Notifications in Laravel

by Sajal Soni 23 Apr 2018

Difficulty: Beginner Length: Medium Languages: English

Laravel 5 PHP Web Development

In this article, we're going to explore the notification system in the Laravel web framework. The notification system in Laravel allows you to send notifications to users over different channels. Today, we'll discuss how you can send notifications over the mail channel.

Basics of Notifications

During application development, you often need to notify users about different state changes. It could be either sending email notifications when the order status is changed or sending an SMS about their login activity for security purposes. In particular, we're talking about messages that are short and just provide insight into the state changes.

Laravel already provides a built-in feature that helps us achieve something similar—notifications. In fact, it makes sending notification messages to users a breeze and a fun experience!

The beauty of that approach is that it allows you to choose from different channels notifications will be sent on. Let's quickly go through the different notification channels supported by Laravel.

- **Mail:** The notifications will be sent in the form of email to users.
- **SMS:** As the name suggests, users will receive SMS notifications on their phone.

- **Slack:** In this case, the notifications will be sent on Slack channels.
- **Database:** This option allows you to store notifications in a database should you wish to build a custom UI to display it.

Among different notification channels, we'll use the mail channel in our example use-case that we're going to develop over the course of this tutorial.

In fact, it'll be a pretty simple use-case that allows users of our application to send messages to each user. When users receive a new message in their inbox, we'll notify them about this event by sending an email to them. Of course, we'll do that by using the notification feature of Laravel!

Create a Custom Notification Class

As we discussed earlier, we are going to set up an application that allows users of our application to send messages to each other. On the other hand, we'll notify users when they receive a new message from other users via email.

In this section, we'll create necessary files that are required in order to implement the use-case that we're looking for.

To start with, let's create the `Message` model that holds messages sent by users to each other.

```
$php artisan make:model Message --migration
```

We also need to add a few fields like `to`, `from` and `message` to the `messages` table. So let's change the migration file before running the `migrate` command.

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateMessagesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */

```

```
    public function up()
    {
        Schema::create('messages', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('from', FALSE, TRUE);
            $table->integer('to', FALSE, TRUE);
            $table->text('message');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('messages');
    }
}
```

Now, let's run the migrate command that creates the messages table in the database.

```
$php artisan migrate
```

That should create the `messages` table in the database.

Also, make sure that you have enabled the default Laravel authentication system in the first place so that features like registration and login work out of the box. If you're not sure how to do that, the Laravel documentation provides a quick insight into that.

Since each notification in Laravel is represented by a separate class, we need to create a custom notification class that will be used to notify users. Let's use the following artisan command to create a custom notification class—NewMessage.

```
$php artisan make:notification NewMessage
```

That should create the `app/Notifications/NewMessage.php` class, so let's replace the contents of that file with the following contents.

```
<?php
// app/Notifications/NewMessage.php
namespace App\Notifications;

use Illuminate\Bus\Queueable;
use Illuminate\Notifications\Notification;

```

```php
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use App\User;

class NewMessage extends Notification
{
    use Queueable;
    public $fromUser;

    /**
     * Create a new notification instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->fromUser = $user;
    }

    /**
     * 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)
    {
        $subject = sprintf('%s: You\'ve got a new message from %s!', config('app.
        $greeting = sprintf('Hello %s!', $notifiable->name);

        return (new MailMessage)
                    ->subject($subject)
                    ->greeting($greeting)
                    ->salutation('Yours Faithfully')
                    ->line('The introduction to the notification.')
                    ->action('Notification Action', url('/'))
                    ->line('Thank you for using our application!');
    }

    /**
     * Get the array representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function toArray($notifiable)
```

```
    {
        return [
            //
        ];
    }
}
```

As we're going to use the mail channel to send notifications to users, the `via` method is configured accordingly. So this is the method that allows you to configure the channel type of a notification.

Next, there's the `toMail` method that allows you to configure various email parameters. In fact, the `toMail` method should return the instance of `\Illuminate\Notifications\Messages\MailMessage`, and that class provides useful methods that allow you to configure email parameters.

Among various methods, the `line` method allows you to add a single line in a message. On the other hand, there's the `action` method that allows you to add a call-to-action button in a message.

In this way, you could format a message that will be sent to users. So that's how you're supposed to configure the notification class while you're using the mail channel to send notifications.

At the end, you need to make sure that you implement the necessary methods according to the channel type configured in the `via` method. For example, if you're using the database channel that stores notifications in a database, you don't need to configure the `toMail` method; instead, you should implement the `toArray` method, which formats the data that needs to be stored in a database.

How to Send Notifications

In the previous section, we created a notification class that's ready to send notifications. In this section, we'll create files that demonstrate how you could actually send notifications using the `NewMessage` notification class.

Let's create a controller file at `app/Http/Controllers/NotificationController.php` with the following contents.

```php
<?php
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Message;
use App\User;
use App\Notifications\NewMessage;
use Illuminate\Support\Facades\Notification;

class NotificationController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
    }

    public function index()
    {
        // user 2 sends a message to user 1
        $message = new Message;
        $message->setAttribute('from', 2);
        $message->setAttribute('to', 1);
        $message->setAttribute('message', 'Demo message from user 2 to user 1.');
        $message->save();

        $fromUser = User::find(2);
        $toUser = User::find(1);

        // send notification using the "user" model, when the user receives new m
        $toUser->notify(new NewMessage($fromUser));

        // send notification using the "Notification" facade
        Notification::send($toUser, new NewMessage($fromUser));
    }
}
```

Of course, you need to add an associated route in the `routes/web.php` file.

```php
Route::get('notify/index', 'NotificationController@index');
```

There are two ways Laravel allows you to send notifications: by using either the notifiable entity or the Notification facade.

If the entity model class utilizes the `Illuminate\Notifications\Notifiable` trait, then you could call the `notify` method on that model. The `App\User` class implements the `Notifiable` trait and thus it becomes the notifiable entity. On the other hand, you could also use the `Illuminate\Support\Facades\Notification` Facade to send notifications to users.

Let's go through the `index` method of the controller.

In our case, we're going to notify users when they receive a new message. So we've tried to mimic that behavior in the `index` method in the first place.

Next, we've notified the recipient user about a new message using the `notify` method on the `$toUser` object, as it's the *notifiable* entity.

```
$toUser->notify(new NewMessage($fromUser));
```

You may have noticed that we also pass the `$fromUser` object in the first argument of the `__construct` method, since we want to include the *from* username in a message.

On the other hand, if you want to mimic it using the `Notification` facade, it's pretty easy to do so using the following snippet.

```
Notification::send($toUser, new NewMessage($fromUser));
```

As you can see, we've used the `send` method of the Notification facade to send a notification to a user.

Go ahead and open the URL http://your-laravel-site-domain/notify/index in your browser. If you're not logged in yet, you'll be redirected to the login screen. Once you're logged in, you should receive a notification email at the email address that's attached with the user `1`.

You may be wondering how the notification system detects the `to` address when we haven't configured it anywhere yet. In that case, the notification system tries to find the `email` property in the notifiable object. And the `App\User` object class already has that property as we're using the default Laravel authentication system.

However, if you would like to override this behavior and you want to use a different property other than email, you just need to define the following method in your notification class.

```
public function routeNotificationForMail()
{
    return $this->email_address;
}
```

Now, the notification system should look for the `email_address` property instead of the `email` property to fetch the `to` address.

And that's how to use the notification system in Laravel. That brings us to the end of this article as well!

Conclusion

What we've gone through today is one of the useful, yet least discussed, features in Laravel—notifications. It allows you to send notifications to users over different channels.

After a quick introduction, we implemented a real-world example that demonstrated how to send notifications over the mail channel. In fact, it's really handy in the case of sending short messages about state changes in your application.

For those of you who are either just getting started with Laravel or looking to expand your knowledge, site, or application with extensions, we have a variety of things you can study in Envato Market.

Should you have any queries or suggestions, don't hesitate to post them using the feed below!

Sajal Soni
Software Engineer, INDIA