

Themes

Themes

- [How do I add featured products to the homepage of my store?](#)
- [How do I add a contact form to my storefront?](#)
- [How do I convert a listings page into a landing page?](#)
- [How to add a header image to your listings page](#)
- [How to add sashes to your listing card / product page](#)
- [Anatomy of a theme](#)
- [How do I create a custom twig function?](#)
- [How do I install a theme?](#)
- [How do I create a custom theme?](#)
- [How do I compile my theme styles?](#)
- [What are the themes available?](#)
- [How do I display the prices from my price list?](#)
- [How do I use a custom Vue component in the storefront?](#)
- [How to Implement Subscriptions on the Product Page](#)
- [How do I implement subscriptions on the product page?](#)
- [How do I give split listings names generated from their attributes?](#)
- [Listing items per page](#)

How do I add featured products to the homepage of my store?

Listing collections can be used to group products together and display them on specified pages.

Install

Run this command at the root of your project

```
composer require aerocargo/listing-collections
```

Create your section

Create the file;

```
your_theme_name/resources/sections/listing-collection.twig
```

Add the loop;

```
{% set collection = listings([{"tag_name.en": id}], 8) %}

{% component "listings" with {listings: collection} csr %}
<div class="aero-listing-collection">
  <h3 class="aero-listing-collection__title">{{ title }}</h3>
  <ul class="aero-listings grid grid-cols-4">
    <li v-for="listing in listings.data" class="aero-listing">
      {% snippet "listing-card" %}
    </li>
  </ul>
</div>
```

```
</div>

{% endcomponent %}
```

Reference your section where you'd like it to display and pass through the variables, here we'll use the tag name `best-sellers` this will be referenced in the next step.

```
{% section "listing-collection" with {
  id: 'best-sellers',
  title: 'Best sellers',
} %}
```

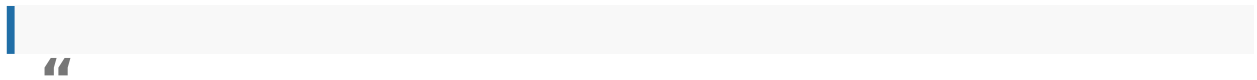
Tag some products

In the admin create a tag which matches the tag name set previously and tag your products.

Add Swiper to create a carousel (optional)

There's no need to add SwiperJS as the files are included in the boilerplate theme.

Reference the css at the top of your section:



We use the once tag so that swiperjs will not be added to the page multiple times if we have more than one instance of the section

```
{% css %}
  {% once 'swipercss' %}
    {{ theme_css('swiper.min.css') }}
  {% endonce %}
{% endcss %}
```

Add the relevant classes to your collection:

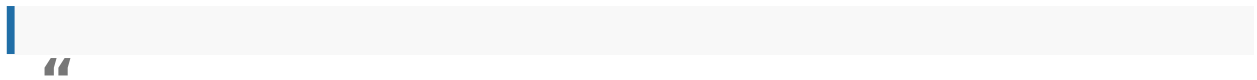
```
{% component "listings" with {listings: collection} csr %}
  <div class="aero-listing-collection">
    <h3 class="aero-listing-collection__title">{{ title }}</h3>
```

```

<div class="aero-listing-collection__carousel swiper-container" id="{{ id }}">
  <ul class="aero-listings swiper-wrapper">
    <li v-for="listing in listings.data" class="aero-listing swiper-slide">
      {% snippet "listing-card" %}
    </li>
  </ul>
</div>
</div>
{% endcomponent %}

```

Add your JS to the bottom of your section:



We use the once tag so that swiperjs will not be added to the page multiple times if we have more than one instance of the section

```

{% js %}
  {% once 'swiperjs' %}
    <script src="{{ theme_asset('swiper.min.js') }}"></script>
  {% endonce %}
  <script>
    function initSwiper({{ id | replace({'-':' '}) }}){
      new Swiper('#{{ id }}', {
        slidesPerView: 2,
        slidesPerGroup: 2,
        spaceBetween: 16,
        longSwipesRatio: 0,
        loop: false,
        allowTouchMove: true,
        watchOverflow: true,
        breakpoints: {
          // @screen lg
          1024: {
            slidesPerView: 3,
            slidesPerGroup: 3,
            spaceBetween: 16,
            allowTouchMove: true,
            pagination: {
              clickable: true,
            },
          },
        },
      });
    }
  </script>

```

```
    },
    // @screen xl
    1280: {
      slidesPerView: 4,
      slidesPerGroup: 4,
      spaceBetween: 24,
      allowTouchMove: true,
      pagination: {
        clickable: true,
      },
    },
  }
});

}

</script>
{% endjs %}
```

How do I add a contact form to my storefront?

Introduction

Forms Location

```
your_theme_name>resources>views>forms
```

Any files created inside this directory can be included on a page using `{% form "contact" %}`

Create a Contact Form

Create your form in the forms location, for example `contact.twig`:

```
<form action="{{ route('form', 'contact') }}" method="post">
  {{ csrf_field() }}
  <div>
    <label for="name">Name:</label>
    <input id="name" type="text" name="name" autocomplete="name" required
placeholder="First Name" value="{{ old('name') }}">
  </div>
  <div>
    <label for="name">Email Address:</label>
    <input id="email" type="text" name="email" autocomplete="email" required
placeholder="Email Address" value="{{ old('email') }}">
  </div>
  <div>
    <label for="name">Your Message</label>
    <textarea id="message" type="text" name="message" autocomplete="off" required
placeholder="Your message">{{ old('message') }}</textarea>
```

```

</div>
<div>
    <button type="submit">Submit</button>
</div>
</form>

```

Session message

```

{% if session('success') %}
    <div role="alert">
        <span>Your message has been sent.</span>
    </div>
{% endif %}

```

Redirect to success page (optional)

Add the following to your project in `app>providers>AppServiceProvider.php`

```

public function boot()
{
    \Aero\Store\Http\Responses\FormSubmit::extend(function ($builder) {
        if ($builder->form === 'contact') {
            $builder->setRedirect('/success');
        }
    });
}

```

Create success page

1. Log in to the admin and navigate to `/admin/content/pages` 2. Create new page
2. Set page name to success 2. Add your content

1. Log in to the admin and navigate to `/admin/configuration/mail` 2. Create new mail
2. Set label to a name of your choice 2. Select "System" layout 3. Add the email address of the recipient 4. Select "Form Submitted" from the event list 1. Set this to match your route, in our case **contact**
3. Continue to the next step
4. Configure your email

5. Set the subject to the subject of your email 2. Input the snippet of code that will be outputted in the body of your email (see example below)
6. Save

```
{% for field, value in fields %}  
    <p>{{ field }}: {{ value }}</p>  
{% endfor %}
```

Test your email

You'll need to set up a mail server for example when testing locally you could use [Mailtrap](#)

Include your form on a contact page

1. Log in to the admin and navigate to `/admin/content/pages`
2. Create a new page
2. Set the name to what you'd like the title of your page to be 2. In the content area add `{% form "contact" %}` where you'd like the form to display.
3. Save

How do I convert a listings page into a landing page?

Update AppServiceProvider

Add the following namespaces:

```
use Aero\Catalog\Models\Category;
use Aero\Responses\ResponseBuilder;
use Aero\Store\Http\Responses\ListingsPage;
use Illuminate\Support\ServiceProvider;
use Closure;
```

Extend the listings page:

```
class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        ListingsPage::extend(function (ResponseBuilder $builder, Closure $next){
            //Get the first slug which is a category
            $category = $builder->slugs->models()->whereInstanceOf(Category::class)->last();

            if ($category) {
                if (($landingPage = $category->additional('landing_page'))
                    && ($builder->request->has('categories') || $category-
>additional('always_landing_page')) {
                    $builder->setView( $landingPage);
                    $builder->setData('block_group', $category->additional('block_group'));
                }
            }

            return $next($builder);
        });
    }
}
```

```
}  
}
```

Create your landing page

Create your landing page twig file for example:

```
your_theme_name>resources>views>example.twig
```

1. Log in to the admin and navigate to `/admin/catalog/categories`
2. Select the category you wish to convert to a landing page
3. Set your additional attributes against the category:

Name	Value
landing_page	example
always_landing_page	true
block_group	example

1. Save

Edit your page

Update your `example.twig` file, you are now able to reference `block_group` to create custom blocks:

In this simple example, we load the listing page with a loop of blocks set by our `block_group` `example.promos`, followed by a loop of our listing-cards.

```
{% layout "main" %}  
  
{% set variables = {  
    categories: categories,  
    filters: filters,  
    listings: listings,  
    search_term: search_term,  
    sort_by_options: sort_by_options,  
    summary: summary | default(null),  
    summary_length: summary | striptags | length,  
    summary_all: false,  
    description: description,
```

```

    heading: heading,
    filters_open: false,
    similar_products: similar_products,
    sortBy: false,
} %}

{% component "listings" with variables %}
    <div data-listings-scroll="">
        {% if block_exists(block_group ~ '.promos') %}
            {% set promos = block_items(block_group ~ '.promos', null, {
                sizes: {
                    small: {
                        width: 168,
                        height: 160,
                    },
                    medium: {
                        width: 227,
                        height: 208,
                    },
                    large: {
                        width: 236,
                        height: 216,
                    },
                },
            }) %}
            {% endif %}
            <div class="aero-promos">
                {% for promo in promos %}
                    <div class="aero-promo">
                        {% if block_exists(block_group ~ '.promos') %}
                            <div class="aero-promo__image">
                                {{ promo | raw }}
                            </div>
                        {% endif %}
                    </div>
                {% endfor %}
            </div>
            <template v-if="listings.data.length">
                <template v-for="listing in listings.data">
                    {% snippet "listing-card" %}

```

</template>

</template>

</div>

{% endcomponent %}

How to add a header image to your listings page

On a listings page we can pass through an image from the admin using the `combination` variable in the `listings.twig` view. The combination is the data model that holds the information for the particular listings page being viewed - since listings can be made up of a category, a manufacturer or both.

We'll output the entire image block:

```
{{ combination.image_block | raw }}
```

However we'll need to pass some sizes:

```
{{ combination.image_block.render({
  sizes: {
    small: {
      width: 360,
      height: 360,
    },
    medium: {
      width: 1024,
      height: 540,
    },
    large: {
      width: 1920,
      height: 900,
    },
  },
}) | raw }}
```

Finally we can add some styling to our header to create a nice component for our theme (optional):

```
<div class="aero-listing-header flex flex-col lg:flex-row gap-4 lg:gap-8">
  <div class="aero-listing-header__image w-full lg:flex-1">
    {{ combination.image_block.render({sizes: {small: {width: 360, height: 360}, medium:
```

```
{width: 1024, height: 540}, large: {width: 1920, height: 900}})) | raw }}  
  
  </div>  
  <div class="aero-listing-header__content">  
    <h1 class="aero-listing-header__title text-xl lg:text-3xl">  
      <template v-if="search_term">Search for "{ { search_term } }" </template>  
      <template v-else>{ { heading } } </template>  
    </h1>  
  </div>  
</div>
```

How to add sashes to your listing card / product page

There are several ways to achieve this feature. We'll highlight a few below.

Method 1: Using tags

Assigning a tag group for use as "sashes" will allow you to bulk tag many products so that they have the same sash.

Update AppServiceProvider

Add the following namespaces:

```
use Aero\Catalog\Models\Tag;  
use Aero\Catalog\Models\TagGroup;  
use Aero\Common\Facades\Settings;  
use Aero\Search\Elastic\Documents\ListingDocument;
```

Create a setting for your tag_sash id and extend the listings document:

```
class AppServiceProvider extends ServiceProvider  
{  
    public function boot()  
    {  
        Settings::group('search', function($group){  
            $group->eloquent('tag_sash', TagGroup::class);  
        });  
  
        ListingDocument::add('search-result', function ($document) {  
            if (! setting('search.tag_sash')) {  
                return [];  
            }  
        })  
    }  
}
```

```

    $listing = $document->getModel();

    $sashes = $listing->allTags
        ->where('tag_group_id', setting('search.tag_sash')->id)
        ->map(function (Tag $tag) use ($document) {
            return $tag->getTranslation('name', $document->getLanguage());
        })->unique()->values()->all();

    return ['sashes' => $sashes];
});
}
}

```

Add the following to your `listing-card.twig` snippet:

```

<template v-if="listing.sashes">
    <span v-for="sash in listing.sashes"
        class="aero-product-sash"
        v-text="sash.replace('-', ' ')"></span>
</template>

```

Set the Tag Sash setting in the admin

1. Log in to the admin and navigate to `/admin/settings/manage/search`
2. Set your **Tag Sashes** the tag collection that you'd like to display as sashes
3. Save

Method 2: Using additional attributes

Setting the sash values in additional attributes gives you control over aspects of the sash, for example its colour or position. Don't forget, additional attributes can be set via a bulk action in the admin, which will save time applying these sashes to multiple products at once.

Update AppServiceProvider

Add the following namespace:

```
use Aero\Search\Elastic\Documents\ListingDocument;
```

Extend the listing document so that the sash data is available when looping over each listing on the listings page:

```
class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        ListingDocument::add('search-result', function ($document) {
            $product = $document->getModel()->product;

            return [
                'sash' => [
                    'text' => $product->additional('sash-text'),
                    'colour' => $product->additional('sash-colour'),
                ],
            ];
        });
    }
}
```

Listing card snippet

Add the following to your `listing-card.twig` snippet:

```
<template v-if="listing.sash && listing.sash.text">
    <span class="aero-product-sash" :style="{backgroundColor: listing.sash.colour}" v-
text="listing.sash.text"></span>
</template>
```

Anatomy of a theme

```
└─ layout
  └─ page
    └─ section
      └─ snippet
        └─ element
          └─ element
      └─ section
        └─ snippet
          └─ element
            └─ element
        └─ snippet
          └─ element
            └─ element
```

Elements are the base level of the Aero design system these can be used for things such as Buttons which don't require other components.

```
{% element 'example' with {
  key: 'value',
} %}
```

Snippets are a collection of two or more elements which can be used in more multiple places.

```
{% snippet 'example' with {
  key: 'value',
} %}
```

Sections are a collection of two or more snippets and elements which can be used in more multiple places.

```
{% section 'example' with {  
    key: 'value',  
} %}
```

Javascript and CSS

The js and css tag will push any javascript to the bottom of the DOM and css to the top. This allows javascript to be used within sections, snippets and elements without breaking the flow of the DOM.

```
{% css %}  
    {{ theme_css('example.css') }}  
{% endcss %}
```

```
{% js %}  
    <script>  
        alert( 'Hello, world!' );  
    </script>  
{% endjs %}
```

Using the once tag you are able to create a re-usable section without having duplicate stylesheets / javascript.

```
{% css %}  
    {% once 'example' %}  
        {{ theme_css('example.css') }}  
    {% endonce %}  
    {% once 'example' %}  
        {{ theme_css('example.css') }}  
    {% endonce %}  
{% endcss %}
```

```
{{ theme_css('example.css') }}
```

How do I create a custom twig function?

Aero can extend Twig to add custom functions that can then be used throughout the shop. To create a custom twig function, we have to access a service provider that is of our concern - e.g. **AppServiceProvider** or a **Service Provider** belonging to a module.

In order to create a custom twig function, we have to add the following piece of code to a **Service Provider** of our choice:

```
TwigFunctions::add(new TwigFunction('new_function', function () {  
    // Handle the function call  
}));
```

Anything within the above function will be processed after **new_function()** is ran in the frontend.

Twig functions can also be passed values directly from the frontend.

Example:

```
new TwigFunction('image_factory', static function ($width, $height, $path = '') {  
    return new ImageFactory($width, $height, $path);  
})
```

The above twig function requires the width and the height to call another function in the backend responsible for creating an Image Factory.

If instantiated correctly, the function should now be callable by its defined name.

Options and functions

There are a number of options and functions that are readily available by Twig Functions. The usage of these functions can be explained in the [official documentation](#)

How do I install a theme?

Install the Aero Theme components

To install a theme we must first install our theme UI package to the vendor directory of your project by running this command in the root of your project;

```
composer require aerocommerce/theme-ui --dev
```

The theme UI package gives you access to a number of themes, our themes require a couple of additional modules to add extra functionality which we'll install next.

Install the required modules

Our themes make use of the listing collections and account area modules to further enhance both the admin and the storefront.

```
composer require aerocargo/listing-collections aerocommerce/account-area
```

What's next?

Now we've installed all of the required modules we can move on to;

- [What are the themes available?](#) - [How do I create a custom theme?](#)

How do I create a custom theme?

If you've created a custom .yml config file you can build a theme by referencing it as the --config, for example if you created your file in the root of your project you can run:

```
php artisan theme:build your_theme_name --config=custom.yml
```

Additional information

When running the theme build command you will be prompted to run `npm run init` this will compile your theme styles upon build and is therefore recommended. This will be followed by a prompt to enable the theme, if you choose not to enable the theme this can be done later.

It's possible to confirm these prompts during the install by adding the following options to the end of your command:

Option	Description
-e	Enable the theme
-r	Compile styles
-er	Enable the theme & compile styles

For example:

```
php artisan theme:build your_theme_name --config=vendor/aerocommerce/theme-ui/config/shadow.yml -er
```

Switching between themes

To switch theme you can do so by adding/updating this line of your `.env` file:

```
AERO_THEME=your_theme_name
```

How do I compile my theme styles?

To compile the styles of your theme you first need to change directory in terminal to your theme folder:

```
cd your_project/themes/your_theme_name
```

Compiling your styles

Our themes are built to use npm, the `webpack.mix.js` file contains the required code to compile tailwind styles into a minified css file.

Development

Running the following command in terminal will constantly watch changes made to your theme and add any tailwind helpers to the `main.css` file if you add any. This is the best method to use **while in development**, however the css file size will be much bigger and disrupt your site speeds.

```
npm run watch
```

Production

When your project is ready for production you can run this alternative command in your terminal this will minify your styles and remove any unused css from the final build giving you optimal site speeds.

```
npm run production
```

What are the themes available?

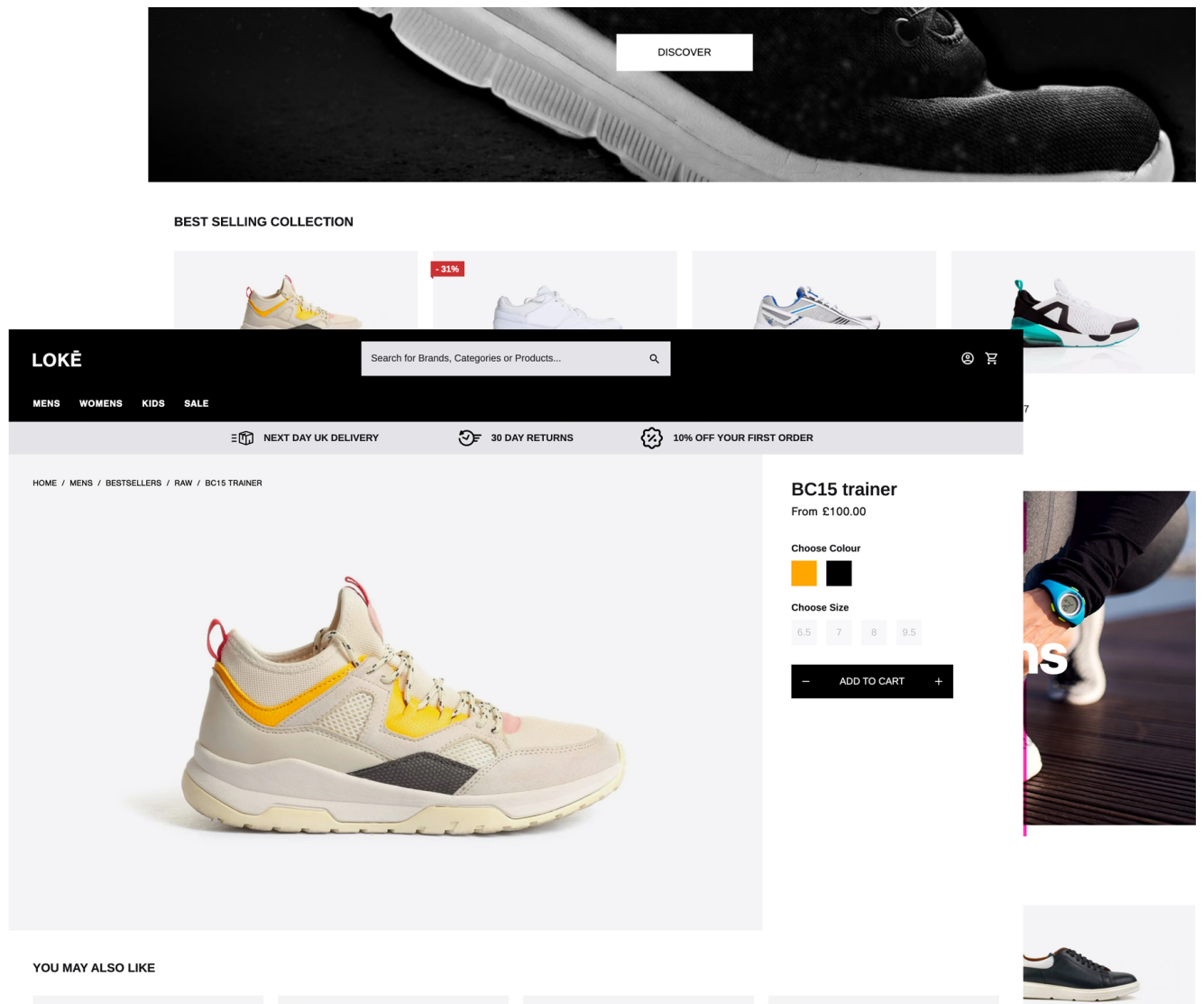
We have a number of official themes available. These themes are designed to be used as a starting point for your project and have been meticulously crafted with conversion, speed and responsive design at their core.

Our official themes are built using Tailwind CSS and can be modified using the `tailwind.config.js` file and editing the various Twig files.

During the set up of a new Aero store, a theme can be selected and will automatically be installed and configured. To install a theme after this point, please see [How do I install a theme?](#)

Phantom

Phantom is our fashion focused theme, it is bold and striking, the use of large imagery keeps the products as the main focus of your store.



Demo

Our demo store Lokē is built using the Phantom theme;

<https://loke.store.aerocommerce.com/>

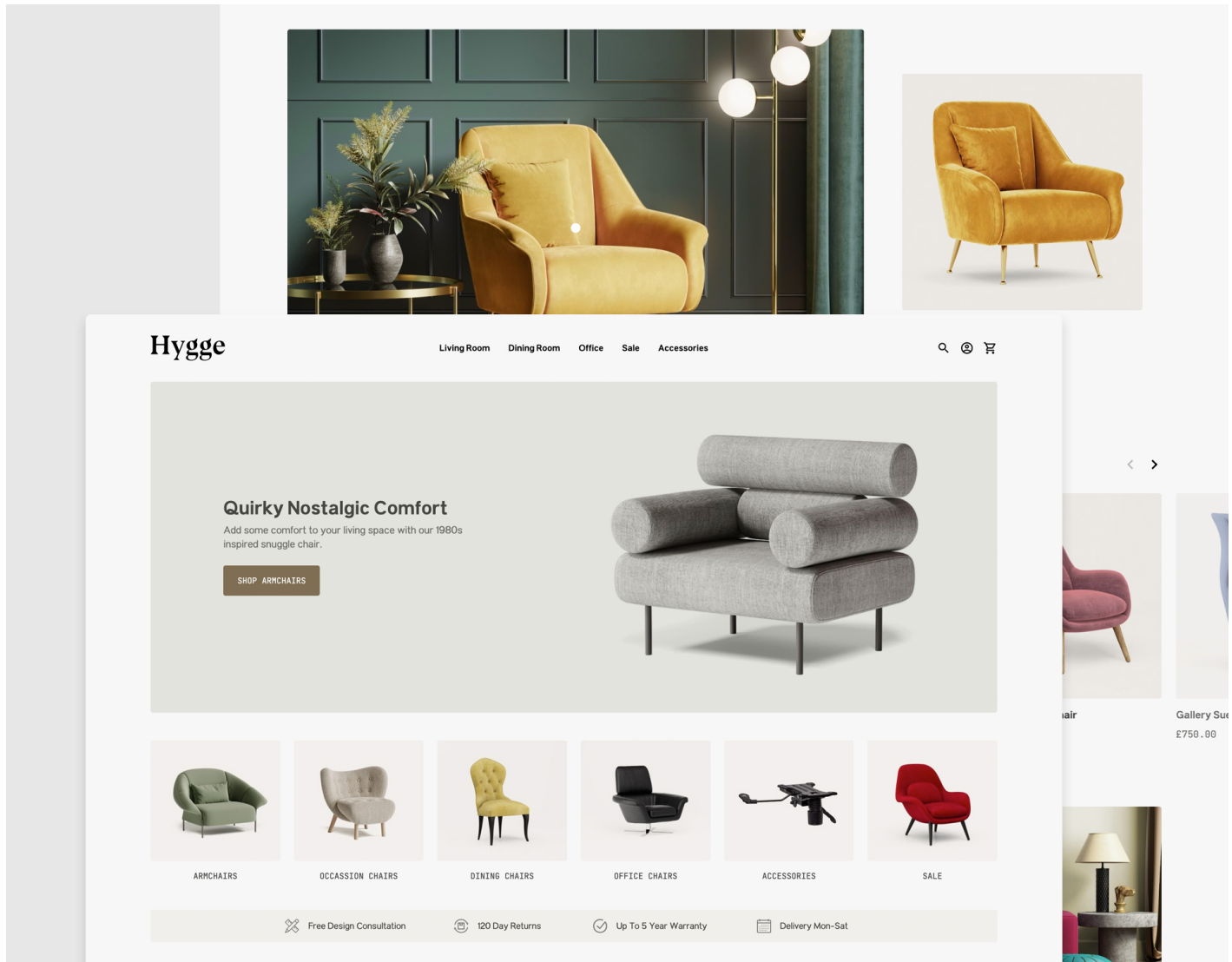
Install

To install the Phantom theme run the following command;

```
php artisan theme:build your_theme_name --config=vendor/aerocommerce/theme-ui/config/phantom.yml
```

Shadow

Shadow is our furniture and homewear theme, it is clean and simple to use for the customer and makes great use of whitespace to provide a sense of luxury.



Demo

Our demo store Hygge is built using the Shadow theme;

<https://hygge.store.aerocommerce.com/>

Install

To install the Shadow theme run the following command;

```
php artisan theme:build your_theme_name --config=vendor/aerocommerce/theme-ui/config/shadow.yml
```

How do I display the prices from my price list?

You can display what the sales price was before the price list was applied, how much of a discount the customer is getting, and whether the price they're seeing is specifically for them (such as a price caused by them being in a specific customer group).

Listings Page Code Snippet

```
<span v-if="listing.price && listing.price.not_regular_sale_price">
  <span class="text-red">
    <span v-if="listing.price.not_regular_sale_price && listing.price.price_list_entry &&
listing.price.price_list_entry.is_for_customer">Your price </span>
    <span v-else-if="listing.price.is_reduced && listing.price.is_ranged">Now from </span>
    <span v-else-if="listing.price.is_reduced">Now </span>
    <span v-else-if="listing.price.is_ranged">From </span>
    <span v-html="listing.price.sale_value.inc"></span>
  </span>
  <span class="block text-xs font-normal">
    <span class="line-through" v-html="listing.price.regular_sale_value.inc"></span>
    <span class="text-red">
      Save
      <span v-html="listing.price.price_list_diff_value.inc"></span>
      <span v-if="listing.price.price_list_entry.is_decrease &&
listing.price.price_list_entry.is_percentage">&nbsp;<span v-
html="listing.price.price_list_entry.display"></span></span>
    </span>
  </span>
</span>
<span v-else-if="listing.price && listing.price.is_reduced">
  <span class="text-red">
    <span v-if="listing.price.is_ranged" class="text-xs">Now from </span>
    <span v-else class="text-xs">Now </span>
```

```

        <span v-html="listing.price.sale_value.inc"></span>
    </span>
    <span class="block text-xs font-normal">
        <span class="line-through" v-html="listing.price.value.inc"></span>
        <span class="text-red"> Save <span v-
html="listing.price.saving_value.inc"></span></span>
    </span>
</span>
<span v-else-if="listing.price">
    <span class="text-xs" v-if="listing.price.is_ranged">From </span>
    <span v-html="listing.price.sale_value.inc"></span>
    <span v-if="listing.price.not_retail" class="block text-xs font-normal">RRP <span v-
html="listing.price.retail_value.inc"></span></span>
</span>

```

Product Page Code Snippet

```

<div v-if="has_price">
    <span class="font-semibold" :class="{ 'text-red': is_reduced }">
        <span v-if="not_regular_sale_price && price_list_entry &&
price_list_entry.is_for_customer">Your price </span>
        <span v-else-if="is_reduced && is_ranged">Now from </span>
        <span v-else-if="is_reduced">Now </span>
        <span v-else-if="is_ranged">From </span>
        <span class="text-xl">{{ "{{ sale_price.inc }}" }}</span>
    </span>
    <span v-if="not_regular_sale_price">Normally {{ "{{ regular_sale_price.inc }}" }} - <span
class="text-red"> Save {{ "{{ price_list_diff_price.inc }}" }}</span></span>
    <span v-else-if="is_reduced" class="text-sm">
        <span class="line-through">{{ "{{ price.inc }}" }}</span>
        <span class="text-red"> Save {{ "{{ saving_price.inc }}" }}</span>
    </span>
    <span v-else-if="not_retail" class="text-sm">
        RRP <span>{{ "{{ retail_price.inc }}" }}</span>
    </span>
</div>

```


How do I use a custom Vue component in the storefront?

This repo (<https://github.com/aerocargo/example-vue>)

You need to make your component **.vue**file(s) and your main **.js**file that will register all of the components. An example javascript structure can be found here in the repo (

<https://github.com/aerocargo/example-vue/tree/master/resources/js>)

You then need to add the **assetLinks**function to your modules service provider so that your modules assets are publicly linkable. You then need to register the javascript file using the **Aero\Components\Facades\Component**facade **import**method. An example module service provider can be found here in the repo (<https://github.com/aerocargo/example-vue/blob/master/src/ServiceProvider.php>)

Once you've got your files setup you need to build the javascript files. You can do this using npm (**npm run dev** or **npm run production**). After this you will be able to use your custom Vue components as described below.

Usage

A custom component can be included in any code contained within the `{% component %}` Twig tag offered by the **aerocommerce/components**package:

```
{% component "product" with {count: 100} %}  
<div>  
    <example-component :count="count" />  
</div>  
{% endcomponent %}
```

To include in a Twig template file outside of the `{% component %}` tag, or on a page not utilizing the `{% component %}` tag:

```
{% vue %}  
<example-component count="100" />
```

```
{% endvue %}
```

To share data between multiple sibling components you can declare the data within the opening Twig tag:

```
{% vue with {count: 100} %}  
<div>  
  <example-component :count="count" />  
  <another-example-component v-model="count" />  
</div>  
{% endvue %}
```

As with Vue.js templates, you should ensure there is only one root element within the Twig tag.

By default, the `{% vue %}` Twig tag renders the Vue template on the client-side. To make use of server-side rendering, simply specify the `ssr` option at the end of the opening Twig tag:

```
{% vue ssr %}  
// ...  
{% endvue %}  
  
// or  
  
{% vue with {count: 100} ssr %}  
// ...  
{% endvue %}
```

How to Implement Subscriptions on the Product Page

To implement subscriptions on your product page you'll need to add some code to your themes **product.twig** file that allows customers to see the available subscription plans and select the one they want.

The code snippet below provides radio options that will allow customers to select a subscription plan (or select that they would like to make a one-time order if the product doesn't require a subscription).

```
<div v-if="has_subscription_plans">
  <div v-for="plan in subscription_plans">
    <input type="radio" name="subscription" :id="'subscription-' + plan.id"
    @click="selectSubscriptionPlan(plan.id)" :checked="has_selected_subscription_plan &&
    selected_subscription_plan.id === plan.id">
    <label :for="'subscription-' + plan.id">
      <span v-text="plan.name + ' ' + plan.price.inc"></span>
      <span v-if="plan.has_saving_price" v-text="'saving: ' +
plan.saving_price.inc"></span>
    </label>
  </div>
  <div v-if="!requires_subscription_plan">
    <input type="radio" name="subscription" id="one-time-order"
    @click="deselectSubscriptionPlan()" :checked="!has_selected_subscription_plan">
    <label for="one-time-order">
      One-time order
    </label>
  </div>
</div>
```

How do I implement subscriptions on the product page?

To implement subscriptions on your product page you'll need to add some code to your themes `product.twig` file that allows customers to see the available subscription plans and select the one they want.

The code snippet below provides radio options that will allow customers to select a subscription plan (or select that they would like to make a one-time order if the product doesn't require a subscription).

```
<div v-if="has_subscription_plans">
  <div v-for="plan in subscription_plans">
    <input type="radio" name="subscription" :id="'subscription-' + plan.id"
    @click="selectSubscriptionPlan(plan.id)" :checked="has_selected_subscription_plan &&
    selected_subscription_plan.id === plan.id">
      <label :for="'subscription-' + plan.id">
        <span v-text="plan.name + ' ' + plan.price.inc"></span>
        <span v-if="plan.has_saving_price" v-text="'saving: ' +
plan.saving_price.inc"></span>
      </label>
    </div>
    <div v-if="!requires_subscription_plan">
      <input type="radio" name="subscription" id="one-time-order"
      @click="deselectSubscriptionPlan()" :checked="!has_selected_subscription_plan">
        <label for="one-time-order">
          One-time order
        </label>
      </div>
    </div>
```

How do I give split listings names generated from their attributes?

This code example extends the `Aero\Search\Elastic\Documents\ListingDocument` document to generate names for split listings from the attributes that the variant is made up of. You can [learn more about extending the listing document here](#)

It's important to note that you need to reindex for changes to the listing document to take effect. You can reindex using the php artisan `aero:search:reindex` command. [You can learn more about reindexing here](#)

```
<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Search\Elastic\Documents\ListingDocument;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        ListingDocument::add('search-result', function ($document) {
            if (! $name = $document->getData()['search-result']['name'] ?? null) return [];
            if (! $variant = $document->getModel()->variants->first()) return [];
            if (! $attributeGroups = $document->getModel()->product-
>attribute_groups_to_split_by ?? []) return [];

            $attributes = $variant->attributes->whereIn('attribute_group_id',
$attributeGroups);

            if ($attributes->isEmpty()) return [];
```

```
        $attributeNames = $attributes->map->getTranslation('name', $document->getLanguage())->join(' ');
```

```
        return ['name' => "$attributeNames $name"];
```

```
    });
```

```
}
```

```
}
```

Listing items per page

Update AppServiceProvider

```
app>Providers>AppServiceProvider.php
```

Add the following namespaces:

```
use Aero\Store\Http\Responses\ListingsJson;  
use Aero\Store\Http\Responses\ListingsPage;  
use Aero\Store\Http\Responses\SearchJson;  
use Aero\Store\Http\Responses\SearchPage;  
use Illuminate\Support\ServiceProvider;
```

Create a setting for per_page

```
class AppServiceProvider extends ServiceProvider  
{  
    public function boot()  
    {  
        ListingsPage::extend($perPage = function ($page) {  
            $selected = (int) $page->request->input('per_page', setting('search.per_page'));  
  
            $perPage = collect([24, 48, 96])>map(function ($key) use ($page, $selected) {  
                return [  
                    'key' => $key,  
                    'name' => $key,  
                    'selected' => $key === $selected,  
                    'url' => rtrim($page->request->fullUrlWithQuery(['per_page' => $key,  
                        'page' => null]), '?'),  
                ];  
            });  
  
            $page->setData('per_page_options', $perPage);  
        });  
  
        ListingsJson::extend($perPage);  
    }  
}
```

```
SearchPage::extend($perPage);  
SearchJson::extend($perPage);  
}  
}
```

Listing page variables

Add the following to the variables at the top of your `listing.twig` file

```
{% set variables = {  
    per_page_options: per_page_options,  
} %}
```

Listing page snippet

Add the following inside the component tags of the `listing.twig` file

```
<ul>  
  <li v-for="per_page in per_page_options" :key="per_page.key">  
    <a :href="per_page.url"  
      @click.prevent="updateWithoutScroll(per_page.url)"  
      class="w-full flex items-center p-3 space-x-4">  
      <span :class="per_page.selected ? 'font-bold' : ''">{{ "{{ per_page.name }}"  
    }}</span>  
    </a>  
  </li>  
</ul>
```