

---

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

*Release 0.22.2*

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

**Aug 28, 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>Demos</b>	<b>9</b>
4.1	Online demo . . . . .	9
4.2	Local demo . . . . .	9
<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 . . . . .	31
12.3.2.1.3	Publisher model . . . . .	31
12.3.2.1.4	Author model . . . . .	32
12.3.2.1.5	Tag model . . . . .	32
12.3.2.1.6	Book model . . . . .	33
12.3.2.2	Admin classes . . . . .	34
12.3.2.3	Create database tables . . . . .	35

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

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

12.7.1.5.1	Search	80
12.7.1.5.2	Filtering	81
12.7.1.5.3	Ordering	82
12.7.1.6	Ids filter	83
12.7.1.7	Faceted search	83
12.7.1.8	Post-filter	84
12.7.1.8.1	Sample view	84
12.7.1.8.2	Sample queries	86
12.7.1.9	Geo-spatial features	86
12.7.1.9.1	Filtering	87
12.7.1.9.2	Ordering	87
12.7.1.9.3	Geo-shape	87
12.7.1.10	Suggestions	89
12.7.1.10.1	Completion suggesters	89
12.7.1.10.1.1	Document definition	89
12.7.1.10.1.2	Serializer definition	91
12.7.1.10.1.3	ViewSet definition	91
12.7.1.10.1.4	Sample requests/responses	93
12.7.1.10.1.5	Suggestions on Array/List field	95
12.7.1.10.1.6	Sample requests/responses	95
12.7.1.10.1.7	Context suggesters	96
12.7.1.10.1.8	<i>category</i> context	96
12.7.1.10.1.9	<i>geo</i> context	97
12.7.1.10.2	Term and Phrase suggestions	99
12.7.1.10.2.1	Document definition	99
12.7.1.10.2.2	ViewSet definition	101
12.7.1.10.2.3	Sample requests/responses	105
12.7.1.10.2.4	Completion	105
12.7.1.10.2.5	Term	107
12.7.1.10.2.6	Phrase	108
12.7.1.11	Functional suggestions	108
12.7.1.11.1	Document definition	108
12.7.1.11.2	ViewSet definition	110
12.7.1.12	Highlighting	112
12.7.1.13	Pagination	114
12.7.1.13.1	Page number pagination	114
12.7.1.13.2	Limit/offset pagination	114
12.7.1.13.3	Customisations	115
12.8	Nested fields usage examples	115
12.8.1	Example app	116
12.8.1.1	Sample models	116
12.8.1.2	Sample document	121
12.8.1.2.1	Index definition	121
12.8.1.2.1.1	Settings	121
12.8.1.2.1.2	Document index	122
12.8.1.3	Sample serializer	125
12.8.1.4	Sample view	125
12.8.1.5	Usage example	128
12.8.1.5.1	Sample queries	128
12.8.1.5.1.1	Search	128
12.8.1.5.1.2	Nested filtering	128
12.8.1.5.1.3	Nested search	128
12.8.1.5.1.4	Sample models	129
12.8.1.5.1.5	Sample document	129

	12.8.1.5.1.6	Sample view . . . . .	131
	12.8.1.5.1.7	Sample request . . . . .	133
	12.8.1.5.1.8	Filtering . . . . .	133
	12.8.1.5.1.9	Ordering . . . . .	134
	12.8.1.6	Suggestions . . . . .	134
	12.8.1.7	Nested aggregations/facets . . . . .	134
12.9	More like this . . . . .		138
	12.9.1	Usage example . . . . .	138
		12.9.1.1 Sample document . . . . .	138
		12.9.1.2 Sample view . . . . .	140
		12.9.1.3 Sample request . . . . .	141
		12.9.1.4 Generated query . . . . .	141
		12.9.1.5 Options . . . . .	141
12.10	Global aggregations . . . . .		141
	12.10.1	Sample view . . . . .	142
	12.10.2	Sample request . . . . .	142
	12.10.3	Generated query . . . . .	142
	12.10.4	Sample response . . . . .	143
12.11	Configuration tweaks . . . . .		145
	12.11.1	Ignore certain Elasticsearch exceptions . . . . .	145
12.12	Source filtering backend . . . . .		145
12.13	Pagination . . . . .		146
	12.13.1	Page number pagination . . . . .	146
	12.13.2	Query friendly page number pagination . . . . .	146
	12.13.3	Limit/offset pagination . . . . .	147
		12.13.3.1 Customisations . . . . .	147
12.14	Indexing troubleshooting . . . . .		148
	12.14.1	Timeout . . . . .	148
	12.14.2	Chunk size . . . . .	148
	12.14.3	Use parallel indexing . . . . .	149
	12.14.4	Limit the number of items indexed at once . . . . .	149
12.15	FAQ . . . . .		150
	12.15.1	Questions and answers . . . . .	150
12.16	Demo . . . . .		151
	12.16.1	Run demo locally . . . . .	151
	12.16.2	Prerequisites . . . . .	152
12.17	frontend demo for django-elasticsearch-dsl-drf . . . . .		152
	12.17.1	Quick start . . . . .	152
		12.17.1.1 Install the django requirements . . . . .	152
		12.17.1.2 Install Elasticsearch requirements . . . . .	152
		12.17.1.3 Run Elasticsearch . . . . .	153
		12.17.1.4 Build Elasticsearch index . . . . .	153
		12.17.1.5 Install React requirements . . . . .	153
		12.17.1.6 Run Django . . . . .	153
		12.17.1.7 Run React demo app . . . . .	153
12.18	Release history and notes . . . . .		154
	12.18.1	0.22.2 . . . . .	154
	12.18.2	0.22.1 . . . . .	154
	12.18.3	0.22 . . . . .	154
	12.18.4	0.21 . . . . .	154
	12.18.5	0.20.9 . . . . .	155
	12.18.6	0.20.8 . . . . .	155
	12.18.7	0.20.7 . . . . .	155
	12.18.8	0.20.6 . . . . .	155

12.18.9 0.20.5 . . . . .	155
12.18.100.20.4 . . . . .	156
12.18.110.20.3 . . . . .	156
12.18.120.20.2 . . . . .	156
12.18.130.20.1 . . . . .	156
12.18.140.20 . . . . .	156
12.18.150.19 . . . . .	156
12.18.160.18 . . . . .	157
12.18.170.17.7 . . . . .	157
12.18.180.17.6 . . . . .	157
12.18.190.17.5 . . . . .	157
12.18.200.17.4 . . . . .	158
12.18.210.17.3 . . . . .	158
12.18.220.17.2 . . . . .	158
12.18.230.17.1 . . . . .	158
12.18.240.17 . . . . .	158
12.18.250.16.3 . . . . .	158
12.18.260.16.2 . . . . .	159
12.18.270.16.1 . . . . .	159
12.18.280.16 . . . . .	159
12.18.290.15.1 . . . . .	160
12.18.300.15 . . . . .	160
12.18.310.14 . . . . .	160
12.18.320.13.2 . . . . .	160
12.18.330.13.1 . . . . .	160
12.18.340.13 . . . . .	160
12.18.350.12 . . . . .	161
12.18.360.11 . . . . .	161
12.18.370.10 . . . . .	162
12.18.380.9 . . . . .	162
12.18.390.8.4 . . . . .	162
12.18.400.8.3 . . . . .	163
12.18.410.8.2 . . . . .	163
12.18.420.8.1 . . . . .	163
12.18.430.8 . . . . .	163
12.18.440.7.2 . . . . .	164
12.18.450.7.1 . . . . .	164
12.18.460.7 . . . . .	164
12.18.470.6.4 . . . . .	164
12.18.480.6.3 . . . . .	164
12.18.490.6.2 . . . . .	164
12.18.500.6.1 . . . . .	165
12.18.510.6 . . . . .	165
12.18.520.5.1 . . . . .	165
12.18.530.5 . . . . .	165
12.18.540.4.4 . . . . .	165
12.18.550.4.3 . . . . .	166
12.18.560.4.2 . . . . .	166
12.18.570.4.1 . . . . .	166
12.18.580.4 . . . . .	166
12.18.590.3.12 . . . . .	166
12.18.600.3.11 . . . . .	167
12.18.610.3.10 . . . . .	167
12.18.620.3.9 . . . . .	167



12.18.630.3.8	167
12.18.640.3.7	167
12.18.650.3.6	167
12.18.660.3.5	167
12.18.670.3.4	168
12.18.680.3.3	168
12.18.690.3.2	168
12.18.700.3.1	168
12.18.710.3	168
12.18.720.2.6	168
12.18.730.2.5	168
12.18.740.2.4	169
12.18.750.2.3	169
12.18.760.2.2	169
12.18.770.2.1	169
12.18.780.2	169
12.18.790.1.8	169
12.18.800.1.7	169
12.18.810.1.6	170
12.18.820.1.5	170
12.18.830.1.4	170
12.18.840.1.3	170
12.18.850.1.2	170
12.18.860.1.1	170
12.18.870.1	171
12.19 django_elasticsearch_dsl_drf package	171
12.19.1 Subpackages	171
12.19.1.1 django_elasticsearch_dsl_drf.fields package	171
12.19.1.1.1 Submodules	171
12.19.1.1.2 django_elasticsearch_dsl_drf.fields.common module	171
12.19.1.1.3 django_elasticsearch_dsl_drf.fields.helpers module	172
12.19.1.1.4 django_elasticsearch_dsl_drf.fields.nested_fields module	172
12.19.1.1.5 Module contents	173
12.19.1.2 django_elasticsearch_dsl_drf.filter_backends package	175
12.19.1.2.1 Subpackages	175
12.19.1.2.1.1 django_elasticsearch_dsl_drf.filter_backends.aggregations package	175
12.19.1.2.1.2 Submodules	175
12.19.1.2.1.3 django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations module	175
12.19.1.2.1.4 django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations module	175
12.19.1.2.1.5 django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations module	175
12.19.1.2.1.6 Module contents	175
12.19.1.2.1.7 django_elasticsearch_dsl_drf.filter_backends.filtering package	175
12.19.1.2.1.8 Submodules	175
12.19.1.2.1.9 django_elasticsearch_dsl_drf.filter_backends.filtering.common module	175
12.19.1.2.1.10 django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial module	183
12.19.1.2.1.11 django_elasticsearch_dsl_drf.filter_backends.filtering.ids module	188
12.19.1.2.1.12 django_elasticsearch_dsl_drf.filter_backends.filtering.nested module	190
12.19.1.2.1.13 django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter module	192

12.19.1.2.1.14	Module contents . . . . .	193
12.19.1.2.1.15	django_elasticsearch_dsl_drf.filter_backends.ordering package . .	209
12.19.1.2.1.16	Submodules . . . . .	209
12.19.1.2.1.17	django_elasticsearch_dsl_drf.filter_backends.ordering.common module . . . . .	209
12.19.1.2.1.18	django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial module . . . . .	212
12.19.1.2.1.19	Module contents . . . . .	214
12.19.1.2.1.20	django_elasticsearch_dsl_drf.filter_backends.search package . . .	218
12.19.1.2.1.21	Subpackages . . . . .	218
12.19.1.2.1.22	django_elasticsearch_dsl_drf.filter_backends.search.query_backends package . . . . .	218
12.19.1.2.1.23	Submodules . . . . .	218
12.19.1.2.1.24	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base module . . . . .	218
12.19.1.2.1.25	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match module . . . . .	219
12.19.1.2.1.26	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase module . . . . .	219
12.19.1.2.1.27	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix module . . . . .	220
12.19.1.2.1.28	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match module . . . . .	220
12.19.1.2.1.29	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested module . . . . .	221
12.19.1.2.1.30	django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string module . . . . .	222
12.19.1.2.1.31	Module contents . . . . .	223
12.19.1.2.1.32	Submodules . . . . .	227
12.19.1.2.1.33	django_elasticsearch_dsl_drf.filter_backends.search.base module .	227
12.19.1.2.1.34	django_elasticsearch_dsl_drf.filter_backends.search.compound module . . . . .	228
12.19.1.2.1.35	django_elasticsearch_dsl_drf.filter_backends.search.historical module . . . . .	228
12.19.1.2.1.36	django_elasticsearch_dsl_drf.filter_backends.search.multi_match module . . . . .	230
12.19.1.2.1.37	django_elasticsearch_dsl_drf.filter_backends.search.query_string module . . . . .	231
12.19.1.2.1.38	django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string module . . . . .	231
12.19.1.2.1.39	Module contents . . . . .	231
12.19.1.2.1.40	django_elasticsearch_dsl_drf.filter_backends.suggester package . .	235
12.19.1.2.1.41	Submodules . . . . .	235
12.19.1.2.1.42	django_elasticsearch_dsl_drf.filter_backends.suggester.functional module . . . . .	235
12.19.1.2.1.43	django_elasticsearch_dsl_drf.filter_backends.suggester.native module . . . . .	239
12.19.1.2.1.44	Module contents . . . . .	244
12.19.1.2.2	Submodules . . . . .	251
12.19.1.2.3	django_elasticsearch_dsl_drf.filter_backends.faceted_search module . . .	251
12.19.1.2.4	django_elasticsearch_dsl_drf.filter_backends.highlight module . . . . .	253
12.19.1.2.5	django_elasticsearch_dsl_drf.filter_backends.mixins module . . . . .	255
12.19.1.2.6	django_elasticsearch_dsl_drf.filter_backends.source module . . . . .	256
12.19.1.2.7	Module contents . . . . .	257

12.19.1.3	django_elasticsearch_dsl_drf.management package	258
12.19.1.3.1	Subpackages	258
12.19.1.3.1.1	django_elasticsearch_dsl_drf.management.commands package	258
12.19.1.3.1.2	Submodules	258
12.19.1.3.1.3	django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes module	258
12.19.1.3.1.4	Module contents	260
12.19.1.3.2	Module contents	260
12.19.1.4	django_elasticsearch_dsl_drf.tests package	260
12.19.1.4.1	Submodules	260
12.19.1.4.2	django_elasticsearch_dsl_drf.tests.base module	260
12.19.1.4.3	django_elasticsearch_dsl_drf.tests.data_mixins module	260
12.19.1.4.4	django_elasticsearch_dsl_drf.tests.test_faceted_search module	260
12.19.1.4.5	django_elasticsearch_dsl_drf.tests.test_filtering_common module	260
12.19.1.4.6	django_elasticsearch_dsl_drf.tests.test_filtering_geo_spatial module	260
12.19.1.4.7	django_elasticsearch_dsl_drf.tests.test_filtering_global_aggregations module	260
12.19.1.4.8	django_elasticsearch_dsl_drf.tests.test_filtering_nested module	260
12.19.1.4.9	django_elasticsearch_dsl_drf.tests.test_filtering_post_filter module	260
12.19.1.4.10	django_elasticsearch_dsl_drf.tests.test_functional_suggesters module	260
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	260
12.19.1.4.16	django_elasticsearch_dsl_drf.tests.test_pagination module	260
12.19.1.4.17	django_elasticsearch_dsl_drf.tests.test_pip_helpers module	260
12.19.1.4.18	django_elasticsearch_dsl_drf.tests.test_query_friendly_pagination module	260
12.19.1.4.19	django_elasticsearch_dsl_drf.tests.test_search module	260
12.19.1.4.20	django_elasticsearch_dsl_drf.tests.test_search_multi_match module	260
12.19.1.4.21	django_elasticsearch_dsl_drf.tests.test_search_simple_query_string module	260
12.19.1.4.22	django_elasticsearch_dsl_drf.tests.test_serializers module	260
12.19.1.4.23	django_elasticsearch_dsl_drf.tests.test_source module	260
12.19.1.4.24	django_elasticsearch_dsl_drf.tests.test_suggesters module	260
12.19.1.4.25	django_elasticsearch_dsl_drf.tests.test_versions module	260
12.19.1.4.26	django_elasticsearch_dsl_drf.tests.test_views module	261
12.19.1.4.27	django_elasticsearch_dsl_drf.tests.test_wrappers module	261
12.19.1.4.28	Module contents	261
12.19.2	Submodules	261
12.19.3	django_elasticsearch_dsl_drf.analyzers module	261
12.19.4	django_elasticsearch_dsl_drf.apps module	261
12.19.5	django_elasticsearch_dsl_drf.compat module	261
12.19.6	django_elasticsearch_dsl_drf.constants module	262
12.19.7	django_elasticsearch_dsl_drf.elasticsearch_helpers module	262
12.19.8	django_elasticsearch_dsl_drf.helpers module	262
12.19.9	django_elasticsearch_dsl_drf.pagination module	263
12.19.10	django_elasticsearch_dsl_drf.pip_helpers module	265
12.19.11	django_elasticsearch_dsl_drf.serializers module	265
12.19.12	django_elasticsearch_dsl_drf.utils module	266
12.19.13	django_elasticsearch_dsl_drf.versions module	267
12.19.14	django_elasticsearch_dsl_drf.viewsets module	267
12.19.15	django_elasticsearch_dsl_drf.wrappers module	268
12.19.16	Module contents	269

<b>13 Indices and tables</b>	<b>271</b>
<b>Python Module Index</b>	<b>273</b>
<b>Index</b>	<b>275</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.



## DOCUMENTATION

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

Make sure to read [FAQ](#).





## PREREQUISITES

- Django 2.2, 3.1 and 3.2.
- Python 3.6, 3.7, 3.8 and 3.9.
- Elasticsearch 6.x, 7.x. For older versions use `django-elasticsearch-dsl-drf` version 0.18.



## MAIN FEATURES AND HIGHLIGHTS

- *Dynamic serializer for Documents.*
- *Search filter backend.*
- *Ordering filter backend.*
- *Filtering filter backend* (big variety of native- and functional- query lookups, such as `gt`, `gte`, `lt`, `lte`, `endswith`, `contains`, `wildcard`, `exists`, `exclude`, `isnull`, `range`, `in`, `prefix` (same as `startswith`), `term` and `terms` is implemented).
- *Geo-spatial filtering filter backend* (the following filters implemented: `geo_distance`, `geo_polygon` and `geo_bounding_box`).
- *Geo-spatial ordering filter backend* (the following filters implemented: `geo_distance`).
- *Faceted search filter backend.*
- *Post-filter filter backend.*
- *Nested filtering filter backend.*
- *Highlight backend.*
- *Suggester filter backend.*
- *Functional suggester filter backend.*
- *Pagination (Page number and limit/offset pagination).*
- *Ids filter backend.*
- *Multi match search filter backend.*
- *Simple search query search filter backend.*
- *More-like-this support (detail action).*
- *Global aggregations support.*
- *Source filter backend.*

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



## 4.1 Online demo

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

## 4.2 Local demo

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

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

```
docker-compose up
```

It will set up:

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



## INSTALLATION

- (1) Install latest stable version from PyPI:

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

or latest stable version from GitHub:

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

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

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





## **QUICK START**

Perhaps the easiest way to get acquainted with `django-elasticsearch-dsl-drf` is to read the [\*quick start tutorial\*](#). See it as a guide of diving into integration of Elasticsearch with Django with very low knowledge entry level.



## TESTING

Project is covered with tests.

To test with all supported Python/Django versions type:

```
tox
```

To test against specific environment, type:

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

To test just your working environment type:

```
./runtests.py
```

To run a single test in your working environment type:

```
./runtests.py src/django_elasticsearch_dsl_drf/tests/test_filtering.py
```

Or:

```
./manage.py test django_elasticsearch_dsl_drf.tests.test_ordering
```

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

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

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

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



## WRITING DOCUMENTATION

Keep the following hierarchy.

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



**LICENSE**

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





**SUPPORT**

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



---

CHAPTER  
**ELEVEN**

---

**AUTHOR**

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



## PROJECT DOCUMENTATION

Contents:

### Table of Contents

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

## 12.1 Dependencies

### elasticsearch and elasticsearch-dsl

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

Current compatibility matrix is:

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

## django-elasticsearch-dsl

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

The following versions have been tested and work well together:

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

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

## Django/ Django REST Framework

Initial version of this package was written for [djangorestframework](#) 3.6.2.

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

Current compatibility matrix is:

Django	Django REST Framework
1.11	3.9.3
2.0	3.9.3
2.1	3.9.3
2.2	3.9.3
3.0	3.11.0

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

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

## 12.2 Installing Elasticsearch

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

---

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

---

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

### 12.2.1 Docker

Using `docker-compose` (persistent):

```
docker-compose up elasticsearch
```

#### 12.2.1.1 6.x

##### 6.3.2

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

##### 6.4.0

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

#### 12.2.1.2 7.x

##### 7.3.0

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

## 12.3 Quick start

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

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

Contents:

## Table of Contents

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



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

### 12.3.1 Installation

- (1) Install latest stable version from PyPI:

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

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

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

    # Django Elasticsearch integration
    'django_elasticsearch_dsl',

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

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

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

# Elasticsearch configuration
```

(continues on next page)

(continued from previous page)

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

## 12.3.2 Example app

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

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

To keep things separate, our Django models will reside in the books app. Elasticsearch documents and Django REST framework views will be defined in a search\_indexes app. Both of the apps should be added to the INSTALLED\_APPS.

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

### 12.3.2.1 Sample models

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

#### 12.3.2.1.1 Required imports

Imports required for model definition.

books/models/book.py

```
import json

from django.conf import settings
from django.db import models
from django.utils.translation import gettext, gettext_lazy as _
```

### 12.3.2.1.2 Book statuses

*books/models/book.py*

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

### 12.3.2.1.3 Publisher model

*books/models/book.py*

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

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

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

        ordering = ["id"]

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

(continues on next page)

(continued from previous page)

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

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

#### 12.3.2.1.4 Author model

*books/models/author.py*

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

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

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

        ordering = ["id"]

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

#### 12.3.2.1.5 Tag model

*books/models/tag.py*

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

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

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

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

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

### 12.3.2.1.6 Book model

*books/models/book.py*

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

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

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

        ordering = ["isbn"]

    def __str__(self):
        return self.title

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

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

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

(continues on next page)

(continued from previous page)

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

### 12.3.2.2 Admin classes

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

*books/admin.py*

```
from django.contrib import admin

from .models import *

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

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

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

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

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

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

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

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

### 12.3.2.3 Create database tables

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

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

### 12.3.2.4 Fill in some data

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

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

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

### 12.3.2.5 Sample document

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

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

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

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

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

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

#### 12.3.2.5.1 Required imports

*search\_indexes/documents/book.py*

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

from books.models import Book
```

### 12.3.2.5.2 Index definition

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

#### 12.3.2.5.2.1 Settings

---

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

---

*settings/base.py*

---

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

---

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

*settings/testing.py*

---

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

---

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

*settings/production.py*

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

#### 12.3.2.5.2.2 Document index

*search\_indexes/documents/book.py*

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

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

(continues on next page)



(continued from previous page)

```

    number_of_replicas=1
)

```

### 12.3.2.5.3 Custom analyzers

```

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

```

### 12.3.2.5.4 Document definition

*search\_indexes/documents/book.py*

```

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

)

publication_date = fields.DateField()

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

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

price = fields.FloatField()

pages = fields.IntegerField()

stock_count = fields.IntegerField()

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

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

    model = Book # The model associate with this Document

```

### 12.3.2.6 Syncing Django's database with Elasticsearch indexes

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

### 12.3.2.6.1 Full database sync

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

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

- (1) Create Elasticsearch indexes:

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

- (2) Sync the data:

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

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

### 12.3.2.6.2 Sample partial sync (using custom signals)

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

#### 12.3.2.6.2.1 Required imports

*search\_indexes/signals.py*

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

from django_elasticsearch_dsl.registries import registry
```

#### 12.3.2.6.2.2 Update book index on related model change

*search\_indexes/signals.py*

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

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

    if app_label == 'book':
        # If it is `books.Publisher` that is being updated.
        if model_name == 'publisher':
```

(continues on next page)

(continued from previous page)

```

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

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

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

```

### 12.3.2.6.2.3 Update book index on related model removal

*search\_indexes/signals.py*

```

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

    Updates Book document from index if related `books.Publisher`
    (`publisher`), `books.Author` (`authors`), `books.Tag` (`tags`) fields
    have been removed from database.
    """

    app_label = sender._meta.app_label
    model_name = sender._meta.model_name
    instance = kwargs['instance']

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

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

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

```

(continues on next page)

(continued from previous page)

```
registry.update(_instance)
# registry.delete(_instance, raise_on_error=False)
```

### 12.3.2.7 Sample serializer

At this step we're going to define a serializer to be used in the Django REST framework ViewSet. Content of the `search_indexes/serializers.py` file. Additionally, see the code comments.

#### 12.3.2.7.1 Required imports

*search\_indexes/serializers/book.py*

```
import json

from rest_framework import serializers
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from .documents import BookDocument
```

#### 12.3.2.7.2 Serializer definition

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

*search\_indexes/serializers/book.py*

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

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

        # Specify the correspondent document class
        document = BookDocument

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

(continues on next page)

(continued from previous page)

```
        'tags',  
    )
```

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

*search\_indexes/serializers/book.py*

```
class BookDocumentSerializer(serializers.Serializer):  
    """Serializer for the Book document."""  
  
    id = serializers.IntegerField(read_only=True)  
  
    title = serializers.CharField(read_only=True)  
    description = serializers.CharField(read_only=True)  
    summary = serializers.CharField(read_only=True)  
  
    publisher = serializers.CharField(read_only=True)  
    publication_date = serializers.DateField(read_only=True)  
    state = serializers.CharField(read_only=True)  
    isbn = serializers.CharField(read_only=True)  
    price = serializers.FloatField(read_only=True)  
    pages = serializers.IntegerField(read_only=True)  
    stock_count = serializers.IntegerField(read_only=True)  
    tags = serializers.SerializerMethodField()  
  
    class Meta:  
        """Meta options."""  
  
        # List the serializer fields. Note, that the order of the fields  
        # is preserved in the ViewSet.  
        fields = (  
            'id',  
            'title',  
            'description',  
            'summary',  
            'publisher',  
            'publication_date',  
            'state',  
            'isbn',  
            'price',  
            'pages',  
            'stock_count',  
            'tags',  
        )  
  
    def get_tags(self, obj):  
        """Get tags."""  
        if obj.tags:  
            return list(obj.tags)  
        else:  
            return []
```

### 12.3.2.8 ViewSet definition

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

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

#### 12.3.2.8.1 Required imports

*search\_indexes/viewsets/book.py*

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

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

#### 12.3.2.8.2 ViewSet definition

*search\_indexes/viewsets/book.py*

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

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)



(continued from previous page)

```

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

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

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

```

### 12.3.2.9 URLs

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

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

#### 12.3.2.9.1 Required imports

*search\_indexes/urls.py*

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

from .views import BookDocumentView
```

#### 12.3.2.9.2 Router definition

*search\_indexes/urls.py*

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

#### 12.3.2.9.3 URL patterns

*search\_indexes/urls.py*

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

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

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

#### 12.3.2.10.1 URLs

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

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

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

(continues on next page)

(continued from previous page)

```
# ...
]
```

### 12.3.2.10.2 Test in browser

Open the following URL in your browser.

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

Perform a number of lookups:

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

## 12.3.3 Development and debugging

### 12.3.3.1 Profiling tools

Looking for profiling tools for Elasticsearch?

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

#### 12.3.3.1.1 Installation

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

#### 12.3.3.1.2 Configuration

Change your development settings in the following way:

*settings/dev.py*

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

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

DEBUG_TOOLBAR_CONFIG = {
```

(continues on next page)

(continued from previous page)

```

    'INTERCEPT_REDIRECTS': False,
}

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

```

### 12.3.3.2 Debugging

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

## 12.4 Filter usage examples

Example usage of filtering backends.

Contents:

### Table of Contents

- *Filter usage examples*
  - *Search*
    - \* *Search in all fields*
    - \* *Search a single term on specific field*
    - \* *Search for multiple terms*
    - \* *Search for multiple terms in specific fields*
  - *Filtering*
    - \* *Supported lookups*
      - *Native*
      - *term*

- *terms*
- *range*
- *exists*
- *prefix*
- *wildcard*
- *ids*
- *Functional*
- *contains*
- *in*
- *gt*
- *gte*
- *lt*
- *lte*
- *startswith*
- *endswith*
- *isnull*
- *exclude*
- *Usage examples*

### 12.4.1 Search

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

Multiple search terms are joined with OR.

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

#### 12.4.1.1 Search in all fields

Search in all fields (name, address, city, state\_province and country) for word “reilly”.

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

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

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

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

#### 12.4.1.3 Search for multiple terms

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

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

#### 12.4.1.4 Search for multiple terms in specific fields

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

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

### 12.4.2 Filtering

#### 12.4.2.1 Supported lookups

##### 12.4.2.1.1 Native

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

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

#### 12.4.2.1.1.1 term

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

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

#### 12.4.2.1.1.2 terms

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

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

#### 12.4.2.1.1.3 range

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

**From, to**

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

**From, to, boost**

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

#### 12.4.2.1.1.4 exists

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

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

#### 12.4.2.1.1.5 prefix

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

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

#### 12.4.2.1.1.6 wildcard

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

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

Should match: *delusional insanity*.

#### 12.4.2.1.1.7 ids

Find documents with the specified type and IDs.

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

#### 12.4.2.1.2 Functional

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

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

##### 12.4.2.1.2.1 contains

Case-insensitive containment test.

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

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

##### 12.4.2.1.2.2 in

In a given list.

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

##### 12.4.2.1.2.3 gt

Greater than.

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



#### 12.4.2.1.2.4 gte

Greater than or equal to.

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

#### 12.4.2.1.2.5 lt

Less than.

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

#### 12.4.2.1.2.6 lte

Less than or equal to.

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

#### 12.4.2.1.2.7 startswith

Case-sensitive starts-with.

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

Should match: *biography*, *bio mechanics*

#### 12.4.2.1.2.8 endswith

Case-sensitive ends-with.

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

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

#### 12.4.2.1.2.9 isnull

Takes either True or False.

##### True

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

##### False

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

#### 12.4.2.1.2.10 exclude

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

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

### 12.4.3 Usage examples

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

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

Additionally, see:

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

## 12.5 Search backends

### 12.5.1 Compound search filter backend

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

#### 12.5.1.1 Multi-match

##### 12.5.1.1.1 Sample view

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

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookCompoundSearchBackendDocumentViewSet(DocumentViewSet):

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

(continues on next page)

(continued from previous page)

```

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

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

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

```

#### 12.5.1.1.2 Sample request

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

#### 12.5.1.1.3 Generated query

```

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

```

(continues on next page)

(continued from previous page)

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

### 12.5.1.2 Fuzzy search

#### 12.5.1.2.1 Sample view

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

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

#### 12.5.1.2.2 Sample request

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

#### 12.5.1.2.3 Generated query

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

(continues on next page)

(continued from previous page)

```

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

```

## 12.5.2 Multi match search filter backend

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

### 12.5.2.1 Sample view

```

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

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookMultiMatchSearchFilterBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

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

```

(continues on next page)

(continued from previous page)

```
# ...  
]  
  
multi_match_search_fields = {  
    'title': {'boost': 4},  
    'summary': {'boost': 2},  
    'description': None,  
}  
  
ordering = ('_score', 'id', 'title', 'price',)
```

### 12.5.2.2 Sample request

---

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

---

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

### 12.5.2.3 Generated query

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

#### 12.5.2.4 Options

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

Selective list of available options:

- `operator`
- `type`
- `analyzer`
- `tie_breaker`

##### 12.5.2.4.1 Type options

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

- `best_fields`
- `most_fields`
- `cross_fields`
- `phrase`
- `phrase_prefix`

##### Example

```
class BookMultiMatchSearchFilterBackendDocumentViewSet(DocumentViewSet):  
  
    # ...  
  
    multi_match_options = {  
        'type': 'phrase'  
    }
```

##### 12.5.2.4.2 Operator options

Can be either `and` or `or`.

### 12.5.3 Simple query string filter backend

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

### 12.5.3.1 Sample view

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

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookSimpleQueryStringSearchFilterBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

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

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

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

### 12.5.3.2 Sample request 1

---

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

---

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



### 12.5.3.3 Generated query 1

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

### 12.5.3.4 Sample request 2

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

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

### 12.5.3.5 Generated query 2

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

(continues on next page)

(continued from previous page)

```

    "id",
    "title",
    "price"
],
"from": 0,
"size": 2
}

```

### 12.5.3.6 Sample request 3

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

```

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

```

### 12.5.3.7 Generated query 3

```

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

```

### 12.5.3.8 Options

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

Selective list of available options:

- `default_operator`

#### 12.5.3.8.1 Default Operator options

Can be either `and` or `or`.

#### Example

```
class BookSimpleQueryStringSearchFilterBackendDocumentViewSet(DocumentViewSet):

    # ...

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

## 12.6 Basic usage examples

Basic Django REST framework integration example

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

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

Contents:

#### Table of Contents

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

- *Filtering*
- *Ordering*

## 12.6.1 Example app

### 12.6.1.1 Sample models

*books/models/publisher.py*

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

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

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

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

### 12.6.1.2 Sample document

*search\_indexes/documents/publisher.py*

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

from books.models import Publisher

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

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

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

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

(continues on next page)

(continued from previous page)

```
website = fields.StringField()

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

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

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

### 12.6.1.3 Sample serializer

*search\_indexes/serializers/book.py*

```
import json

from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

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

    location = serializers.SerializerMethodField()

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

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

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

### 12.6.1.4 Sample view

*search\_indexes/views/publisher.py*

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_GEO_DISTANCE,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    OrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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

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

    document = PublisherDocument
    serializer_class = PublisherDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'name',
        'info',
        'address',
        'city',
        'state_province',
        'country',
    )
    # Define filtering fields
    filter_fields = {
        'id': None,
        'name': 'name.raw',
        'city': 'city.raw',
        'state_province': 'state_province.raw',
        'country': 'country.raw',
    }
    # Define ordering fields
    ordering_fields = {
        'id': None,
        'name': None,
        'city': None,
        'country': None,
```

(continues on next page)

(continued from previous page)

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

### 12.6.1.5 Usage example

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

#### 12.6.1.5.1 Sample queries

##### 12.6.1.5.1.1 Search

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

Multiple search terms are joined with OR.

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

##### Search in all fields

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

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

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

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

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

##### Search for multiple terms

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

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

##### Search for multiple terms in specific fields

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

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



### 12.6.1.5.1.2 Filtering

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

Multiple filter terms are joined with AND.

#### Filter documents by single field

Filter documents by field (city) "yerevan".

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

#### Filter documents by multiple fields

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

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

#### Filter document by a single field

Filter documents by (field country) "Armenia".

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

#### Filter documents by multiple fields

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

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

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

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

If you want the same as above, but joined with AND, add \_\_term to each lookup.

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

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

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

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

#### Geo-distance filtering

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

```
http://127.0.0.1:8000/search/publishers/?location__geo_distance=100000km__12.04__-63.93
```

### 12.6.1.5.1.3 Ordering

The - prefix means ordering should be descending.

#### Order documents by field (ascending)

Filter documents by field `city` (ascending).

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

#### Order documents by field (descending)

Filter documents by field `country` (descending).

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

#### Order documents by multiple fields

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

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

## 12.7 Advanced usage examples

Advanced Django REST framework integration examples.

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

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

Contents:

### Table of Contents

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

- *Search*
- *Filtering*
- *Ordering*
- \* *Ids filter*
- \* *Faceted search*
- \* *Post-filter*
  - *Sample view*
  - *Sample queries*
- \* *Geo-spatial features*
  - *Filtering*
  - *Ordering*
  - *Geo-shape*
- \* *Suggestions*
  - *Completion suggesters*
  - *Document definition*
  - *Serializer definition*
  - *ViewSet definition*
  - *Sample requests/responses*
  - *Suggestions on Array/List field*
  - *Sample requests/responses*
  - *Context suggesters*
  - *category context*
  - *geo context*
  - *Term and Phrase suggestions*
  - *Document definition*
  - *ViewSet definition*
  - *Sample requests/responses*
  - *Completion*
  - *Term*
  - *Phrase*
- \* *Functional suggestions*
  - *Document definition*
  - *ViewSet definition*
- \* *Highlighting*
- \* *Pagination*

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

## 12.7.1 Example app

### 12.7.1.1 Sample models

*books/models/publisher.py*

```
import json

from django.conf import settings
from django.db import models
from django.utils.translation import gettext, gettext_lazy as _

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

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

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

(continues on next page)

(continued from previous page)

```

                                default=0)

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

    ordering = ["id"]

def __str__(self):
    return self.name

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

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

```

books/models/author.py

```

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

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

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

        ordering = ["id"]

def __str__(self):
    return self.name

```

books/models/tag.py

```

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

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

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

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

```

(continues on next page)

(continued from previous page)

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

books/models/book.py

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

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

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

        ordering = ["isbn"]

    def __str__(self):
        return self.title

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

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

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

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

### 12.7.1.2 Sample document

#### 12.7.1.2.1 Index definition

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

##### 12.7.1.2.1.1 Settings

*settings/base.py*

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

*settings/testing.py*

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

*settings/production.py*

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

##### 12.7.1.2.1.2 Document index

*search\_indexes/documents/book.py*

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

from books.models import Book

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

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

    publication_date = fields.DateField()

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

```

(continues on next page)



(continued from previous page)

```

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

price = fields.FloatField()

pages = fields.IntegerField()

stock_count = fields.IntegerField()

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

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

    model = Book # The model associate with this Document

```

### 12.7.1.3 Sample serializer

*search\_indexes/serializers/tag.py*

```

import json

from rest_framework import serializers

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

    title = serializers.CharField()

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

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

```

*search\_indexes/serializers/book.py*

```

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

```

(continues on next page)

(continued from previous page)

```
id = serializers.SerializerMethodField()

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

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

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

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

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

#### 12.7.1.4 Sample view

*search\_indexes/viewsets/book.py*

```
from django_elasticsearch_dsl_drf.constants import (
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_RANGE,
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_EXCLUDE,
)
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet

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

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

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'
    filter_backends = [
        FilteringFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
        SearchFilterBackend,
    ]
    # Define search fields
    search_fields = (
        'title',
        'summary',
        'description',
    )
    # Define filtering fields
    filter_fields = {
        'id': {
            'field': '_id',
            'lookups': [
                LOOKUP_FILTER_RANGE,
                LOOKUP_QUERY_IN,
            ],
        },
        'publisher': 'publisher.raw',
    }
```

(continues on next page)

(continued from previous page)

```

'publication_date': 'publication_date',
'isbn': 'isbn.raw',
'tags': {
    'field': 'tags',
    'lookups': [
        LOOKUP_FILTER_TERMS,
        LOOKUP_FILTER_PREFIX,
        LOOKUP_FILTER_WILDCARD,
        LOOKUP_QUERY_IN,
        LOOKUP_QUERY_EXCLUDE,
    ],
},
'tags.raw': {
    'field': 'tags.raw',
    'lookups': [
        LOOKUP_FILTER_TERMS,
        LOOKUP_FILTER_PREFIX,
        LOOKUP_FILTER_WILDCARD,
        LOOKUP_QUERY_IN,
        LOOKUP_QUERY_EXCLUDE,
    ],
},
}
# Define ordering fields
ordering_fields = {
    'id': 'id',
    'title': 'title.raw',
    'price': 'price.raw',
    'state': 'state.raw',
    'publication_date': 'publication_date',
}
# Specify default ordering
ordering = ('id', 'title',)

```

### 12.7.1.5 Usage example

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

#### 12.7.1.5.1 Search

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

Multiple search terms are joined with OR.

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

#### Search in all fields

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

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

### Search a single term on specific field

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

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

### Search for multiple terms

In order to search for multiple terms “education”, “technology” add multiple search query params.

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

### Search for multiple terms on specific fields

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

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

### Search with boosting

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

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

    # ...

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

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

    # ...
```

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

#### 12.7.1.5.2 Filtering

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

Multiple filter terms are joined with AND.

#### Filter documents by field

Filter documents by field (state) “published”.

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

### **Filter documents by multiple fields**

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

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

### **Filter document by a single field**

Filter documents by (field tag) “education”.

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

### **Filter documents by multiple fields**

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

```
http://127.0.0.1:8080/search/books/?tags__in=education__economy
```

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

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

If you want the same as above, but joined with AND, add \_\_term to each lookup.

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

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

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

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

## **12.7.1.5.3 Ordering**

The - prefix means ordering should be descending.

### **Order documents by field (ascending)**

Order documents by field price (ascending).

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

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

Order documents by field price (descending).

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

### **Order documents by multiple fields**

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

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

### 12.7.1.6 Ids filter

Filters documents that only have the provided ids.

```
http://127.0.0.1:8000/api/articles/?ids=68__64__58
```

Or, alternatively:

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

### 12.7.1.7 Faceted search

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

*search\_indexes/viewsets/book.py*

```
# ...

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

# ...

from elasticsearch_dsl import (
    DateHistogramFacet,
    RangeFacet,
    TermsFacet,
)

# ...

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

    # ...

    filter_backends = [
        # ...
        FacetedSearchFilterBackend,
    ]

    # ...

    faceted_search_fields = {
        'state': 'state.raw', # By default, TermsFacet is used
        'publisher': {
            'field': 'publisher.raw',
            'facet': TermsFacet, # But we can define it explicitly
            'enabled': True,
        },
        'publication_date': {
```

(continues on next page)

(continued from previous page)

```

        'field': 'publication_date',
        'facet': DateHistogramFacet,
        'options': {
            'interval': 'year',
        },
    },
    'pages_count': {
        'field': 'pages',
        'facet': RangeFacet,
        'options': {
            'ranges': [
                ("<10", (None, 10)),
                ("11-20", (11, 20)),
                ("20-50", (20, 50)),
                (">50", (50, None)),
            ]
        },
    },
}

# ...

```

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

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

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

### 12.7.1.8 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/viewssets/book.py*

```

# ...

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

# ...

```

(continues on next page)



(continued from previous page)

```

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):
        """
```

(continues on next page)

(continued from previous page)

```
Indexing circle geo_shape with 10km radius.
Used in Elasticsearch indexing/tests of `geo_shape` native filter.
"""

return {
    'type': 'circle',
    'coordinates': [self.latitude, self.longitude],
    'radius': '10km',
}
```

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

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

### Supported shapes & queries

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

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

Potentially supported but untested shapes are : multipoint and linestring

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

### Shapes intersects

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

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

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

### 12.7.1.10 Suggestions

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

---

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

---

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

---

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

---

#### 12.7.1.10.1 Completion suggesters

##### 12.7.1.10.1.1 Document definition

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

*search\_indexes/documents/publisher.py*

```
from django.conf import settings

from django_elasticsearch_dsl import Document, Index, fields

from books.models import Publisher

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

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

(continues on next page)

(continued from previous page)

```
info = fields.StringField()

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

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

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

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

website = fields.StringField()

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

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

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

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

### 12.7.1.10.1.2 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:
        """Meta options."""

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

### 12.7.1.10.1.3 ViewSet definition

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

---

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

---

*search\_indexes/viewsets/publisher.py*

```
# ...

from django_elasticsearch_dsl_drf.constants import SUGGESTER_COMPLETION
from django_elasticsearch_dsl_drf.filter_backends import (
    # ...
    SuggesterFilterBackend,
)

# ...

class PublisherDocumentViewSet(DocumentViewSet):
    """The PublisherDocument view."""
```

(continues on next page)

(continued from previous page)

```

document = PublisherDocument

# ...

filter_backends = [
    # ...
    SuggesterFilterBackend,
]

# ...

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

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

```

(continues on next page)



(continued from previous page)

}

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

#### 12.7.1.10.1.4 Sample requests/responses

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

##### Request

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

##### Response

```
{
  "_shards": {
    "failed": 0,
    "successful": 1,
    "total": 1
  },
  "country_suggest__completion": [
    {
      "options": [
        {
          "score": 1.0,
          "text": "Armenia"
        },
        {
          "score": 1.0,
          "text": "Argentina"
        }
      ],
      "offset": 0,
      "length": 2,
      "text": "Ar"
    }
  ]
}
```

You can also have multiple suggesters per request.

##### Request

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

##### Response

```
{
  "_shards": {
    "successful": 1,
    "total": 1,
```

(continues on next page)

(continued from previous page)

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

#### 12.7.1.10.1.5 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,
          "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),
            '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(

```

(continues on next page)

(continued from previous page)

```

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

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

    # ...

```

ViewSet should altered as follows:

### ViewSet definition

```

class BookFrontendDocumentViewSet(DocumentViewSet):

    # ...

    # Suggester fields
    suggester_fields = {
        'street_suggest_context': {
            'field': 'street.suggest_context',
            'default_suggester': SUGGESTER_COMPLETION,
            # We want to be able to filter the completion filter
            # results on the following params: tag, state and publisher.
            # We also want to provide the size value.
            # See the "https://www.elastic.co/guide/en/elasticsearch/
            # reference/6.1/suggester-context.html" for the reference.
            'completion_options': {
                'geo_filters': {
                    'title_suggest_loc': 'loc',
                },
            },
            'options': {
                'size': 10, # By default, number of results is 5.
                'skip_duplicates': True, # Whether duplicate suggestions should be
    ↪ filtered out.

```

(continues on next page)

(continued from previous page)

```

        },
    },
}

# ...

```

And finally we can narrow our suggestions as follows:

### Sample request

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

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

Same query with boosting (boost value 2.0):

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

## 12.7.1.10.2 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(

```

(continues on next page)

(continued from previous page)

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

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

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

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

    # Publication date
    publication_date = fields.DateField()

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

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

    # Price
    price = fields.FloatField()
```

(continues on next page)



(continued from previous page)

```

# Pages
pages = fields.IntegerField()

# Stock count
stock_count = fields.IntegerField()

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

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

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

    model = Book # The model associate with this Document

```

#### 12.7.1.10.2.2 ViewSet definition

---

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

---

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

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

URL shall be constructed in the following way:

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

Example for completion suggerer:

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

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

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

Example for term suggerer:

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

Example for phrase suggester:

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

*search\_indexes/viewsets/book.py*

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

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

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

```

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

### 12.7.1.10.2.3 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",

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

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

#### 12.7.1.10.2.5 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
        },
        {
          "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
```

(continues on next page)



(continued from previous page)

```

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

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

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

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

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

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

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

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

    # ...

    class Meta:

```

(continues on next page)

(continued from previous page)

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

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

### 12.7.1.11.2 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,
)
```

(continues on next page)

(continued from previous page)

```

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

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

    # ...

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

```

---

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

---

### 12.7.1.12 Highlighting

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

#### ViewSet definition

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

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

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

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

    # ...

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

    # ...
```

#### Request

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

(continues on next page)

(continued from previous page)

**Response**

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

(continues on next page)

(continued from previous page)

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

### 12.7.1.13 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)
        __data.append(
            ('current_page', int(self.request.query_params.get('page', 1)))
        )
        __data.append(
            ('page_size', self.get_page_size(self.request))
        )

        return sorted(__data)
```

Same applies to the customisations of the `LimitOffsetPagination`.

## 12.8 Nested fields usage examples

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

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

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

Contents:

### Table of Contents

- *Nested fields usage examples*
  - *Example app*
    - \* *Sample models*
    - \* *Sample document*
      - *Index definition*
      - *Settings*

- *Document index*
- \* *Sample serializer*
- \* *Sample view*
- \* *Usage example*
  - *Sample queries*
  - *Search*
  - *Nested filtering*
  - *Nested search*
  - *Sample models*
  - *Sample document*
  - *Sample view*
  - *Sample request*
  - *Filtering*
  - *Ordering*
- \* *Suggestions*
- \* *Nested aggregations/facets*

## 12.8.1 Example app

### 12.8.1.1 Sample models

*books/models/continent.py*

```
from django.db import models

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

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

(continues on next page)



(continued from previous page)

```

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

    ordering = ["id"]

def __str__(self):
    return self.name

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

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

```

books/models/country.py

```

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

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

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

        ordering = ["id"]

def __str__(self):

```

(continues on next page)

(continued from previous page)

```

    return self.name

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

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

```

books/models/city.py

```

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

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

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

        ordering = ["id"]

    def __str__(self):
        return self.name

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

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

```

(continues on next page)

(continued from previous page)

```

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

```

books/models/address.py

```

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

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

    street = models.CharField(max_length=255)
    house_number = models.CharField(max_length=60)
    appendix = models.CharField(max_length=30, null=True, blank=True)
    zip_code = models.CharField(max_length=60)
    city = models.ForeignKey('books.City', on_delete=models.CASCADE)
    latitude = models.DecimalField(
        null=True,
        blank=True,
        decimal_places=15,
        max_digits=19,
        default=0
    )
    longitude = models.DecimalField(
        null=True,
        blank=True,
        decimal_places=15,
        max_digits=19,
        default=0
    )

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

        ordering = ["id"]

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

    Example:

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

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

    return wrapper

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

    Example:

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

    :return:
    """

```

(continues on next page)

(continued from previous page)

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

return wrapper
```

### 12.8.1.2 Sample document

#### 12.8.1.2.1 Index definition

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

##### 12.8.1.2.1.1 Settings

*settings/base.py*

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

*settings/testing.py*

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

*settings/production.py*

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

### 12.8.1.2.1.2 Document index

*search\_indexes/documents/address.py*

```
from django.conf import settings

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

from books.models import Address

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

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

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

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

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

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

    house_number = StringField(analyzer=html_strip)

    appendix = StringField(analyzer=html_strip)

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

    model = Address # The model associate with this Document

```



### 12.8.1.3 Sample serializer

*search\_indexes/serializers/address.py*

```
from django_elasticsearch_dsl_drf.serializers import DocumentSerializer

from ..documents import AddressDocument

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

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

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

### 12.8.1.4 Sample view

*search\_indexes/viewsets/address.py*

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

from ..documents import AddressDocument
```

(continues on next page)

(continued from previous page)

```
from ..serializers import AddressDocumentSerializer

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

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

(continues on next page)

(continued from previous page)

```

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

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

```

(continues on next page)

(continued from previous page)

```
}

```

### 12.8.1.5 Usage example

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

#### 12.8.1.5.1 Sample queries

##### 12.8.1.5.1.1 Search

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

###### Search in all fields

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

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

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

*books/models/country.py*

```
from django.db import models

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

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

#### 12.8.1.5.1.5 Sample document

*documents/city.py*

```
from django.conf import settings

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

(continues on next page)

(continued from previous page)

```

from books.models import City

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

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

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

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

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

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

    info = StringField(analyzer=html_strip)

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

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

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

    model = City # The model associate with this Document

```

#### 12.8.1.5.1.6 Sample view

viewsets/city.py

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

    # Define ordering fields
    ordering_fields = {
        'id': None,
    }

```

(continues on next page)



(continued from previous page)

```

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

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

```

#### 12.8.1.5.1.7 Sample request

##### Request

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

#### 12.8.1.5.1.8 Filtering

##### Filter documents by field

Filter documents by field (city) “Dublin”.

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

##### Filter documents by multiple fields

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

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

#### 12.8.1.5.1.9 Ordering

The - prefix means ordering should be descending.

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

Order documents by field country (ascending).

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

#### 12.8.1.6 Suggestions

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

---

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

---

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

---

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

---

Suggest completion for field country.

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

Suggest completion for field city.

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

#### 12.8.1.7 Nested aggregations/facets

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

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

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

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

    faceted_search_param = 'nested_facet'

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

(continues on next page)

(continued from previous page)

```

:param request: Django REST framework request.
:type request: rest_framework.request.Request
:return: List of search query params.
:rtype: list
"""

query_params = request.query_params.copy()
return query_params.getlist(self.faceted_search_param, [])

def filter_queryset(self, request, queryset, view):
    """Filter the queryset.
    :param request: Django REST framework request.
    :param queryset: Base queryset.
    :param view: View.
    :type request: rest_framework.request.Request
    :type queryset: elasticsearch_dsl.search.Search
    :type view: rest_framework.viewsets.ReadOnlyModelViewSet
    :return: Updated queryset.
    :rtype: elasticsearch_dsl.search.Search
    """

    facets = self.get_faceted_search_query_params(request)

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

    return queryset

```

The view will look as follows:

*search\_indexes/viewsets/address.py*

```

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

```

## 12.9 More like this

More like this functionality.

### 12.9.1 Usage example

#### 12.9.1.1 Sample document

```

from django.conf import settings

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

from books.models import Book

from .analyzers import html_strip

INDEX = Index(settings.ELASTICSEARCH_INDEX_NAMES[__name__])

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

```

(continues on next page)

(continued from previous page)

```

        blocks={'read_only_allow_delete': False}
    )

@INDEX.doc_type
class BookDocument(Document):

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

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

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

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

    # ...

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

        model = Book # The model associate with this Document

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

```

### 12.9.1.2 Sample view

```
from django_elasticsearch_dsl_drf.filter_backends import (
    FilteringFilterBackend,
    IdsFilterBackend,
    OrderingFilterBackend,
    PostFilterFilteringFilterBackend,
    SearchFilterBackend,
)
from django_elasticsearch_dsl_drf.viewsets import (
    DocumentViewSet,
    MoreLikeThisMixin,
)
from .serializers import BookDocumentSerializer

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

    # ...

    document = BookDocument
    lookup_field = 'id'
    serializer_class = BookDocumentSerializer

    # ...

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

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



### 12.9.1.3 Sample request

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

### 12.9.1.4 Generated query

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

### 12.9.1.5 Options

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

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

## 12.10 Global aggregations

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

### 12.10.1 Sample view

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

from .documents import BookDocument
from .serializers import BookDocumentSerializer

class BookCompoundSearchBackendDocumentViewSet(DocumentViewSet):

    document = BookDocument
    serializer_class = BookDocumentSerializer
    lookup_field = 'id'

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

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

### 12.10.2 Sample request

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

### 12.10.3 Generated query

```
{
  "from": 0,
  "query": {
    "bool": {
      "filter": [
        {
          "terms": {
            "state.raw": [
```

(continues on next page)

(continued from previous page)

```

        "rejected"
    ]
}
},
"size": 25,
"aggs": {
    "_filter_state_global": {
        "aggs": {
            "state_global": {
                "terms": {
                    "field": "state.raw"
                }
            }
        },
        "global": {
        }
    }
},
"sort": [
    "id",
    "title",
    "price"
]
}

```

#### 12.10.4 Sample response

```

{
    "count": 25,
    "next": null,
    "previous": null,
    "facets": {
        "_filter_state_global": {
            "state_global": {
                "buckets": [
                    {
                        "doc_count": 29,
                        "key": "not_published"
                    },
                    {
                        "doc_count": 25,
                        "key": "in_progress"
                    },
                    {
                        "doc_count": 25,
                        "key": "rejected"
                    }
                ]
            }
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        },
        {
            "doc_count": 21,
            "key": "cancelled"
        },
        {
            "doc_count": 17,
            "key": "published"
        }
    ],
    "sum_other_doc_count": 0,
    "doc_count_error_upper_bound": 0
},
"doc_count": 117
}
},
"results": [
    {
        "id": 1007489,
        "title": "Cupiditate qui nulla itaque maxime impedit.",
        "description": null,
        "summary": "Aut recusandae architecto incidunt quaerat odio .",
        "authors": [
            "Evy Vermeulen",
            "Tycho Weijland",
            "Rik Zeldenrust"
        ],
        "publisher": "Overdijk Inc",
        "publication_date": "2014-02-28",
        "state": "rejected",
        "isbn": "978-0-15-184366-4",
        "price": 6.53,
        "pages": 82,
        "stock_count": 30,
        "tags": [
            "Trilogy"
        ],
        "highlight": {},
        "null_field": null,
        "score": null
    },
    # ...
]
}

```

## 12.11 Configuration tweaks

### 12.11.1 Ignore certain Elasticsearch exceptions

```
class BookIgnoreIndexErrorsDocumentViewSet(DocumentViewSet):

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

## 12.12 Source filtering backend

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

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

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

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

# Local article document definition
from .documents import ArticleDocument

# Local article document serializer
from .serializers import ArticleDocumentSerializer

class ArticleDocumentView(BaseDocumentViewSet):

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

To disable `_source` retrieval set to `False`:

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

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

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

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

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

---

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

---

## 12.13 Pagination

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

### 12.13.1 Page number pagination

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

Example:

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

### 12.13.2 Query friendly page number pagination

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

*search\_indexes/viewsets/book.py*

```
# ...

from django_elasticsearch_dsl_drf.pagination import QueryFriendlyPageNumberPagination

# ...

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

    # ...

    pagination_class = QueryFriendlyPageNumberPagination

    # ...
```

### 12.13.3 Limit/offset pagination

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

*search\_indexes/viewsets/book.py*

```
# ...

from django_elasticsearch_dsl_drf.pagination import LimitOffsetPagination

# ...

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

    # ...

    pagination_class = LimitOffsetPagination

    # ...
```

Example:

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

#### 12.13.3.1 Customisations

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

```
from django_elasticsearch_dsl_drf.pagination import PageNumberPagination

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

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

        return sorted(__data)
```

Same applies to the customisations of the `LimitOffsetPagination`.

## 12.14 Indexing troubleshooting

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

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

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

### 12.14.1 Timeout

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

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

*settings/indexing.py*

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

# Override the elasticsearch configuration and provide a custom timeout
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'localhost:9200',
        'timeout': 60, # Custom timeout
    },
}
```

Then rebuild your search index specifying the indexing settings:

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

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

### 12.14.2 Chunk size

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

Install it as follows:

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

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

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



### 12.14.3 Use parallel indexing

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

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

Install it as follows:

*barseghyanartur/django-elasticsearch-dsl fork*

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

*mjl/django-elasticsearch-dsl fork*

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

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

*yourapp/documents.py*

```
class LocationDocument(Document):

    # ...

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

        model = Location
        parallel_indexing = True
```

### 12.14.4 Limit the number of items indexed at once

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

*yourapp/documents.py*

```
class LocationDocument(Document):

    # ...

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

        model = Location
        queryset_pagination = 50
```

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

*settings/base.py*

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

*settings/indexing.py*

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

*yourapp/documents.py*

```
from django.conf import settings

# ...

class LocationDocument(Document):

    # ...

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

        model = Location
        queryset_pagination = settings.ELASTICSEARCH_DSL_QUERYSET_PAGINATION
```

## 12.15 FAQ

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

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

### 12.15.1 Questions and answers

#### Question

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

#### Answer

Yes. There are many ways doing this in Elasticsearch.

To mention a couple:

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

---

#### Question

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

#### Answer

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

---

### Question

How can I order search results overall relevance

### Answer

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

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

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

---

### Question

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

### Answer

It's documented [here](#).

---

### Question

How can I sync my database with Elasticsearch indexes.

### Answer

It's documented [here](#).

---

### Question

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

### Answer

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

---

## 12.16 Demo

### 12.16.1 Run demo locally

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

## 12.16.2 Prerequisites

- Python 3
- Docker

Grab and run Elasticsearch:

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

Grab and run the latest `django_elasticsearch_dsl_drf_demo_installer.sh`:

```
wget -O - https://raw.githubusercontent.com/barseghyanartur/django-elasticsearch-dsl-drf/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 0.22.2

2021-08-29

- Tested against Django 3.2.

### 12.18.2 0.22.1

2021-05-01

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

### 12.18.3 0.22

2021-03-26

---

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

---

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

### 12.18.4 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*.

## 12.18.5 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.

## 12.18.6 0.20.8

2020-04-10

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

## 12.18.7 0.20.7

2020-04-10

- Make `skip_duplicates` available for native suggester.

## 12.18.8 0.20.6

2020-04-04

- Basic implementation of RegExp field lookup.

## 12.18.9 0.20.5

2019-12-30

- Minor fixes.

### 12.18.10 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.

### 12.18.11 0.20.3

2019-09-20

- Testing the auxiliary versions module.

### 12.18.12 0.20.2

2019-08-30

- Minor improvements in test coverage.

### 12.18.13 0.20.1

2019-08-18

- Minor Elasticsearch 7.x compatibility fixes.

### 12.18.14 0.20

2019-08-17

- Adding Elasticsearch 7.x support.

### 12.18.15 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.



### 12.18.16 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.

### 12.18.17 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.

### 12.18.18 0.17.6

2019-04-08

- Minor fixes.
- Additions to the docs.

### 12.18.19 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.

### 12.18.20 0.17.4

2019-03-13

- Source backend.

### 12.18.21 0.17.3

2019-02-08

- Obey object permissions.

### 12.18.22 0.17.2

2019-01-07

- Add nested ordering.

### 12.18.23 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.

### 12.18.24 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.

### 12.18.25 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.

### 12.18.26 0.16.2

2018-09-21

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

### 12.18.27 0.16.1

2018-09-18

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

### 12.18.28 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.

### 12.18.29 0.15.1

2018-08-22

- More tests.
- Fixes in docs.

### 12.18.30 0.15

2018-08-10

- Global aggregations.

### 12.18.31 0.14

2018-08-06

- More like this support through detail action.

### 12.18.32 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).

### 12.18.33 0.13.1

2018-07-26

- Minor fix in suggesters on Elasticsearch 6.x.

### 12.18.34 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.

### 12.18.35 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.

### 12.18.36 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.

### 12.18.37 0.10

2018-07-06

- Elasticsearch 6.x support.
- Minor fixes.

### 12.18.38 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`.

### 12.18.39 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.

### 12.18.40 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.

### 12.18.41 0.8.2

2018-06-05

- Minor fixes.

### 12.18.42 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.

### 12.18.43 0.8

2018-06-01

---

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

---

---

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

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

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

## **12.18.44 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.

## **12.18.45 0.7.1**

2018-04-04

---

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

---

- Add query *boost* support for search fields.

## **12.18.46 0.7**

2018-03-08

---

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

---

- CoreAPI/CoreSchema support.

## **12.18.47 0.6.4**

2018-03-05

- Minor fix: explicitly use DocType in the ViewSets.

## **12.18.48 0.6.3**

2018-01-03

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

## **12.18.49 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.



### 12.18.50 0.6.1

2017-11-28

- Documentation fixes.

### 12.18.51 0.6

2017-11-28

- Added highlight backend.
- Added nested search functionality.

### 12.18.52 0.5.1

2017-10-18

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

### 12.18.53 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.

### 12.18.54 0.4.4

2017-10-02

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

### 12.18.55 0.4.3

2017-09-28

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

### 12.18.56 0.4.2

2017-09-28

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

### 12.18.57 0.4.1

2017-09-26

- Fixes in docs.

### 12.18.58 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).

### 12.18.59 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.

### 12.18.60 0.3.11

2017-09-21

- Added query argument to `more_like_this` helper.

### 12.18.61 0.3.10

2017-09-20

- Minor fixes.
- Simplified Elasticsearch version check.

### 12.18.62 0.3.9

2017-09-12

- Python 2.x compatibility fix.

### 12.18.63 0.3.8

2017-09-12

- Fixes tests on some environments.

### 12.18.64 0.3.7

2017-09-07

- Docs fixes.

### 12.18.65 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.

### 12.18.66 0.3.5

2017-08-24

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

### 12.18.67 0.3.4

2017-08-23

- Minor fixes in the ordering backend.

### 12.18.68 0.3.3

2017-07-13

- Minor fixes and improvements.

### 12.18.69 0.3.2

2017-07-12

- Minor fixes and improvements.

### 12.18.70 0.3.1

2017-07-12

- Minor Python2 fixes.
- Minor documentation fixes.

### 12.18.71 0.3

2017-07-11

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

### 12.18.72 0.2.6

2017-07-11

- Minor fixes.
- Fixes in documentation.

### 12.18.73 0.2.5

2017-07-11

- Fixes in documentation.

### 12.18.74 0.2.4

2017-07-11

- Fixes in documentation.

### 12.18.75 0.2.3

2017-07-11

- Fixes in documentation.

### 12.18.76 0.2.2

2017-07-11

- Fixes in documentation.

### 12.18.77 0.2.1

2017-07-11

- Fixes in documentation.

### 12.18.78 0.2

2017-07-11

- Initial faceted search support.
- Pagination support.

### 12.18.79 0.1.8

2017-06-26

- Python2 fixes.
- Documentation and example project improvements.

### 12.18.80 0.1.7

2017-06-25

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

### 12.18.81 0.1.6

2017-06-23

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

### 12.18.82 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.

### 12.18.83 0.1.4

2017-06-22

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

### 12.18.84 0.1.3

2017-06-21

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

### 12.18.85 0.1.2

2017-06-20

- Minor fixes in tests.

### 12.18.86 0.1.1

2017-06-20

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

## 12.18.87 0.1

2017-06-19

- Initial beta release.

## 12.19 django\_elasticsearch\_dsl\_drf package

### 12.19.1 Subpackages

#### 12.19.1.1 django\_elasticsearch\_dsl\_drf.fields package

##### 12.19.1.1.1 Submodules

##### 12.19.1.1.2 django\_elasticsearch\_dsl\_drf.fields.common module

Common fields.

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

Bases: `rest_framework.fields.BooleanField`

Object field.

```
get_value(dictionary)
```

Get value.

```
to_representation(value)
```

To representation.

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

Bases: `rest_framework.fields.CharField`

Object field.

```
get_value(dictionary)
```

Get value.

```
to_representation(value)
```

To representation.

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

Bases: `rest_framework.fields.DateField`

Object field.

```
get_value(dictionary)
```

Get value.

```
to_representation(value)
```

To representation.

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

Bases: `rest_framework.fields.FloatField`

Object field.

```
get_value(dictionary)
```

Get value.

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.common.**IPAddressField**(\*args, \*\*kwargs)

Bases: rest\_framework.fields.IPAddressField

Object field.

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

Get value.

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.common.**IntegerField**(\*args, \*\*kwargs)

Bases: rest\_framework.fields.IntegerField

Object field.

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

Get value.

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

To representation.

#### 12.19.1.1.3 django\_elasticsearch\_dsl\_drf.fields.helpers module

Helpers.

django\_elasticsearch\_dsl\_drf.fields.helpers.**to\_representation**(*value*)

To representation.

#### 12.19.1.1.4 django\_elasticsearch\_dsl\_drf.fields.nested\_fields module

Nested fields.

**class** django\_elasticsearch\_dsl\_drf.fields.nested\_fields.**GeoPointField**(\*args, \*\*kwargs)

Bases: [django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#)

Geo point field.

**class** django\_elasticsearch\_dsl\_drf.fields.nested\_fields.**GeoShapeField**(\*args, \*\*kwargs)

Bases: [django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#)

Geo shape field.

**class** django\_elasticsearch\_dsl\_drf.fields.nested\_fields.**ListField**(\*args, \*\*kwargs)

Bases: rest\_framework.fields.Field

List field.

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

Get value.

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

To internal value.

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

To representation.



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

Nested field.

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

Object field.

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

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

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

#### 12.19.1.1.5 Module contents

Fields.

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

Object field.

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

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

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

Object field.

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

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

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

Object field.

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

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

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

Object field.

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

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**GeoPointField**(\*args, \*\*kwargs)

Bases: [django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#)

Geo point field.

**class** django\_elasticsearch\_dsl\_drf.fields.**GeoShapeField**(\*args, \*\*kwargs)

Bases: [django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#)

Geo shape field.

**class** django\_elasticsearch\_dsl\_drf.fields.**IPAddressField**(\*args, \*\*kwargs)

Bases: `rest_framework.fields.IPAddressField`

Object field.

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

Get value.

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**IntegerField**(\*args, \*\*kwargs)

Bases: `rest_framework.fields.IntegerField`

Object field.

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

Get value.

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**ListField**(\*args, \*\*kwargs)

Bases: `rest_framework.fields.Field`

List field.

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

Get value.

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

To internal value.

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

To representation.

**class** django\_elasticsearch\_dsl\_drf.fields.**NestedField**(\*args, \*\*kwargs)

Bases: [django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_fields.ObjectField](#)

Nested field.

**class** django\_elasticsearch\_dsl\_drf.fields.**ObjectField**(\*args, \*\*kwargs)

Bases: `rest_framework.fields.Field`

Object field.

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

Get value.

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

To internal value.

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

## 12.19.1.2 django\_elasticsearch\_dsl\_drf.filter\_backends package

### 12.19.1.2.1 Subpackages

#### 12.19.1.2.1.1 django\_elasticsearch\_dsl\_drf.filter\_backends.aggregations package

##### 12.19.1.2.1.2 Submodules

##### 12.19.1.2.1.3 django\_elasticsearch\_dsl\_drf.filter\_backends.aggregations.bucket\_aggregations module

##### 12.19.1.2.1.4 django\_elasticsearch\_dsl\_drf.filter\_backends.aggregations.metrics\_aggregations module

##### 12.19.1.2.1.5 django\_elasticsearch\_dsl\_drf.filter\_backends.aggregations.pipeline\_aggregations module

##### 12.19.1.2.1.6 Module contents

##### 12.19.1.2.1.7 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering package

##### 12.19.1.2.1.8 Submodules

##### 12.19.1.2.1.9 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common module

Common filtering backend.

**class**

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

Filtering filter backend for Elasticsearch.

Example:

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

(continues on next page)

(continued from previous page)

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

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

Apply *prefix* filter.

Syntax:

`/endpoint/?field_name__prefix={value}`

Example:

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

**Parameters**

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

**Returns** Modified queryset.

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

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

Apply *range* filter.

Syntax:

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

Example:

[http://localhost:8000/api/users/?age\\_\\_range=16\\_\\_67\\_\\_2.0](http://localhost:8000/api/users/?age__range=16__67__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 queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

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

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

Apply *terms* filter.

Syntax:

/endpoint/?field\_name\_\_terms={ value1 }\_\_ { value2 } /endpoint/?field\_name\_\_terms={ value1 }

Note, that number of values is not limited.

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *contains* filter.

Syntax:

/endpoint/?field\_name\_\_contains={ value }

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *endswith* filter.

Syntax:

/endpoint/?field\_name\_\_endswith={ value }

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *exclude* functional query.

Syntax:

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

Note, that number of values is not limited.

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *exists* filter.

Syntax:

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

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *gt* functional query.

Syntax:

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

Example:

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

#### Parameters

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

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

**Returns** Modified queryset.

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

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

Apply *gte* functional query.

Syntax:

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

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *in* functional query.

Syntax:

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

Note, that number of values is not limited.

Example:

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

#### Parameters

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

**Returns** Modified queryset.

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

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

Apply *isnull* functional query.

Syntax:

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

Example:

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



**Parameters**

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

**Returns** Modified queryset.

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

**classmethod** `apply_query_lt(queryset, options, value)`

Apply *lt* functional query.

Syntax:

`/endpoint/?field_name__lt={value}__{boost} /endpoint/?field_name__lt={value}`

Example:

`http://localhost:8000/api/articles/?id__lt=1__2.0 http://localhost:8000/api/articles/?id__lt=1`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_lte(queryset, options, value)`

Apply *lte* functional query.

Syntax:

`/endpoint/?field_name__lte={value}__{boost} /endpoint/?field_name__lte={value}`

Example:

`http://localhost:8000/api/articles/?id__lte=1__2.0 http://localhost:8000/api/articles/?id__lte=1`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_wildcard(queryset, options, value)`

Apply *wildcard* filter.

Syntax:

`/endpoint/?field_name__wildcard={value}* /endpoint/?field_name__wildcard={value} /end-point/?field_name__wildcard={value}*`

Example:

`http://localhost:8000/api/articles/?tags__wildcard=child*`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**get\_coreschema\_field** (*field*)

**get\_filter\_query\_params** (*request, view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** `dict`

**classmethod get\_gte\_lte\_params** (*value, lookup*)

Get params for *gte*, *gt*, *lte* and *lt* query.

Syntax:

`/endpoint/?field_name__gt={lower}__{boost} /endpoint/?field_name__gt={lower}`

Example:

`http://localhost:8000/api/articles/?id__gt=1 http://localhost:8000/api/articles/?id__gt=1__2.0`

**Parameters**

- **value** (*str*) –
- **lookup** (*str*) –

**Returns** Params to be used in *range* query.

**Return type** `dict`

**classmethod** `get_range_params(value)`

Get params for *range* query.

Syntax:

```
/endpoint/?field_name__range={lower}__{upper}__{boost} /end-
point/?field_name__range={lower}__{upper}
```

Example:

```
http://localhost:8000/api/users/?age__range=16__67__2.0 http://localhost:8000/api/users/?age__
_range=16__67 http://localhost:8000/api/users/?age__range=16
```

**Parameters** `value` –

**Type** `str`

**Returns** Params to be used in *range* query.

**Return type** `dict`

**get\_schema\_fields**(*view*)

**classmethod** `prepare_filter_fields(view)`

Prepare filter fields.

**Parameters** `view` (`rest_framework.viewsets.ReadOnlyModelViewSet`) –

**Returns** Filtering options.

**Return type** `dict`

#### 12.19.1.2.1.10 `django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial` module

Geo spatial filtering backend.

Elasticsearch supports two types of geo data:

- `geo_point` fields which support lat/lon pairs
- `geo_shape` fields, which support points, lines, circles, polygons, multi-polygons etc.

The queries in this group are:

- `geo_shape` query: Find document with geo-shapes which either intersect, are contained by, or do not intersect with the specified geo-shape.
- `geo_bounding_box` query: Finds documents with geo-points that fall into the specified rectangle.
- `geo_distance` query: Finds document with geo-points within the specified distance of a central point.
- `geo_distance_range` query: Like the `geo_distance` query, but the range starts at a specified distance from the central point. Note, that this one is deprecated and this isn't implemented.
- `geo_polygon` query: Find documents with geo-points within the specified polygon.

**class** `django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.`

**GeoSpatialFilteringFilterBackend**

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Geo-spatial filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_GEO_DISTANCE,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialFilteringFilterBackend,]
>>>     geo_spatial_filter_fields = {
>>>         'loc': 'location',
>>>         'location': {
>>>             'field': 'location',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_GEO_DISTANCE,
>>>             ],
>>>         }
>>> }
```

**classmethod** `apply_query_geo_bounding_box(queryset, options, value)`

Apply `geo_bounding_box` query.

**Parameters**

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (`dict`) – Filter options.
- **value** (`str`) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_geo_distance(queryset, options, value)`

Apply `geo_distance` query.

**Parameters**

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (`dict`) – Filter options.
- **value** (`str`) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_geo_polygon(queryset, options, value)`

Apply *geo\_polygon* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_geo_shape(queryset, options, value)`

Apply *geo\_shape* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_filter\_query\_params** (*request, view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** *dict*

**classmethod** `get_geo_bounding_box_params(value, field)`

Get params for *geo\_bounding\_box* query.

Example:

`/api/articles/?location__geo_bounding_box=40.73,-74.1__40.01,-71.12`

Example:

`/api/articles/?location__geo_polygon=40.73,-74.1 __40.01,-71.12 __name,myname __validation_method,IGNORE_MALFORMED __type,indexed`

Elasticsearch:

```
{
  "query": {
    "bool": [{
      "must": [{ "match_all": {} }],
      "filter": {
        "geo_bounding_box": [{
          "person.location": [{
            "top_left": [{ "lat": 40.73, "lon": -74.1
            }, "bottom_right": {
              "lat": 40.01, "lon": -71.12
            }
          }
        ]
      }
    ]
  }
}
```

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_bounding\_box* query.

**Return type** dict

**classmethod** **get\_geo\_distance\_params**(*value, field*)

Get params for *geo\_distance* query.

Example:

`/api/articles/?location__geo_distance=2km__43.53__-12.23`

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** dict

**classmethod** **get\_geo\_polygon\_params**(*value, field*)

Get params for *geo\_polygon* query.

Example:

`/api/articles/?location__geo_polygon=40,-70__30,-80__20,-90`

Example:

`/api/articles/?location__geo_polygon=40,-70 __30,-80 __20,-90 __name,myname __validation_method,IGNORE_MALFORMED`

Elasticsearch:

```
{
    "query": {
        "bool": [{
            "must": [{ "match_all": {} }],
            "filter": {
                "geo_polygon": [{
                    "person.location": [{
                        "points": [[ { "lat": 40, "lon": -70 }, { "lat": 30, "lon": -80 }, { "lat": 20, "lon": -90 } ]
                    }
                ]
            }
        ]
    }
}
```

#### Parameters

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** dict

**classmethod** `get_geo_shape_params(value, field)`

Get params for *geo\_shape* query.

Example:

```
/search/publishers/?location__geo_shape=48.9864453,6.37977 __relation,intersects
__type,circle __radius,20km
```

Example:

```
/search/publishers/?location__geo_shape=48.906254,6.378593 __48.985850,6.479359
__relation,within __type,envelope
```

Elasticsearch:

```
{
    "query": {
        "bool": [{
            "must": [{ "match_all": {} }],
            "filter": {
                "geo_shape": [{
                    "location": [{
                        "shape": { "type": "circle", "coordinates": [48.9864453,
                            6.37977], "radius": "20km"
                        }, "relation": "intersects"
                    }
                ]
            }
        ]
    }
}
```

```

        }
    }
}
}
}

```

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_shape* query.

**Return type** dict

**classmethod** **prepare\_filter\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

#### 12.19.1.2.1.11 `django_elasticsearch_dsl_drf.filter_backends.filtering.ids` module

Ids filtering backend.

Filters documents that only have the provided ids. Note, this query uses the *\_uid* field.

Elastic query:

```

{
  "query": {
    "ids": { "type": "book_document", "values": ["68", "64", "58"]
    }
  }
}

```

REST framework request equivalent:

- [http://localhost:8000/api/articles/?ids=68\\_\\_64\\_\\_58](http://localhost:8000/api/articles/?ids=68__64__58)
- <http://localhost:8000/api/articles/?ids=68&ids=64&ids=58>

Official Elastic docs:

- <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-ids-query.html>

**class** `django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Ids filter backend for Elasticsearch.

Example:



```

>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     IdsFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [IdsFilterBackend]

```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ids\_query\_params**(*request*)

Get search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**get\_ids\_values**(*request, view*)

Get ids values for query.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**ids\_query\_param** = 'ids'

### 12.19.1.2.1.12 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested module

Nested filtering backend.

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.

**NestedFilteringFilterBackend**

Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend*

Nested filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     NestedFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [NestedFilteringFilterBackend,]
>>>     nested_filter_fields = {
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_TERM,
>>>                 LOOKUP_FILTER_TERMS,
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>         }
>>>     }
```

**classmethod** apply\_filter(*queryset, options=None, args=None, kwargs=None*)

Apply filter.

#### Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

#### Returns

**classmethod** **apply\_query**(*queryset, options=None, args=None, kwargs=None*)

Apply query.

#### Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

#### Returns

**get\_coreschema\_field**(*field*)

**get\_filter\_field\_nested\_path**(*filter\_fields, field\_name*)

Get filter field path to be used in nested query.

#### Parameters

- **filter\_fields** –
- **field\_name** –

#### Returns

**get\_filter\_query\_params**(*request, view*)

Get query params to be filtered on.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**get\_schema\_fields**(*view*)

**classmethod** **prepare\_filter\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

### 12.19.1.2.1.13 django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter module

The post\_filter filtering backend.

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.

**PostFilterFilteringFilterBackend**

Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend*

The post\_filter filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     PostFilterFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [PostFilterFilteringFilterBackend,]
>>>     post_filter_fields = {
>>>         'title': 'title.raw',
>>>         'state': {
>>>             'field': 'state.raw',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>         }
>>> }
```

**classmethod** **apply\_filter**(*queryset, options=None, args=None, kwargs=None*)

Apply filter.

**Parameters**

- **queryset** –

- **options** –
- **args** –
- **kwargs** –

**Returns**

**classmethod** **apply\_query**(*queryset, options=None, args=None, kwargs=None*)

Apply query.

**Parameters**

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

**Returns**

**get\_coreschema\_field**(*field*)

**get\_schema\_fields**(*view*)

**classmethod** **prepare\_filter\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

#### 12.19.1.2.1.14 Module contents

Term level filtering and post\_filter backends.

**class** `django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend`  
**Bases:** `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
```

(continues on next page)

(continued from previous page)

```
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [FilteringFilterBackend,]
>>>     filter_fields = {
>>>         'title': 'title.raw',
>>>         'state': {
>>>             'field': 'state.raw',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>             'default_lookup': LOOKUP_FILTER_WILDCARD,
>>>         }
>>>     }
```

**classmethod** `apply_filter_prefix(queryset, options, value)`

Apply *prefix* filter.

Syntax:

`/endpoint/?field_name__prefix={value}`

Example:

`http://localhost:8000/api/articles/?tags__prefix=bio`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_filter_range(queryset, options, value)`

Apply *range* filter.

Syntax:

`/endpoint/?field_name__range={lower}__{upper}__{boost}` `/end-`  
`point/?field_name__range={lower}__{upper}`

Example:

`http://localhost:8000/api/users/?age__range=16__67__2.0` `http://localhost:8000/api/users/`  
`?age__range=16__67` `http://localhost:8000/api/users/?age__range=16`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_filter_regexp(queryset, options, value)`

Apply *regexp* filter.

Syntax:

`/endpoint/?field_name__regexp={regexp}`

Example:

`http://localhost:8000/api/users/?age__regexp=1[{}6-9]`      `http://localhost:8000/api/users/?age__regexp=2.*`

#### Parameters

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_filter_term(queryset, options, value)`

Apply *term* filter.

Syntax:

`/endpoint/?field_name={value}`

Example:

`http://localhost:8000/api/articles/?tags=children`

#### Parameters

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_filter_terms(queryset, options, value)`

Apply *terms* filter.

Syntax:

`/endpoint/?field_name__terms={value1}__{value2} /endpoint/?field_name__terms={value1}`

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__terms=children__python`      `http://localhost:8000/api/articles/?tags__terms=children`

#### Parameters

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*mixed: either str or iterable (list, tuple)*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_contains(queryset, options, value)`

Apply *contains* filter.

Syntax:

`/endpoint/?field_name__contains={value}`

Example:

[http://localhost:8000/api/articles/?state\\_\\_contains=lis](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](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__python) [http://localhost:8000/api/articles/?tags\\_\\_exclude=children](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=true) [http://localhost:8000/api/articles/?tags\\_\\_exists=false](http://localhost:8000/api/articles/?tags__exists=false)

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.



- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_gt(queryset, options, value)`

Apply *gt* functional query.

Syntax:

`/endpoint/?field_name__gt={value}__{boost} /endpoint/?field_name__gt={value}`

Example:

`http://localhost:8000/api/articles/?id__gt=1__2.0 http://localhost:8000/api/articles/?id__gt=1`

**Parameters**

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_gte(queryset, options, value)`

Apply *gte* functional query.

Syntax:

`/endpoint/?field_name__gte={value}__{boost} /endpoint/?field_name__gte={value}`

Example:

`http://localhost:8000/api/articles/?id__gte=1__2.0 http://localhost:8000/api/articles/?id__gte=1`

**Parameters**

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_in(queryset, options, value)`

Apply *in* functional query.

Syntax:

`/endpoint/?field_name__in={value1}__{value2} /endpoint/?field_name__in={value1}`

Note, that number of values is not limited.

Example:

`http://localhost:8000/api/articles/?tags__in=children__python`

**Parameters**

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_query_isnull(queryset, options, value)`

Apply *isnull* functional query.

Syntax:

`/endpoint/?field_name__isnull=true /endpoint/?field_name__isnull=false`

Example:

`http://localhost:8000/api/articles/?tags__isnull=true http://localhost:8000/api/articles/?tags__isnull=false`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_lt(queryset, options, value)`

Apply *lt* functional query.

Syntax:

`/endpoint/?field_name__lt={value}__{boost} /endpoint/?field_name__lt={value}`

Example:

`http://localhost:8000/api/articles/?id__lt=1__2.0 http://localhost:8000/api/articles/?id__lt=1`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_lte(queryset, options, value)`

Apply *lte* functional query.

Syntax:

`/endpoint/?field_name__lte={value}__{boost} /endpoint/?field_name__lte={value}`

Example:

`http://localhost:8000/api/articles/?id__lte=1__2.0 http://localhost:8000/api/articles/?id__lte=1`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_wildcard(queryset, options, value)`

Apply *wildcard* filter.

Syntax:

`/endpoint/?field_name__wildcard={value}* /endpoint/?field_name__wildcard={value}*  
/endpoint/?field_name__wildcard={value}* /endpoint/?field_name__wildcard={value}*`

Example:

`http://localhost:8000/api/articles/?tags__wildcard=child*`

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_coreschema\_field**(*field*)

**get\_filter\_query\_params**(*request, view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** *dict*

**classmethod get\_gte\_lte\_params**(*value, lookup*)

Get params for *gte*, *gt*, *lte* and *lt* query.

Syntax:

`/endpoint/?field_name__gt={lower}__{boost} /endpoint/?field_name__gt={lower}`

Example:

`http://localhost:8000/api/articles/?id__gt=1 http://localhost:8000/api/articles/?id__gt=1__2.0`

**Parameters**

- **value** (*str*) –
- **lookup** (*str*) –

**Returns** Params to be used in *range* query.

**Return type** *dict*

**classmethod get\_range\_params**(*value*)

Get params for *range* query.

Syntax:

`/endpoint/?field_name__range={lower}__{upper}__{boost} /end-point/?field_name__range={lower}__{upper}`

Example:

```
http://localhost:8000/api/users/?age__range=16__67__2.0 http://localhost:8000/api/users/
?age__range=16__67 http://localhost:8000/api/users/?age__range=16
```

**Parameters value** –

**Type** str

**Returns** Params to be used in *range* query.

**Return type** dict

**get\_schema\_fields**(view)

**classmethod prepare\_filter\_fields**(view)

Prepare filter fields.

**Parameters view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**class**

`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Geo-spatial filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_GEO_DISTANCE,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialFilteringFilterBackend,]
>>>     geo_spatial_filter_fields = {
>>>         'loc': 'location',
>>>         'location': {
>>>             'field': 'location',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_GEO_DISTANCE,
>>>             ],
>>>         },
>>>     }
```

**classmethod** `apply_query_geo_bounding_box(queryset, options, value)`

Apply *geo\_bounding\_box* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_geo_distance(queryset, options, value)`

Apply *geo\_distance* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_geo_polygon(queryset, options, value)`

Apply *geo\_polygon* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_query_geo_shape(queryset, options, value)`

Apply *geo\_shape* query.

**Parameters**

- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_filter\_query\_params** (*request, view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod** `get_geo_bounding_box_params(value, field)`

Get params for *geo\_bounding\_box* query.

Example:

`/api/articles/?location__geo_bounding_box=40.73,-74.1__40.01,-71.12`

Example:

`/api/articles/?location__geo_polygon=40.73,-74.1 __40.01,-71.12 __name,myname  
__validation_method,IGNORE_MALFORMED __type,indexed`

Elasticsearch:

```
{
  "query": {
    "bool": [{
      "must": [{ "match_all": {} }],
      "filter": {
        "geo_bounding_box": [{
          "person.location": [{
            "top_left": [{"lat": 40.73, "lon": -74.1},
            "bottom_right": {
              "lat": 40.01, "lon": -71.12
            }
          }
        ]
      }
    ]
  }
}
```

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_bounding\_box* query.

**Return type** dict

**classmethod** `get_geo_distance_params(value, field)`

Get params for *geo\_distance* query.

Example:

`/api/articles/?location__geo_distance=2km__43.53__-12.23`

**Parameters**

- **value** (*str*) –

- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** dict

**classmethod** `get_geo_polygon_params(value, field)`

Get params for *geo\_polygon* query.

Example:

`/api/articles/?location__geo_polygon=40,-70__30,-80__20,-90`

Example:

`/api/articles/?location__geo_polygon=40,-70__30,-80__20,-90__name,myname__validation_method,IGNORE_MALFORMED`

Elasticsearch:

```
{
  "query": {
    "bool": [{
      "must": [{ "match_all": {} }],
      "filter": {
        "geo_polygon": [{
          "person.location": [{
            "points": [[ { "lat": 40, "lon": -70 }, { "lat": 30, "lon": -80 }, { "lat": 20, "lon": -90 } ]
          }
        ]
      }
    ]
  }
}
```

**Parameters**

- **value** (*str*) –

- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** dict

**classmethod** `get_geo_shape_params(value, field)`

Get params for *geo\_shape* query.

Example:

`/search/publishers/?location__geo_shape=48.9864453,6.37977__relation,intersects__type,circle__radius,20km`

Example:

`/search/publishers/?location__geo_shape=48.906254,6.378593__48.985850,6.479359__relation,within__type,envelope`

Elasticsearch:

```
{
  "query": {
```

```

        "bool" [{}
            "must" [{} "match_all": {}
            ], "filter": {
                "geo_shape" [{}
                    "location" [{}
                        "shape": { "type": "circle", "coordinates": [48.9864453,
                            6.37977], "radius": "20km"
                        }, "relation": "intersects"
                    }
                ]
            }
        ]
    }
}

```

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_shape* query.

**Return type** dict

**classmethod** `prepare_filter_fields(view)`

Prepare filter fields.

**Parameters** *view* (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**class** `django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Ids filter backend for Elasticsearch.

Example:

```

>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     IdsFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):

```

(continues on next page)



(continued from previous page)

```
>>>
>>> document = ArticleDocument
>>> serializer_class = ArticleDocumentSerializer
>>> filter_backends = [IdsFilterBackend]
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ids\_query\_params**(*request*)

Get search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**get\_ids\_values**(*request, view*)

Get ids values for query.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**ids\_query\_param** = 'ids'

**class** *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.NestedFilteringFilterBackend*

Bases: *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend*

Nested filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     NestedFilteringFilterBackend
>>> )
```

(continues on next page)

(continued from previous page)

```
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [NestedFilteringFilterBackend,]
>>>     nested_filter_fields = {
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>             'lookups': [
>>>                 LOOKUP_FILTER_TERM,
>>>                 LOOKUP_FILTER_TERMS,
>>>                 LOOKUP_FILTER_PREFIX,
>>>                 LOOKUP_FILTER_WILDCARD,
>>>                 LOOKUP_QUERY_EXCLUDE,
>>>                 LOOKUP_QUERY_ISNULL,
>>>             ],
>>>         }
>>>     }
```

**classmethod** `apply_filter(queryset, options=None, args=None, kwargs=None)`

Apply filter.

**Parameters**

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

**Returns**

**classmethod** `apply_query(queryset, options=None, args=None, kwargs=None)`

Apply query.

**Parameters**

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

**Returns**

`get_coreschema_field(field)`

**get\_filter\_field\_nested\_path**(*filter\_fields*, *field\_name*)

Get filter field path to be used in nested query.

**Parameters**

- **filter\_fields** –
- **field\_name** –

**Returns**

**get\_filter\_query\_params**(*request*, *view*)

Get query params to be filtered on.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**get\_schema\_fields**(*view*)

**classmethod prepare\_filter\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

**class**

`django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilterFilteringFilterBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend`

The post\_filter filtering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_ISNULL,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     PostFilterFilteringFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
```

(continues on next page)

(continued from previous page)

```
>>> document = ArticleDocument
>>> serializer_class = ArticleDocumentSerializer
>>> filter_backends = [PostFilterFilteringFilterBackend,]
>>> post_filter_fields = {
>>>     'title': 'title.raw',
>>>     'state': {
>>>         'field': 'state.raw',
>>>         'lookups': [
>>>             LOOKUP_FILTER_PREFIX,
>>>             LOOKUP_FILTER_WILDCARD,
>>>             LOOKUP_QUERY_EXCLUDE,
>>>             LOOKUP_QUERY_ISNULL,
>>>         ],
>>>     }
>>> }
```

**classmethod** `apply_filter(queryset, options=None, args=None, kwargs=None)`

Apply filter.

**Parameters**

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

**Returns**

**classmethod** `apply_query(queryset, options=None, args=None, kwargs=None)`

Apply query.

**Parameters**

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

**Returns**

`get_coreschema_field(field)`

`get_schema_fields(view)`

**classmethod** `prepare_filter_fields(view)`

Prepare filter fields.

**Parameters** `view` (`rest_framework.viewsets.ReadOnlyModelViewSet`) –

**Returns** Filtering options.

**Return type** dict

### 12.19.1.2.1.15 django\_elasticsearch\_dsl\_drf.filter\_backends.ordering package

### 12.19.1.2.1.16 Submodules

### 12.19.1.2.1.17 django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common module

Ordering backend.

**class**

django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.**DefaultOrderingFilterBackend**  
 Bases: rest\_framework.filters.BaseFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.OrderingMixin

Default ordering filter backend for Elasticsearch.

Make sure this is your last ordering backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     DefaultOrderingFilterBackend,
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [
>>>         DefaultOrderingFilterBackend,
>>>         OrderingFilterBackend,
>>>     ]
>>>     ordering_fields = {
>>>         'id': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>         'city': {
```

(continues on next page)

(continued from previous page)

```
>>>         'field': 'continent.country.city.name.raw',
>>>         'path': 'continent.country.city',
>>>     }
>>> }
>>> ordering = 'city'
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod get\_default\_ordering\_params**(*view*)

Get the default ordering params for the view.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**get\_ordering\_query\_params**(*request, view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**ordering\_param** = 'ordering'

**class** *django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.OrderingFilterBackend*

Bases: *rest\_framework.filters.BaseFilterBackend*, *django\_elasticsearch\_dsl\_drf.*

*filter\_backends.ordering.common.OrderingMixin*

Ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
```

(continues on next page)

(continued from previous page)

```

>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [OrderingFilterBackend,]
>>>     ordering_fields = {
>>>         'id': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = ('id', 'title',)

```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ordering\_query\_params**(*request, view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**get\_schema\_fields**(*view*)

**ordering\_param** = 'ordering'

### 12.19.1.2.1.18 django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.geo\_spatial module

Geo-spatial ordering backend.

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.geo\_spatial.

**GeoSpatialOrderingFilterBackend**

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Geo-spatial ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialOrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialOrderingFilterBackend,]
>>>     geo_spatial_ordering_fields = {
>>>         'location': {
>>>             'field': 'location',
>>>         }
>>>     }
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** **get\_geo\_distance\_params**(*value, field*)

Get params for *geo\_distance* ordering.

Example:

`/api/articles/?ordering=-location__45.3214__-34.3421__km__planes`

**Parameters**

- **value** (*str*) –



- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** dict

**get\_geo\_spatial\_field\_name**(*request, view, name*)

Get geo-spatial field name.

We have to deal with a couple of situations here:

Example 1:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': None,
>>> }
```

Example 2:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': 'location',
>>> }
```

Example 3:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': {
>>>         'field': 'location'
>>>     },
>>> }
```

#### Parameters

- **request** –
- **view** –
- **name** –

#### Returns

**get\_ordering\_query\_params**(*request, view*)

Get ordering query params.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**ordering\_param** = 'ordering'

### 12.19.1.2.1.19 Module contents

Ordering backends.

```
class django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingFilterBackend
    Bases:      rest_framework.filters.BaseFilterBackend,      django_elasticsearch_dsl_drf.
               filter_backends.ordering.common.OrderingMixin
```

Default ordering filter backend for Elasticsearch.

Make sure this is your last ordering backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     DefaultOrderingFilterBackend,
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [
>>>         DefaultOrderingFilterBackend,
>>>         OrderingFilterBackend,
>>>     ]
>>>     ordering_fields = {
>>>         'id': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = 'city'
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod get\_default\_ordering\_params**(*view*)

Get the default ordering params for the view.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**get\_ordering\_query\_params**(*request, view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**ordering\_param** = 'ordering'

**class**

`django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Geo-spatial ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     GeoSpatialOrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [GeoSpatialOrderingFilterBackend,]
```

(continues on next page)

(continued from previous page)

```
>>> geo_spatial_ordering_fields = {
>>>     'location': {
>>>         'field': 'location',
>>>     }
>>> }
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** **get\_geo\_distance\_params**(*value, field*)

Get params for *geo\_distance* ordering.

Example:

```
/api/articles/?ordering=-location__45.3214__-34.3421__km__planes
```

**Parameters**

- **value** (*str*) –
- **field** –

**Returns** Params to be used in *geo\_distance* query.

**Return type** *dict*

**get\_geo\_spatial\_field\_name**(*request, view, name*)

Get geo-spatial field name.

We have to deal with a couple of situations here:

Example 1:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': None,
>>> }
```

Example 2:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': 'location',
>>> }
```

Example 3:

```
>>> geo_spatial_ordering_fields = {
>>>     'location': {
>>>         'field': 'location'
>>>     },
>>> }
```

**Parameters**

- **request** –
- **view** –
- **name** –

#### Returns

**get\_ordering\_query\_params**(*request*, *view*)

Get ordering query params.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**ordering\_param** = 'ordering'

**class** `django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.ordering.common.OrderingMixin`

Ordering filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     OrderingFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [OrderingFilterBackend,]
>>>     ordering_fields = {
>>>         'id': None,
>>>         'title': 'title.raw',
>>>         'date_submitted': 'date_submitted',
>>>         'continent': {
>>>             'field': 'continent.name.raw',
>>>             'path': 'continent',
>>>         }
>>>         'country': {
>>>             'field': 'continent.country.name.raw',
>>>             'path': 'continent.country',
>>>         }
>>>     }
```

(continues on next page)

(continued from previous page)

```
>>>         'city': {
>>>             'field': 'continent.country.city.name.raw',
>>>             'path': 'continent.country.city',
>>>         }
>>>     }
>>>     ordering = ('id', 'title',)
```

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_ordering\_query\_params**(*request, view*)

Get ordering query params.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Ordering params to be used for ordering.

**Return type** list

**get\_schema\_fields**(*view*)

**ordering\_param** = 'ordering'

#### 12.19.1.2.1.20 `django_elasticsearch_dsl_drf.filter_backends.search` package

#### 12.19.1.2.1.21 Subpackages

#### 12.19.1.2.1.22 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends` package

#### 12.19.1.2.1.23 Submodules

#### 12.19.1.2.1.24 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.`

**BaseSearchQueryBackend**

Bases: object

Search query backend.

**classmethod** **construct\_search**(*request, view, search\_backend*)

Construct search.

**Parameters**

- **request** –

- `view` –
- `search_backend` –

**Returns**

#### 12.19.1.2.1.25 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match.`

**MatchQueryBackend**

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

**Parameters**

- `request` –
- `view` –
- `search_backend` –

**Returns**

`query_type = 'match'`

#### 12.19.1.2.1.26 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase.`

**MatchPhraseQueryBackend**

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match phrase query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

**Parameters**

- `request` –
- `view` –
- `search_backend` –

**Returns**

`query_type = 'match_phrase'`

#### 12.19.1.2.1.27 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.`

`match_phrase_prefix.MatchPhrasePrefixQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match phrase prefix query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

**Parameters**

- **request** –
- **view** –
- **search\_backend** –

**Returns**

`query_type = 'match_phrase_prefix'`

#### 12.19.1.2.1.28 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match.`

`MultiMatchQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Multi match query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_multi_match=lorem ipsum /search/books/?search_multi_match=title,summary:lorem ipsum
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_multi_match=title,summary:lorem ipsum
&search_multi_match=author,publisher=o'reilly
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
multi_match_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'summary':
    { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
multi_match_search_fields = ( 'title', 'summary', 'description',
)
```



#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**classmethod** `get_field(field, options)`

Get field.

#### Parameters

- **field** –
- **options** –

#### Returns

**classmethod** `get_query_options(request, view, search_backend)`

`query_type = 'multi_match'`

### 12.19.1.2.1.29 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.`

#### **NestedQueryBackend**

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Nested query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

Dictionary key is the GET param name. The path option stands for the path in Elasticsearch.

Type 1:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': ['name'],
    }, 'city': {
        'path': 'country.city', 'fields': ['name'],
    },
}
```

Type 2:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': [{ 'name': { 'boost': 2 } }}
    }, 'city': {
        'path': 'country.city', 'fields': [{ 'name': { 'boost': 2 } }}
    },
}
```

#### Parameters

- **request** –

- **view** –
- **search\_backend** –

Returns

**query\_type** = 'nested'

#### 12.19.1.2.1.30 `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string` module

**class** `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.`

`simple_query_string.SimpleQueryStringQueryBackend`

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Simple query string query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_simple_query_string= "fried eggs"%2B(eggplant|potato)-frittata
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata &search_simple_query_string= author,publisher="fried eggs" +(egg-
plant | potato) -frittata
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
simple_query_string_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'sum-
mary': { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
simple_query_string_search_fields = ( 'title', 'summary', 'description',
)
```

Query examples:

```
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22 -considering
```

**Parameters**

- **request** –
- **view** –
- **search\_backend** –

**Returns**

```
classmethod get_field(field, options)
    Get field.
    Parameters
        • field –
        • options –
    Returns
classmethod get_query_options(request, view, search_backend)
query_type = 'simple_query_string'
```

#### 12.19.1.2.1.31 Module contents

Search query backends.

```
class
django_elasticsearch_dsl_drf.filter_backends.search.query_backends.BaseSearchQueryBackend
    Bases: object
    Search query backend.
classmethod construct_search(request, view, search_backend)
    Construct search.
    Parameters
        • request –
        • view –
        • search_backend –
    Returns
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
MatchPhrasePrefixQueryBackend
    Bases: django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backends.base.
BaseSearchQueryBackend
    Match phrase prefix query backend.
classmethod construct_search(request, view, search_backend)
    Construct search.
    Parameters
        • request –
        • view –
        • search_backend –
    Returns
    query_type = 'match_phrase_prefix'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.query_backends.
MatchPhraseQueryBackend
    Bases: django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backends.base.
BaseSearchQueryBackend
    Match phrase query backend.
classmethod construct_search(request, view, search_backend)
    Construct search.
```

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**query\_type** = 'match\_phrase'

#### class

`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchQueryBackend`  
 Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Match query backend.

**classmethod construct\_search**(*request, view, search\_backend*)

Construct search.

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**query\_type** = 'match'

#### class

`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MultiMatchQueryBackend`  
 Bases: `django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend`

Multi match query backend.

**classmethod construct\_search**(*request, view, search\_backend*)

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_multi_match=lorem ipsum /search/books/?search_multi_match=title,summary:lorem ipsum
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_multi_match=title,summary:lorem ipsum
&search_multi_match=author,publisher=o'reily
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
multi_match_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'summary':
    { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
multi_match_search_fields = ( 'title', 'summary', 'description',
)
```

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**classmethod** `get_field(field, options)`

Get field.

#### Parameters

- **field** –
- **options** –

#### Returns

**classmethod** `get_query_options(request, view, search_backend)`

`query_type = 'multi_match'`

#### class

`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.NestedQueryBackend`

Bases: [`django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backends.base.BaseSearchQueryBackend`](#)

Nested query backend.

**classmethod** `construct_search(request, view, search_backend)`

Construct search.

Dictionary key is the GET param name. The path option stands for the path in Elasticsearch.

Type 1:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': ['name'],
    }, 'city': {
        'path': 'country.city', 'fields': ['name'],
    },
}
```

Type 2:

```
search_nested_fields = {
    'country': { 'path': 'country', 'fields': [{ 'name': { 'boost': 2 } } ]
    }, 'city': {
        'path': 'country.city', 'fields': [{ 'name': { 'boost': 2 } } ]
    },
}
```

#### Parameters

- **request** –
- **view** –

- **search\_backend** –

#### Returns

**query\_type** = 'nested'

**class** django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backends.

**SimpleQueryStringQueryBackend**

Bases: [django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search.query\\_backends.base.BaseSearchQueryBackend](#)

Simple query string query backend.

**classmethod** **construct\_search**(*request, view, search\_backend*)

Construct search.

In case of multi match, we always look in a group of fields. Thus, matching per field is no longer valid use case here. However, we might want to have multiple fields enabled for multi match per view set, and only search in some of them in specific request.

Example:

```
/search/books/?search_simple_query_string= "fried eggs"%2B(eggplant|potato)-frittata
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata
```

Note, that multiple searches are not supported (would not raise an exception, but would simply take only the first):

```
/search/books/?search_simple_query_string= title,summary:"fried eggs" +(eggplant |
potato) -frittata &search_simple_query_string= author,publisher="fried eggs" +(egg-
plant | potato) -frittata
```

In the view-set fields shall be defined in a very simple way. The only accepted argument would be boost (per field).

Example 1 (complex):

```
simple_query_string_search_fields = { 'title': { 'field': 'title.english', 'boost': 4}, 'sum-
mary': { 'boost': 2}, 'description': None,
}
```

Example 2 (simple list):

```
simple_query_string_search_fields = ( 'title', 'summary', 'description',
)
```

Query examples:

```
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22
http://localhost:8000/search /books-simple-query-string-search-backend
/?search_simple_query_string=%22Pool%20of%20Tears%22 -considering
```

#### Parameters

- **request** –
- **view** –
- **search\_backend** –

#### Returns

**classmethod** **get\_field**(*field, options*)

Get field.

#### Parameters

- **field** –
- **options** –

### Returns

```
classmethod get_query_options(request, view, search_backend)
query_type = 'simple_query_string'
```

#### 12.19.1.2.1.32 Submodules

#### 12.19.1.2.1.33 django\_elasticsearch\_dsl\_drf.filter\_backends.search.base module

Base search backend.

```
class django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend
    Bases: rest_framework.filters.BaseFilterBackend, django_elasticsearch_dsl_drf.
           filter_backends.mixins.FilterBackendMixin
```

Base search filter backend.

```
filter_queryset(request, queryset, view)
    Filter the queryset.
```

### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

```
get_coreschema_field(field)
```

```
get_query_backends(request, view)
    Get query backends.
```

### Returns

```
get_schema_fields(view)
```

```
get_search_query_params(request)
    Get search query params.
```

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

```
matching = 'should'
```

```
query_backends = []
```

```
search_param = 'search'
```

#### 12.19.1.2.1.34 django\_elasticsearch\_dsl\_drf.filter\_backends.search.compound module

Compound search backend.

**class**

`django_elasticsearch_dsl_drf.filter_backends.search.compound.CompoundSearchFilterBackend`  
Bases: `django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend`

Compound search backend.

```
query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.
query_backends.match.MatchQueryBackend'>, <class
'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.
NestedQueryBackend'>]
```

#### 12.19.1.2.1.35 django\_elasticsearch\_dsl\_drf.filter\_backends.search.historical module

Search backend. Most likely to be deprecated soon.

**class** `django_elasticsearch_dsl_drf.filter_backends.search.historical.SearchFilterBackend`  
Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Search filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SearchFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [SearchFilterBackend,]
>>>     search_fields = (
>>>         'title',
>>>         'content',
>>>     )
>>>     search_nested_fields = {
>>>         'state': ['name'],
>>>         'documents.author': ['title', 'description'],
>>>     }
```



**construct\_nested\_search**(*request, view*)

Construct nested search.

We have to deal with two types of structures:

Type 1:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': ['name'],
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': ['name'],
>>>     },
>>> }
```

Type 2:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': [{ 'name': { 'boost': 2 } }]
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': [{ 'name': { 'boost': 2 } }]
>>>     },
>>> }
```

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**construct\_search**(*request, view*)

Construct search.

We have to deal with two types of structures:

Type 1:

```
>>> search_fields = (
>>>     'title',
>>>     'description',
>>>     'summary',
>>> )
```

Type 2:

```
>>> search_fields = {
>>>     'title': {'boost': 2},
>>>     'description': None,
>>>     'summary': None,
>>> }
```

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_coreschema\_field**(*field*)

**get\_schema\_fields**(*view*)

**get\_search\_query\_params**(*request*)

Get search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**search\_param** = 'search'

### 12.19.1.2.1.36 `django_elasticsearch_dsl_drf.filter_backends.search.multi_match` module

Multi match search filter backend.

**class** `django_elasticsearch_dsl_drf.filter_backends.search.multi_match.`

**MultiMatchSearchFilterBackend**

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend`

Multi match search filter backend.

**matching** = 'must'

**query\_backends** = [`<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match.MultiMatchQueryBackend'>`]

**search\_param** = 'search\_multi\_match'

### 12.19.1.2.1.37 `django_elasticsearch_dsl_drf.filter_backends.search.query_string` module

### 12.19.1.2.1.38 `django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string` module

Simple query string search filter backend.

**class** `django_elasticsearch_dsl_drf.filter_backends.search.simple_query_string.`

**SimpleQueryStringSearchFilterBackend**

Bases: `django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend`

Simple query string search filter backend.

**matching** = 'must'

**query\_backends** = [`<class 'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.simple_query_string.SimpleQueryStringQueryBackend'>`]

**search\_param** = 'search\_simple\_query\_string'

### 12.19.1.2.1.39 Module contents

Search filter backends.

**class** `django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Base search filter backend.

**filter\_queryset**(*request*, *queryset*, *view*)

Filter the queryset.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**get\_coreschema\_field**(*field*)

**get\_query\_backends**(*request*, *view*)

Get query backends.

**Returns**

**get\_schema\_fields**(*view*)

**get\_search\_query\_params**(*request*)

Get search query params.

**Parameters** **request** (`rest_framework.request.Request`) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**matching** = 'should'

**query\_backends** = []

```
search_param = 'search'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.CompoundSearchFilterBackend
    Bases: django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchFilterBackend
```

Compound search backend.

```
query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.
query_backends.match.MatchQueryBackend'>, <class
'django_elasticsearch_dsl_drf.filter_backends.search.query_backends.nested.
NestedQueryBackend'>]
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchSearchFilterBackend
    Bases: django\_elasticsearch\_dsl\_drf.filter\_backends.search.base.BaseSearchFilterBackend
```

Multi match search filter backend.

```
matching = 'must'
```

```
query_backends = [<class 'django_elasticsearch_dsl_drf.filter_backends.search.
query_backends.multi_match.MultiMatchQueryBackend'>]
```

```
search_param = 'search_multi_match'
```

```
class django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend
    Bases: rest\_framework.filters.BaseFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin
```

Search filter backend for Elasticsearch.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SearchFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [SearchFilterBackend,]
>>>     search_fields = (
>>>         'title',
>>>         'content',
>>>     )
>>>     search_nested_fields = {
>>>         'state': ['name'],
```

(continues on next page)

(continued from previous page)

```
>>> 'documents.author': ['title', 'description'],
>>> }
```

**construct\_nested\_search**(*request*, *view*)

Construct nested search.

We have to deal with two types of structures:

Type 1:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': ['name'],
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': ['name'],
>>>     },
>>> }
```

Type 2:

```
>>> search_nested_fields = {
>>>     'country': {
>>>         'path': 'country',
>>>         'fields': [{ 'name': { 'boost': 2 } }],
>>>     },
>>>     'city': {
>>>         'path': 'country.city',
>>>         'fields': [{ 'name': { 'boost': 2 } }],
>>>     },
>>> }
```

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**construct\_search**(*request*, *view*)

Construct search.

We have to deal with two types of structures:

Type 1:

```
>>> search_fields = (
>>>     'title',
>>>     'description',
>>>     'summary',
>>> )
```

Type 2:

```
>>> search_fields = {
>>>     'title': {'boost': 2},
>>>     'description': None,
>>>     'summary': None,
>>> }
```

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**get\_coreschema\_field**(*field*)

**get\_schema\_fields**(*view*)

**get\_search\_query\_params**(*request*)

Get search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**search\_param** = 'search'

**class**

**django\_elasticsearch\_dsl\_drf.filter\_backends.search.SimpleQueryStringSearchFilterBackend**

Bases: [django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search.base.BaseSearchFilterBackend](#)

Simple query string search filter backend.

**matching** = 'must'

**query\_backends** = [[<class 'django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search.query\\_backends.simple\\_query\\_string.SimpleQueryStringQueryBackend'>](#)]

**search\_param** = 'search\_simple\_query\_string'

#### 12.19.1.2.1.40 django\_elasticsearch\_dsl\_drf.filter\_backends.suggester package

#### 12.19.1.2.1.41 Submodules

#### 12.19.1.2.1.42 django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional module

Functional suggesters backend.

It's assumed, that fields you're planning to query suggestions for have been properly indexed using fields.CompletionField.

Example:

```

>>> from django_elasticsearch_dsl import Document, Index, fields
>>>
>>> from books.models import Publisher
>>>
>>> # Name of the Elasticsearch index
>>> PUBLISHER_INDEX = Index(PUBLISHER_INDEX_NAME)
>>> # See Elasticsearch Indices API reference for available settings
>>> PUBLISHER_INDEX.settings(
>>>     number_of_shards=1,
>>>     number_of_replicas=1
>>> )
>>>
>>> @PUBLISHER_INDEX.doc_type
>>> class PublisherDocument(Document):
>>>     "Publisher Elasticsearch document."
>>>
>>>     id = fields.IntegerField(attr='id')
>>>
>>>     name = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     info = fields.StringField()
>>>
>>>     address = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword')
>>>         }
>>>     )
>>>
>>>     city = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>

```

(continues on next page)

(continued from previous page)

```

>>> state_province = fields.StringField(
>>>     fields={
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> country = fields.StringField(
>>>     fields={
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> website = fields.StringField()
>>>
>>> class Meta:
>>>     "Meta options."
>>>
>>> model = Publisher # The model associate with this Document

```

**class** `django_elasticsearch_dsl_drf.filter_backends.suggester.functional.`

**FunctionalSuggesterFilterBackend**

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Suggester filter backend for Elasticsearch.

Suggestion functionality is exclusive. Once you have queried the `FunctionalSuggesterFilterBackend`, the latter will transform your current search query into another search query (altered). Therefore, always add it as the very last filter backend.

Example:

```

>>> from django_elasticsearch_dsl_drf.constants import (
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_TERM_MATCH,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FunctionalSuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
>>>

```

(continues on next page)



(continued from previous page)

```

>>> document = PublisherDocument
>>> serializer_class = PublisherDocumentSerializer
>>> filter_backends = [
>>>     # ...
>>>     FunctionalSuggesterFilterBackend,
>>> ]
>>> # Suggester fields
>>> functional_suggester_fields = {
>>>     'name_suggest': {
>>>         'field': 'name.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>>     'city_suggest': {
>>>         'field': 'city.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>>     'state_province_suggest': {
>>>         'field': 'state_province.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>         ],
>>>     },
>>>     'country_suggest': {
>>>         'field': 'country.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>> }

```

**classmethod** `apply_query_size(queryset, options)`

Apply query size.

**Parameters**

- **queryset** –
- **options** –

**Returns**

**classmethod** `apply_suggester_completion_match(suggester_name, queryset, options, value)`

Apply *completion* suggester match.

This is effective when used with Ngram fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_suggester_completion_prefix(suggester_name, queryset, options, value)`

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**clean\_queryset(queryset)**

Clean the queryset.

- Remove aggregations.
- Remove highlight.
- Remove sorting options.

**Parameters** **queryset** –

**Returns**

**extract\_field\_name(field\_name)**

Extract field name.

For instance, “name.suggest” or “name.raw” becomes “name”.

**Parameters** **field\_name** –

**Returns**

**Return type** `str`

**filter\_queryset(request, queryset, view)**

Filter the queryset.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**get\_suggester\_query\_params(request, view)**

Get query params to be for suggestions.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Request query params to filter on.

**Return type** `dict`

**classmethod** `prepare_suggester_fields(view)`

Prepare filter fields.

**Parameters** **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) –

**Returns** Filtering options.

**Return type** dict

**serialize\_queryset**(*queryset, suggester\_name, value, serializer\_field*)

Serialize queryset.

This shall be done here, since we don't want to delegate it to pagination.

**Parameters**

- **queryset** –
- **suggester\_name** –
- **value** –
- **serializer\_field** –

**Returns**

#### 12.19.1.2.1.43 django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.native module

Suggesters backend.

It's assumed, that fields you're planning to query suggestions for have been properly indexed using fields. CompletionField.

Example:

```
>>> from django_elasticsearch_dsl import Document, Index, fields
>>>
>>> from books.models import Publisher
>>>
>>> # Name of the Elasticsearch index
>>> PUBLISHER_INDEX = Index(PUBLISHER_INDEX_NAME)
>>> # See Elasticsearch Indices API reference for available settings
>>> PUBLISHER_INDEX.settings(
>>>     number_of_shards=1,
>>>     number_of_replicas=1
>>> )
>>>
>>> @PUBLISHER_INDEX.doc_type
>>> class PublisherDocument(Document):
>>>     "Publisher Elasticsearch document."
>>>
>>>     id = fields.IntegerField(attr='id')
>>>
>>>     name = fields.StringField(
>>>         fields={
>>>             'raw': fields.StringField(analyzer='keyword'),
>>>             'suggest': fields.CompletionField(),
>>>         }
>>>     )
>>>
>>>     info = fields.StringField()
>>>
>>>     address = fields.StringField(
>>>         fields={
```

(continues on next page)

(continued from previous page)

```
>>>         'raw': fields.StringField(analyzer='keyword')
>>>     }
>>> )
>>>
>>> city = fields.StringField(
>>>     fields={
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> state_province = fields.StringField(
>>>     fields={
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> country = fields.StringField(
>>>     fields={
>>>         'raw': fields.StringField(analyzer='keyword'),
>>>         'suggest': fields.CompletionField(),
>>>     }
>>> )
>>>
>>> website = fields.StringField()
>>>
>>> class Meta:
>>>     "Meta options."
>>>
>>>     model = Publisher # The model associate with this Document
```

## class

`django_elasticsearch_dsl_drf.filter_backends.suggester.native.SuggesterFilterBackend`  
 Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Suggester filter backend for Elasticsearch.

Suggestion functionality is exclusive. Once you have queried the `SuggesterFilterBackend`, the latter will transform your current search query into suggestion search query (which is very different). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     SUGGESTER_TERM,
>>>     SUGGESTER_PHRASE,
>>>     SUGGESTER_COMPLETION,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
```

(continues on next page)

(continued from previous page)

```

>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(BaseDocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         SuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     suggester_fields = {
>>>         'name_suggest': {
>>>             'field': 'name.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_TERM,
>>>                 SUGGESTER_PHRASE,
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'city_suggest': {
>>>             'field': 'city.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'state_province_suggest': {
>>>             'field': 'state_province.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>         'country_suggest': {
>>>             'field': 'country.suggest',
>>>             'suggesters': [
>>>                 SUGGESTER_COMPLETION,
>>>             ],
>>>         },
>>>     }
>>> }

```

**classmethod** `apply_suggester_completion(suggester_name, queryset, options, value)`

Apply completion suggester.

#### Parameters

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.

- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_suggester_phrase(suggester_name, queryset, options, value)`

Apply *phrase* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_suggester_term(suggester_name, queryset, options, value)`

Apply *term* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**filter\_queryset(request, queryset, view)**

Filter the queryset.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `get_suggester_context(field, suggester_name, request, view)`

Get suggester context.

Given the following definition (in ViewSets):

```
>>> # Suggester fields
>>> suggester_fields = {
>>>     'title_suggest': {
>>>         'field': 'title.suggest',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>     },
>>>     'title_suggest_context': {
>>>         'field': 'title.suggest_context',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>         'completion_options': {
>>>             'filters': {
```

(continues on next page)

(continued from previous page)

```
>>>         'title_suggest_tag': 'tag',
>>>         'title_suggest_state': 'state',
>>>         'title_suggest_publisher': 'publisher',
>>>     },
>>>     'size': 10,
>>> }
>>> },
>>> }
```

[http://localhost:8000/search/books-frontend/suggest/?title\\_suggest\\_context=M](http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M)

When talking about the queries made, we have the following. Multiple values per field are combined with OR:

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': ['History', 'Drama'],
>>>     }
>>> }
```

The following works with OR as well, so it seems we have OR only. However, the following construction is more handy, since it allows us to play with boosting nicely. Also, it allows to provide *prefix* param (which in case of the example given below means that suggestions shall match both categories with “Child”, “Children”, “Childrend’s”). Simply put it’s treated as *prefix*, rather than *term*.

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': [
>>>             {'context': 'History'},
>>>             {'context': 'Drama', 'boost': 2},
>>>             {'context': 'Children', 'prefix': True},
>>>         ],
>>>     },
>>> }
```

Sample query for *category* filter:

`/search/books-frontend/suggest/ ?title_suggest_context=M &title_suggest_tag=Art__2.0 &title_suggest_tag=Documentary__2.0__prefix &title_suggest_publisher=Apress`

The query params would be:

```
query_params: <QueryDict: {
  'title_suggest_context': ['M'], 'title_suggest_tag': ['Art__2.0', 'Documentary__2.0__prefix'], 'title_suggest_publisher': ['Apress']
}>
```

Sample query for *geo* filter:

`/search/address/suggest/ ?street_suggest_context=M &street_suggest_loc=43.66__-79.22__2.0__10000km`

The query params would be:

```
query_params: <QueryDict: {
  'street_suggest_context': ['M'], 'street_suggest_loc': ['Art__43.66__-79.22__2.0__10000km'],
}
```

```
>
```

#### Returns

**get\_suggester\_query\_params**(*request*, *view*)

Get query params to be for suggestions.

#### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod prepare\_suggester\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

### 12.19.1.2.1.44 Module contents

Suggester filtering backends.

#### class

`django_elasticsearch_dsl_drf.filter_backends.suggester.FunctionalSuggesterFilterBackend`

Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Suggester filter backend for Elasticsearch.

Suggestion functionality is exclusive. Once you have queried the `FunctionalSuggesterFilterBackend`, the latter will transform your current search query into another search query (altered). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>     FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_PHRASE_MATCH,
>>>     FUNCTIONAL_SUGGESTER_TERM_MATCH,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FunctionalSuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(DocumentViewSet):
```

(continues on next page)



(continued from previous page)

```

>>>
>>> document = PublisherDocument
>>> serializer_class = PublisherDocumentSerializer
>>> filter_backends = [
>>>     # ...
>>>     FunctionalSuggesterFilterBackend,
>>> ]
>>> # Suggester fields
>>> functional_suggester_fields = {
>>>     'name_suggest': {
>>>         'field': 'name.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>>     'city_suggest': {
>>>         'field': 'city.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>>     'state_province_suggest': {
>>>         'field': 'state_province.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_MATCH,
>>>         ],
>>>     },
>>>     'country_suggest': {
>>>         'field': 'country.suggest',
>>>         'suggesters': [
>>>             FUNCTIONAL_SUGGESTER_COMPLETION_PREFIX,
>>>         ],
>>>     },
>>> }

```

**classmethod** `apply_query_size(queryset, options)`

Apply query size.

**Parameters**

- **queryset** –
- **options** –

**Returns**

**classmethod** `apply_suggester_completion_match(suggester_name, queryset, options, value)`

Apply *completion* suggester match.

This is effective when used with Ngram fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.

- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**classmethod** `apply_suggester_completion_prefix(suggester_name, queryset, options, value)`

Apply *completion* suggester prefix.

This is effective when used with Keyword fields.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** `elasticsearch_dsl.search.Search`

**clean\_queryset(queryset)**

Clean the queryset.

- Remove aggregations.
- Remove highlight.
- Remove sorting options.

**Parameters** **queryset** –

**Returns**

**extract\_field\_name(field\_name)**

Extract field name.

For instance, “name.suggest” or “name.raw” becomes “name”.

**Parameters** **field\_name** –

**Returns**

**Return type** `str`

**filter\_queryset(request, queryset, view)**

Filter the queryset.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **queryset** (`elasticsearch_dsl.search.Search`) – Base queryset.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Updated queryset.

**Return type** `elasticsearch_dsl.search.Search`

**get\_suggester\_query\_params(request, view)**

Get query params to be for suggestions.

**Parameters**

- **request** (`rest_framework.request.Request`) – Django REST framework request.
- **view** (`rest_framework.viewsets.ReadOnlyModelViewSet`) – View.

**Returns** Request query params to filter on.

**Return type** `dict`

**classmethod** `prepare_suggester_fields(view)`

Prepare filter fields.

**Parameters** `view` (`rest_framework.viewsets.ReadOnlyModelViewSet`) –  
**Returns** Filtering options.  
**Return type** dict

**serialize\_queryset** (`queryset`, `suggester_name`, `value`, `serializer_field`)  
 Serialize queryset.

This shall be done here, since we don't want to delegate it to pagination.

**Parameters**

- `queryset` –
- `suggester_name` –
- `value` –
- `serializer_field` –

**Returns**

**class** `django_elasticsearch_dsl_drf.filter_backends.suggester.SuggesterFilterBackend`  
 Bases: `rest_framework.filters.BaseFilterBackend`, `django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin`

Suggester filter backend for Elasticsearch.

Suggestion functionality is exclusive. Once you have queried the `SuggesterFilterBackend`, the latter will transform your current search query into suggestion search query (which is very different). Therefore, always add it as the very last filter backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.constants import (
>>>     SUGGESTER_TERM,
>>>     SUGGESTER_PHRASE,
>>>     SUGGESTER_COMPLETION,
>>> )
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     SuggesterFilterBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import DocumentViewSet
>>>
>>> # Local PublisherDocument definition
>>> from .documents import PublisherDocument
>>>
>>> # Local PublisherDocument serializer
>>> from .serializers import PublisherDocumentSerializer
>>>
>>> class PublisherDocumentView(BaseDocumentViewSet):
>>>
>>>     document = PublisherDocument
>>>     serializer_class = PublisherDocumentSerializer
>>>     filter_backends = [
>>>         # ...
>>>         SuggesterFilterBackend,
>>>     ]
>>>     # Suggester fields
>>>     suggester_fields = {
>>>         'name_suggest': {
```

(continues on next page)

(continued from previous page)

```
>>>         'field': 'name.suggest',
>>>         'suggesters': [
>>>             SUGGESTER_TERM,
>>>             SUGGESTER_PHRASE,
>>>             SUGGESTER_COMPLETION,
>>>         ],
>>>     },
>>>     'city_suggest': {
>>>         'field': 'city.suggest',
>>>         'suggesters': [
>>>             SUGGESTER_COMPLETION,
>>>         ],
>>>     },
>>>     'state_province_suggest': {
>>>         'field': 'state_province.suggest',
>>>         'suggesters': [
>>>             SUGGESTER_COMPLETION,
>>>         ],
>>>     },
>>>     'country_suggest': {
>>>         'field': 'country.suggest',
>>>         'suggesters': [
>>>             SUGGESTER_COMPLETION,
>>>         ],
>>>     },
>>> }
```

**classmethod** `apply_suggester_completion(suggester_name, queryset, options, value)`

Apply *completion* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_suggester_phrase(suggester_name, queryset, options, value)`

Apply *phrase* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod** `apply_suggester_term(suggester_name, queryset, options, value)`

Apply *term* suggester.

**Parameters**

- **suggester\_name** (*str*) –
- **queryset** (*elasticsearch\_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

**Returns** Modified queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**filter\_queryset** (*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

**classmethod get\_suggester\_context** (*field, suggester\_name, request, view*)

Get suggester context.

Given the following definition (in ViewSets):

```
>>> # Suggester fields
>>> suggester_fields = {
>>>     'title_suggest': {
>>>         'field': 'title.suggest',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>     },
>>>     'title_suggest_context': {
>>>         'field': 'title.suggest_context',
>>>         'default_suggester': SUGGESTER_COMPLETION,
>>>         'completion_options': {
>>>             'filters': {
>>>                 'title_suggest_tag': 'tag',
>>>                 'title_suggest_state': 'state',
>>>                 'title_suggest_publisher': 'publisher',
>>>             },
>>>             'size': 10,
>>>         },
>>>     },
>>> }
```

[http://localhost:8000/search/books-frontend/suggest/?title\\_suggest\\_context=M](http://localhost:8000/search/books-frontend/suggest/?title_suggest_context=M)

When talking about the queries made, we have the following. Multiple values per field are combined with OR:

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': ['History', 'Drama'],
```

(continues on next page)

(continued from previous page)

```
>>>     }
>>> }
```

The following works with OR as well, so it seems we have OR only. However, the following construction is more handy, since it allows us to play with boosting nicely. Also, it allows to provide *prefix* param (which in case of the example given below means that suggestions shall match both categories with “Child”, “Children”, “Childrend’s”). Simply put it’s treated as *prefix*, rather than *term*.

```
>>> completion={
>>>     'field': options['field'],
>>>     'size': 10,
>>>     'contexts': {
>>>         'tag': [
>>>             {'context': 'History'},
>>>             {'context': 'Drama', 'boost': 2},
>>>             {'context': 'Children', 'prefix': True},
>>>         ],
>>>     },
>>> }
```

Sample query for *category* filter:

```
/search/books-frontend/suggest/?title_suggest_context=M &title_suggest_tag=Art__2.0 &title_suggest_tag=Documentary__2.0__prefix &title_suggest_publisher=Apress
```

The query params would be:

```
query_params: <QueryDict: {
  'title_suggest_context': ['M'], 'title_suggest_tag': ['Art__2.0', 'Documentary__2.0__prefix'], 'title_suggest_publisher': ['Apress']}
>
```

Sample query for *geo* filter:

```
/search/address/suggest/?street_suggest_context=M &street_suggest_loc=43.66__-79.22__2.0__10000km
```

The query params would be:

```
query_params: <QueryDict: {
  'street_suggest_context': ['M'], 'street_suggest_loc': ['Art__43.66__-79.22__2.0__10000km'],
}>
```

### Returns

**get\_suggester\_query\_params**(*request*, *view*)

Get query params to be for suggestions.

### Parameters

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Request query params to filter on.

**Return type** dict

**classmethod prepare\_suggester\_fields**(*view*)

Prepare filter fields.

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Filtering options.

**Return type** dict

### 12.19.1.2.2 Submodules

#### 12.19.1.2.3 django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search module

Faceted search backend.

**class**

django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search.FacetedSearchFilterBackend

Bases: rest\_framework.filters.BaseFilterBackend

Faceted search backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     FacetedSearchFilterBackend
>>> )
>>> from elasticsearch_dsl import TermsFacet, DateHistogramFacet
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
>>>
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [FacetedSearchFilterBackend,]
>>>     faceted_search_fields = {
>>>         'title': 'title.raw', # Uses `TermsFacet` by default
>>>         'state': {
>>>             'field': 'state.raw',
>>>             'facet': TermsFacet,
>>>         },
>>>         'publisher': {
>>>             'field': 'publisher.raw',
>>>             'facet': TermsFacet,
>>>             'enabled': False,
>>>         },
>>>         'date_published': {
>>>             'field': 'date_published.raw',
>>>             'facet': DateHistogramFacet,
>>>             'options': {
>>>                 'interval': 'month',
>>>             },
>>>             'enabled': True,
>>>         },
>>>     }
>>> }
```

Facets make queries to be more heavy. That's why by default all facets are disabled and enabled only explicitly either in the filter options (*enabled* set to True) or via query params *?facet=state&facet=date\_published*.

**aggregate**(*request*, *queryset*, *view*)

Aggregate.

**Parameters**

- **request** –
- **queryset** –
- **view** –

**Returns**

**construct\_facets**(*request*, *view*)

Construct facets.

Turns the following structure:

```
>>> {
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
>>>         'enabled': False,
>>>     }
>>>     'date_published': {
>>>         'field': 'date_published',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

Into the following structure:

```
>>> {
>>>     'publisher': TermsFacet(field='publisher.raw'),
>>>     'publishing_frequency': DateHistogramFacet(
>>>         field='date_published.raw',
>>>         interval='month'
>>>     ),
>>> }
```

**faceted\_search\_param** = 'facet'

**filter\_queryset**(*request*, *queryset*, *view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*



**get\_faceted\_search\_query\_params**(*request*)

Get faceted search query params.

**Parameters** **request** (*rest\_framework.request.Request*) – Django REST framework request.

**Returns** List of search query params.

**Return type** list

**classmethod prepare\_faceted\_search\_fields**(*view*)

Prepare faceted search fields.

Prepares the following structure:

```
>>> {
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
>>>         'enabled': False,
>>>     }
>>>     'date_published': {
>>>         'field': 'date_published.raw',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }
```

**Parameters** **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) –

**Returns** Faceted search fields options.

**Return type** dict

#### 12.19.1.2.4 django\_elasticsearch\_dsl\_drf.filter\_backends.highlight module

Highlight backend.

**class** `django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend`

Bases: `rest_framework.filters.BaseFilterBackend`

Highlight backend.

Example:

```
>>> from django_elasticsearch_dsl_drf.filter_backends import (
>>>     HighlightBackend
>>> )
>>> from django_elasticsearch_dsl_drf.viewsets import (
>>>     BaseDocumentViewSet,
>>> )
>>>
>>> # Local article document definition
>>> from .documents import ArticleDocument
>>>
>>> # Local article document serializer
>>> from .serializers import ArticleDocumentSerializer
```

(continues on next page)

(continued from previous page)

```

>>>
>>> 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': {

```

(continues on next page)

(continued from previous page)

```
>>>         '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
>>>
```

(continues on next page)

(continued from previous page)

```
>>> class ArticleDocumentView(BaseDocumentViewSet):
>>>
>>>     document = ArticleDocument
>>>     serializer_class = ArticleDocumentSerializer
>>>     filter_backends = [SourceBackend,]
>>>     source = ["title"]
```

Example 2 (complex):

```
>>> # ...
>>>     source = ["title", "author.*"]
```

Example 3 (even more complex):

```
>>> # ...
>>>     source = {
>>>         "includes": ["title", "author.*"],
>>>         "excludes": [ "*.description" ]
>>>     }
```

Source can make queries lighter. However, it can break current functionality. Use it with caution.

**filter\_queryset**(*request, queryset, view*)

Filter the queryset.

**Parameters**

- **request** (*rest\_framework.request.Request*) – Django REST framework request.
- **queryset** (*elasticsearch\_dsl.search.Search*) – Base queryset.
- **view** (*rest\_framework.viewsets.ReadOnlyModelViewSet*) – View.

**Returns** Updated queryset.

**Return type** *elasticsearch\_dsl.search.Search*

### 12.19.1.2.7 Module contents

All filter backends.

### 12.19.1.3 django\_elasticsearch\_dsl\_drf.management package

#### 12.19.1.3.1 Subpackages

##### 12.19.1.3.1.1 django\_elasticsearch\_dsl\_drf.management.commands package

##### 12.19.1.3.1.2 Submodules

##### 12.19.1.3.1.3 django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch\_remove\_indexes module

```
class django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command(stdout=None,
                                                                                          stderr=None,
                                                                                          no_color=False,
                                                                                          force_color=False)
```

Bases: `django.core.management.base.BaseCommand`

**add\_arguments**(*parser*)

Entry point for subclassed commands to add custom arguments.

**handle**(\*args, \*\*options)

The actual logic of the command. Subclasses must implement this method.

**help** = 'Remove all indexes from Elasticsearch'



#### 12.19.1.3.1.4 Module contents

#### 12.19.1.3.2 Module contents

#### 12.19.1.4 django\_elasticsearch\_dsl\_drf.tests package

##### 12.19.1.4.1 Submodules

##### 12.19.1.4.2 django\_elasticsearch\_dsl\_drf.tests.base module

##### 12.19.1.4.3 django\_elasticsearch\_dsl\_drf.tests.data\_mixins module

##### 12.19.1.4.4 django\_elasticsearch\_dsl\_drf.tests.test\_faceted\_search module

##### 12.19.1.4.5 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_common module

##### 12.19.1.4.6 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_geo\_spatial module

##### 12.19.1.4.7 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_global\_aggregations module

##### 12.19.1.4.8 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_nested module

##### 12.19.1.4.9 django\_elasticsearch\_dsl\_drf.tests.test\_filtering\_post\_filter module

##### 12.19.1.4.10 django\_elasticsearch\_dsl\_drf.tests.test\_functional\_suggesters module

##### 12.19.1.4.11 django\_elasticsearch\_dsl\_drf.tests.test\_helpers module

##### 12.19.1.4.12 django\_elasticsearch\_dsl\_drf.tests.test\_highlight module

##### 12.19.1.4.13 django\_elasticsearch\_dsl\_drf.tests.test\_more\_like\_this module

##### 12.19.1.4.14 django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_common module

##### 12.19.1.4.15 django\_elasticsearch\_dsl\_drf.tests.test\_ordering\_geo\_spatial module

##### 12.19.1.4.16 django\_elasticsearch\_dsl\_drf.tests.test\_pagination module

##### 12.19.1.4.17 django\_elasticsearch\_dsl\_drf.tests.test\_pip\_helpers module

##### 12.19.1.4.18 django\_elasticsearch\_dsl\_drf.tests.test\_query\_friendly\_pagination module

##### 12.19.1.4.19 django\_elasticsearch\_dsl\_drf.tests.test\_search module

##### 12.19.1.4.20 django\_elasticsearch\_dsl\_drf.tests.test\_search\_multi\_match module

##### 12.19.1.4.21 django\_elasticsearch\_dsl\_drf.tests.test\_search\_simple\_query\_string module

##### 12.19.1.4.22 django\_elasticsearch\_dsl\_drf.tests.test\_serializers module

##### 12.19.1.4.23 django\_elasticsearch\_dsl\_drf.tests.test\_source module



Tests of `django_elasticsearch_dsl_drf.versions` module.

**setUp()**

Hook method for setting up the test fixture before exercising it.

**test\_elasticsearch\_dsl\_6\_3\_0()**

Tests as if we were using `elasticsearch_dsl==6.3.0`.

**test\_elasticsearch\_dsl\_7\_0\_0()**

Tests as if we were using `elasticsearch_dsl==7.0.0`.

#### 12.19.1.4.26 `django_elasticsearch_dsl_drf.tests.test_views` module

#### 12.19.1.4.27 `django_elasticsearch_dsl_drf.tests.test_wrappers` module

#### 12.19.1.4.28 Module contents

### 12.19.2 Submodules

#### 12.19.3 `django_elasticsearch_dsl_drf.analyzers` module

Analyzers.

#### 12.19.4 `django_elasticsearch_dsl_drf.apps` module

Apps.

**class** `django_elasticsearch_dsl_drf.apps.Config(app_name, app_module)`

Bases: `django.apps.config AppConfig`

Config.

**label** = `'django_elasticsearch_dsl_drf'`

**name** = `'django_elasticsearch_dsl_drf'`

#### 12.19.5 `django_elasticsearch_dsl_drf.compat` module

Transitional compatibility module. Contains various field wrappers and helpers for painless (testing of) Elastic 2.x to Elastic 5.x transition. This module is not supposed to solve all transition issues for you. Better move to Elastic 5.x as soon as possible.

**class** `django_elasticsearch_dsl_drf.compat.KeywordField(attr=None, **kwargs)`

Bases: `django_elasticsearch_dsl.fields.DEDField, elasticsearch_dsl.field.Keyword`

`django_elasticsearch_dsl_drf.compat.StringField(**kwargs)`

String field.

**Parameters** `kwargs` –

**Returns**

### 12.19.6 django\_elasticsearch\_dsl\_drf.constants module

Constants module. Contains Elasticsearch constants, lookup constants, functional constants, suggesters, etc.

### 12.19.7 django\_elasticsearch\_dsl\_drf.elasticsearch\_helpers module

`django_elasticsearch_dsl_drf.elasticsearch_helpers.delete_all_indices(with_protected=False)`

Delete all indices.

**Args:** `with_protected` (bool):

**Returns:** tuple: Tuple of two lists with removed and errored indices.

`django_elasticsearch_dsl_drf.elasticsearch_helpers.get_all_indices(with_protected=False)`

Get all indices.

**Args:** `with_protected` (bool):

**Returns:** list: List of indices.

### 12.19.8 django\_elasticsearch\_dsl\_drf.helpers module

Helpers.

`django_elasticsearch_dsl_drf.helpers.get_document_for_model(model)`

Get document for model given.

**Parameters** `model` (Subclass of `django.db.models.Model`.) – Model to get document index for.

**Returns** Document index for the given model.

**Return type** Subclass of `django_elasticsearch_dsl.Document`.

`django_elasticsearch_dsl_drf.helpers.get_index_and_mapping_for_model(model)`

Get index and mapping for model.

**Parameters** `model` (Subclass of `django.db.models.Model`.) – Django model for which to get index and mapping for.

**Returns** Index and mapping values.

**Return type** tuple.

`django_elasticsearch_dsl_drf.helpers.more_like_this(obj, fields, max_query_terms=25,  
min_term_freq=2, min_doc_freq=5,  
max_doc_freq=0, query=None)`

More like this.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-mlt-query.html>

**Parameters**

- `obj` (Instance of `django.db.models.Model` (sub-classed) model.) – Django model instance for which similar objects shall be found.
- `fields` (list) – Fields to search in.
- `max_query_terms` (int) –
- `min_term_freq` (int) –
- `min_doc_freq` (int) –
- `max_doc_freq` (int) –
- `query` (`elasticsearch_dsl.query.Q`) – Q query

**Returns** List of objects.

**Return type** `elasticsearch_dsl.search.Search`

Example:

```
>>> from django_elasticsearch_dsl_drf.helpers import more_like_this
>>> from books.models import Book
```

(continues on next page)

(continued from previous page)

```
>>> book = Book.objects.first()
>>> similar_books = more_like_this(
>>>     book,
>>>     ['title', 'description', 'summary']
>>> )
```

`django_elasticsearch_dsl_drf.helpers.sort_by_list(unsorted_dict, sorted_keys)`  
Sort an `OrderedDict` by list of sorted keys.

**Parameters**

- **unsorted\_dict** (*collections.OrderedDict*) – Source dictionary.
- **sorted\_keys** (*list*) – Keys to sort on.

**Returns** Sorted dictionary.

**Return type** `collections.OrderedDict`

### 12.19.9 django\_elasticsearch\_dsl\_drf.pagination module

Pagination.

**class** `django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination(*args, **kwargs)`  
Bases: `rest_framework.pagination.LimitOffsetPagination`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

A limit/offset pagination.

Example:

`http://api.example.org/accounts/?limit=100`   `http://api.example.org/accounts/?offset=400&limit=100`

**get\_facets**(*facets=None*)

Get facets.

**Parameters** *facets* –

**Returns**

**get\_paginated\_response**(*data*)

Get paginated response.

**Parameters** *data* –

**Returns**

**get\_paginated\_response\_context**(*data*)

Get paginated response data.

**Parameters** *data* –

**Returns**

**paginate\_queryset**(*queryset, request, view=None*)

**class** `django_elasticsearch_dsl_drf.pagination.Page(object_list, number, paginator, facets)`  
Bases: `django.core.paginator.Page`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

Page for Elasticsearch.

**class** `django_elasticsearch_dsl_drf.pagination.PageNumberPagination(*args, **kwargs)`  
Bases: `rest_framework.pagination.PageNumberPagination`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

Page number pagination.

A simple page number based style that supports page numbers as query parameters.

Example:

`http://api.example.org/accounts/?page=4`    `http://api.example.org/accounts/?page=4&page_size=100`

**django\_paginator\_class**

alias of *django\_elasticsearch\_dsl\_drf.pagination.Paginator*

**get\_facets**(*page=None*)

Get facets.

**Parameters** *page* –

**Returns**

**get\_paginated\_response**(*data*)

Get paginated response.

**Parameters** *data* –

**Returns**

**get\_paginated\_response\_context**(*data*)

Get paginated response data.

**Parameters** *data* –

**Returns**

**paginate\_queryset**(*queryset, request, view=None*)

Paginate a queryset.

Paginate a queryset if required, either returning a page object, or *None* if pagination is not configured for this view.

**Parameters**

- **queryset** –
- **request** –
- **view** –

**Returns**

**class** *django\_elasticsearch\_dsl\_drf.pagination.Paginator*(*object\_list, per\_page, orphans=0, allow\_empty\_first\_page=True*)

Bases: *django.core.paginator.Paginator*

Paginator for Elasticsearch.

**page**(*number*)

Returns a Page object for the given 1-based page number.

**Parameters** *number* –

**Returns**

**class** *django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNumberPagination*(\*args, \*\*kwargs)

Bases: *django\_elasticsearch\_dsl\_drf.pagination.PageNumberPagination*

Page number pagination.

A simple page number based style that supports page numbers as query parameters.

Example:

`http://api.example.org/accounts/?page=4`    `http://api.example.org/accounts/?page=4&page_size=100`

**django\_paginator\_class**

alias of *django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPaginator*

**orphans\_query\_param** = 'orphans'

**page\_size\_query\_param** = 'page\_size'

**paginate\_queryset**(*queryset, request, view=None*)

Paginate a queryset.

Paginate a queryset if required, either returning a page object, or *None* if pagination is not configured for this view.

**Parameters**

- **queryset** –
- **request** –
- **view** –

**Returns**

**class** `django_elasticsearch_dsl_drf.pagination.QueryFriendlyPaginator`(*object\_list, per\_page, orphans=0, allow\_empty\_first\_page=True*)

Bases: `django_elasticsearch_dsl_drf.pagination.Paginator`, `django_elasticsearch_dsl_drf.pagination.GetCountMixin`

Paginator for Elasticsearch.

**page**(*number*)

Returns a Page object for the given 1-based page number.

**Parameters** **number** –

**Returns**

## 12.19.10 django\_elasticsearch\_dsl\_drf.pip\_helpers module

Pip helpers module.

`django_elasticsearch_dsl_drf.pip_helpers.check_if_installed`(*package, installed\_packages=None*)

Check if package is installed.

**Parameters**

- **package** (*str*) –
- **installed\_packages** (*iterable*) –

**Returns**

**Return type** bool

`django_elasticsearch_dsl_drf.pip_helpers.get_installed_packages`(*with\_versions=False*)

Get installed packages.

**Parameters** **with\_versions** (*bool*) – If set to True, returned with versions.

**Returns**

**Return type** list

## 12.19.11 django\_elasticsearch\_dsl\_drf.serializers module

Serializers.

**class** `django_elasticsearch_dsl_drf.serializers.DocumentSerializer`(\*args, \*\*kwargs)

Bases: `rest_framework.serializers.Serializer`

A dynamic DocumentSerializer class.

**create**(*validated\_data*)

Create.

Do nothing.

**Parameters** `validated_data` –

**Returns**

**get\_fields()**

Get the required fields for serializing the result.

**update**(*instance, validated\_data*)

Update.

Do nothing.

**Parameters**

- **instance** –

- **validated\_data** –

**Returns**

**class** `django_elasticsearch_dsl_drf.serializers.DocumentSerializerMeta`(*name, bases, attrs*)

Bases: `rest_framework.serializers.SerializerMetaclass`

Metaclass for the DocumentSerializer.

Ensures that all declared subclasses implemented a Meta.

**class** `django_elasticsearch_dsl_drf.serializers.Meta`(*name, bases, attrs*)

Bases: `type`

Template for the DocumentSerializerMeta.Meta class.

**exclude** = ()

**field\_aliases** = {}

**field\_options** = {}

**fields** = ()

**ignore\_fields** = ()

**index\_aliases** = {}

**index\_classes** = ()

**search\_fields** = ()

**serializers** = ()

## 12.19.12 `django_elasticsearch_dsl_drf.utils` module

Utils.

**class** `django_elasticsearch_dsl_drf.utils.DictionaryProxy`(*mapping, meta=None*)

Bases: `object`

Dictionary proxy.

**to\_dict()**

To dict.

**Returns**

**class** `django_elasticsearch_dsl_drf.utils.EmptySearch`(\*args, \*\*kwargs)

Bases: `object`

Empty Search.

```

execute(*args, **kwargs)
highlight(*args, **kwargs)
property hits
sort(*args, **kwargs)
to_dict(*args, **kwargs)

```

### 12.19.13 django\_elasticsearch\_dsl\_drf.versions module

Contains information about the current Elasticsearch version in use, including (LTE and GTE).

`django_elasticsearch_dsl_drf.versions.get_elasticsearch_version(default=(2, 0, 0))`

Get Elasticsearch version.

**Parameters** **default** (*tuple*) – Default value. Mainly added for building the docs when 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.

**dictionary\_proxy**

alias of `django_elasticsearch_dsl_drf.utils.DictionaryProxy`

**document** = `None`

**document\_uid\_field** = `'id'`

**get\_object()**

Get object.

**get\_queryset()**

Get queryset.

**ignore** = `[]`

**pagination\_class**

alias of `django_elasticsearch_dsl_drf.pagination.PageNumberPagination`

**class** `django_elasticsearch_dsl_drf.viewsets.DocumentViewSet(*args, **kwargs)`

Bases: `django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`,  
`django_elasticsearch_dsl_drf.viewsets.SuggestMixin`, `django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin`

DocumentViewSet with suggest and functional-suggest mix-ins.

**class** `django_elasticsearch_dsl_drf.viewsets.FunctionalSuggestMixin`

Bases: `object`

Functional suggest mixin.

**functional\_suggest**(*request*)

Functional suggest functionality.

**Parameters** `request` –

**Returns**

**class** `django_elasticsearch_dsl_drf.viewsets.MoreLikeThisMixin`

Bases: `object`

More-like-this mixin.

**more\_like\_this**(*request*, *pk=None*, *id=None*)

More-like-this functionality detail view.

**Parameters** `request` –

**Returns**

**class** `django_elasticsearch_dsl_drf.viewsets.SuggestMixin`

Bases: `object`

Suggest mixin.

**suggest**(*request*)

Suggest functionality.

### 12.19.15 `django_elasticsearch_dsl_drf.wrappers` module

**class** `django_elasticsearch_dsl_drf.wrappers.Wrapper`

Bases: `object`

Wrapper.

Example: 

```
>>> from django_elasticsearch_dsl_drf.wrappers import dict_to_obj
>>> >>> mapping = { >>>
'country': { >>> 'name': 'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': {
>>> 'name': 'Amsterdam', >>> } >>> } >>> } >>> } >>> >>> wrapper = dict_to_obj(mapping) >>> wrap-
per.country.name >>> "Netherlands" >>> wrapper.country.province.name >>> "North Holland" >>> wrap-
per.country.province.city.name >>> "Amsterdam" >>> wrapper.as_dict >>> { >>> 'country': { >>> 'name':
'Netherlands', >>> 'province': { >>> 'name': 'North Holland', >>> 'city': { >>> 'name': 'Amsterdam', >>> }
>>> } >>> } >>> } >>> str(wrapper) >>> "Netherlands"
```

**property** `as_dict`

As dict.

**Returns**

**Return type** `dict`

**property** `as_json`

As JSON.

**Returns**

**Return type** `str`

`django_elasticsearch_dsl_drf.wrappers.dict_to_obj`(*mapping*)

dict to obj mapping.

**Parameters** `mapping` (*dict*) –

**Returns**

**Return type** `Wrapper`

`django_elasticsearch_dsl_drf.wrappers.obj_to_dict`(*obj*)

Wrapper to dict.

**Parameters** `obj` (`obj:Wrapper:`) –

**Returns**

**Return type** `dict`



### 12.19.16 Module contents

Integrate Elasticsearch DSL with Django REST framework.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### d

`django_elasticsearch_dsl_drf`, 269

`django_elasticsearch_dsl_drf.analyzers`, 261

`django_elasticsearch_dsl_drf.apps`, 261

`django_elasticsearch_dsl_drf.compat`, 261

`django_elasticsearch_dsl_drf.constants`, 262

`django_elasticsearch_dsl_drf.elasticsearch_helpers`, 262

`django_elasticsearch_dsl_drf.fields`, 173

`django_elasticsearch_dsl_drf.fields.common`, 171

`django_elasticsearch_dsl_drf.fields.helpers`, 172

`django_elasticsearch_dsl_drf.fields.nested_fields`, 172

`django_elasticsearch_dsl_drf.filter_backends`, 257

`django_elasticsearch_dsl_drf.filter_backends.aggregations`, 175

`django_elasticsearch_dsl_drf.filter_backends.aggregations.bucket_aggregations`, 175

`django_elasticsearch_dsl_drf.filter_backends.aggregations.metrics_aggregations`, 175

`django_elasticsearch_dsl_drf.filter_backends.aggregations.pipeline_aggregations`, 175

`django_elasticsearch_dsl_drf.filter_backends.faceted_search`, 251

`django_elasticsearch_dsl_drf.filter_backends.filtering`, 193

`django_elasticsearch_dsl_drf.filter_backends.filtering.common`, 175

`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial`, 183

`django_elasticsearch_dsl_drf.filter_backends.filtering.ids`, 188

`django_elasticsearch_dsl_drf.filter_backends.filtering.nested`, 190

`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter`, 192

`django_elasticsearch_dsl_drf.filter_backends.highlight`, 253

`django_elasticsearch_dsl_drf.filter_backends.mixins`, 255

`django_elasticsearch_dsl_drf.filter_backends.ordering`, 214

`django_elasticsearch_dsl_drf.filter_backends.ordering.common`, 209

`django_elasticsearch_dsl_drf.filter_backends.ordering.geo`, 212

`django_elasticsearch_dsl_drf.filter_backends.search`, 231

`django_elasticsearch_dsl_drf.filter_backends.search.base`, 227

`django_elasticsearch_dsl_drf.filter_backends.search.compound`, 228

`django_elasticsearch_dsl_drf.filter_backends.search.history`, 228

`django_elasticsearch_dsl_drf.filter_backends.search.multi`, 230

`django_elasticsearch_dsl_drf.filter_backends.search.query`, 223

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations`, 218

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 219

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 219

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.bucket_aggregations`, 220

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 220

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 221

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.bucket_aggregations`, 222

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.metrics_aggregations`, 231

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.pipeline_aggregations`, 231

`django_elasticsearch_dsl_drf.filter_backends.search.query.aggregations.bucket_aggregations`, 256

`django_elasticsearch_dsl_drf.filter_backends.suggester`, 244

`django_elasticsearch_dsl_drf.filter_backends.suggester.fur`

[235](#)  
django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.native,  
[239](#)  
django\_elasticsearch\_dsl\_drf.helpers, [262](#)  
django\_elasticsearch\_dsl\_drf.management, [260](#)  
django\_elasticsearch\_dsl\_drf.management.commands,  
[260](#)  
django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch\_remove\_indexes,  
[258](#)  
django\_elasticsearch\_dsl\_drf.pagination, [263](#)  
django\_elasticsearch\_dsl\_drf.pip\_helpers, [265](#)  
django\_elasticsearch\_dsl\_drf.serializers, [265](#)  
django\_elasticsearch\_dsl\_drf.tests, [261](#)  
django\_elasticsearch\_dsl\_drf.tests.test\_versions,  
[260](#)  
django\_elasticsearch\_dsl\_drf.utils, [266](#)  
django\_elasticsearch\_dsl\_drf.versions, [267](#)  
django\_elasticsearch\_dsl\_drf.viewsets, [267](#)  
django\_elasticsearch\_dsl\_drf.wrappers, [268](#)

## INDEX

### A

- `add_arguments()` (`django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command` method), 258
- `aggregate()` (`django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchFilterBackend` method), 252
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 190
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 206
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 192
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 208
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 208
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 255
- `apply_filter()` (`django_elasticsearch_dsl_drf.filter_backends.mixins.FilterBackendMixin` class method), 255
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 176
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 195
- `apply_filter_prefix()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 194
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 176
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 196
- `apply_filter_range()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 194
- `apply_filter_regexp()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 177
- `apply_filter_regexp()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 194
- `apply_filter_regexp()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 179
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 177
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 196
- `apply_filter_term()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 195
- `apply_filter_terms()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 177
- `apply_filter_terms()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 195
- `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 191
- `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend` class method), 206
- `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 193
- `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 208
- `apply_query()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.post_filter.PostFilterFilteringFilterBackend` class method), 255
- `apply_query_contains()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 178
- `apply_query_contains()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 195
- `apply_query_endswith()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 178
- `apply_query_endswith()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_query_exclude()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 178
- `apply_query_exclude()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_query_exists()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend` class method), 179
- `apply_query_exists()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_query_exists()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend` class method), 196
- `apply_query_geo_bounding_box()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilterBackend` class method), 184
- `apply_query_geo_bounding_box()` (`django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilterBackend` class method), 184

class method), 200  
 apply\_query\_geo\_distance() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_distance\_filtering.FilterBackend class method), 184  
 apply\_query\_geo\_distance() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.GeoDistanceFilterBackend class method), 201  
 apply\_query\_geo\_polygon() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_polygon\_filtering.FilterBackend class method), 184  
 apply\_query\_geo\_polygon() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.GeoPolygonFilterBackend class method), 201  
 apply\_query\_geo\_shape() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_shape\_filtering.FilterBackend class method), 185  
 apply\_query\_geo\_shape() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.GeoShapeFilterBackend class method), 201  
 apply\_query\_gt() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.gt\_filtering.FilterBackend class method), 179  
 apply\_query\_gt() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.GtFilterBackend class method), 197  
 apply\_query\_gte() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.gte\_filtering.FilterBackend class method), 180  
 apply\_query\_gte() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.GteFilterBackend class method), 197  
 apply\_query\_in() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.in\_filtering.FilterBackend class method), 180  
 apply\_query\_in() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.InFilterBackend class method), 197  
 apply\_query\_isnull() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.isnull\_filtering.FilterBackend class method), 180  
 apply\_query\_isnull() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.IsnullFilterBackend class method), 197  
 apply\_query\_lt() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.lt\_filtering.FilterBackend class method), 181  
 apply\_query\_lt() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.LtFilterBackend class method), 198  
 apply\_query\_lte() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.lte\_filtering.FilterBackend class method), 181  
 apply\_query\_lte() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.LteFilterBackend class method), 198  
 apply\_query\_size() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.size\_filtering.FilterBackend class method), 237  
 apply\_query\_size() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.SizeFilterBackend class method), 245  
 apply\_query\_wildcard() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.wildcard\_filtering.FilterBackend class method), 181  
 apply\_query\_wildcard() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.WildcardFilterBackend class method), 198  
 class method), 198  
 apply\_suggester\_completion() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.native.Suggester class method), 241  
 apply\_suggester\_completion() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.Suggester class method), 248  
 apply\_suggester\_completion\_match() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional.Suggester class method), 237  
 apply\_suggester\_completion\_match() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FunctionalSuggester class method), 245  
 apply\_suggester\_completion\_prefix() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional.Suggester class method), 238  
 apply\_suggester\_completion\_prefix() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FunctionalSuggester class method), 246  
 apply\_suggester\_phrase() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.Suggester class method), 248  
 apply\_suggester\_term() (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.Suggester class method), 248  
 as\_dict() (django\_elasticsearch\_dsl\_drf.wrappers.Wrapper class method), 268  
 as\_json() (django\_elasticsearch\_dsl\_drf.wrappers.Wrapper class method), 268  
 BaseDocumentViewSet (class in django\_elasticsearch\_dsl\_drf.viewsets), 267  
 BaseSearchFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search), 227  
 BaseSearchFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search.base), 227  
 BaseSearchQueryBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search.query\_backend), 227  
 BooleanField (class in django\_elasticsearch\_dsl\_drf.fields), 173  
 BooleanField (class in django\_elasticsearch\_dsl\_drf.filter\_backends.filtering)



`django_elasticsearch_dsl_drf.fields.common`),  
171

## C

`CharField` (class in `django_elasticsearch_dsl_drf.fields`),  
173

`CharField` (class in `django_elasticsearch_dsl_drf.fields.common`),  
171

`check_if_installed()` (in module `django_elasticsearch_dsl_drf.pip_helpers`),  
265

`clean_queryset()` (`django_elasticsearch_dsl_drf.filter_backends.suggester_functional.FunctionalSuggesterFilterBackend` method), 238

`clean_queryset()` (`django_elasticsearch_dsl_drf.filter_backends.suggester_functional.FunctionalSuggesterFilterBackend` method), 246

## D

`Command` (class in `django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes`),  
258

`CompoundSearchFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.search.compound`),  
232

`CompoundSearchFilterBackend` (class in `DefaultOrderingFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.ordering`),  
228

`Config` (class in `django_elasticsearch_dsl_drf.apps`),  
261

`construct_facets()` (`django_elasticsearch_dsl_drf.filter_backends.suggested_search.FacetedSearchFilterBackend` method), 252

`construct_nested_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.history.SearchFilterBackend` method), 228

`construct_nested_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend` method), 233

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.history.SearchFilterBackend` method), 229

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.base.BaseSearchQueryBackend` class method), 218

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.BaseSearchQueryBackend` class method), 223

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match.MatchQueryBackend` class method), 219

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase.MatchPhraseQueryBackend` class method), 219

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.match_phrase_prefix.MatchPhrasePrefixQueryBackend` class method), 220

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchPhrasePrefixQueryBackend` class method), 223

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchPhraseQueryBackend` class method), 223

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.MatchQueryBackend` class method), 224

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search.query_backends.multi_match.MultiMatchQueryBackend` class method), 220

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search` class method), 224

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search` class method), 225

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search` class method), 222

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search` class method), 226

`construct_search()` (`django_elasticsearch_dsl_drf.filter_backends.search` class method), 233

`create()` (`django_elasticsearch_dsl_drf.serializers.DocumentSerializer` method), 265

`DateField` (class in `django_elasticsearch_dsl_drf.fields`),  
173

`DateField` (class in `django_elasticsearch_dsl_drf.fields.common`),  
171

`DefaultOrderingFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.ordering`),  
214

`DefaultOrderingFilterBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.ordering.common`),  
214

`delete_all_indices()` (in module `django_elasticsearch_dsl_drf.elasticsearch_helpers`),  
261

`dict_to_obj()` (in module `django_elasticsearch_dsl_drf.wrappers`),  
261

`dictionary_proxy` (`django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet` method), 261

`DictionaryProxy` (class in `django_elasticsearch_dsl_drf.viewsets.BaseDocumentViewSet`), 261

`django_elasticsearch_dsl_drf`  
module, 266

`django_elasticsearch_dsl_drf.analyzers`  
module, 266

`django_elasticsearch_dsl_drf.apps`  
module, 266

`django_elasticsearch_dsl_drf.compat`  
module, 266

`django_elasticsearch_dsl_drf.constants`  
module, 266

`django_elasticsearch_dsl_drf.elasticsearch_helpers`  
module, 266

`django_elasticsearch_dsl_drf.fields`  
module, 266

`django_elasticsearch_dsl_drf.fields.common`  
module, 266

`django_elasticsearch_dsl_drf.fields.helpers`  
module, 172



document\_uid\_field(*django\_elasticsearch\_dsl\_drf.views.FilterBackendMixin* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.ordering*), 267

DocumentSerializer (class in *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.ordering*), 211

DocumentSerializerMeta (class in *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.ordering*), 214

DocumentViewSet (class in *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.ordering*), 216

E

EmptySearch (class in *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.search*), 227

exclude (*django\_elasticsearch\_dsl\_drf.serializers.Meta* *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.search*), 231

execute() (*django\_elasticsearch\_dsl\_drf.utils.EmptySearch* *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.search*), 230

extract\_field\_name() *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.search*), 234

extract\_field\_name() (*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester* *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester*), 238

extract\_field\_name() (*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester* *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester*), 238

F

faceted\_search\_param *filter\_queryset()* (*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester*), 242

FacetedSearchFilterBackend (class in *FilterBackendMixin* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.mixins*), 255

field\_aliases (*django\_elasticsearch\_dsl\_drf.serializers.Meta* *FilteringFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering*), 193

field\_options (*django\_elasticsearch\_dsl\_drf.serializers.Meta* *FilteringFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common*), 175

fields (*django\_elasticsearch\_dsl\_drf.serializers.Meta* *FilteringFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common*), 173

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search.FacetedSearchFilterBackend* *FloatField* (class in *django\_elasticsearch\_dsl\_drf.fields*), 171

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend* *FloatField* (class in *django\_elasticsearch\_dsl\_drf.fields.common*), 171

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.FilteringFilterBackend* *functional\_suggest()* (*django\_elasticsearch\_dsl\_drf.viewsets.FunctionalSuggesterFilterBackend*), 267

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_spatial.GeoSpatialFilteringFilterBackend* *FunctionalSuggesterFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.suggester*), 244

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.ids.IdsFilterBackend* *FunctionalSuggesterFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.functional\_suggester*), 236

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.ids.IdsFilterBackend* *FunctionalSuggesterFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.viewsets*), 267

filter\_queryset() (*django\_elasticsearch\_dsl\_drf.filter\_backends.highlight.HighlightFilterBackend* (class in *django\_elasticsearch\_dsl\_drf.viewsets*), 267

## G

<b>G</b>					(django_elasticsearch_dsl_drf.filter_backends.search.SearchFilterBackend), 234
GeoPointField	(class in	django_elasticsearch_dsl_drf.fields), 174	get_default_ordering_params()		(django_elasticsearch_dsl_drf.filter_backends.ordering.common.DefaultOrderingParams), 210
GeoPointField	(class in	django_elasticsearch_dsl_drf.fields.nested_fields), 172	get_default_ordering_params()		(django_elasticsearch_dsl_drf.filter_backends.ordering.DefaultOrderingParams), 215
GeoShapeField	(class in	django_elasticsearch_dsl_drf.fields), 174	get_document_for_model()	(in module	django_elasticsearch_dsl_drf.helpers), 262
GeoShapeField	(class in	django_elasticsearch_dsl_drf.fields.nested_fields), 172	get_elasticsearch_version()	(in module	django_elasticsearch_dsl_drf.versions), 267
GeoSpatialFilteringFilterBackend	(class in	django_elasticsearch_dsl_drf.filter_backends.filtering), 200	get_faceted_search_query_params()		(django_elasticsearch_dsl_drf.filter_backends.faceted_search.FacetedSearchQueryParams), 252
GeoSpatialFilteringFilterBackend	(class in	django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial), 183	get_facets()	(django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination), 263	
GeoSpatialOrderingFilterBackend	(class in	django_elasticsearch_dsl_drf.filter_backends.ordering), 215	get_facets()	(django_elasticsearch_dsl_drf.pagination.PageNumberPagination), 264	
GeoSpatialOrderingFilterBackend	(class in	django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial), 212	get_field()	(django_elasticsearch_dsl_drf.filter_backends.search.query_params), 221	
get_all_indices()	(in module	django_elasticsearch_dsl_drf.elasticsearch_helpers), 262	get_field()	(django_elasticsearch_dsl_drf.filter_backends.search.query_params), 222	
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend method), 182	get_field()	(django_elasticsearch_dsl_drf.filter_backends.search.query_params), 226	
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend method), 199	get_fields()	(django_elasticsearch_dsl_drf.serializers.DocumentSerializer), 266	
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend method), 191	get_filter_field_nested_path()		(django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend method), 191
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend method), 191	get_filter_field_nested_path()		(django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend method), 206
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend method), 206	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilteringFilterBackend method), 182
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.post_filters.PostFilteringFilterBackend method), 193	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend method), 199
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.filtering.PostFilteringFilterBackend method), 208	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.GeoSpatialFilteringFilterBackend method), 185
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchFilterBackend method), 227	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend method), 201
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchFilterBackend method), 231	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.nested.NestedFilteringFilterBackend method), 191
get_coreschema_field()		(django_elasticsearch_dsl_drf.filter_backends.search.historical_search.HistoricalSearchFilterBackend method), 230	get_filter_query_params()		(django_elasticsearch_dsl_drf.filter_backends.filtering.NestedFilteringFilterBackend method), 207
get_coreschema_field()			get_geo_bounding_box_params()		



<code>(django_elasticsearch_dsl_drf.filter_backends.filtering.backends.FilterBackend class method), 185</code>	<code>get_index_and_mapping_filter_model() (in module django_elasticsearch_dsl_drf.helpers), 262</code>
<code>get_geo_bounding_box_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 202</code>	<code>get_installed_packages() (in module django_elasticsearch_dsl_drf.pip_helpers), 265</code>
<code>get_geo_distance_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.C26SpatialFilteringFilterBackend class method), 186</code>	<code>get_object() (django_elasticsearch_dsl_drf.viewsets.BaseDocumentView class method), 265</code>
<code>get_geo_distance_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 202</code>	<code>get_ordering_query_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.common.C26SpatialOrderingFilterBackend class method), 217</code>
<code>get_geo_distance_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.C26SpatialOrderingFilterBackend class method), 212</code>	<code>get_ordering_query_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.common.C26SpatialOrderingFilterBackend class method), 215</code>
<code>get_geo_distance_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend class method), 216</code>	<code>get_ordering_query_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.C26SpatialFilteringFilterBackend class method), 186</code>
<code>get_geo_polygon_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.C26SpatialFilteringFilterBackend class method), 186</code>	<code>get_ordering_query_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend class method), 203</code>
<code>get_geo_shape_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.geo_spatial.C26SpatialFilteringFilterBackend class method), 187</code>	<code>get_ordering_query_params() (django_elasticsearch_dsl_drf.filter_backends.ordering.OrderingFilterBackend class method), 187</code>
<code>get_geo_shape_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.GeoSpatialFilteringFilterBackend class method), 203</code>	<code>get_paginated_response() (django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination class method), 263</code>
<code>get_geo_spatial_field_name() (django_elasticsearch_dsl_drf.filter_backends.ordering.geo_spatial.C26SpatialOrderingFilterBackend method), 213</code>	<code>get_paginated_response() (django_elasticsearch_dsl_drf.pagination.PageNumberPagination class method), 263</code>
<code>get_geo_spatial_field_name() (django_elasticsearch_dsl_drf.filter_backends.ordering.GeoSpatialOrderingFilterBackend method), 216</code>	<code>get_paginated_response_context() (django_elasticsearch_dsl_drf.pagination.LimitOffsetPagination class method), 263</code>
<code>get_gte_lte_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.common.FilterBackend class method), 182</code>	<code>get_paginated_response_context() (django_elasticsearch_dsl_drf.pagination.PageNumberPagination class method), 263</code>
<code>get_gte_lte_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.FilteringFilterBackend class method), 199</code>	<code>get_query_backends() (django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchBackend class method), 221</code>
<code>get_highlight_query_params() (django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend class method), 254</code>	<code>get_query_options() (django_elasticsearch_dsl_drf.filter_backends.search.BaseSearchBackend class method), 221</code>
<code>get_ids_query_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend class method), 189</code>	<code>get_query_options() (django_elasticsearch_dsl_drf.filter_backends.search.query_backend.QueryBackend class method), 225</code>
<code>get_ids_query_params() (django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend class method), 205</code>	<code>get_query_options() (django_elasticsearch_dsl_drf.filter_backends.search.query_backend.QueryBackend class method), 225</code>
<code>get_ids_values() (django_elasticsearch_dsl_drf.filter_backends.filtering.ids.IdsFilterBackend class method), 189</code>	<code>get_query_options() (django_elasticsearch_dsl_drf.filter_backends.search.query_backend.QueryBackend class method), 225</code>
<code>get_ids_values() (django_elasticsearch_dsl_drf.filter_backends.filtering.IdsFilterBackend class method), 205</code>	<code>get_query_options() (django_elasticsearch_dsl_drf.filter_backends.search.query_backend.QueryBackend class method), 225</code>

class method), 227  
 get\_queryset() (django\_elasticsearch\_dsl\_drf.viewsets.BaseDjangoElasticsearchDrfViewSet, Context)  
 method), 267  
 get\_range\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.FilteringFilterBackend  
 class method), 182  
 get\_range\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.NestedFilteringFilterBackend, filter\_backends.suggester.SuggesterFilterBackend)  
 class method), 199  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.combined\_filtering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.function\_based\_suggester.FilteringFilterBackend)  
 method), 183  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FunctionBasedSuggesterFilterBackend)  
 method), 200  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested\_filtering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.native\_suggester.FilteringFilterBackend)  
 method), 191  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.NestedFilteringFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.SuggesterFilterBackend)  
 method), 207  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filtering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.SuggesterFilterBackend)  
 method), 193  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filtering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.BooleanField)  
 method), 208  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.CharField)  
 method), 211  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.DateField)  
 method), 218  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.FloatField)  
 method), 211  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.IntegerField)  
 method), 227  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.IPAddressField)  
 method), 231  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.ListField)  
 method), 230  
 get\_schema\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.ObjectField)  
 method), 234  
 get\_search\_query\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.ListField)  
 method), 227  
 get\_search\_query\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.ListField)  
 method), 231  
 get\_search\_query\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.fields.common.ObjectField)  
 method), 230  
 get\_search\_query\_params() (django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.combined\_ordering.FilteringFilterBackend, django\_elasticsearch\_dsl\_drf.management.commands.elasticsearch)  
 method), 234

## H

method), 258

help(`django_elasticsearch_dsl_drf.management.commands.elasticsearch_remove_indexes.Command` attribute), 258

highlight()`(django_elasticsearch_dsl_drf.utils.EmptySearch` method), 267

highlight\_param(`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend` attribute), 254

HighlightBackend (class in `django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend` attribute), 227

hits (`django_elasticsearch_dsl_drf.utils.EmptySearch` property), 267

I

ids\_query\_param(`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend` attribute), 189

ids\_query\_param(`django_elasticsearch_dsl_drf.filter_backends.highlight.HighlightBackend` attribute), 205

IdsFilterBackend (class in `MatchPhrasePrefixQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering`), `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 204

IdsFilterBackend (class in `MatchPhrasePrefixQueryBackend` (class in `django_elasticsearch_dsl_drf.filter_backends.filtering.ids`), `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 188

ignore(`django_elasticsearch_dsl_drf.viewsets.BaseDocumentView` attribute), 267

ignore\_fields(`django_elasticsearch_dsl_drf.serializers.Meta` attribute), 266

index\_aliases(`django_elasticsearch_dsl_drf.serializers.Meta` attribute), 266

index\_classes(`django_elasticsearch_dsl_drf.serializers.Meta` attribute), 266

IntegerField (class in `django_elasticsearch_dsl_drf.fields`), 174

IntegerField (class in `django_elasticsearch_dsl_drf.fields.common`), 172

IPAddressField (class in `django_elasticsearch_dsl_drf.fields`), 174

IPAddressField (class in `django_elasticsearch_dsl_drf.fields.common`), 172

K

KeywordField (class in `django_elasticsearch_dsl_drf.compat`), 261

L

label (`django_elasticsearch_dsl_drf.apps.Config` attribute), 261

LimitOffsetPagination (class in `django_elasticsearch_dsl_drf.pagination`), 263

ListField(class in `django_elasticsearch_dsl_drf.fields`), 174

ListField(class in `django_elasticsearch_dsl_drf.fields.nested_fields`), 172

M

matching(`django_elasticsearch_dsl_drf.filter_backends.search.base.BaseSearchBackend` attribute), 231

matching(`django_elasticsearch_dsl_drf.filter_backends.search.multi_match.MultiMatchBackend` attribute), 230

matching(`django_elasticsearch_dsl_drf.filter_backends.search.MultiMatchBackend` attribute), 232

matching(`django_elasticsearch_dsl_drf.filter_backends.search.simple_query.SimpleQueryBackend` attribute), 231

matching(`django_elasticsearch_dsl_drf.filter_backends.search.SimpleQueryBackend` attribute), 234

MatchPhrasePrefixQueryBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 223

MatchPhraseQueryBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 219

MatchQueryBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 224

MatchQueryBackend (class in `django_elasticsearch_dsl_drf.filter_backends.search.query_backend` attribute), 219

Meta(class in `django_elasticsearch_dsl_drf.serializers`), 266

module

`django_elasticsearch_dsl_drf`, 269

`django_elasticsearch_dsl_drf.analyzers`, 261

`django_elasticsearch_dsl_drf.apps`, 261

`django_elasticsearch_dsl_drf.compat`, 261

`django_elasticsearch_dsl_drf.constants`, 262

`django_elasticsearch_dsl_drf.elasticsearch_helpers`, 262

`django_elasticsearch_dsl_drf.fields`, 173

`django_elasticsearch_dsl_drf.fields.common`, 171

`django_elasticsearch_dsl_drf.fields.helpers`, 172

[django\\_elasticsearch\\_dsl\\_drf.fields.nested\\_field](#), 172  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends](#), 220  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 257  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 220  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 175  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 221  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 175  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 222  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 175  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 231  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 175  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.search\\_query\\_backend](#), 231  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.source](#), 251  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 193  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 175  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 183  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 239  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 188  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 244  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 190  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 244  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 192  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 244  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 253  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 255  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 214  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 214  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 209  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 261  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 212  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 260  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 231  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 266  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 227  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 227  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 228  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 228  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 230  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 230  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 223  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 268  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 218  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 218  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 219  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 224  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 219  
[django\\_elasticsearch\\_dsl\\_drf.filter\\_backends.suggester](#), 219



220  
MultiMatchSearchFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common),  
django\_elasticsearch\_dsl\_drf.filter\_backends.search), 210  
232 orphans\_query\_param  
MultiMatchSearchFilterBackend (class in (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNum  
django\_elasticsearch\_dsl\_drf.filter\_backends.search.multi\_match), 264  
230

## N

name (django\_elasticsearch\_dsl\_drf.apps.Config attribute), 261  
NestedField (class in django\_elasticsearch\_dsl\_drf.fields), 174  
NestedField (class in django\_elasticsearch\_dsl\_drf.fields.nested\_fields), 172  
NestedFilteringFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common), 205  
NestedFilteringFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested), 190  
NestedQueryBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search.common), 225  
NestedQueryBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search.nested), 221

## O

obj\_to\_dict() (in module django\_elasticsearch\_dsl\_drf.wrappers), 268  
ObjectField (class in django\_elasticsearch\_dsl\_drf.fields), 174  
ObjectField (class in django\_elasticsearch\_dsl\_drf.fields.nested\_fields), 173  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 210  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 211  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 215  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 213  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 217  
ordering\_param(django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common.DefaultOrderingFilterBackend attribute), 218  
OrderingFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.ordering.common), 217

## P

Page (class in django\_elasticsearch\_dsl\_drf.pagination), 263  
page() (django\_elasticsearch\_dsl\_drf.pagination.Paginator method), 264  
page() (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPaginator method), 265  
page\_size\_query\_param (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNumber attribute), 264  
PageNumberPagination (class in django\_elasticsearch\_dsl\_drf.pagination), 263  
paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.LimitOffsetPagination method), 263  
paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.PageNumberPagination method), 264  
paginate\_queryset() (django\_elasticsearch\_dsl\_drf.pagination.QueryFriendlyPageNumber method), 265  
pagination\_class(django\_elasticsearch\_dsl\_drf.viewsets.BaseDocumentView attribute), 267  
Paginator (class in django\_elasticsearch\_dsl\_drf.pagination), 264  
PostFilterFilteringFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.filtering), 207  
PostFilterFilteringFilterBackend (class in django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter), 192  
prepare\_faceted\_search\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.faceted\_search.FacetedSearch class method), 253  
prepare\_faceted\_search\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.common.FilteringFilterBackend class method), 200  
prepare\_faceted\_search\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_spatial.GeoSpatialOrderingFilterBackend class method), 204  
prepare\_faceted\_search\_fields() (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.geo\_spatial.GeoSpatialOrderingFilterBackend class method), 204

[prepare\\_filter\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.nested.NestedFilteringFilterBackend class method), 191

[prepare\\_filter\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.PostFilterFilteringFilterBackend class method), 207

[prepare\\_filter\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.PostFilterFilteringFilterBackend class method), 193

[prepare\\_filter\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.filtering.post\_filter.PostFilterFilteringFilterBackend class method), 208

[prepare\\_highlight\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.highlight.HighlightFilterBackend class method), 254

[prepare\\_suggester\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FriendlyPageNumberPaginator class method), 238

[prepare\\_suggester\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FriendlyPageNumberPaginator class method), 246

[prepare\\_suggester\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FriendlyPageNumberPaginator class method), 244

[prepare\\_suggester\\_fields\(\)](#)  
 (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FriendlyPageNumberPaginator class method), 250

## Q

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 227

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 231

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 228

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 232

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 230

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 232

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 234

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 231

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 234

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 231

[query\\_backends](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 234

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 219

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 219

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 220

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 223

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 224

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 221

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 222

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 226

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 227

[query\\_type](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.backends.SearchBackend attribute), 264

[QueryFriendlyPaginator](#) (class in django\_elasticsearch\_dsl\_drf.pagination), 265

[S](#)

[search\\_fields](#) (django\_elasticsearch\_dsl\_drf.serializers.Meta attribute), 266

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 232

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 230

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 230

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 232

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 234

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 231

[search\\_param](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.search.BaseSearchBackend attribute), 234

[SearchFilterBackend](#) (class in django\_elasticsearch\_dsl\_drf.filter\_backends), 231

[SimpleQueryStringSearchFilterBackend](#) (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search), 234

[SimpleQueryStringSearchFilterBackend](#) (class in django\_elasticsearch\_dsl\_drf.filter\_backends.search), 228

[serialize\\_queryset](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FunctionalSuggesterBackend method), 247

[serialize\\_queryset](#) (django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.FunctionalSuggesterBackend method), 247

serializers(*django\_elasticsearch\_dsl\_drf.serializers.Meta* method), 261  
*attribute*), 266 *to\_dict()* (*django\_elasticsearch\_dsl\_drf.utils.DictionaryProxy*  
*setUp()* (*django\_elasticsearch\_dsl\_drf.tests.test\_versions.VersionsTest* method), 266  
*method*), 261 *to\_dict()* (*django\_elasticsearch\_dsl\_drf.utils.EmptySearch*  
SimpleQueryStringQueryBackend (class in method), 267  
*django\_elasticsearch\_dsl\_drf.filter\_backends.search\_backend.internal\_value()*  
226 (*django\_elasticsearch\_dsl\_drf.fields.ListField*  
SimpleQueryStringQueryBackend (class in method), 174  
*django\_elasticsearch\_dsl\_drf.filter\_backends.search\_backend.internal\_value()* (*simple\_query\_string*),  
222 (*django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ListField*  
SimpleQueryStringSearchFilterBackend (class in method), 172  
*django\_elasticsearch\_dsl\_drf.filter\_backends.search\_backend.internal\_value()*  
234 (*django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*  
SimpleQueryStringSearchFilterBackend (class in method), 173  
*django\_elasticsearch\_dsl\_drf.filter\_backends.search\_backend.internal\_value()*,  
231 (*django\_elasticsearch\_dsl\_drf.fields.ObjectField*  
*sort()* (*django\_elasticsearch\_dsl\_drf.utils.EmptySearch* method), 174  
*method*), 267 *to\_representation()*  
*sort\_by\_list()* (in module (*django\_elasticsearch\_dsl\_drf.fields.BooleanField*  
*django\_elasticsearch\_dsl\_drf.helpers*), 263 *method*), 173  
SourceBackend (class in *to\_representation()*  
*django\_elasticsearch\_dsl\_drf.filter\_backends.source*), (*django\_elasticsearch\_dsl\_drf.fields.CharField*  
256 *method*), 173  
split\_lookup\_complex\_multiple\_value() *to\_representation()*  
(*django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin* *django\_elasticsearch\_dsl\_drf.fields.common.BooleanField*  
*class method*), 255 *method*), 171  
split\_lookup\_complex\_value() *to\_representation()*  
(*django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin* *django\_elasticsearch\_dsl\_drf.fields.common.CharField*  
*class method*), 256 *method*), 171  
split\_lookup\_filter() *to\_representation()*  
(*django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin* *django\_elasticsearch\_dsl\_drf.fields.common.DateField*  
*class method*), 256 *method*), 171  
split\_lookup\_name() *to\_representation()*  
(*django\_elasticsearch\_dsl\_drf.filter\_backends.mixins.FilterBackendMixin* *django\_elasticsearch\_dsl\_drf.fields.common.FloatField*  
*class method*), 256 *method*), 171  
StringField() (in module *to\_representation()*  
*django\_elasticsearch\_dsl\_drf.compat*), 261 (*django\_elasticsearch\_dsl\_drf.fields.common.IntegerField*  
*suggest()* (*django\_elasticsearch\_dsl\_drf.viewsets.SuggestMixin* method), 172  
*method*), 268 *to\_representation()*  
SuggesterFilterBackend (class in (*django\_elasticsearch\_dsl\_drf.fields.common.IPAddressField*  
*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester*), *method*), 172  
247 *to\_representation()*  
SuggesterFilterBackend (class in (*django\_elasticsearch\_dsl\_drf.fields.DateField*  
*django\_elasticsearch\_dsl\_drf.filter\_backends.suggester.native* *method*), 173  
240 *to\_representation()*  
SuggestMixin (class in (*django\_elasticsearch\_dsl\_drf.fields.FloatField*  
*django\_elasticsearch\_dsl\_drf.viewsets*), 268 *method*), 173  
*to\_representation()*  
T (*django\_elasticsearch\_dsl\_drf.fields.IntegerField*  
*test\_elasticsearch\_dsl\_6\_3\_0()* *method*), 174  
(*django\_elasticsearch\_dsl\_drf.tests.test\_versions.VersionsTest* *to\_representation()*  
*method*), 261 (*django\_elasticsearch\_dsl\_drf.fields.IPAddressField*  
*test\_elasticsearch\_dsl\_7\_0\_0()* *method*), 174  
(*django\_elasticsearch\_dsl\_drf.tests.test\_versions.VersionsTest* *to\_representation()*

*(django\_elasticsearch\_dsl\_drf.fields.ListField*  
*method), 174*  
**to\_representation()**  
*(django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ListField*  
*method), 172*  
**to\_representation()**  
*(django\_elasticsearch\_dsl\_drf.fields.nested\_fields.ObjectField*  
*method), 173*  
**to\_representation()**  
*(django\_elasticsearch\_dsl\_drf.fields.ObjectField*  
*method), 174*  
**to\_representation()** (in module  
*django\_elasticsearch\_dsl\_drf.fields.helpers),*  
 172

## U

**update()** (*django\_elasticsearch\_dsl\_drf.serializers.DocumentSerializer*  
*method), 266*

## V

**VersionsTest** (class in  
*django\_elasticsearch\_dsl\_drf.tests.test\_versions),*  
 260

## W

**Wrapper** (class in *django\_elasticsearch\_dsl\_drf.wrappers),*  
 268