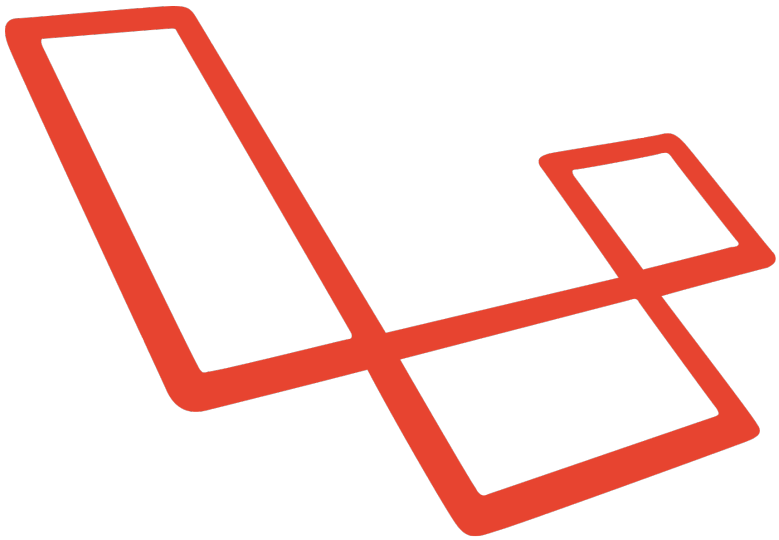


Laravel

5.4

Release Notes

Support Policy

For LTS releases, such as Laravel 5.1, bug fixes are provided for 2 years and security fixes are provided for 3 years. These releases provide the longest window of support and maintenance. For general releases, bug fixes are provided for 6 months and security fixes are provided for 1 year.

Laravel 5.4

Laravel 5.4 continues the improvements made in Laravel 5.3 by adding support for Markdown based emails and notifications, the Laravel Dusk browser automation and testing framework, Laravel Mix, Blade "components" and "slots", route model binding on broadcast channels, higher order messages for Collections, object-based Eloquent events, job-level "retry" and "timeout" settings, "realtime" facades, improved support for Redis Cluster, custom pivot table models, middleware for request input trimming and cleaning, and more. In addition, the entire codebase of the framework was reviewed and refactored for general cleanliness.

This documentation summarizes the most notable improvements to the framework; however, more thorough change logs are always available on GitHub.

Markdown Mail & Notifications

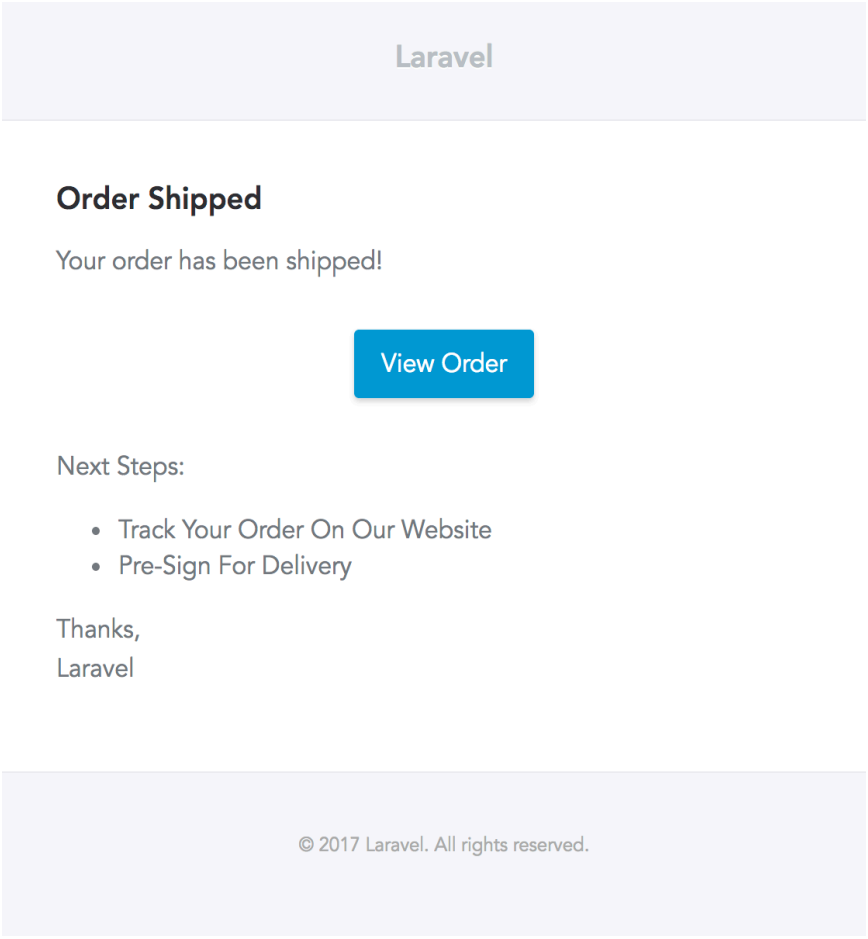
There is a free video tutorial for this feature available on Laracasts.

Markdown mailable messages allow you to take advantage of the pre-built templates and components of mail notifications in your mailables. Since the messages are written in Markdown, Laravel is able to render beautiful, responsive HTML templates for the messages while also automatically generating a plain-text counterpart. For example, a Markdown email might look something like the following:

```
@component('mail::message')
# Order Shipped
```

```
Your order has been shipped!

@component('mail::button', ['url' => $url])
View Order
@endcomponent

Next Steps:

- Track Your Order On Our Website
- Pre-Sign For Delivery

Thanks,<br>
{{ config('app.name') }}
@endcomponent
```

Using this simple Markdown template, Laravel is able to generate a responsive HTML email and plain-text counterpart:



To read more about Markdown mail and notifications, check out the full mail and notification documentation.

You may export all of the Markdown mail components to your own application for customization. To export the components, use the `vendor:publish` Artisan command to publish the `laravel-mail` asset tag.

Laravel Dusk

There is a free video tutorial for this feature available on Laracasts.

Laravel Dusk provides an expressive, easy-to-use browser automation and testing API. By default, Dusk does not require you to install JDK or Selenium on your machine. Instead, Dusk uses a standalone ChromeDriver installation. However, you are free to utilize any other Selenium compatible driver you wish.

Since Dusk operates using a real browser, you are able to easily test and interact with your applications that heavily use JavaScript:

```
/**
 * A basic browser test example.
 *
 * @return void
 */

public function testBasicExample()
{
    $user = factory(User::class)->create([
        'email' => 'taylor@laravel.com',
    ]);

    $this->browse(function ($browser) use ($user) {

        $browser->loginAs($user)
                ->visit('/home')
                ->press('Create Playlist')
                ->whenAvailable('.playlist-modal', function ($modal) {
                    $modal->type('name', 'My Playlist')
                          ->press('Create');
                });

        $browser->waitForText('Playlist Created');
    });
}
```

For more information on Dusk, consult the full Dusk documentation.

Laravel Mix

There is a free video tutorial for this feature available on Laracasts.

Laravel Mix is the spiritual successor of Laravel Elixir, and its entirely based on Webpack instead of Gulp. Laravel Mix provides a fluent API for defining Webpack build steps for your Laravel application using several common CSS and JavaScript pre-processors. Through simple method chaining, you can fluently define your asset pipeline. For example:

```
mix.js('resources/assets/js/app.js', 'public/js')
   .sass('resources/assets/sass/app.scss', 'public/css');
```

Blade Components & Slots

There is a free video tutorial for this feature available on Laracasts.

Blade components and slots provide similar benefits to sections and layouts; however, some may find the mental model of components and slots easier to understand. First, let's imagine a reusable "alert" component we would like to reuse throughout our application:

```
<!-- /resources/views/alert.blade.php -->

<div class="alert alert-danger">

    {{ $slot }}

</div>
```

The `{{ $slot }}` variable will contain the content we wish to inject into the component. Now, to construct this component, we can use the `@component` Blade directive:

```
@component('alert')
    <strong>Whoops!</strong> Something went wrong!
@endcomponent
```

Named slots allow you to provide multiple slots into a single component:

```
<!-- /resources/views/alert.blade.php -->

<div class="alert alert-danger">
    <div class="alert-title">{{ $title }}</div>

    {{ $slot }}
</div>
```

Named slots may be injected using the `@slot` directive. Any content is not within a `@slot` directive will be passed to the component in the `$slot` variable:

```
@component('alert')
    @slot('title')
        Forbidden
    @endslot
```

```
    You are not allowed to access this resource!
@endcomponent
```

To read more about components and slots, consult the full Blade documentation.

Broadcast Model Binding

Just like HTTP routes, channel routes may now take advantage of implicit and explicit route model binding. For example, instead of receiving the string or numeric order ID, you may request an actual `Order` model instance:

```
use App\Order;

Broadcast::channel('order.{order}', function ($user, Order $order) {
    return $user->id === $order->user_id;

});
```

To read more about broadcast model binding, consult the full event broadcasting documentation.

Collection Higher Order Messages

There is a free video tutorial for this feature available on Laracasts.

Collections now provide support for "higher order messages", which are short-cuts for performing common actions on collections. The collection methods that provide higher order messages are: `contains`, `each`, `every`, `filter`, `first`, `map`, `partition`, `reject`, `sortBy`, `sortByDesc`, and `sum`.

Each higher order message can be accessed as a dynamic property on a collection instance. For instance, let's use the `each` higher order message to call a method on each object within a collection:

```
$users = User::where('votes', '>', 500)->get();

$users->each->markAsVip();
```

Likewise, we can use the `sum` higher order message to gather the total number of "votes" for a collection of users:

```
$users = User::where('group', 'Development')->get();

return $users->sum->votes;
```

Object Based Eloquent Events

There is a free video tutorial for this feature available on Laracasts.

Eloquent event handlers may now be mapped to event objects. This provides a more intuitive way of handling Eloquent events and makes it easier to test the events. To get started, define an `$events`property on your Eloquent model that maps various points of the Eloquent model's lifecycle to your own event classes:

```php
<?php

namespace App;

use App\Events\UserSaved;
use App\Events\UserDeleted;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The event map for the model.
     *
     * @var array
     */
    protected $events = [
        'saved' => UserSaved::class,
        'deleted' => UserDeleted::class,
    ];
}
```

Job Level Retry & Timeout

Previously, queue job "retry" and "timeout" settings could only be configured globally for all jobs on the command line. However, in Laravel 5.4, these settings may be configured on a per-job basis by defining them directly on the job class:

```php
<?php

namespace App\Jobs;

class ProcessPodcast implements ShouldQueue
{
    /**
     * The number of times the job may be attempted.
     *
     * @var int
     */
    public $tries = 5;
```

```
    /**
     * The number of seconds the job can run before timing out.
     *
     * @var int
     */
    public $timeout = 120;
}
```

For more information about these settings, consult the full queue documentation.

Request Sanitization Middleware

There is a free video tutorial for this feature available on Laracasts.

Laravel 5.4 includes two new middleware in the default middleware stack: `TrimStrings` and `ConvertEmptyStringsToNull`:

```
/**
 * The application's global HTTP middleware stack.
 *
 * These middleware are run during every request to your application.
 *
 * @var array
 */
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
];
```

These middleware will automatically trim request input values and convert any empty strings to `null`. This helps you normalize the input for every request entering into your application and not have to worry about continually calling the `trim` function in every route and controller.

"Realtime" Facades

There is a free video tutorial for this feature available on Laracasts.

Previously, only Laravel's own built-in services exposed facades, which provide quick, terse access to their methods via the service container. However, in Laravel 5.4, you may easily convert any of your application's classes into a facade in realtime simply by prefixing the imported class name with `Facades`. For example, imagine your application contains a class like the following:

```
<?php

namespace App\Services;
```

```
class PaymentGateway
{
    protected $tax;

    /**
     * Create a new payment gateway instance.
     *
     * @param  TaxCalculator  $tax
     * @return void
     */
    public function __construct(TaxCalculator $tax)
    {
        $this->tax = $tax;
    }

    /**
     * Pay the given amount.
     *
     * @param  int  $amount
     * @return void
     */
    public function pay($amount)
    {
        // Pay an amount...
    }
}
```

You may easily use this class as a facade like so:

```
use Facades\ {
    App\Services\PaymentGateway
};

Route::get('/pay/{amount}', function ($amount) {
    PaymentGateway::pay($amount);
});
```

Of course, if you leverage a realtime facade in this way, you may easily write a test for the interaction using Laravel's facade mocking capabilities:

```
PaymentGateway::shouldReceive('pay')->with('100');
```

Custom Pivot Table Models

In Laravel 5.3, all "pivot" table models for `belongsToMany` relationships used the same built-in `Pivot`model instance. In Laravel 5.4, you may define custom models for your pivot tables. If you

would like to define a custom model to represent the intermediate table of your relationship, use the `using` method when defining the relationship:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany('App\User')->using('App\UserRole');
    }
}
```

Improved Redis Cluster Support

Previously, it was not possible to define Redis connections to single hosts and to clusters in the same application. In Laravel 5.4, you may now define Redis connections to multiple single hosts and multiple clusters within the same application. For more information on Redis in Laravel, please consult the full Redis documentation.

Migration Default String Length

Laravel 5.4 uses the `utf8mb4` character set by default, which includes support for storing "emojis" in the database. If you are upgrading your application from Laravel 5.3, you are not required to switch to this character set.

If you choose to switch to this character set manually and are running a version of MySQL older than the 5.7.7 release, you may need to manually configure the default string length generated by migrations. You may configure this by calling the `Schema::defaultStringLength` method within your `AppServiceProvider`:

```
use Illuminate\Support\Facades\Schema;

/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    Schema::defaultStringLength(191);
```