

Twig

- [What is Twig?](#)
- [What are the available twig functions?](#)
- [How do I add a custom twig function?](#)
- [What are twig functions?](#)

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.

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.