Image Factory

Image Factory

How do I use 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 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 #}
```

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:

Twig/Vue

```
<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">
        <img :src="'{{ image_factory(600, 600) }}/' + image.file"
             :srcset="'{{ image_factory(600, 600).retina() }}/' + image.file + ' 2x'"
             :alt="image.alt"
             :key="image.file">
    </picture>
</div>
```

What are the available options for Image factory?

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

- path - webp - retina - dpr - flip - contain - crop - upscale - rotate - sharpen - pixelate - quality - invert - greyscale - placeholder - options

```
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 %}
<img src="{{ image_factory(150, 300).path(image.file) }}">
```

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 %}
<img src="{{ image_factory(150, 300, image.file) }}">
```

PHP

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

```
webp()
```

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

Twig

```
<img src="{{ image_factory(150, 300).webp() }}">
```

PHP

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

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

```
<img src="{{ image_factory(150, 300).retina() }}">
```

PHP

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

```
dpr($scale)
```

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

Twig

```
<img src="{{ image_factory(150, 300).dpr(3.5) }}">
```

PHP

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

```
flip()
```

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

Twig

```
<img src="{{ image_factory(150, 300).dpr(3.5) }}">
```

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

```
<img src="{{ image_factory(150, 300).contain() }}">
```

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

```
<img src="{{ image_factory(150, 300).crop() }}">
```

PHP

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

```
upscale()
```

The `upscale` option allows an image to be enlarged above it's real dimensions. By default, the image factory will not resize an image larger than it originally is, however, if you wish to force an aspect ratio, this option can be used.

Twig

```
<img src="{{ image_factory(150, 300).upscale() }}">
```

PHP

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

```
rotate($angle)
```

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

Twig

```
<img src="{{ image_factory(150, 300).rotate(90) }}">
```

PHP

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

```
sharpen($amount)
```

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

Twig

```
<img src="{{ image_factory(150, 300).sharpen(25) }}">
```

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

```
<img src="{{ image_factory(150, 300).pixelate(5) }}">
```

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

```
<img src="{{ image_factory(150, 300).quality(95) }}">
```

PHP

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

```
invert()
```

The invert option reverses all colors in the image.

Twig

```
<img src="{{ image_factory(150, 300).invert() }}">
```

PHP

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

```
greyscale()
```

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

Twig

```
<img src="{{ image_factory(150, 300).greyscale() }}">
```

PHP

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

```
placeholder($values)
```

The `placeholder` option returns a placeholder string.

Twig

```
<img src="{{ image_factory(150, 300).placeholder('#ff000') }}">
```

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:

PHP

```
\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

```
<img src="{{ image_factory(150, 300).placeholder({type: 'product', category: 'dress'}) }}">
```

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 specificed by calling one method. The individual options can either be passed as separate arguments or as an array.

Twig

```
<img src="{{ image_factory(150, 300).options('crop', {pixelate: 100}, {quality: 30}) }}">

// or ...

<img src="{{ image_factory(150, 300).options(['crop', {pixelate: 100}, {quality: 30}]) }}">
```

PHP

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

// or ...

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