Laravel Image Documentation

Release 0.9.0

Folklore,David Mongeau-Petitpas

Jan 28, 2019

Documentation:

CHAPTER 1

Installation

1.1 Dependencies

- Laravel 5.x
- Imagine 0.6.x|0.7.x

1.2 Server Requirements

- gd or Imagick or Gmagick
- exif - Required to get image format.

1.3 Steps

1- Require the package via Composer in your `composer.json`.

```
$ composer require folklore/image
```

2- Add the service provider to your `app/config/app.php` file

```
Folklore\Image\ImageServiceProvider::class,
```

3- Add the facade to your `app/config/app.php` file

```
'Image' => Folklore\Image\Facades\Image::class,
```

4- Publish the configuration file and public files

```
$ php artisan vendor:publish --provider="Folklore\Image\ImageServiceProvider"
```

5- Review the configuration file at `config/image.php`

CHAPTER 2

Getting started

This package aims to simplify images manipulation by using the url of an image to determine what filters should be applied. This way, you don't need to implements the creation of various image formats, you simply call the url with the parameters and it automatically generates a new image for you. It also supports static caching so the next request will serve a static image instead of being handled by Laravel.

This works with two components, the Url Generator and the Image Router. First, you generate an url containing the filters you want with the Url Generator. The url is generated according to the format declared in `config/image.php` (look for `url`). When you request that url, a route is declared with a pattern corresponding to the url format and catch the request, applying the filters and responding the new image.

The package provides many built-in filters such as: Thumbnail, Rotation, Colorize, Grayscale, Blur, Negative, etc... and can easily be extended with custom filters.

2.1 Basic Usage

2.1.1 Displaying a thumbnail

```
// Using the helper
<img src="{{ image_url('path/to/your/image.jpg', 100, 100, ['crop' => true]) }}" />

// Or using the facade
<img src="{{ Image::url('path/to/your/image.jpg', 100, 100, ['crop' => true]) }}" />
```

This will translate to: (assuming you haven't changed the default url format in `config/image.php`)

```
<img src="/path/to/your/image-filters(100x100-crop).jpg" />
```

If you call this url, the router will catch the request and respond with a cropped 100x100 version of your image.

2.1.2 Creating a new thumbnail

```
// Using the facade
$thumbnail = Image::make('path/to/your/image.jpg', [
    'width' => 100,
    'height' => 100,
    'crop' => true
]);

// Using the helper
$thumbnail = image()->make('path/to/your/image.jpg', [
    'width' => 100,
    'height' => 100,
    'crop' => true
]);

// or with the shortcut
$thumbnail = image('path/to/your/image.jpg', [
    'width' => 100,
    'height' => 100,
    'crop' => true
]);

// Save the image on the default source (look for `'source'` in `config/image.php`)
image()->save($thumbnail, 'path/to/your/new-image.jpg');

// Or save it on the cloud source
image()->source('cloud')
    ->save($thumbnail, 'path/to/your/new-image.jpg');
```

2.1.3 Defininig a custom filter

While writing all the filters you want in the `url()` method works, you will probably want to declare a custom filter that "group" some values together. This way you can reuse it, and instead of remembering all the values, you can simply apply your filter, by it's name.

To do this, the simplest way is adding a custom filter as an array. In your `AppServiceProvider`, add the following lines in the `boot()` method.

```
public function boot()
{
    parent::boot();

    // ...

    $this->app['image']->filter('thumbnail', [
        'width' => 100,
        'height' => 100,
        'crop' => true,
    ]);
}
```

Now you can simply use the filter by it's name

```
// Using the helper
<img src="{{ image_url('path/to/your/image.jpg', 'thumbnail') }}" />
```

(continues on next page)

(continued from previous page)

```
// Or using the facade
<img src="{{ Image::url('path/to/your/image.jpg', 'thumbnail') }}" />
```

Or combine with others

```
<img src="{{ image_url('path/to/your/image.jpg', ['thumbnail', 'greyscale']) }}" />
```

2.2 Advanced Usage

CHAPTER 3

Sources

In Laravel Image, the place where your images files are stored is called a source. You can have multiple source and each source can implements a different driver. Currently there is two drivers supported: `local` and `filesystem`. The last one is based on Laravel Filesystems and it supports all the same drivers (Amazon S3, Ftp, . . .). Just specify a disk that is defined in `config/filesystems.php` and you are good to go.

3.1 Configuration

Here is the default sources configuration from `config/image.php`:

```
/*
|--------------------------------------------------------------------------
| Default Source
|--------------------------------------------------------------------------
|
| This option define the default source to be used by the Image facade. The
| source determine where the image files are read and saved.
|
*/
'source' => 'public',

/*
|--------------------------------------------------------------------------
| Sources
|--------------------------------------------------------------------------
|
| The list of sources where you store images.
|
| Supported driver: "local", "filesystem"
|
*/
'sources' => [
```

(continues on next page)

(continued from previous page)

```
    'public' => [
        // The local driver use a local path on the machine.
        'driver' => 'local',

        // The path where the images are stored.
        'path' => public_path()
    ],

    'cloud' => [
        // The filesystem driver lets you use the filesystem from laravel.
        'driver' => 'filesystem',

        // The filesystem disk where the images are stored.
        'disk' => 'public',

        // The path on the disk where the images are stored. If set to null,
        // it will start from the root.
        'path' => null,

        // Cache the file on local machine. It can be useful for remote files.
        'cache' => true,

        // The path where you want to put cached files
        'cache_path' => storage_path('image/cache')
    ]

],
```

3.2 Usage

When you interact with the `Image` facade or the `image()` helper, by default the images are taken from the default source defined in the config. If you want to use another source, you can do:

```
$image = image()->source('cloud')->open('path/to/an/image.jpg');

// or
$image = image()->source('cloud')->make('path/to/an/image.jpg', [
    'width' => 100,
    'height' => 100
]);
```

Be carefull the `$image` object returned by the `make()` and `open()` methods implements a `save()` method. This method will save on your local disk only. To save an image on a specific source, use:

```
image()->source('cloud')->save($image, 'path/on/the/source/image.jpg');
```

CHAPTER 4

Filters

4.1 Configuration

4.2 Usage

CHAPTER 5

Javascript Helper

We provide a javascript helper to generate images url in the frontend. The file is published in `public/vendor/folklore/image/image.js`. You can simply add the javascript tag in your layout:

```
<script type="text/javascript" src="{{ asset('vendor/folklore/image/image.js') }}"></
↪script>
```

The helper is now available as a `LaravelImage` global variable. You can use it like this:

```
const url = LaravelImage.url('path/to/image', 300, 300, {
    rotate: 90,
});

// or

const url = LaravelImage.url('path/to/image', {
    width: 300,
    height: 300,
    rotate: 90,
});
```

5.1 Npm package

If you prefer, you can use the npm package:

```
$ npm install laravel-image --save
```

```
import LaravelImage from 'laravel-image';

const url = LaravelImage.url('path/to/image', 300, 300, {
    rotate: 90,
});
```

(continues on next page)

(continued from previous page)

```
// or

import { UrlGenerator } from 'laravel-image';

const urlGenerator = new UrlGenerator({
    // custom pattern options
});
const url = urlGenerator.make('path/to/image', 300, 300, {
    rotate: 90,
});
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search