), 225](#)  
[create\(\) \(django\\_elasticsearch\\_dsl\\_drf.serializers.DocumentSerializer method\), 273](#)  
[create\\_books\(\) \(django\\_elasticsearch\\_dsl\\_drf.tests.data\\_mixins.BookMixin class method\), 250](#)  
[created\\_addresses\(\) \(django\\_elasticsearch\\_dsl\\_drf.tests.data\\_mixins.AddressesMixin class method\), 250](#)  
**D**  
[DateField \(class in django\\_elasticsearch\\_dsl\\_drf.fields\), 165](#)  
[DateField \(class in django\\_elasticsearch\\_dsl\\_drf.fields.common\), 161](#)  
[deep\\_get\(\) \(django\\_elasticsearch\\_dsl\\_drf.tests.test\\_ordering\\_common.TestOrdering static method\), 260](#)  
[DefaultOrderingFilterBackend \(class in django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.ordering\), 206](#)  
[DefaultOrderingFilterBackend \(class in django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.ordering.common\), 201](#)  
[delete\\_all\\_indices\(\) \(in module django\\_elasticsearch\\_dsl\\_drf.elasticsearch\\_helpers\), 269](#)  
[dict\\_to\\_obj\(\) \(in module django\\_elasticsearch\\_dsl\\_drf.wrappers\), 275](#)  
[DictionaryProxy \(class in django\\_elasticsearch\\_dsl\\_drf.utils\), 274](#)  
[django\\_elasticsearch\\_dsl\\_drf \(module\), 276](#)  
[django\\_elasticsearch\\_dsl\\_drf.analyzers \(module\), 268](#)  
[django\\_elasticsearch\\_dsl\\_drf.apps \(module\), 268](#)  
[django\\_elasticsearch\\_dsl\\_drf.compat \(module\), 269](#)  
[django\\_elasticsearch\\_dsl\\_drf.constants \(module\), 269](#)  
[django\\_elasticsearch\\_dsl\\_drf.elasticsearch\\_helpers \(module\), 269](#)  
[django\\_elasticsearch\\_dsl\\_drf.fields \(module\), 164](#)  
[django\\_elasticsearch\\_dsl\\_drf.fields.common \(module\), 160](#)  
[django\\_elasticsearch\\_dsl\\_drf.fields.helpers \(module\), 161](#)  
[django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields \(module\), 162](#)  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends \(module\), 248](#)





**D**

- `DocumentViewSet` (class in `filter_queryset()` (`django_elasticsearch_dsl_dr.f.viewsets`), 275)
- `exclude()` (`django_elasticsearch_dsl_dr.serializers.Meta` attribute), 273
- `execute()` (`django_elasticsearch_dsl_dr.utils.EmptySearch` method), 274
- `extract_field_name()` (`django_elasticsearch_dsl_dr.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend` method), 230
- `extract_field_name()` (`django_elasticsearch_dsl_dr.filter_backends.suggester.FunctionalSuggesterFilterBackend` method), 241

**E**

- `EmptySearch` (class in `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.ordering.common.` method), 202)
- `exclude()` (`django_elasticsearch_dsl_dr.serializers.Meta` attribute), 273
- `execute()` (`django_elasticsearch_dsl_dr.utils.EmptySearch` method), 207
- `extract_field_name()` (`django_elasticsearch_dsl_dr.filter_backends.ordering.geo_spatial.GeoSpatialFilteringFilterBackend` method), 205
- `extract_field_name()` (`django_elasticsearch_dsl_dr.filter_backends.ordering.GeoSpatialFilteringFilterBackend` method), 208

**F**

- `faceted_search_param` (`django_elasticsearch_dsl_dr.filter_backends.faceted_search.FacetedSearchFilterBackend` attribute), 244
- `FacetedSearchFilterBackend` (class in `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.search.BaseSearchFilterBackend` method), 223)
- `field_aliases` (`django_elasticsearch_dsl_dr.serializers.Meta` attribute), 273
- `field_options` (`django_elasticsearch_dsl_dr.serializers.Meta` attribute), 273
- `fields` (`django_elasticsearch_dsl_dr.serializers.Meta` attribute), 273
- `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.faceted_search.FacetedSearchFilterBackend` method), 244
- `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.filtering.common.FilteringFilterBackend` method), 175
- `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.filtering.FilteringFilterBackend` method), 192
- `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.filtering.geo_spatial.GeoSpatialFilteringFilterBackend` method), 178
- `filter_queryset()` (`django_elasticsearch_dsl_dr.filter_backends.filtering.GeoSpatialFilteringFilterBackend` method), 194
- `filter_queryset()` (`FilterBackendMixin` (class in `django_elasticsearch_dsl_dr.filter_backends.mixins`), 246)
- `filter_queryset()` (`FilteringFilterBackend` (class in `django_elasticsearch_dsl_dr.filter_backends.filtering`), 186)
- `filter_queryset()` (`FilteringFilterBackend` (class in `django_elasticsearch_dsl_dr.filter_backends.filtering.common`), 186)

168  
FloatField (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.NestedFilterBackend), 199  
django\_elasticsearch\_dsl\_drf.fields), 165  
FloatField (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter\_backend.PostFilterBackend), 185  
django\_elasticsearch\_dsl\_drf.fields.common), 161  
functional\_suggest() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.PostFilterBackend), 201  
(django\_elasticsearch\_dsl\_drf.viewsets.FunctionalSuggesterMixin), 275  
FunctionalSuggesterFilterBackend (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchBackend), 219  
django\_elasticsearch\_dsl\_drf.filter\_backends.suggester), 239  
FunctionalSuggesterFilterBackend (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend), 223  
django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional), 228  
FunctionalSuggestMixin (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.search.historical.SearchBackend), 222  
django\_elasticsearch\_dsl\_drf.viewsets), 275  
G  
GeoPointField (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.search.SearchFilterBackend), 226  
django\_elasticsearch\_dsl\_drf.fields), 165  
GeoPointField (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.CommonOrderingBackend), 202  
django\_elasticsearch\_dsl\_drf.fields.nested\_fields), 162  
GeoShapeField (class in (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.DefaultOrderingBackend), 207  
django\_elasticsearch\_dsl\_drf.fields), 165  
GeoShapeField (class in get\_document\_for\_model() (in module  
django\_elasticsearch\_dsl\_drf.fields.nested\_fields), django\_elasticsearch\_dsl\_drf.helpers), 269  
162 get\_elasticsearch\_version() (in module  
django\_elasticsearch\_dsl\_drf.versions), 274  
GeoSpatialFilteringFilterBackend (class in get\_faceted\_search\_query\_params() (in module  
django\_elasticsearch\_dsl\_drf.filter\_backends.filtering), (django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search.FacetedSearchBackend), 244  
193 get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
GeoSpatialFilteringFilterBackend (class in get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
django\_elasticsearch\_dsl\_drf.filter\_backends.filtering\_geo\_spatial), 176  
GeoSpatialOrderingFilterBackend (class in get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
django\_elasticsearch\_dsl\_drf.filter\_backends.ordering), 207  
GeoSpatialOrderingFilterBackend (class in get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
django\_elasticsearch\_dsl\_drf.filter\_backends.ordering\_geo\_spatial), 204  
get\_all\_indices() (in module get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
django\_elasticsearch\_dsl\_drf.elasticsearch\_helpers), class method), 215  
269 get\_facets() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPaginator), 270  
get\_coreschema\_field() (class method), 218  
(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend), 175  
get\_coreschema\_field() (class method), 218  
(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.FilteringFilterBackend), 192  
get\_coreschema\_field() (class method), 218  
(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.NestedFilterBackend), 184  
method), 184  
get\_coreschema\_field() (class method), 218  
(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.NestedFilterBackend), 199  
method), 199  
get\_coreschema\_field() (class method), 218  
get\_filter\_query\_params()

<code>(django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringBackend class method), 175</code>	<code>(django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringBackend class method), 175</code>
<code>get_filter_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend class method), 192</code>	<code>get_gte_lte_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend class method), 192</code>
<code>get_filter_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filtering.FilteringBackend class method), 178</code>	<code>get_highlight_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightFilterBackend class method), 246</code>
<code>get_filter_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 194</code>	<code>get_ids_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend class method), 182</code>
<code>get_filter_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.nested_filtering.FilteringBackend class method), 184</code>	<code>get_ids_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend class method), 198</code>
<code>get_filter_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend class method), 199</code>	<code>get_ids_values()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend</code> class method), 198
<code>get_geo_bounding_box_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filtering.FilteringBackend class method), 178</code>	<code>get_ids_values()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend</code> class method), 198
<code>get_geo_bounding_box_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 195</code>	<code>get_ids_values()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend</code> class method), 198
<code>get_geo_distance_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filtering.FilteringBackend class method), 179</code>	<code>get_ids_values()</code> ( <code>django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringBackend</code> class method), 198
<code>get_geo_distance_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 195</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingBackend class method), 203</code>
<code>get_geo_distance_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial_ordering.FilteringBackend class method), 205</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingBackend class method), 204</code>
<code>get_geo_distance_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend class method), 208</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingBackend class method), 207</code>
<code>get_geo_polygon_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filtering.FilteringBackend class method), 179</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial_ordering.FilteringBackend class method), 206</code>
<code>get_geo_polygon_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 195</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend class method), 209</code>
<code>get_geo_shape_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial_filtering.FilteringBackend class method), 180</code>	<code>get_ordering_query_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingBackend class method), 210</code>
<code>get_geo_shape_params()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 196</code>	<code>get_paginated_response()</code> <code>(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination class method), 270</code>
<code>get_geo_spatial_field_name()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial_ordering.FilteringBackend class method), 205</code>	<code>get_paginated_response()</code> <code>(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination class method), 271</code>
<code>get_geo_spatial_field_name()</code> <code>(django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend class method), 208</code>	<code>get_paginated_response_context()</code> <code>(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination class method), 270</code>
<code>get_gte_lte_params()</code>	<code>get_paginated_response_context()</code>



(`django_elasticsearch_dsl_drf.pagination.PageNumberPagination`), 219

`method`), 271

`get_query_backends()` (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend`), 219

`method`), 219

`get_query_backends()` (`django_elasticsearch_dsl_drf.filter_backends.search.historical.SimpleQueryBackend`), 223

`method`), 223

`get_query_options()` (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend`), 213

`class method`), 213

`get_query_options()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backend.MultiMatchQueryBackend`), 217

`class method`), 217

`get_query_options()` (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend`), 233

`class method`), 215

`get_query_options()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backend.SimpleQueryStringQueryBackend`), 233

`class method`), 219

`get_queryset()` (`django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`), 275

`method`), 275

`get_range_params()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.combined_filtering.FilterBackend`), 175

`class method`), 175

`get_range_params()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilterBackend`), 192

`class method`), 192

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.combined_filtering.FilterBackend`), 176

`method`), 176

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilterBackend`), 192

`method`), 192

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested_filtering.FilterBackend`), 184

`method`), 184

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`), 200

`method`), 200

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filtering.FilterBackend`), 185

`method`), 185

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilteringFilterBackend`), 201

`method`), 201

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.ordering.common_ordering.FilterBackend`), 204

`method`), 204

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend`), 210

`method`), 210

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchBackend`), 219

`method`), 219

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend`), 219

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.search.historical.SimpleQueryBackend`), 223

`get_schema_fields()` (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend`), 213

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchBackend`), 233

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backend.MultiMatchQueryBackend`), 217

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend`), 233

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backend.SimpleQueryStringQueryBackend`), 233

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.historical.SimpleQueryBackend`), 223

`get_search_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend`), 213

`get_suggester_context()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterBackend`), 234

`class method`), 234

`get_suggester_context()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterBackend`), 237

`class method`), 237

`get_suggester_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.function_based.SuggesterBackend`), 230

`method`), 230

`get_suggester_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionBasedSuggesterBackend`), 241

`method`), 241

`get_suggester_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterBackend`), 235

`method`), 235

`get_suggester_query_params()` (`django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterBackend`), 239

`method`), 239

`get_value()` (`django_elasticsearch_dsl_drf.fields.BooleanField`), 160

`get_value()` (`django_elasticsearch_dsl_drf.fields.CharField`), 165

`get_value()` (`django_elasticsearch_dsl_drf.fields.common.BooleanField`), 160

`get_value()` (`django_elasticsearch_dsl_drf.fields.common.CharField`), 161

`get_value()` (`django_elasticsearch_dsl_drf.fields.common.DateField`), 161

`get_value()` (`django_elasticsearch_dsl_drf.fields.common.FloatField`), 161

`get_value()` (`django_elasticsearch_dsl_drf.fields.common.IntegerField`), 161

get\_value() (django\_elasticsearch\_dsl\_drf.fields.common.IntegerField (class in method), 161  
 get\_value() (django\_elasticsearch\_dsl\_drf.fields.DateField 161  
 method), 165  
 IPAddressField (class in  
 get\_value() (django\_elasticsearch\_dsl\_drf.fields.FloatField django\_elasticsearch\_dsl\_drf.fields), 166  
 method), 165  
 IPAddressField (class in  
 get\_value() (django\_elasticsearch\_dsl\_drf.fields.IntegerField django\_elasticsearch\_dsl\_drf.fields.common),  
 method), 166  
 161

get\_value() (django\_elasticsearch\_dsl\_drf.fields.IPAddressField  
 method), 166  
 K  
 get\_value() (django\_elasticsearch\_dsl\_drf.fields.ListField KeywordField (class in  
 method), 166  
 django\_elasticsearch\_dsl\_drf.compat), 269

get\_value() (django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ListField  
 method), 164  
 L

get\_value() (django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField  
 method), 163  
 Label (django\_elasticsearch\_dsl\_drf.apps.Config at-  
 tribute), 268

get\_value() (django\_elasticsearch\_dsl\_drf.fields.ObjectField  
 method), 167  
 LimitOffsetPagination (class in  
 django\_elasticsearch\_dsl\_drf.pagination),  
 270

H  
 ListField (class in  
 handle() (django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch\_remove\_index\_connections,  
 method), 249  
 ListField (class in  
 help(django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch\_remove\_index\_connections,  
 attribute), 249  
 164

highlight() (django\_elasticsearch\_dsl\_drf.utils.EmptySearch  
 method), 274  
 M

highlight\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.highlight.HighlightBackend  
 attribute), 246  
 matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchBackend,  
 attribute), 219

HighlightBackend (class in matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend,  
 django\_elasticsearch\_dsl\_drf.filter\_backends.highlight), attribute), 223  
 245

matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.multi\_match.MultiMatchBackend,  
 attribute), 222

matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.MultiMatchBackend,  
 attribute), 222

matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.SimpleQueryStringBackend,  
 attribute), 226

ids\_query\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering\_ids.FilterBackend  
 attribute), 182  
 matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.simple\_query\_string.SimpleQueryStringBackend,  
 attribute), 198

ids\_query\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.filtering\_ids.FilterBackend  
 attribute), 198  
 matching(django\_elasticsearch\_dsl\_drf.filter\_backends.search.SimpleQueryStringBackend,  
 attribute), 226

IdsFilterBackend (class in attribute), 226  
 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering\_ids.FilterBackend (class in  
 197  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.QueryBackend,  
 attribute), 216

IdsFilterBackend (class in 216  
 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering\_ids.FilterBackend (class in  
 181  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.QueryBackend,  
 attribute), 216

ignore(django\_elasticsearch\_dsl\_drf.viewsets.BaseDocumentViewSet  
 attribute), 275  
 MatchPhraseQueryBackend (class in  
 212

ignore\_fields(django\_elasticsearch\_dsl\_drf.serializers.Meta  
 attribute), 273  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.QueryBackend (class in  
 216

index\_aliases(django\_elasticsearch\_dsl\_drf.serializers.Meta  
 attribute), 274  
 MatchPhraseQueryBackend (class in  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.QueryBackend,  
 attribute), 216

index\_classes(django\_elasticsearch\_dsl\_drf.serializers.Meta  
 attribute), 274  
 211

IntegerField (class in  
 MatchQueryBackend (class in  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.QueryBackend,  
 attribute), 166  
 215

MatchQueryBackend (class in ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend), 207), 211  
 Meta (class in django\_elasticsearch\_dsl\_drf.serializers), (attribute), 204  
 more\_like\_this() (django\_elasticsearch\_dsl\_drf.viewsets.MoreLikeThisMixin method), 275  
 more\_like\_this() (in module attribute), 206  
 django\_elasticsearch\_dsl\_drf.helpers), 269  
 MoreLikeThisMixin (class in attribute), 209  
 django\_elasticsearch\_dsl\_drf.viewsets), 275  
 MultiMatchQueryBackend (class in attribute), 210  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend), 216  
 MultiMatchQueryBackend (class in 209  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend.multi\_match), (class in  
 212  
 django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common),  
 MultiMatchSearchFilterBackend (class in 203  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend), 224  
 MultiMatchSearchFilterBackend (class in attribute), 272  
 django\_elasticsearch\_dsl\_drf.filter\_backends.search.multi\_match),  
 222

## P

Page (class in django\_elasticsearch\_dsl\_drf.pagination), 271  
 name (django\_elasticsearch\_dsl\_drf.apps.Config attribute), 268  
 page() (django\_elasticsearch\_dsl\_drf.pagination.Paginator method), 271  
 page() (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPaginator method), 272  
 page\_size\_query\_param (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNumber attribute), 272  
 NestedField (class in 167  
 django\_elasticsearch\_dsl\_drf.fields),  
 NestedField (class in 162  
 django\_elasticsearch\_dsl\_drf.fields.nested\_fields),  
 NestedFilteringFilterBackend (class in PageNumberPagination (class in  
 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering), django\_elasticsearch\_dsl\_drf.pagination),  
 198  
 271  
 NestedFilteringFilterBackend (class in paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPagination method), 271  
 182  
 NestedQueryBackend (class in paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.PageNumberPagination method), 271  
 217  
 NestedQueryBackend (class in paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNumber method), 272  
 213  
 pagination\_class (django\_elasticsearch\_dsl\_drf.viewsets.BaseDocument attribute), 275

## O

obj\_to\_dict() (in module Paginator (class in  
 django\_elasticsearch\_dsl\_drf.wrappers), django\_elasticsearch\_dsl\_drf.pagination),  
 275  
 271  
 ObjectField (class in PostFilterFilteringFilterBackend (class in  
 django\_elasticsearch\_dsl\_drf.fields), 167  
 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering),  
 ObjectField (class in 200  
 django\_elasticsearch\_dsl\_drf.fields.nested\_fields)PostFilterFilteringFilterBackend (class in  
 163  
 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter)

184

`prepare_faceted_search_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend`  
class method), 244

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`  
class method), 176

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`  
class method), 193

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`  
class method), 181

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend`  
class method), 197

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`  
class method), 184

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend`  
class method), 200

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`  
class method), 185

`prepare_filter_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilteringFilterBackend`  
class method), 201

`prepare_highlight_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend`  
class method), 246

`prepare_suggester_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.suggester.functional.FunctionalSuggesterFilterBackend`  
class method), 230

`prepare_suggester_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend`  
class method), 242

`prepare_suggester_fields()`  
(`django_elasticsearch_dsl_drf.filter_backends.suggester.natural.NaturalSuggesterFilterBackend`  
class method), 235

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

`pytestmark (django_elasticsearch_dsl_drf.tests.base.BaseTestCase)`  
attribute), 249

`pytestmark (django_elasticsearch_dsl_drf.tests.base.BaseTestCase)`  
attribute), 249

`pytestmark (django_elasticsearch_dsl_drf.tests.test_faceted_search.TestFacetedSearch)`  
attribute), 250

`pytestmark (django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon)`  
attribute), 251

`pytestmark (django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial)`  
attribute), 254

`pytestmark (django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations.TestFilteringGlobalAggregations)`  
attribute), 255

`pytestmark (django_elasticsearch_dsl_drf.tests.test_filtering_post_filtering.TestFilteringPostFiltering)`  
attribute), 257

`pytestmark (django_elasticsearch_dsl_drf.tests.test_functional_suggester.TestFunctionalSuggester)`  
attribute), 259

`pytestmark (django_elasticsearch_dsl_drf.tests.test_helpers.TestHelpers)`  
attribute), 260

`pytestmark (django_elasticsearch_dsl_drf.tests.test_highlight.TestHighlight)`  
attribute), 260

`pytestmark (django_elasticsearch_dsl_drf.tests.test_more_like_this.TestMoreLikeThis)`  
attribute), 260

`pytestmark (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon)`  
attribute), 261

`pytestmark (django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial.TestOrderingGeoSpatial)`  
attribute), 262

`pytestmark (django_elasticsearch_dsl_drf.tests.test_pagination.TestPagination)`  
attribute), 262

`pytestmark (django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers)`  
attribute), 263

`pytestmark (django_elasticsearch_dsl_drf.tests.test_query_friendly_pagination.TestQueryFriendlyPagination)`  
attribute), 263

`pytestmark (django_elasticsearch_dsl_drf.tests.test_search.TestSearch)`  
attribute), 263

`pytestmark (django_elasticsearch_dsl_drf.tests.test_search.TestSearchC)`  
attribute), 264

`pytestmark (django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestSearchMultiMatch)`  
attribute), 264

`pytestmark (django_elasticsearch_dsl_drf.tests.test_search_simple_query.TestSearchSimpleQuery)`  
attribute), 265

`pytestmark (django_elasticsearch_dsl_drf.tests.test_serializers.TestSerializers)`  
attribute), 265

`pytestmark (django_elasticsearch_dsl_drf.tests.test_source.TestSource)`  
attribute), 266

`pytestmark (django_elasticsearch_dsl_drf.tests.test_suggesters.TestCon)`  
attribute), 266

`pytestmark (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSug)`  
attribute), 266

`pytestmark (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSug)`  
attribute), 267

`pytestmark (django_elasticsearch_dsl_drf.tests.test_views.TestViews)`  
attribute), 267

`pytestmark (django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrap)`  
attribute), 268

**Q**

`pytestmark (django_elasticsearch_dsl_drf.filter_backends.search.QueryBackend)`  
attribute), 219

`pytestmark (django_elasticsearch_dsl_drf.filter_backends.search.QueryBackend)`  
attribute), 223

`pytestmark (django_elasticsearch_dsl_drf.filter_backends.search.QueryBackend)`  
attribute), 220



query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.GoToOpenSearchFilterBackend, attribute), 223

query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchMultiMatchSearchFilterBackend, attribute), 222

query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchMultiMatchSearchFilterBackend, attribute), 224

query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SimpleQueryStringSearchFilterBackend, attribute), 223

query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SimpleQueryStringSearchFilterBackend, attribute), 224

query\_backends (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SimpleQueryStringSearchFilterBackend, attribute), 226

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SuggesterBackend, attribute), 211

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhraseQueryBackend, attribute), 212

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 212

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 216

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 216

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 215

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 213

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 217

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 214

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 218

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 215

query\_type (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.MatchPhrasePrefixQueryBackend, attribute), 219

QueryFriendlyPageNumberPagination (class in django\_elasticsearch\_dsl\_drf.pagination), 272

QueryFriendlyPaginator (class in django\_elasticsearch\_dsl\_drf.pagination), 272

**S**

search\_fields (django\_elasticsearch\_dsl\_drf.serializers.Meta, attribute), 274

search\_param (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.BaseSearchFilterBackend, attribute), 219

search\_param (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.BaseSearchFilterBackend, attribute), 223

search\_param (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.BaseSearchFilterBackend, attribute), 222

search\_param (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.BaseSearchFilterBackend, attribute), 222

search\_param (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.BaseSearchFilterBackend, attribute), 224

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested, class method), 255

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filters, class method), 257

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_functional\_suggester, class method), 259

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_helpers.TestHelpers, class method), 260

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_highlight.TestHighlight, class method), 260

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_more\_like\_this.TestMoreLikeThis, class method), 260

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_comments, class method), 261

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_geo\_points, class method), 262

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_pagination.TestPagination, class method), 262

setUpClass () (django\_elasticsearch\_dsl\_drf.tests.test\_pip\_helpers.TestPipHelpers, class method), 262

```

setUpClass() (django_elasticsearch_dsl_drf.tests.test_query_friendly_pagination.TestQueryFriendlyPagination.mixins.FilterBackend
    class method), 263
setUpClass() (django_elasticsearch_dsl_drf.tests.test_search.TestSearch() (in module
    class method), 263
setUpClass() (django_elasticsearch_dsl_drf.tests.test_search_result.MultiMarchSearchDrf.viewsets.SuggestMixin
    class method), 264
setUpClass() (django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch
    class method), 265
setUpClass() (django_elasticsearch_dsl_drf.tests.test_source.TestSource
    class method), 266
setUpClass() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestCompatSuggesters (class in
    class method), 266
setUpClass() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggesters (class in
    class method), 266
setUpClass() (django_elasticsearch_dsl_drf.tests.test_suggesters.TestSuggestersEmptyIndex
    class method), 267
setUpClass() (django_elasticsearch_dsl_drf.tests.test_views.TestViews
    class method), 267
setUpClass() (django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers
    class method), 268
SimpleQueryStringQueryBackend (class in (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    django_elasticsearch_dsl_drf.filter_backends.search.query_backend), 261
SimpleQueryStringQueryBackend (class in (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    django_elasticsearch_dsl_drf.filter_backends.search.query_backend, simple_query_string),
    214
SimpleQueryStringSearchFilterBackend (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    (class in django_elasticsearch_dsl_drf.filter_backends.search), 261
SimpleQueryStringSearchFilterBackend (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    (class in django_elasticsearch_dsl_drf.filter_backends.search, simple_query_string),
    222
sleep() (django_elasticsearch_dsl_drf.tests.base.SleepMixin (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    class method), 250
SleepMixin (class in test_author_default_order_by()
    django_elasticsearch_dsl_drf.tests.base), (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    250
sort() (django_elasticsearch_dsl_drf.utils.EmptySearch test_author_order_by_field_id_ascending()
    method), 274
sort_by_list() (in module method), 261
    django_elasticsearch_dsl_drf.helpers), 270
SourceBackend (class in (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    django_elasticsearch_dsl_drf.filter_backends.source), method), 261
split_lookup_complex_multiple_value() (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    (django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin
    class method), 247
split_lookup_complex_value() (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    (django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin
    class method), 247
split_lookup_filter() (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    (django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin
    class method), 247
split_lookup_name() (test_book_default_order_by()
    (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    method), 261
    test_book_order_by_field_id_ascending()
    (django_elasticsearch_dsl_drf.tests.test_ordering_common.TestO
    method), 261

```

*method*), 261

`test_book_order_by_field_id_descending()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon`

*method*), 261

`test_book_order_by_field_title_ascending()` *method*), 251

`(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon`

*method*), 261

`test_book_order_by_field_title_descending()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon`

*method*), 261

`test_book_order_by_non_existent_field()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon`

*method*), 261

`test_check_if_installed()` *method*), 251

`(django_elasticsearch_dsl_drf.tests.test_pip_helpers.TestPipHelpers`

*method*), 262

`test_compound_search_boost_by_field()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_search.TestSearch`

*method*), 263

`test_compound_search_by_field()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_search.TestSearch`

*method*), 263

`test_compound_search_by_field_multi_terms()` *method*), 251

`(django_elasticsearch_dsl_drf.tests.test_search.TestSearch`

*method*), 263

`test_compound_search_by_nested_field()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_search.TestSearch`

*method*), 263

`test_default_filter_lookup()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon`

*method*), 251

`test_detail_view()` *method*), 251

`(django_elasticsearch_dsl_drf.tests.test_views.TestViews`

*method*), 267

`test_detail_view_with_custom_lookup()` *method*), 256

`(django_elasticsearch_dsl_drf.tests.test_views.TestViews`

*method*), 267

`test_dict_to_obj()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_wrappers.TestWrappers`

*method*), 268

`test_elasticsearch_dsl_6_3_0()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest`

*method*), 267

`test_elasticsearch_dsl_7_0_0()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_versions.VersionsTest`

*method*), 267

`test_field_filter_contains()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon`

*method*), 251

`test_field_filter_contains()` *method*), 262

`(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested`

*method*), 255

`test_field_filter_contains()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter`

*method*), 257

`test_field_filter_contains()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter`

*method*), 257

`test_field_filter_exclude()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon`

*method*), 251

`test_field_filter_exclude()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested`

*method*), 255

`test_field_filter_exclude()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter`

*method*), 257

`test_field_filter_exists_false()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon`

*method*), 251

`test_field_filter_exists_false()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested`

*method*), 255

`test_field_filter_exists_false()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter`

*method*), 257

`test_field_filter_exists_true()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon`

*method*), 251

`test_field_filter_exists_true()` *method*), 255

`(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested`

*method*), 255

`test_field_filter_exists_true()` *method*), 257

`(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter`

*method*), 257

`test_field_filter_geo_bounding_box()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

*method*), 254

`test_field_filter_geo_bounding_box_fail_test()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

*method*), 254

`test_field_filter_geo_distance()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

*method*), 254

`test_field_filter_geo_distance()` *method*), 262

`(django_elasticsearch_dsl_drf.tests.test_ordering_geo_spatial.TestOrderingGeoSpatial`

*method*), 262

`test_field_filter_geo_distance_distance_type_arc()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

*method*), 254

`test_field_filter_geo_distance_not_enough_args_fail_test()` *method*), 254

`(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFilteringGeoSpatial`

*method*), 254

<code>method)</code> , 254	<code>method)</code> , 258
<code>test_field_filter_geo_polygon()</code>	<code>test_field_filter_isnull_true()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 254	<code>method)</code> , 252
<code>test_field_filter_geo_polygon_fail_test()</code>	<code>test_field_filter_isnull_true()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 254	<code>method)</code> , 258
<code>test_field_filter_geo_polygon_string_options()</code>	<code>test_field_filter_lt()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 254	<code>method)</code> , 252
<code>test_field_filter_geo_polygon_string_options_fail_test()</code>	<code>test_field_filter_lt()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 254	<code>method)</code> , 258
<code>test_field_filter_geo_shape_circle()</code>	<code>test_field_filter_lt_with_boost()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 254	<code>method)</code> , 252
<code>test_field_filter_geo_shape_circle_intersects()</code>	<code>test_field_filter_lt_with_boost()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 254	<code>method)</code> , 258
<code>test_field_filter_geo_shape_envelope()</code>	<code>test_field_filter_lte()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial.TestFieldFilterGeoSpatial</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 254	<code>method)</code> , 252
<code>test_field_filter_gt()</code>	<code>test_field_filter_lte()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 251	<code>method)</code> , 258
<code>test_field_filter_gt()</code>	<code>test_field_filter_prefix()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 257	<code>method)</code> , 252
<code>test_field_filter_gt_with_boost()</code>	<code>test_field_filter_prefix()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFieldFilterNested</code>
<code>method)</code> , 251	<code>method)</code> , 256
<code>test_field_filter_gt_with_boost()</code>	<code>test_field_filter_prefix()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 257	<code>method)</code> , 258
<code>test_field_filter_gte()</code>	<code>test_field_filter_range()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 251	<code>method)</code> , 252
<code>test_field_filter_gte()</code>	<code>test_field_filter_range()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFieldFilterNested</code>
<code>method)</code> , 257	<code>method)</code> , 256
<code>test_field_filter_in()</code>	<code>test_field_filter_range()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 252	<code>method)</code> , 258
<code>test_field_filter_in()</code>	<code>test_field_filter_range_with_boost()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFieldFilterNested</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>
<code>method)</code> , 256	<code>method)</code> , 252
<code>test_field_filter_in()</code>	<code>test_field_filter_range_with_boost()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFieldFilterNested</code>
<code>method)</code> , 258	<code>method)</code> , 256
<code>test_field_filter_isnull_false()</code>	<code>test_field_filter_range_with_boost()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>
<code>method)</code> , 252	<code>method)</code> , 258
<code>test_field_filter_isnull_false()</code>	<code>test_field_filter_regexp()</code>
<code>(django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFieldFilterPostFilter</code>	<code>(django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFieldFilterCommon</code>



method), 253

test\_field\_filter\_term()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_field\_filter\_term()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_field\_filter\_term()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 258

test\_field\_filter\_term\_explicit()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_field\_filter\_term\_explicit()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_field\_filter\_term\_explicit()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_field\_filter\_terms\_list()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_field\_filter\_terms\_list()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_field\_filter\_terms\_list()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_field\_filter\_terms\_string()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_field\_filter\_terms\_string()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_field\_filter\_terms\_string()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_field\_filter\_wildcard()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_field\_filter\_wildcard()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_field\_filter\_wildcard()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_filter\_by\_field()  
(django\_elasticsearch\_dsl\_drf.tests.test\_helpers.TestHelpers method), 260

test\_get\_installed\_packages()  
(django\_elasticsearch\_dsl\_drf.tests.test\_pip\_helpers.TestPipHelpers method), 263

test\_get\_installed\_packages\_with\_versions()  
(django\_elasticsearch\_dsl\_drf.tests.test\_pip\_helpers.TestPipHelpers method), 263

method), 263

test\_ids\_empty\_filter()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_ids\_filter()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_ids\_filter()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_list\_results()  
(django\_elasticsearch\_dsl\_drf.tests.test\_source.TestSource method), 266

test\_list\_results\_with\_facets()  
(django\_elasticsearch\_dsl\_drf.tests.test\_faceted\_search.TestFacetedSearch method), 250

test\_list\_results\_with\_facets()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_global\_aggregation.TestFilteringGlobalAggregation method), 255

test\_list\_results\_with\_facets()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

test\_list\_results\_with\_highlights()  
(django\_elasticsearch\_dsl\_drf.tests.test\_highlight.TestHighlight method), 260

test\_listing\_view()  
(django\_elasticsearch\_dsl\_drf.tests.test\_views.TestViews method), 268

test\_more\_like\_this()  
(django\_elasticsearch\_dsl\_drf.tests.test\_more\_like\_this.TestMoreLikeThis method), 260

test\_nested\_field\_filter\_term()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_nested\_fields\_suggesters\_completion()  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestSuggesters method), 266

test\_obj\_to\_dict()  
(django\_elasticsearch\_dsl\_drf.tests.test\_wrappers.TestWrappers method), 268

test\_pagination()  
(django\_elasticsearch\_dsl\_drf.tests.test\_pagination.TestPagination method), 262

test\_pagination()  
(django\_elasticsearch\_dsl\_drf.tests.test\_query\_friendly\_pagination.TestQueryFriendlyPagination method), 263

test\_schema\_field\_not\_required()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon method), 253

test\_schema\_field\_not\_required()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested.TestFilteringNested method), 256

test\_schema\_field\_not\_required()  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter method), 259

`method)`, 259  
`test_schema_field_not_required()` (`django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon` method), 262  
`test_schema_field_not_required()` (`django_elasticsearch_dsl_drf.tests.test_search.TestSearch` method), 264  
`test_schema_field_not_required()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_schema_field_not_required()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_schema_field_not_required()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_filtering_common.TestFilteringCommon` method), 253  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_filtering_nested.TestFilteringNested` method), 256  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_filtering_post_filter.TestFilteringPostFilter` method), 259  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_ordering_common.TestOrderingCommon` method), 262  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_schema_fields_with_filter_fields_list()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265  
`test_search()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_search_any_word_in_indexed_fields()` (`django_elasticsearch_dsl_drf.tests.test_search.TestSearch` method), 264  
`test_search_boost()` (`django_elasticsearch_dsl_drf.tests.test_search.TestSearch` method), 263  
`test_search_boost()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_search_boost_compound()` (`django_elasticsearch_dsl_drf.tests.test_search.TestSearch` method), 263  
`test_search_boost_selected_fields()` (`django_elasticsearch_dsl_drf.tests.test_search_multi_match.TestMultiMatchSearch` method), 264  
`test_search_boost_selected_fields()` (`django_elasticsearch_dsl_drf.tests.test_search_simple_query_string.TestSimpleQueryStringSearch` method), 265

test\_suggesters\_completion\_context() TestHighlight (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestContextSuggesters),  
method), 266 260

test\_suggesters\_completion\_no\_args\_provided() TestMoreLikeThis (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_functional\_suggesters.TestFunctionalSuggesters),  
method), 259 260

test\_suggesters\_completion\_no\_args\_provided() TestMultiMatchSearch (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestSuggesters),  
method), 267 264

test\_suggesters\_on\_empty\_index() TestOrdering (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestSuggesters),  
method), 267 260

test\_suggesters\_phrase() TestOrderingGeoSpatial (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestSuggesters),  
method), 267 262

test\_suggesters\_term() TestPagination (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_suggesters.TestSuggesters),  
method), 267 262

test\_various\_complex\_fields() TestPipHelpers (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common.TestFilteringCommon),  
method), 253 262

test\_various\_complex\_fields() TestQueryFriendlyPagination (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter.TestFilteringPostFilter),  
method), 259 263

test\_wrapper\_as\_json() TestSearch (class in  
(django\_elasticsearch\_dsl\_drf.tests.test\_wrappers.TestWrappers),  
method), 268 263

TestContextSuggesters (class in TestSearchCustomCases (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_suggesters), django\_elasticsearch\_dsl\_drf.tests.test\_search),  
266 264

TestFacetedSearch (class in TestSerializers (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_faceted\_search), django\_elasticsearch\_dsl\_drf.tests.test\_serializers),  
250 265

TestFilteringCommon (class in TestSimpleQueryStringSearch (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common), django\_elasticsearch\_dsl\_drf.tests.test\_search\_simple\_query\_str),  
250 264

TestFilteringGeoSpatial (class in TestSource (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_geo\_spatial), django\_elasticsearch\_dsl\_drf.tests.test\_source),  
253 266

TestFilteringGlobalAggregations (class in TestSuggesters (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_global\_aggregations), django\_elasticsearch\_dsl\_drf.tests.test\_suggesters),  
255 266

TestFilteringNested (class in TestSuggestersEmptyIndex (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested), django\_elasticsearch\_dsl\_drf.tests.test\_suggesters),  
255 267

TestFilteringPostFilter (class in TestViews (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter), django\_elasticsearch\_dsl\_drf.tests.test\_views),  
257 267

TestFunctionalSuggesters (class in TestWrappers (class in  
django\_elasticsearch\_dsl\_drf.tests.test\_functional\_suggesters), django\_elasticsearch\_dsl\_drf.tests.test\_wrappers),  
259 268

TestHelpers (class in to\_dict() (django\_elasticsearch\_dsl\_drf.utils.DictionaryProxy  
django\_elasticsearch\_dsl\_drf.tests.test\_helpers), method), 274  
260 to\_dict() (django\_elasticsearch\_dsl\_drf.utils.EmptySearch

[method](#)), 274  
[to\\_internal\\_value\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.ListField](#) [method](#)), 166  
[to\\_internal\\_value\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ListField](#) [method](#)), 164  
[to\\_internal\\_value\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#) [method](#)), 163  
[to\\_internal\\_value\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#) [method](#)), 163 (in module [django\\_elasticsearch\\_dsl\\_drf.fields.helpers](#)), 161  
[to\\_internal\\_value\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.ObjectField](#) [method](#)), 167  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.BooleanField](#) [method](#)), 165  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.CharField](#) [method](#)), 165  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.BooleanField](#) [method](#)), 160  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.CharField](#) [method](#)), 161  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.DateField](#) [method](#)), 161  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.FloatField](#) [method](#)), 161  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.IntegerField](#) [method](#)), 161  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.common.IPAddressField](#) [method](#)), 161  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.DateField](#) [method](#)), 165  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.FloatField](#) [method](#)), 165  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.IntegerField](#) [method](#)), 166  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.IPAddressField](#) [method](#)), 166  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.ListField](#) [method](#)), 166  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ListField](#)

[method](#)), 164  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#) [method](#)), 163  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.fields.ObjectField](#) [method](#)), 167  
[to\\_representation\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.serializers.DocumentSerializer](#) [method](#)), 273  
[update\(\)](#) ([django\\_elasticsearch\\_dsl\\_drf.serializers.DocumentSerializer](#) [method](#)), 273  
[VersionsTest](#) (class in [django\\_elasticsearch\\_dsl\\_drf.tests.test\\_versions](#)), 267  
[Wrapper](#) (class in [django\\_elasticsearch\\_dsl\\_drf.wrappers](#)), 275