

Listings And Search

Listings And Search

- [How do I create a custom sort order for the listings pages?](#)
- [What are the available documents for listings and search?](#)
- [How do I re-index listing and search documents?](#)
- [How do I add custom facets to listing and search?](#)
- [How do I setup the search index?](#)
- [How do I re-build the listings and search index?](#)
- [How do I extend the document structure for listings and search?](#)
- [How do I filter listings?](#)
- [How to Extend the Elastic Search Listing Model](#)

How do I create a custom sort order for the listings pages?

When you set the sort get parameter in a listing pages url (e.g. ?sort=name-az), a method is called on the `\Aero\Search\Elastic\Query\Builder` class (for name-az the method **sortByNameAz** is called). You can find the default sort by methods in the **Aero\Search\Elastic\Concerns\CanBeSorted** trait.

To add a custom sort to the listings page you therefore just need to macro a method to the `\Aero\Search\Elastic\Query\Builder` class using the correct naming convention.

This example adds an ascending and descending sort by for the summary (to use it you need to add ?sort=summary-az or ?sort=summary-za to the listing page url):

```
<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        \Aero\Search\Elastic\Query\Builder::macro('sortBySummaryAz', function () {
            $this->query->addSort([
                'string-sort.summary' => [
                    'order' => 'asc',
                ],
            ]);
        });

        \Aero\Search\Elastic\Query\Builder::macro('sortBySummaryZa', function () {
```

```

        $this->query->addSort([
            'string-sort.summary' => [
                'order' => 'desc',
            ],
        ]);
    });
}
}

```

If you try the above code you will notice that it throws an **Elastica\Exception\ResponseException** error. This is because the **string-sort.summary** field used in the Elasticsearch sort by query does not exist.

To add **summary** as a field to the **string-sort** key you need to use the ****\Aero\Search\Elastic\Documents\ListingDocument::add()** method to define a closure that will return an array of data to merge into the document structure. It's important to know that when you change the document structure you need to reindex. You can reindex using the **php artisan aero:search:reindex** command.

```

<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        \Aero\Search\Elastic\Documents\ListingDocument::add('string-sort', function ($document)
        {
            return [
                'summary' => $document->getModel()->product->getTranslation('summary',
request()->store()->language),
            ];
        });

        \Aero\Search\Elastic\Query\Builder::macro('sortBySummaryAz', function () {
            $this->query->addSort([
                'string-sort.summary' => [

```

```

        'order' => 'asc',
    ],
    ]);
});

\Aero\Search\Elastic\Query\Builder::macro('sortBySummaryZa', function () {
    $this->query->addSort([
        'string-sort.summary' => [
            'order' => 'desc',
        ],
    ],
    ]);
});
}
}

```

Advanced Query

It's possible to do a more advanced query for your Elasticsearch sort by. You can make use of [Elasticsearch's Painless scripting language](#)

This example code shows how to add a rand sort that will execute a Java script to sort the listings.

```

<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;
use Elastica\Script\Script;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        \Aero\Search\Elastic\Query\Builder::macro('sortByRand', function () {
            $this->query->addSort([
                '_script' => [
                    'type' => 'number',

```

```

        'script' => [
            'lang' => Script::LANG_PAINLESS,
            'source' => $this->loadJavaFile(__DIR__.'../../scripts/random-
sort.java'),
            'params' => $this->parameters,
        ],
        'order' => 'asc',
    ],
    1);
});
}
}

```

This Java file is located in a scripts folder outside of the src folder:

```

def sorts = new ArrayList();

Random rand = new Random();
sorts.add(rand.nextInt(1000000));

if (sorts.size() == 0) {
    sorts.add(0);
}

Collections.min(sorts.stream().map(Double::doubleToRawLongBits).collect(Collectors.toList()));

```

What are the available documents for listings and search?

There are 2 documents available:

```
Aero\Search\Elastic\Documents\ListingsDocument
```

This document is used to hold information about the listings that are shown on the storefront.

```
Aero\Search\Elastic\Documents\ProductDocument
```

This document is used to hold information about the products within the store's catalog. It is generally used in the backend admin, to allow for quick access when managing products.

How do I re-index listing and search documents?

Aero automatically manages indexing changes to documents. For example, if a product's stock level changes the document is updated to reflect the current version. Adding and removing documents is also taken care of for you, sparing the need to schedule in daily tasks to update the store's entire catalog. However, in cases where a large amount of data has been manually changed, or an addition to the document structure has been made, a re-index of all documents can be carried out using the following command:

```
php artisan aero:search:reindex
```

If there have only been modifications to a certain document (for example, only the listing document structure has been extended) then the document **type** can be passed as an option, which will only re-index the documents with the provided type. The available types correspond to the names of the elastic documents, **listing**, and **product**.

```
php artisan aero:search:reindex --type=listing
```

How do I add custom facets to listing and search?

Faceted navigation in eCommerce allows customers the ability to refine down a catalog of products based on their filter criteria.

In order to define facets against a document, you must first register the facet group. This should be done within the **boot()** method of a **ServiceProvider** or the **setup()** method of a module **ServiceProvider**.

```
\Aero\Search\FacetSettings::registerFacet('stock', 'Stock Status');
```

The document structure can then be extended to add the facet to the document's data.

```
\Aero\Search\Elastic\Documents\ListingDocument::add('string-facets', function ($document) {  
    $stock = $document->getModel()->availableStock() > 0;  
  
    return [  
        \Aero\Search\Elastic\Facet::create('stock', 'Stock Status', (int) $stock, $stock ?  
        'In-stock' : 'Out of stock'),  
    ];  
});
```

Re-index the documents after any modifications to the document structure.

How do I setup the search index?

Firstly, you should ensure that the Elasticsearch host information is defined in the store's `.env`. If you're running the service directly on your machine (either natively or through Docker), you'll most likely use the following details:

```
ELASTICSEARCH_HOST=localhost  
ELASTICSEARCH_PORT=9200
```

When using a remote connection, ensure the host does not contain the port and URI scheme.

It is also important to ensure the **STORE_IDENTIFIER** is defined. This value is automatically added to the project's `.env` when using the Aero CLI tool to install and configure a store.

To create the Elasticsearch index and apply the mappings, run the following command within the root directory of your project:

```
php artisan aero:search:install
```

How do I re-build the listings and search index?

At times (especially during the development phase of a store build), it may be necessary to delete and documents and freshly index them. This process can be achieved using the following command:

```
php artisan aero:search:rebuild
```

Be careful running the re-build command on a production store.

Due to the nature of how this process runs, there will be a temporary period where there are no listings or products shown.

How do I extend the document structure for listings and seach?

One of the most useful features of the listings and search system is the ability to add custom data into the document. Typically, this will be data to be consumed in the code that forms the storefront listings page.

To extend a document, use the **`**add()`** method, to define a closure that will return an **array** of data to merge into the document structure. This is typically done from within a **ServiceProvider**. If the code is unique to a particular store it could be placed in the boot method of the **AppServiceProvider**. If it is to be included as part of a module, then it can be added in the setup method of the module's **ServiceProvider**.

```
\Aero\Search\Elastic\Documents\ListingDocument::add('search-result', function ($document) {  
    $colors = $document->getModel()->visibleVariants()->pluck('attributes')->flatten()-  
>filter(function ($attribute) {  
        return strpos(strtolower($attribute->group->name), 'color') !== false;  
    })->pluck('name')->unique()->values()->all();  
  
    return [  
        'colors' => $colors,  
    ];  
});
```

The available section of the document that can be extended are:

- search-result - search-data

How do I filter listings?

There may be occasions where you need to pre-filter listings. For example, you may want to ensure that all listings are connected to a certain sub-store or Dropship provider, or all share a tag that is set by your application rather than applied by the customer. Below is an example that extends the Elasticsearch listings query to scope the results to only those that have stock:

```
\Aero\Search\Elastic\Repositories\ElasticListings::extend(function ($listings) {  
    $filter = new \Elastica\Query\Term(['number-sort.has-stock' => ['value' => 1]]);  
  
    $listings->base()->getQuery()->addFilter($filter);  
});
```

How to Extend the Elastic Search Listing Model

The `Aero\Search\Elastic\Models\Listing` model is built up from an array of data returned from elastic search. You can modify the model's attributes through the `Aero\Search\Elastic\Pipelines\ListingAttributes` pipeline.

For example you can change every listing's name to be "Example".

It's important to note that this pipeline is run for every listing that is loaded, everytime. If you use this approach to add a resource intensive calculation it's important to take steps such as caching so performance isn't impacted.

```
<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Search\Elastic\Pipelines\ListingAttributes;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        ListingAttributes::extend(function ($attributes) {
            $attributes['name'] = 'Example';

            return $attributes;
        });
    }
}
```