

Storefront

- [General](#)
 - [How can I add content into the body of the storefront?](#)
 - [How can I add content into the head of the storefront?](#)
- [Pages](#)
 - [What are the standard storefront pages?](#)
- [Listings Page](#)
 - [How do I extend the listings page query?](#)
- [Products Page](#)
 - [How can I remove the manufacturer from the product page breadcrumb?](#)
- [Blocks](#)
 - [How do I display content blocks on the storefront?](#)
- [Breadcrumb](#)
 - [How do I display the breadcrumb?](#)
 - [How do I add something to the breadcrumb?](#)
 - [How do I clear the breadcrumb?](#)
 - [How can I filter the breadcrumb?](#)
 - [How can I display the request referrer?](#)
 - [How can I disable the breadcrumb functionality?](#)
- [Routing](#)
 - [How do I set a slug validator?](#)
- [Image Factory](#)

- [What is the image factory?](#)
- [How do I use the image factory?](#)
- [Currencies](#)
 - [How do I change the store currency?](#)
 - [What is a fascia currency and how do I use it?](#)
- [Events](#)
 - [What are the available events?](#)
 - [How do I run a custom event?](#)
- [Twig](#)
 - [What is Twig?](#)
 - [What are the available twig functions?](#)
 - [How do I add a custom twig function?](#)
 - [What are twig functions?](#)
- [Vue](#)
 - [How do I use a custom vue component in the storefront?](#)
- [Tutorials](#)
 - [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 add a header image to the listings page?](#)
 - [How can I add a tax toggle to my store?](#)
 - [How can I add a cart panel to my store?](#)

General

How can I add content into the body of the storefront?

The content for body pipeline can be used to inject content into the `<body>` element of the page, such as stylesheets, meta tags, scripts, or other body elements:

```
// Globally
\Aero\Store\Pipelines\ContentForBody::extend(function (string $content, \Closure $next) {
    return $next($content . view('acme::script'))
});

// Product Page Only
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    \Aero\Store\Pipelines\ContentForBody::extend(function (string $content, \Closure $next) {
        return $next($content . view('acme::script'))
    });
});
```

Parameters

Parameter	Type	Description
<code>\$content</code>	<code>string</code>	The content to be injected into the body.
<code>\$next</code>	<code>\Closure</code>	The next step in the query pipeline.

How can I add content into the head of the storefront?

The content for head pipeline can be used to inject content into the `<head>` element of the page, such as stylesheets, meta tags, scripts, or other head elements.

```
// Globally
\Aero\Store\Pipelines\ContentForHead::extend(function (string $content, \Closure $next) {
    return $next($content . view('acme::script'))
});

// Product Page Only
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    \Aero\Store\Pipelines\ContentForHead::extend(function (string $content, \Closure $next) {
        return $next($content . view('acme::script'))
    });
});
```

Parameters

Parameter	Type	Description
<code>\$content</code>	<code>string</code>	The content to be injected into the body.
<code>\$next</code>	<code>\Closure</code>	The next step in the query pipeline.

Pages

What are the standard storefront pages?

Aero includes the following standard e-commerce pages.

These are automatically created when you create a new Aero store:

- **Homepage:** Displays the main landing page of the store.
- **Listings:** Displays product listings based on searches or category selection.
- **Product:** Displays a product details and let's customers add it to their cart.
- **Cart:** Displays the cart, e.g. the products the customer has added to it.
- **Search:** Search results will point to the listings page
- **Information:** Displays static information, e.g. About Us, Returns Policy etc

Listings Page

How do I extend the listings page query?

You can extend the listings page query by extending the `\Aero\Store\Pipelines\ListingsPageQuery` pipeline.

For example, you can add functionality to have products that only certain customer groups can view and buy.

First, add a product setting to control this behaviour:

```
\Aero\Catalog\Models\Product::settings('customer_groups', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->eloquent('allowed_customer_groups', \Aero\Account\Models\CustomerGroup::class)
        ->hint('Leave blank to allow any customer group to view/buy this product')
        ->multiple()
        ->dontLoadModels();
});
```

Then add this data to the `\Aero\Search\Elastic\Documents\ListingDocument`:

```
\Aero\Search\Elastic\Documents\ListingDocument::add('search-data', function
(\Aero\Search\Elastic\Documents\ListingDocument $document) {
    $product = optional($document->getModel())->product ?? null;

    $allowedGroups = optional($product)->setting('customer_groups.allowed_customer_groups') ??
    [];

    return [
        'customer_groups' => $allowedGroups,
        'does_not_require_customer_group' => empty($allowedGroups),
    ];
});
```

You must run `php artisan aero:search:reindex --type=listing` for these document changes to be applied to existing listings.

Then extend the query to use factor this in when resolving what listings to show on the listings page:

```
\Aero\Store\Pipelines\ListingsPageQuery::extend(function
(\Aero\Store\Pipelines\ListingsPageQuery $builder) {
    $bool = new \Elastica\Query\BoolQuery();

    // Always show products that don't require a customer group
    $bool->addShould(
        new \Elastica\Query\Term([
            'search-data.does_not_require_customer_group' => true,
        ]),
    );

    // When a customer is in a group, also show products that require that group
    if ($group = optional($builder->builder->request->user())->customer_group_id) {
        $bool->addShould(
            new \Elastica\Query\Terms('search-data.customer_groups', [$group]),
        );
    }

    // At least one of the above conditions must be met
    $bool->setMinimumShouldMatch(1);

    $builder->query->getQuery()->addMust($bool);
});
```

Finally extend the product page to prevent people viewing products when they lack the required customer group:

```
\Aero\Store\Http\Responses\ProductPage::extend(function
(\Aero\Store\Http\Responses\ProductPage $page, \Closure $next) {
    $allowedGroups = $page->product->setting('customer_groups.allowed_customer_groups');

    if (empty($allowedGroups)) {
        return $next($page);
    }
}
```

```
$customer = $page->request->user();

$customerGroupId = optional($customer)->customer_group_id ?? null;

abort_unless(in_array($customerGroupId, $allowedGroups), 404);

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

Products Page

How can I remove the manufacturer from the product page breadcrumb?

By default, when there is no referrer (i.e. the page was hit directly - not through a filtering or searching), the product page breadcrumb contains a crumb entry for the manufacturer.

This can be disabled:

```
\Aero\Catalog\Models\Product::$breadcrumbContainsManufacturer = false;
```

Blocks

How do I display content blocks on the storefront?

The structure of the views for the block items is shown below. If you are creating your own theme or custom storefront, you must ensure these files are present in order for the content blocks to display correctly:

```
└─ resources
  └─ views
    └─ blocks
      └─ html.twig
      └─ image.twig
      └─ link.twig
      └─ menu.twig
      └─ text.twig
```

Block Items

A single block item

To show a single block item, use `block_item` twig function, passing the block's key:

```
{{ block_item('product-page.confidence-builder') }}
```

This will show the first available block item in the block.

To show a different block item, provide the position as the second parameter:

```
{% if block_exists('homepage.brand-heroes') %}
  <div class="flex w-full flex-wrap">
    <div class="w-full md:w-1/2">
      {{ block_item('homepage.brand-heroes', 0) }}
```

```

    </div>

    <div class="w-full md:w-1/2">
        {{ block_item('homepage.brand-heroes', 1) }}
    </div>

</div>

{% endif %}

```

A collection of block items

To show all block items in a content block, use the `block_items` twig function, passing the block's key:

```

{{ block_items('product-page.confidence-builders') }}

```

If you need to limit the number of items returned in the collection, pass a limit as the second parameter:

```

{{ block_items('product-page.confidence-builders', 3) }}

```

If the block items need to be wrapped in elements to style their position, the following approach can be used:

```

{% if block_exists('homepage.main-hero') %}
    {% set main_hero = block_items('homepage.main-hero', null, {
        sizes: {
            small: { width: 360, height: 360 },
            medium: { width: 660, height: 560 },
            large: { width: 1280, height: 600 }
        }
    }) %}
    <div class="flex w-full flex-wrap">
        {% for hero in main_hero %}
            <div class="w-full">{{ hero | raw }}</div>
        {% endfor %}
    </div>
{% endif %}

```

You must use the `raw` filter to ensure the block item HTML isn't escaped.

Passing data

In some cases (as seen in the previous example), a block item may need data to be passed from the theme.

This is especially useful for themes, which dictate the image dimensions:

```
{{ block_item('listings.loyalty-banner', 0, {  
  sizes: {  
    small: { width: 400, height: 200 },  
    medium: { width: 600, height: 260 },  
    large: { width: 1024, height: 300 },  
  },  
  options: ['contain', {quality: 90}],  
}) }}
```

Breadcrumb

How do I display the breadcrumb?

To access the breadcrumb on the storefront, use the `breadcrumb()` function inside Twig files.

It can be displayed by looping over each crumb like so:

```
<ul class="breabcrumb">
  {% for crumb in breadcrumb() %}
    <li class="crumb">
      {% if loop.last %}
        <span>{{ crumb.name }}</span>
      {% else %}
        <a href="{{ crumb.url }}">{{ crumb.name }}</a>
      {% endif %}
    </li>
  {% endfor %}
</ul>
```

How do I add something to the breadcrumb?

You can use the `add()` method to add a crumb:

```
\Aero\Store\Facades\Breadcrumb::add(function (\Aero\Store\Services\Breadcrumb $breadcrumb) {  
    $breadcrumb->add(new \Aero\Store\Crumb('Name', '/url'));  
});
```

You can provide a Closure that returns an `\Aero\Store\Crumb` instance instead if you need to defer until the breadcrumb is rendered.

How do I clear the breadcrumb?

You can use the `clear()` method to clear the crumbs:

```
\Aero\Store\Facades\Breadcrumb::clear();
```

How can I filter the breadcrumb?

You can provide a filter callback to remove unwanted crumbs, it should return `true` when a crumb should be kept and `false` otherwise:

```
\Aero\Store\Facades\Breadcrumb::setFilter(function ($crumb) {  
    if ($crumb->getName() === 'Foo') {  
        return false;  
    }  
  
    return true;  
});
```

How can I display the request referrer?

By default, the product page breadcrumb is built based on the referrer.

For example, if a customer performs a search for "shirt" and then clicks on a listing result, the breadcrumb on the product page will be `Home > Search for "shirt" > Casual Shirt`.

To disable this functionality, use `dontUseReferer()`:

```
\Aero\Store\Facades\Breadcrumb::dontUseReferer();
```

How can I disable the breadcrumb functionality?

There may be cases (such as an API) where the breadcrumb functionality should not use the request at all.

To do this, use `dontUseRequest()`:

```
\Aero\Store\Facades\Breadcrumb::dontUseRequest();
```

Routing

How do I set a slug validator?

You can control if a collection of slugs is valid or not by setting your own validator:

```
\Aero\Store\Routing\Slugs::setValidator(function ($slugs, $resolver) {  
    if ($resolver !== 'listings') return true;  
  
    if ($slugs->count() < 3 && $slugs->where('model_type', 'tag')->count()) return false;  
  
    return true;  
});
```

Image Factory

What is the image factory?

The Image Factory is a tool for generating optimised, resized versions of images on demand. It allows you to define the dimensions and apply transformations such as cropping, resizing, format conversion, and quality adjustments, while generating a signed URL for the resulting image.

Introduction

The image factory can be called using the `image_factory()` helper function from within PHP (.php and .blade.php) and Twig (.twig) files. When initializing an instance, the width and height of the resized image must be provided (in pixels).

The image factory outputs a signed URL to the resized image with an embedded validation token. This prevents being able to generate an infinite number of uniquely configured URLs that would drain the resources on the server.

PHP

```
image_factory(600, 800);  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800  
  
image_factory(600, 800)->path('/images/example.jpg');  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg  
  
image_factory(600, 800)->path('/images/example.jpg')->retina();  
// outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg
```

Twig

```
{{ image_factory(600, 800) }}  
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800 #}
```

```
{{ image_factory(600, 800).path('/images/example.jpg') }}
```

```
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg #}
```



```
{{ image_factory(600, 800).path('/images/example.jpg').retina() }}
```

```
{# outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg #}
```

Twig / Vue

If using the Aero component system (Vue.js template within a .twig file), the image URL can be built by merging the image factory prefix with the JavaScript variable:

```
<div v-if="image">
  <picture>
    <source :srcset="'{{ image_factory(600, 600) }}/' + image.file + '.webp 1x,{{
image_factory(600, 800).retina() }}/' + image.file + '.webp 2x'"
    type="image/webp">
    
  </picture>
</div>
```

Available Options

There are several options that can be used to manipulate the image. These can be chained.

path(\$path)

The `path` option is used to specify the path to the original image within the `storage/app/public` directory.

Twig:

```
{% set image = listing.images | first %}
```

```

```

PHP:

```
$image = image_factory(150, 300)->path($path);
```

Alternatively, the path can be provided as the third parameter in the initialisation of the instance:

Twig:

```
{% set image = listing.images | first %}  

```

PHP:

```
$image = image_factory(150, 300, $path);
```

webp()

The `webp` option is used to serve a `.webp` (a modern lossless compression) image.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->webp();
```

retina()

The `retina` option is used to serve an image that is twice the size (in pixels) of the designated dimensions.

For example, an image generated from `image_factory(500, 500)->retina()` would result in an image with dimensions 1000px by 1000px.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->retina();
```

dpr(\$scale)

The `dpr` option is used to scale the image dimensions.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->dpr(3.5);
```

flip()

The `flip` option transforms the image by flipping the x-axis.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->flip();
```

contain()

The `contain` option keeps the resized image within the dimensional constraints whilst retaining the original aspect ratio.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->contain();
```

crop()

The `crop` option scales the image to the dimensional constraints and removes any overflowing image from the canvas.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->crop();
```

rotate(\$angle)

The `rotate` option rotates the image by the given angle (in degrees counter-clockwise).

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->rotate(90);
```

sharpen(\$amount)

The `sharpen` option sharpens the image by the given amount (0 - 100).

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->sharpen(25);
```

pixelate(\$size)

The `pixelate` option applies a pixelation effect to the image with a given size of pixels.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->pixelate(5);
```

quality(\$percentage)

The `quality` option sets the quality of the outputted image. The default is 80%.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->quality(95);
```

invert()

The `invert` option reverses all colors in the image.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->invert();
```

greyscale()

The `greyscale` option turns the image into a greyscale version.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->greyscale();
```

placeholder(\$values)

The `placeholder` option returns a placeholder string.

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->placeholder('#ff0000');
```

By default, this is an encoded inline SVG path, which can be changed by setting a custom resolver:

```
\Aero\Store\Services\ImageFactory::setPlaceholderResolver(function ($image) {  
    return asset('placeholder.svg');  
});
```

If using a custom resolver, the values passed into the placeholder can then be accessed using the `$placeholderValues` property:

Twig:

```

```

PHP:

```
$image = image_factory(150, 300)->placeholder(['type' => 'product', 'category' => 'dress']);
```

```
\Aero\Store\Services\ImageFactory::setPlaceholderResolver(function ($image) {  
    return asset("placeholder-{$image->placeholderValues['type']}.svg#{$image->placeholderValues['category']}");  
});
```

options(\$options)

The `options` option allows multiple options to be specified by calling one method. The individual options can either be passed as separate arguments or as an array.

Twig:

```


// or ...


```

PHP:

```
$image = image_factory(150, 300)->options('crop', ['pixelate' => 100], ['quality' => 30]);

// or ...

$image = image_factory(150, 300)->options(['crop', ['pixelate' => 100], ['quality' => 30]]);
```

How do I use the image factory?

The image factory can be called using the `image_factory()` helper function from within PHP (.php and .blade.php) and Twig (.twig) files. When initialising an instance, the width and height of the resized image must be provided (in pixels).

The image factory outputs a signed URL to the resized image with an embedded validation token. This prevents being able to generate an infinite number of uniquely configured URLs that would drain the resources on the server.

PHP

```
image_factory(600, 800);  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800  
  
image_factory(600, 800)->path('/images/example.jpg');  
// outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg  
  
image_factory(600, 800)->path('/images/example.jpg')->retina();  
// outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg
```

Twig

```
{{ image_factory(600, 800) }}  
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800 #}  
  
{{ image_factory(600, 800).path('/images/example.jpg') }}  
{# outputs: https://store.com/image-factory/640fa397eb07791f~600x800/images/example.jpg #}  
  
{{ image_factory(600, 800).path('/images/example.jpg').retina() }}  
{# outputs: https://store.com/image-factory/632ab6d293827ff6~600x800@2x/images/example.jpg #}
```

Twig / Vue

If using the Aero component system (Vue.js template within a .twig file), the image URL can be built by merging the image factory prefix with the JavaScript variable:

```
<div v-if="image">
  <picture>
    <source :srcset="'{{ image_factory(600, 600) }}/' + image.file + '.webp 1x,{{
image_factory(600, 800).retina() }}/' + image.file + '.webp 2x'"
    type="image/webp">
    
  </picture>
</div>
```

Currencies

How do I change the store currency?

In order to change the store currency we need to publish the store configuration file:

```
php artisan vendor:publish --tag aero::store-config
```

Then locate the `configurations` section of the published `config/aero/store.php` file and update the `currency_code` value of the active store configuration.

Ensure the new `currency_code` exists in the `currencies` database table of the store.

What is a fascia currency and how do I use it?

A fascia currency can be set to render a different currency to the one specified in the store configuration.

Setting the fascia currency

The easiest way to check for and apply the fascia currency is through `store` middleware.

The example snippet below details how to check for the presence of a `currency` cookie passed with the request:

```
class SetFasciaCurrency
{
    public function handle(\Illuminate\Http\Request $request, \Closure $next): void
    {
        \Aero\Store\Http\Middleware\EncryptCookies::$exceptions[] = 'currency';

        if ($code = $request->cookie('currency')) {
            $currency = \Aero\Common\Models\Currency::find($code);

            $request->store()->setFasciaCurrency($currency);
        }

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

The middleware class can then be pushed to the `store` middleware group within the `register` method of a service provider:

```
public function register(): void
{
```

```
$this->app['router']->pushMiddlewareToGroup('store', SetFasciaCurrency::class);  
}
```

Remember to call `parent::register()` if inside a module service provider.

If the fascia currency needs to be set when running a console command:

```
if ($this->app->runningInConsole() && ($currency = env('CURRENCY_CODE'))) {  
    $this->app->booted(function () use ($currency) {  
  
        \Aero\Store\Facades\Store::setFasciaCurrency(\Aero\Common\Models\Currency::find($currency));  
    });  
}
```

Additional Considerations

- The exchange-rate of the fascia currency is based on the store's default configuration currency.
- If your store theme uses the Aero component system, the fascia currency will automatically be rendered for listings and products. However, if your theme utilises the `range-slider` component, you should ensure `currency('[value]')` and `currency().fascia_exchange_rate` are passed through as the `formatter` and `multiplier` attributes, respectively.
- Any event which is queued and is an instance of `Aero\Events\ManagedEvent` will automatically handle re-applying the fascia currency when running from the queue worker.

To force rendering the payment currency in mail notifications, use the `.base` modifier.

For example: `{{ order.total_price.base }}`.

Events

What are the available events?

Products

Event	Description
<code>product.loaded</code>	Fired when the product component has finished mounting and is marked as loaded.
<code>product.images-updating</code>	Fired when the product images are about to be updated because the image set changed.
<code>product.images-updated</code>	Fired after the product images have been updated. Provides the updated <code>images</code> .
<code>product.variant-selected</code>	Fired when the selected product variant changes. Provides the variant, quantity, price, and total sale price.
<code>product.attribute-selected</code>	Fired when one or more product attributes are selected or changed. Provides the selected attributes.
<code>product.quantity-updated</code>	Fired when the product quantity changes. Provides the new quantity.
<code>product.subscription-plan-deselected</code>	Fired when the currently selected subscription plan is removed/deselected.
<code>product.subscription-plan-selected</code>	Fired when a subscription plan is selected. Provides the plan ID and selected plan.
<code>product.add-to-cart</code>	Fired before the product is sent to the cart API. This event can modify the cart payload or return <code>false</code> / <code>null</code> to prevent adding the item.
<code>product.added-to-cart</code>	Fired after the cart API successfully adds the product. Provides the returned cart/add-to-cart data.
<code>product.failed-adding-to-cart</code>	Fired when adding the product to the cart fails. Provides the returned errors.

Listings

Event	Description
-------	-------------

<code>listings.loaded</code>	Fired after the listings component has mounted and its initial data has been loaded.
<code>listings.update</code>	Listener event that triggers a listings update/navigation to the supplied URL.
<code>listings.updating</code>	Fired after a listings request returns but before the response data is applied to the Vue component. Can modify the returned listings data.
<code>listings.updated</code>	Fired after the returned listings data has been applied to the component and Vue has completed the next tick.
<code>listings.navigated</code>	Fired when listings data is updated through a navigation that does not involve the component's normal history push/replace handling.

How do I run a custom event?

You can run your own custom event using the `runEvent()` method:

```
window.Aero.runEvent('acme', { foo: 'bar' });
```

And listen for it wherever needed:

```
window.AeroEvents.on('acme', (data) => {  
  console.log(data);  
});
```

Twig

Twig

What is Twig?

Storefront pages are built using the [Twig](#) template engine.

Twig is a fast, secure and flexible PHP templating solution. Before developing with Aero, we recommend that you become acquainted with the core concepts of Twig. The official Twig documentation is a great place to start:

[Click here for the official Twig documentation](#)

What are the available twig functions?

Twig function	Description	Typical Usage / Notes
<code>aero(key)</code>	Retrieves a value from the current store using the supplied key.	Useful for accessing store-level properties dynamically.
<code>locale()</code>	Returns the current store locale as a two-letter language code, e.g. <code>en</code> .	Useful when language-specific behaviour or markup is required.
<code>page_title()</code>	Returns the current page's SEO title.	Used for page titles and SEO-related markup.
<code>store_name()</code>	Returns the current store's name.	Useful for displaying the store name in templates.
<code>store_logo()</code>	Returns the URL of the current store's logo, or an empty string if no logo is configured.	Useful for displaying the store logo.
<code>email_logo()</code>	Returns the URL of the store's email logo, or an empty string if none is configured.	Intended for email templates.
<code>store_address()</code>	Returns the current store's address as an array.	Useful when rendering store address information.
<code>active(...)</code>	Determines whether a given item/route is currently active using the application's active service.	Commonly used to apply active states to navigation elements.
<code>blocks(key, limit = null, data = [])</code>	Returns the block items belonging to a specified block/group.	Can optionally limit the number of items or filter by block type.
<code>block_items(key, limit = null, data = [])</code>	Alias of <code>blocks()</code> . Returns block items for a specified block/group.	Provides an alternative, more descriptive name when working with block collections.
<code>block_item(key, index = 0, data = [])</code>	Returns a specific block item from a block/group by its index.	Useful when a template needs a particular block item rather than the whole collection.
<code>block_exists(key, count = null)</code>	Checks whether a block/group contains any items. Can also check whether it contains at least a specified number of items.	Useful for conditionally rendering block-based content.

<code>block_item_exists(key, index = 0)</code>	Checks whether a block contains an item at a specified index.	Useful for safely checking whether a particular block item exists before rendering it.
<code>breadcrumb(...)</code>	Returns the current breadcrumb/crumb collection.	Used to render breadcrumb navigation.
<code>base64_encode(content)</code>	Base64-encodes the supplied content.	Useful when content needs to be represented as Base64 within a template.
<code>image_factory(width, height, path = '')</code>	Creates an <code>ImageFactory</code> instance for generating/manipulating an image with the specified dimensions and optional path.	Used when template-level image generation or processing is required.
<code>theme_asset(path, secure = null)</code>	Generates the URL for an asset within the current theme.	Useful for referencing theme CSS, JS, images, fonts, etc.
<code>theme_mix(path)</code>	Generates the URL for a theme asset processed through Laravel Mix.	Used for versioned/compiled theme assets.
<code>theme_vite(entrypoints, directory = 'build')</code>	Generates Vite asset tags/URLs for entry points within the current theme.	Used to load Vite-built theme assets.
<code>vite(entrypoints, directory = 'build')</code>	Generates Vite asset tags/URLs for the supplied entry points.	Used to load application-level Vite assets.
<code>theme_css(path)</code>	Generates a stylesheet reference for a CSS file within the current theme.	Handles theme CSS and can render it inline where appropriate.
<code>view_exists(view)</code>	Checks whether a specified view exists.	Useful for conditionally rendering optional views.
<code>setting(...)</code>	Retrieves an application/store setting using its key.	Provides access to configured settings from Twig.
<code>cart(...)</code>	Retrieves the current cart from the request.	Used to access cart information in storefront templates.
<code>currency(amount = null)</code>	Returns the current currency, or formats a supplied amount using the current currency.	Useful for displaying correctly formatted prices.
<code>css(path, url)</code>	Generates CSS output for a stylesheet, either as an inline <code><style></code> block or a <code><link></code> element depending on the context.	Used to include CSS dynamically in templates.
<code>section(name, properties = [])</code>	Renders a named view from the <code>sections</code> view directory and passes it the supplied properties.	Useful for reusable page sections.
<code>snippet(name, properties = [])</code>	Renders a named view from the <code>snippets</code> view directory and passes it the supplied properties.	Useful for small, reusable pieces of template markup.
<code>form(name, properties = [])</code>	Renders a named form view and processes the resulting template through Twig.	Used to render reusable forms from Twig. Renders <code>forms.{name}</code> view.

<code>has_route(name)</code>	Checks whether a named Laravel route exists.	Useful when conditionally generating links or navigation based on route availability.
------------------------------	--	---

Twig

How do I add a custom twig function?

You can use the `add()` method to add a custom twig function:

```
\Aero\Store\Twig\Extensions\TwigFunctions::add([
    new \Twig\TwigFunction('foo', function () {
        return 'bar';
    }),
]);
```

And then use it in twig:

```
{{ foo() }}
```

What are twig functions?

Twig functions are reusable functions that can be called from within Twig templates to perform a specific task or return a value.

They are typically used to keep template logic simple and readable by moving more complex or reusable operations into functions. A Twig function can accept parameters, perform some processing, and return a result that can then be used in the template.

For example, you can use `{{ store_name() }}` in `twig` to output your store's name.

See [here](#) for a list of all available twig functions, and [here](#) for how to add your own.

Vue

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

1. You need to add the `assetLinks` function to your modules service provider so that your modules assets are publicly linkable. An example module service provider can be found here:

<https://github.com/aerocargo/admin-example-vue/blob/master/src/ServiceProvider.php>

2. You also 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:

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

3. You need to run `php artisan aero:link` in your terminal to link the assets.
4. You then need to register the javascript file using the `Aero\Components\Facades\Component` facade `import` method:

```
\Aero\Components\Facades\Component::import('/modules/aerocargo/acme/components.js');
```

This facade is provided by the `aerocommerce/components` module.

5. Once you've got your files setup, have ran the link command and have built the javascript files (using npm: `npm run dev` or `npm run production`).

Usage

{% component %}

The `component` tag will render the component in line with the `aero.ssr.enabled` config value, if `true` then the component will be server-side rendered:

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

Using `csr` means that the component will always be client-side rendered.
Using `ssr` means the component will always be server-side rendered.

{% vue %}

The `vue` tag will render the component client-side by default:

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

Using `ssr` means the component will be server-side rendered.

Tutorials

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

The recommended way to add featured products to the homepage of your store is to use the [aerocargo/listing-collections](#) package.

Install

Run this command in the root of your project:

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

Create a section

1. Create the file:

```
themes/acme/resources/sections/best-sellers.twig
```

Replace `acme` with the name of your theme.

2. Add the twig:

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

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

```
<h3 class="aero-listing-collection__title">Best Sellers</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>
{% endcomponent %}
```

Use the section

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 "best-sellers" %}
```

Now products tagged with `best-sellers` will be rendered wherever our section is used.

See more info in the [aerocargo/listing-collections docs](#).

How do I add a contact form to my storefront?

1. Create the file:

```
themes/acme/resources/forms/contact.twig
```

Replace `acme` with the name of your theme.

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

2. Add the 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>
```

3. Display a session success message:

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

Mail Notification

You can create a mail notification in the admin for the `Form Submitted` event (leave `value` empty to target all forms, or put `contact` to target this one specifically)

Then add this to the body of the mail notification, to output the submission:

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

How do I add a header image to the 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, passing widths & heights for each size:

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

Optionally, we can add some styling to our header to create a nice component for our theme:

```
<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 can I add a tax toggle to my store?

First, install `aerocargo/tax-toggler`:

```
composer require aerocargo/tax-toggler
```

This module adds a `tax_state()` twig function that returns `inc` or `ex` depending on configuration, see [here](#) for more information.

Next let's create the `layouts/partials/tax-toggle.twig` file:

```
<label class="inline-flex items-center cursor-pointer">
  <span class="hidden md:flex text-xs font-medium uppercase text-gray-900 mr-2">
    <span class="price--inc">inc VAT</span>
    <span class="price--ex">ex VAT</span>
  </span>
  <input type="checkbox" id="tax-toggler" value class="sr-only peer" {{ tax_state() == "inc"
? "checked" : "" }}>
  <span class="relative w-9 h-5 bg-gray-200 peer-focus:outline-none rounded-full peer peer-
checked:after:translate-x-full peer-checked:after:border-white after:content-['']
after:absolute after:top-[2px] after:start-[2px] after:bg-white after:border-gray-300
after:border after:rounded-full after:h-4 after:w-4 after:transition-all"></span>
</label>

{% js %}
  <script>
    (() => {
      let state = "{{ tax_state() }}"

      function togglePrices(state) {
        let date = new Date();
        date.setTime(date.getTime() + 2592000000);
```

```

        document.body.classList.remove(state === 'inc' ? 'prices--ex' : 'prices--inc');

        document.body.classList.add(state === 'inc' ? 'prices--inc' : 'prices--ex');
        document.cookie = `prices=${state}; path=/; expires=${date.toUTCString()}`;
    }

    togglePrices(state);

    let toggle = document.getElementById('tax-toggler');

    toggle.checked = state === 'inc';

    toggle.addEventListener('change', () => {
        togglePrices(toggle.checked ? 'inc' : 'ex');
    });
    })();
</script>
{% endjs %}

{% css %}
<style>
    body.prices--inc .price--inc, body.prices--ex .price--ex { display: initial; }
    body.prices--inc .price--ex, body.prices--ex .price--inc { display: none; }
</style>
{% endcss %}

```

`2592000000` is 30 days in milliseconds.

Then let's include this file in our `header.twig`:

```
{% include "layouts/partial/tax-toggle.twig" %}
```

Now we just have to mark any prices we display to use this toggle, for example in `product.twig`:

```

{# Before #}
<span class="text-xl">{{ "{{ sale_price.inc }}" }}</span>

{# After #}
<span class="text-xl price--inc">{{ "{{ sale_price.inc }}" }}</span>
<span class="text-xl price--ex">{{ "{{ sale_price.ex }}" }}</span>

```


How can I add a cart panel to my store?

This tutorial will provide the code needed to setup a cart panel slide-out system for your store.

First, lets add the `layouts/partials/cart-panel.twig` code:

```
<div id="cart-panel" class="cart-panel isolate fixed z-[999] bottom-0 right-0 max-h-full overflow-y-auto overscroll-contain h-[90%] md:h-dvh w-full md:max-w-2xl bg-white [body.cart-panel--open_&]:shadow-[0_0_50px_0_rgba(0,0,0,0.3)]">
  <button type="button" class="absolute z-10 right-0 top-0 flex items-center justify-center p-3 m-4" onclick="document.getElementById('cart-panel-cover').click()" aria-label="Close cart">
    <svg viewBox="0 0 24 24" fill="currentColor" class="size-6">
      <path fill-rule="evenodd" d="M5.47 5.47a.75.75 0 0 1 1.06 0L12 10.94l5.47-5.47a.75.75 0 1 1 1.06 1.06L13.06 12l5.47 5.47a.75.75 0 1 1-1.06 1.06L12 13.06l-5.47 5.47a.75.75 0 0 1-1.06-1.06L10.94 12 5.47 6.53a.75.75 0 0 1 0-1.06Z" clip-rule="evenodd" />
    </svg>
  </button>
  <div class="relative px-4 pt-6 lg:pt-12 pb-6 lg:pb-12 mx-auto min-h-full w-full max-w-xl items-center text-center">
    <h2 class="copy copy-heading text-base lg:text-lg font-medium">My Cart</h2>

    <div class="pt-2.5 md:pt-3.5 lg:pt-6 pb-2.5 md:pb-3.5 lg:pb-6">
      {% if cart().items is empty %}
        <div class="space-y-4">
          <p class="copy copy-minor-heading leading-6 text-sm">Your Shopping Cart is Empty</p>

          <div class="copy copy-text text-sm leading-6 [&a]:underline">You haven't put any items in your shopping cart.</div>
        </div>
      {% else %}
        {% for item in cart().items %}
          {% if item.isVisible() %}
```

```

        {% snippet "cart-item" %}
    {% endif %}
{% endfor %}
{% endif %}
</div>

<div class="pb-2.5 md:pb-3.5 lg:pb-6 w-full text-sm text-left">
    <form action="{{ route('cart.discount') }}" method="post">
        {{ csrf_field() }}
        <div class="flex flex-col space-y-2">
            <label for="cart-panel-discount-input" class="copy copy-text text-sm">Discount Code?</label>
            <div class="flex space-x-2">
                <input id="cart-panel-discount-input"
                    type="text"
                    class="border border-gray-300 w-full px-2.5 py-2"
                    placeholder="Enter Code"
                    name="code"
                    autocapitalize="off"
                    autocomplete="off"
                    spellcheck="false" />
                <button type="submit" class="px-5 py-3 text-xs font-medium transition border border-[color:#000000] text-[color:#000000] hover:text-[color:#030712]">Apply</button>
            </div>
        </div>
    </form>
</div>

<div class="w-full [&*]:py-1 text-sm">
    <div class="flex">
        <div class="font-medium">Subtotal</div>
        <div class="flex-1 text-right">{{ cart().subtotal }}</div>
    </div>
    {% for discount in cart().discounts %}
        <div class="flex">
            <div class="font-medium">
                {% if discount.code is not empty %}
                    Discount ({{ discount.code }})
                <form action="{{ route('cart.discount') }}" method="post">
                    {{ method_field("delete") }}

```

```

        {{ csrf_field() }}
        <button type="submit"
            name="key" value="{{ discount.key }}"
            class="mt-3 inline-block text-xs font-semibold text-
gray-600 transition hover:text-gray-900 lg:mt-0">
            × remove
        </button>
    </form>
    {% else %}
        {{ discount.name }}
    {% endif %}
</div>
<div class="flex-1 text-right">- {{ discount.price.inc }}</div>
</div>
{% endfor %}
{% if cart().shippable %}
    <div class="flex">
        <div class="font-medium">Shipping</div>
        <div class="flex-1 text-right">
            {% if shipping %}
                {{ shipping.inc }}
            {% else %}
                <span class="text-xs text-gray-600">Calculated at next step</span>
            {% endif %}
        </div>
    </div>
{% endif %}
{% for surcharge in cart().surcharges %}
    <div class="flex">
        <div class="font-medium">{{ surcharge.name }}</div>
        <div class="flex-1 text-right">{{ surcharge.total_price }}</div>
    </div>
{% endfor %}
{% for voucher in cart().vouchers %}
    <div class="flex">
        <div class="font-medium">
            {% if voucher.code is not empty %}
                Voucher ({{ voucher.code }})
            {% else %}
                Voucher
            {% endif %}
        </div>
    </div>
{% endfor %}

```

```

        {% endif %}
        <form action="{{ route('cart.discount') }}" method="post">
            {{ method_field("delete") }}
            {{ csrf_field() }}
            <button type="submit"
                name="key" value="{{ voucher.key }}"
                class="mt-3 inline-block text-xs font-semibold text-gray-
600 transition hover:text-gray-900 lg:mt-0">
                × remove
            </button>
        </form>
    </div>
    <div class="flex-1 text-right">- {{ voucher.price.inc }}</div>
</div>
{% endfor %}
<div class="flex text-base">
    <div class="flex flex-wrap items-center text-left">
        <span class="block w-full font-medium">Total</span>
        {% if cart().total.hasTax() %}
            <span class="block text-xs text-gray-600">Including taxes of {{
cart().total.tax }}</span>
        {% endif %}
    </div>
    <div class="flex-1 text-right text-lg font-medium">{{ cart().total.inc
}}</div>
</div>
</div>

<div class="sticky bottom-0 pt-2.5 md:pt-3.5 lg:pt-6 pb-1 lg:pb-2.5">
    <a href="{{ route('checkout') }}" class="px-5 py-3 font-medium transition bg-
[color:#000000] text-[color:#FFFFFF] hover:bg-[color:#030712] cta-button inline-block mx-auto
text-sm text-center w-full">Checkout Securely</a>
</div>
</div>
</div>

{% js %}
    <div id="cart-panel-cover" class="cart-cover cart-cover--panel fixed inset-0 z-[998]"
style="background-color: rgba(0, 0, 0, 0.3)"></div>
<script>

```

```

!(() => {
  let timeout, animate = true;

  document.querySelector('body').classList.add('reduced-animation');

  const open = () => {
    clearTimeout(timeout);

    if (animate) {
      document.querySelector('body').classList.remove('reduced-animation');
    }

    document.querySelector('body').classList.add('cart-panel--open');

    timeout = setTimeout(() =>
document.querySelector('body').classList.add('reduced-animation'), 500);
  }

  const close = () => {
    clearTimeout(timeout);

    if (animate) {
      document.querySelector('body').classList.remove('reduced-animation');
    }

    document.querySelector('body').classList.remove('cart-panel--open');

    timeout = setTimeout(() =>
document.querySelector('body').classList.add('reduced-animation'), 500);
  }

  AeroEvents.on('product.added-to-cart', function (data) {
    open();

    const parser = new DOMParser();
    const doc = parser.parseFromString(data.cart_panel, 'text/html');

    document.getElementById('cart-panel').innerHTML = doc.querySelector('#cart-
panel').innerHTML;
  });

```

```

        window.addEventListener('DOMContentLoaded', () => {
            document.querySelector('#cart-panel-cover').addEventListener('click', close);

            document.querySelectorAll('.header-link--cart').forEach((el) =>
el.addEventListener('click', (e) => {
                e.preventDefault();

                open();
            }));
        });
    }));
</script>
{% endjs %}

{% css %}
<style>
    .cart-panel { transform: translateY(100%); @media (min-width: 768px) { transform:
translateX(100%); } }
    .cart-cover { transform: translateX(100%); opacity: 0; }
    .cart-panel--open .cart-panel { transform: translateY(0); @media (min-width: 768px) {
transform: translateX(0); } }
    .cart-panel--open .cart-cover--panel { transform: translateX(0); opacity: 1; }
    body:not(.reduced-animation) .cart-panel { transition: transform 300ms; }
    body:not(.reduced-animation) .cart-cover { transition: opacity 300ms; }
</style>
{% endcss %}

```

This file uses the `snippets/cart-item.twig` snippet:

```

<div class="flex flex-wrap py-1.5 first:pt-0 last:pb-0 text-left">
    <div class="flex items-start py-2 pr-2 text-xs lg:py-4 max-w-[80px]">
        {% if item.url %}<a href="{{ item.url }}" aria-label="{{ item.name }}">{% endif %}
            {% set image_width = 320 %}
            {% set image_height = 320 %}
            {% if item.image %}
                
    {% else %}
        
    {% endif %}
    {% if item.url %}</a>{% endif %}
</div>
<div class="relative flex flex-1 flex-wrap">
    <div class="w-full py-2 lg:w-4/6 lg:flex-1 lg:p-4">
        <div class="flex">
            {% if item.url %}<a href="{{ item.url }}" class="inline-block text-sm no-
underline lg:hover:underline">{% endif %}
                {{ item.name }}
            {% if item.url %}</a>{% endif %}
        </div>
        {% if item.options|length %}
            <div class="mt-2 text-xs text-gray-600">
                {% for option in item.options %}
                    <div class="mt-px">
                        {% set value = option.display_value|default(option.value) %}
                        {% if value is empty %}
                            <span class="font-medium">{{ option.name }}</span>
                        {% else %}
                            <span class="font-medium">{{ option.name }}:</span> {{ value
}}

                        {% endif %}
                        {% if option.hasPrice() %}
                            ({{ option.price.inc }})
                        {% endif %}
                        {% if option.hasFiles() %}
                            [{{ option.fileCount() }} attached files]
                        {% endif %}
                    </div>
                {% endfor %}
            </div>
        {% endif %}
    </div>
{% endif %}

```

```

{% if item.hasSubItems() %}
    <div class="text-grey-600 mt-2 text-sm">
        {% for sub_item in item.subItems %}
            {% if sub_item.isHidden() %}
                <div class="{% if errors.has(sub_item.id) or sub_item.hasError()
%}text-red-800{% endif %}">
                    {{ sub_item.quantity }} x {{ sub_item.name }}
                    {% if not sub_item.isFree() %}
                        &ndash; {{ sub_item.subtotal.inc }}
                    {% endif %}
                    {% if errors.has(sub_item.id) or sub_item.hasError() %}
                        <span class="mt-3 block w-full text-sm font-medium text-
red-800" role="alert">{{ errors.first(sub_item.id)|default(sub_item.error) }}</span>
                    {% endif %}
                    {% if sub_item.canBeRemoved() %}
                        <form class="ml-1 inline-block" action="{{
route("cart.update") }}" method="post">
                            {{ method_field("put") }}
                            {{ csrf_field() }}
                            <input type="hidden" name="cart_item_id" value="{{
sub_item.id }}">
                                (<button type="submit" name="quantity" value="0"
class="inline-block text-xs font-semibold text-gray-600 transition hover:text-gray-900">x
remove</button>)
                                </form>
                            {% endif %}
                        </div>
                    {% endif %}
                {% endfor %}
            </div>
        {% endif %}
    {% for file in item.files %}
        {% if file.hasExtension(["jpg", "jpeg", "png"]) %}
            
                {% endif %}
        {% endfor %}
    
```

```

        {% if errors.has(item.id) or item.hasError() %}
            <span class="mt-3 block w-full text-sm font-medium text-red-800"
role="alert">{{ errors.first(item.id)|default(item.error) }}</span>
        {% endif %}
    </div>

    <div class="flex w-full flex-wrap items-center justify-end py-2 lg:w-2/6 lg:flex-col
lg:p-4 lg:pr-0">
        <div class="flex-1 order-1 w-1/2 text-right lg:order-none lg:w-full">
            {% if not item.isFree() %}
                <div class="block text-sm {% if item.isReduced() %}text-red-700{% endif
%}">

                    <span class="price">{{ item.subtotal.inc }}</span>
                </div>
                {% if item.isReduced() %}
                    <div class="block text-xs line-through">
                        <span class="price">{{ item.fullSubtotal.inc }}</span>
                    </div>
                {% endif %}
                {% if item.quantity > 1 %}
                    {% set unit_price %}
                        <span class="price">{{ item.price.inc }}</span>
                    {% endset %}
                    <div class="block text-xs text-gray-600">{{ unit_price }} each</div>
                {% endif %}
            {% endif %}
        </div>
        <div class="w-1/2 lg:order-1 lg:mt-2 lg:w-full lg:text-right">
            {% set quantity %}
                <form action="{{ route('cart.update') }}" method="post" class="inline-flex
items-center">

                    {{ method_field("put") }}
                    {{ csrf_field() }}
                    <input type="hidden" name="cart_item_id" value="{{ item.id }}">
                    {% if item.canBeChanged() %}
                        <select name="quantity"
                            class="ml-2 font-medium transition border border-
[color:#000000] text-[color:#000000] hover:text-[color:#030712] py-1.5 pl-1.5 text-xs"
                            onchange="this.form.submit()">
                            {% if item.canBeRemoved() %}
                                <option value="0">0</option>

```

```

        {% endif %}
        {% for qty in range(item.quantity > 2 ? item.quantity - 1 : 1,
item.quantity + 5) %}

            <option value="{{ qty }}" {% if qty == item.quantity
%}selected{% endif %}>{{ qty }}</option>

            {% endfor %}
        </select>
    {% else %}
        <span class="ml-1">{{ item.quantity }}</span>
    {% endif %}
    {% if item.canBeRemoved() %}
        <button type="submit" class="button button--icon ml-2 -mr-1.5
hidden p-1 sm:flex"

            name="quantity" value="0" aria-label="Remove item">
            <svg class="h-5 w-5" width="24" height="24" fill="none"
viewBox="0 0 24 24">

                <path stroke="currentColor" stroke-linecap="round" stroke-
linejoin="round" stroke-width="1.5" d="M6.75 7.75L7.59115 17.4233C7.68102 18.4568 8.54622
19.25 9.58363 19.25H14.4164C15.4538 19.25 16.319 18.4568 16.4088 17.4233L17.25 7.75" />

                <path stroke="currentColor" stroke-linecap="round" stroke-
linejoin="round" stroke-width="1.5" d="M9.75 7.5V6.75C9.75 5.64543 10.6454 4.75 11.75
4.75H12.25C13.3546 4.75 14.25 5.64543 14.25 6.75V7.5" />

                <path stroke="currentColor" stroke-linecap="round" stroke-
linejoin="round" stroke-width="1.5" d="M5 7.75H19" />

            </svg>
        </button>
    {% endif %}
</form>
{% endset %}
<div class="flex items-center whitespace-nowrap text-xs lg:justify-end">
    <span class="font-medium text-gray-600">Quantity:</span>&nbsp;{{ quantity
}}

    </div>
</div>
</div>
</div>
</div>

```

Then we must extend the `\Aero\Store\Http\Responses\CartItemAdd` response builder in order to update the cart panel:

```
\Aero\Store\Http\Responses\CartItemAdd::extend(function
(\Aero\Store\Http\Responses\CartItemAdd $action, \Closure $next) {
    $action->setData('cart_panel', (string) view('layouts.partials.cart_panel'));
});
```

The `cart_panel` is retrieved by our `product.added-to-cart` listener and used to effectively re-render the cart panel to use the new cart data.

Now we can include the cart panel in `header.twig`:

```
{% include "layouts/partials/cart-panel.twig" %}
```

Finally, if you want the cart panel to open when the cart button (typically an anchor tag with `href={{ route('cart') }}`) is clicked (rather than redirecting to the cart page), then add the `header-link--cart` class to it.