# Laravel notification

## Notifications

Most of the mail that is sent from web apps really has the purpose of notifying users that a particular action has happened or needs to happen. As users' communication preferences grow increasingly diverse, we gather ever increasingly disparate packages to communicate via Slack, SMS, and other means. Laravel 5.3 introduced a new concept in Laravel called, fittingly, notifications. Just like mailable, a notification is a PHP class that represents a single communication that you might want to send to your users. For now, let us imagine we are notifying users of our physical training app that they have a new workout available on our app. Each class represents all of the information necessary to send notifications to your users using one or many notification channels. A single notification could send an email, send an SMS via Nexmo, send a WebSockets ping, add a record to a database, send a message to a Slack channel, and much more. So, let us create our notification:

php artisan make: notification WorkoutAvailable

this gives us:

```php
<?php

namespace App\Notifications;

use Illuminate\Bus\Queueable;
use Illuminate\Notifications\Notification;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;

class WorkoutAvailable extends Notification
{
    use Queueable;

    /**
     * Create a new notification instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Get the notification's delivery channels.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function via($notifiable)
    {
        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        return (new MailMessage)
                    ->line('The introduction to the notification.')
                    ->action('Notification Action', 'https://laravel.com')
                    ->line('Thank you for using our application!');
    }

    /**
     * Get the array representation of the notification.



         *
         * @param  mixed  $notifiable
         * @return array
         */
        public function toArray($notifiable)
        {
            return [
                //
            ];
        }
    }
```

We can learn a few things here. First, we are going to pass in relevant data to the constructor. Second, there's a via() method that allows us to define, for a given user, which notification channels to use ($notifiable represents whatever entities you want to notify in your system; for most apps, it will be a user, but that is not always the case). And third, there are individual methods for each notification channel that allow you to specifically define how to send one of these notifications through that channel.

**Defining the via () method for your notifiable**

As, we are somehow responsible for deciding, for each notification and each notifiable, which notification channel we are going to use. You could just send everything as mail or just send everything as SMS.

Or you could create a method on each notifiable that allows for some complex notification logic. For example, you could notify the user over certain channels during work hours and other channels in the evening. What is important is that via() is a PHP class method, so you can do whatever com- plex logic you want there.

**Sending Notifications**

There are two ways to send a notification: using the Notification facade or adding the Notifiable trait to an Eloquent class (likely your User class).

1. Sending notifications using the notifiable trait

Any model that imports the Laravel\Notifications\Notifiable trait (which the App\User class does by default) has a notify() method that can be passed a notification.

2. Sending notifications with the notification façade

The Notification facade is the clumsier of the two methods since you have to pass both the notifiable and the notification. However, it is Fhelpful because you can choose to pass more than one notifiable in at the same time.

## Queuing Notifications

Most of the notification drivers need to send HTTP requests to send their notifications, which could slow down your user experience, so you want to queue your notifications. All notifications import the Queueable trait by default, so all you need to do is add implements ShouldQueue to your notification and Laravel will instantly move it to a queue. As with any other queued features, you will need to make sure you have your queue setting configured correctly and queue worker running.

## Out-of-the-Box Notification types

Out of the box, Laravel comes with notification drivers for email, database, broadcast, Nexmo SMS, and Slack. I will cover each briefly, but I would recommend referring to the docs for more thorough introductions to each. It is also easy to create your own notification drivers, and dozens of people already have; you can find them at Laravel Notification Channels website.

- Email notifications
- Database notifications
- Broadcast notifications
- SMS notifications
- Slack notifications