Course - Laravel Framework

Notification

Ngoài hỗ trợ gửi email, Laravel còn hỗ trợ gửi thông báo trên nhiều kênh gửi, bao gồm email, SMS (qua Vonage, trước đây gọi là Nexmo) và Slack.

Tags: notification, thong bao, laravel

Giới thiệu

Ngoài hỗ trợ gửi email, Laravel còn hỗ trợ gửi thông báo trên nhiều kênh khác, bao gồm email, SMS (qua Vonage, trước đây gọi là Nexmo) và Slack. Ngoài ra, nhiều kênh thông báo do cộng đồng xây dựng đã được tạo ra để gửi thông báo qua hàng chục kênh khác nhau! Thông báo cũng có thể được lưu trữ trong cơ sở dữ liệu để chúng có thể được hiển thị trong giao diện web của bạn.

Thông thường, thông báo phải là những tin nhắn ngắn gọn, mang tính thông tin để thông báo cho người dùng về điều gì đó đã xảy ra trong ứng dụng của bạn. Ví dụ: nếu bạn đang viết ứng dụng thanh toán, bạn có thể gửi thông báo "Hóa đơn đã thanh toán" cho người dùng của mình qua các kênh email và SMS.

Tạo thông báo

Trong Laravel, mỗi thông báo được đặt trong một class duy nhất và thường được lưu trữ trong thư mục *app/Notifications*. Đừng lo lắng nếu bạn không thấy thư mục này trong ứng dụng của mình - nó sẽ tự động tạo cho bạn khi bạn chạy lệnh Artisan `make:notification`:

```
php artisan make:notification InvoicePaid
```

Lệnh này sẽ đặt một class thông báo mới trong thư mục *app/Notifications* của bạn. Mỗi class notification chứa một phương thức `via` và một số phương thức xây dựng nên thông điệp, chẳng hạn như `toMail` hoặc `toDatabase`, chuyển đổi thông báo thành một thông điệp được điều chỉnh cho kênh cụ thể đó.

Gửi thông báo

Sử dụng đặc điểm đáng chú ý

Notification có thể được gửi theo hai cách: sử dụng phương thức notify của trait Notifiable hoặc sử dụng facade Notification. Trait Notifiable được bao gồm trên model mặc định `App\Models\User`:

```
<?php

namespace App\Models;
```

```php
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
  use Notifiable;
}
```

Phương thức `notify` được cung cấp bởi trait này sẽ nhận được một trường hợp thông báo:

```php
use App\Notifications\InvoicePaid;

$user->notify(new InvoicePaid($invoice));
```

> Hãy nhớ rằng, bạn có thể sử dụng trait `Notifiable` trên bất kỳ mô hình nào của mình. Bạn không bị giới hạn chỉ bao gồm nó trên mô hình `User` của bạn.

Sử dụng facade Notification

Ngoài ra, bạn có thể gửi thông báo qua facade `Notification`. Cách tiếp cận này hữu ích khi bạn cần gửi notification đến nhiều đối tượng cần thông báo khác, chẳng hạn như một tập hợp người dùng. Để gửi notification bằng cách sử dụng facade, hãy truyền tất cả các đối tượng này và thông điệp cụ thể vào phương thức `send`:

```php
use Illuminate\Support\Facades\Notification;

Notification::send($users, new InvoicePaid($invoice));
```

Bạn cũng có thể gửi notification ngay lập tức bằng phương thức `sendNow`. Phương thức này sẽ gửi notification ngay lập tức ngay cả khi notification triển khai interface `ShouldQueue`:

```php
Notification::sendNow($developers, new DeploymentCompleted($deployment));
```

Chỉ định các kênh phân phối

Mỗi class notification đều có phương thức `via` để xác định được kênh nào thì notification sẽ được gửi đi. Notification có thể được gửi trên các kênh `mail`, `database`, `boardcast`, `vonage` và `slack`.

> **Chú ý:** Nếu bạn muốn sử dụng các kênh phân phối khác như Telegram hoặc Pusher, hãy xem trang web về Kênh Notification Laravel do cộng đồng điều hành.

Phương thức `via` nhận được một cá thể `$notifiable`, đây sẽ là một cá thể của class mà thông báo đang được gửi đến. Bạn có thể sử dụng `$notifiable` để xác định kênh nào sẽ gửi thông báo:

```
/**
 * Get the notification's delivery channels.
 *
 * @param  mixed  $notifiable
 * @return array
 */
public function via($notifiable)
{
  return $notifiable->prefers_sms ? ['nexmo'] : ['mail', 'database'];
}
```

Xếp hàng chờ các thông báo

> **Chú ý:** Trước khi notification được xếp hàng, bạn nên cấu hình hàng chờ của mình và bắt đầu một worker.

Việc gửi notification có thể mất thời gian, đặc biệt nếu channel cần thực hiện lệnh gọi API bên ngoài để gửi notification. Và để tăng tốc thời gian phản hồi của ứng dụng, hãy để notification của bạn được xếp hàng đợi bằng cách thêm interface ShouldQueue và trait Queueable vào class của bạn. Interface và trait đã được nhập cho tất cả các thông báo được tạo bằng lệnh `make:notification`, vì vậy bạn có thể ngay lập tức thêm chúng vào class notification của mình:

```
<?php
```

```php
namespace App\Notifications;

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

class InvoicePaid extends Notification implements ShouldQueue
{
  use Queueable;

  // ...
}
```

Khi interface `ShouldQueue` đã được thêm vào notification của bạn, bạn có thể gửi notification như bình thường. Laravel sẽ phát hiện interface `ShouldQueue` trên class và tự động xếp hàng gửi notification:

```php
$user->notify(new InvoicePaid($invoice));
```

Nếu bạn muốn trì hoãn việc gửi notification, bạn có thể gói tiếp nối phương thức delay vào sau phần khởi tạo notification của mình:

```php
$delay = now()->addMinutes(10);

$user->notify((new InvoicePaid($invoice))->delay($delay));
```

Bạn có thể truyền một mảng cho phương thức `delay` để chỉ định lượng thời gian trễ cho các kênh cụ thể:

```php
$user->notify((new InvoicePaid($invoice))->delay([
  'mail' => now()->addMinutes(5),
  'sms' => now()->addMinutes(10),
]));
```

Khi xếp hàng thông báo, một công việc đã xếp hàng sẽ được tạo cho từng người nhận trên

mỗi kênh. Ví dụ: sáu công việc sẽ được gửi đến hàng chờ nếu notification của bạn có ba người nhận và hai kênh.

Tùy chỉnh kết nối hàng chờ thông báo

Theo mặc định, các notification đã xếp hàng sẽ được xếp vào hàng đợi bằng cách sử dụng kết nối hàng chờ mặc định của ứng dụng của bạn. Nếu bạn muốn chỉ định một kết nối khác sẽ được sử dụng cho một notification cụ thể nào đó, thì bạn có thể xác định thuộc tính `$connection` trên notification class:

```
/**
 * The name of the queue connection to use when queueing the notification.
 *
 * @var string
 */
public $connection = 'redis';
```

Tùy chỉnh hàng chờ kênh thông báo

Nếu bạn muốn chỉ định một hàng đợi cụ thể sẽ được sử dụng cho từng kênh notification trong tính năng notification, bạn có thể xác định phương thức `viaQueues` trên notification của mình. Phương thức này sẽ trả về một mảng các cặp <tên kênh/tên hàng chờ>:

```
/**
 * Determine which queues should be used for each notification channel.
 *
 * @return array
 */
public function viaQueues()
{
  return [
    'mail' => 'mail-queue',
    'slack' => 'slack-queue',
  ];
}
```

Thông báo đã xếp hàng chờ & Transaction cơ sở dữ liệu

Khi các notification xếp hàng đợi được gửi đi trong các transaction của cơ sở dữ liệu, chúng có thể được hàng đợi xử lý trước khi transaction của cơ sở dữ liệu được thông qua. Khi điều này xảy ra, bất kỳ cập nhật nào bạn thực hiện trên các model hoặc record của cơ sở dữ liệu trong quá trình transaction của cơ sở dữ liệu rất có thể không được phản ánh trong cơ sở dữ liệu. Ngoài ra, bất kỳ model hoặc record của cơ sở dữ liệu nào được tạo trong transaction có thể không tồn tại trong cơ sở dữ liệu. Nếu notification của bạn phụ thuộc vào các kiểu models này, các lỗi không mong muốn có thể xảy ra khi công việc gửi notification đã xếp hàng chờ được xử lý.

Nếu tùy chọn cấu hình `after_commit` của kết nối hàng chờ của bạn được đặt thành `false`, thì bạn cũng vẫn có thể cho ứng dụng của bạn gửi đi một notification đã được xếp hàng chờ cụ thể nào đó sau khi tất cả các transaction của cơ sở dữ liệu đang mở đã được thông qua bằng cách gọi phương thức `afterCommit` khi gửi notification:

```
use App\Notifications\InvoicePaid;

$user->notify((new InvoicePaid($invoice))->afterCommit());
```

Ngoài ra, bạn có thể gọi phương thức `afterCommit` từ phương thức constructor của notification class:

```
<?php

namespace App\Notifications;

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

class InvoicePaid extends Notification implements ShouldQueue
{
  use Queueable;

  /**
   * Create a new notification instance.
   *
```

```
     * @return void
     */
    public function __construct()
    {
        $this->afterCommit();
    }
}
```

> **Chú ý:** Để tìm hiểu thêm về cách giải quyết những vấn đề này, vui lòng xem lại tài liệu liên quan đến các công việc được xếp hàng đợi và giao dịch cơ sở dữ liệu.

Xác định xem có nên gửi thông báo đã xếp hàng chờ hay không

Sau khi notification xếp hàng chờ đã được gửi vào hàng chờ để xử lý dưới background, notification đó thường sẽ được hàng chờ chấp nhận và gửi đến người nhận.

Tuy nhiên, nếu bạn muốn đưa ra quyết định cuối cùng về việc liệu notification xếp hàng chờ có được gửi sau khi nó đang được xử lý bởi hàng chờ hay không, thì bạn có thể khai báo phương thức `shouldSend` trên notification class. Nếu phương thức này trả về `false`, thông báo sẽ không được gửi:

```
/**
 * Determine if the notification should be sent.
 *
 * @param  mixed  $notifiable
 * @param  string  $channel
 * @return bool
 */
public function shouldSend($notifiable, $channel)
{
    return $this->invoice->isPaid();
}
```

Notification theo yêu cầu

Đôi khi bạn có thể cần phải gửi notification cho một người nào đó không được lưu trữ với tư cách là "người dùng" ứng dụng của bạn. Bằng cách sử dụng phương thức route của

facade Notification, bạn có thể chỉ định thông tin routing notification ad-hoc trước khi gửi thông báo:

```
Notification::route('mail', 'taylor@example.com')
  ->route('nexmo', '5555555555')
  ->route('slack', 'https://hooks.slack.com/services/...')
  ->notify(new InvoicePaid($invoice));
```

Nếu bạn muốn cung cấp tên của người nhận khi gửi notification theo yêu cầu đến mail route, bạn có thể cung cấp một mảng chứa địa chỉ email làm khóa và tên là giá trị của phần tử đầu tiên trong mảng:

```
Notification::route('mail', [
  'barrett@example.com' => 'Barrett Blair',
])->notify(new InvoicePaid($invoice));
```

Mail Notification

Định dạng thông điệp mail

Nếu một notification được gửi dưới dạng email, bạn nên khai báo phương thức `toMail` trên notification class. Phương thức này sẽ nhận được `$notifiable` và sẽ trả về một đối tượng `Illuminate\Notifications\Messages\MailMessage`.

Lớp `MailMessage` chứa một vài phương thức đơn giản để giúp bạn xây dựng các tin nhắn email transaction. Tin nhắn này có thể chứa các dòng văn bản cũng như "call to action". Hãy xem một ví dụ về phương thức `toMail`:

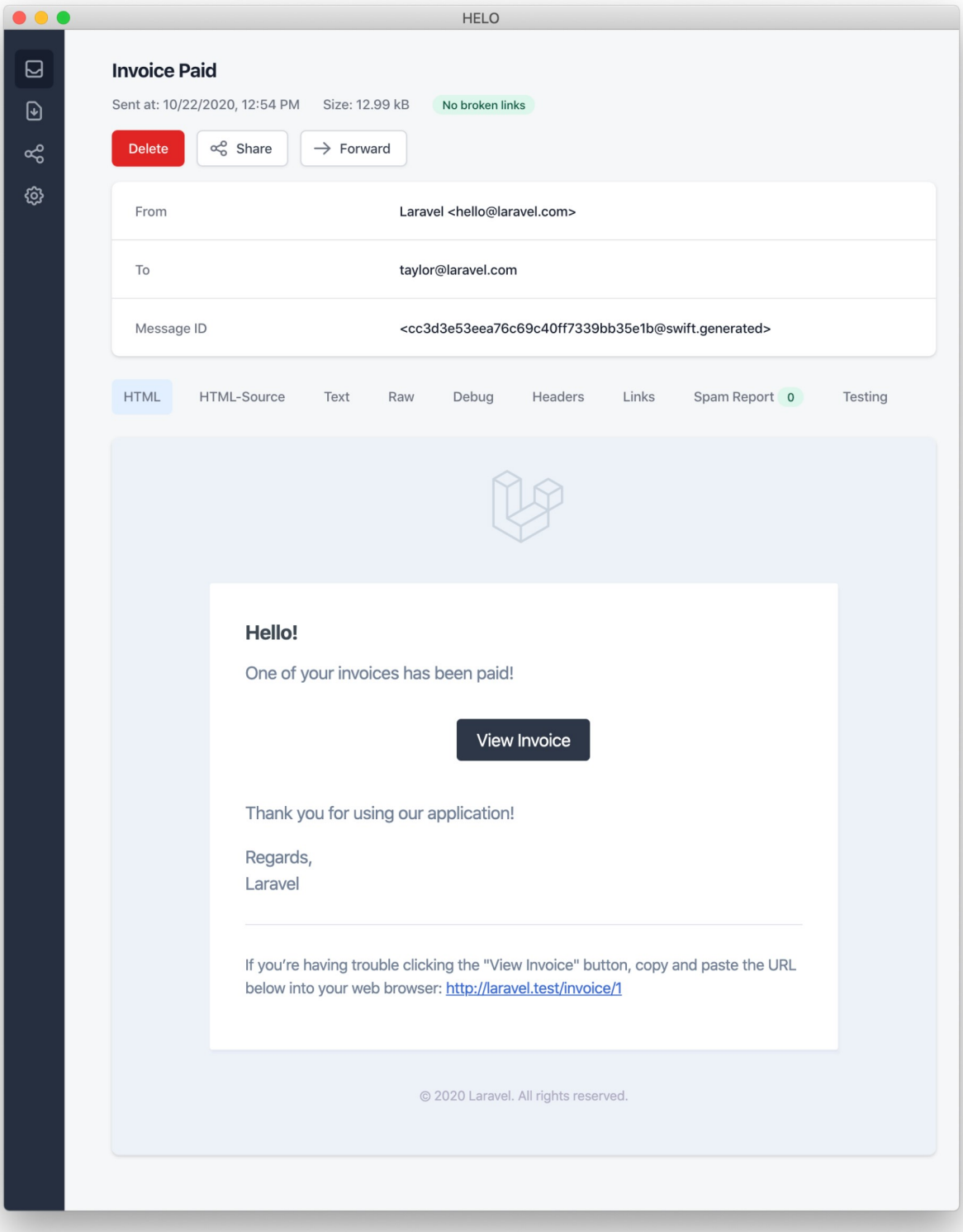
```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
```

```
    $url = url('/invoice/'.$this->invoice->id);

    return (new MailMessage)
        ->greeting('Hello!')
        ->line('One of your invoices has been paid!')
        ->action('View Invoice', $url)
        ->line('Thank you for using our application!');
}
```

> **Chú ý:** Lưu ý rằng chúng tôi đang sử dụng `$this->invoice->id` trong phương thức `toMail` của chúng tôi. Bạn có thể truyền bất kỳ dữ liệu nào mà notification của bạn cần để tạo thông điệp vào phương thức constructor của class notification.

Trong ví dụ này, chúng tôi đăng ký một lời chào, một dòng văn bản, một cuộc gọi đến action và sau đó là một dòng văn bản khác. Các phương thức này được cung cấp bởi đối tượng `MailMessage` sẽ làm cho nó đơn giản và nhanh chóng trong việc định dạng các transactional email nhỏ. Mail channel sau đó sẽ dịch các component thông điệp thành HTML template. Dưới đây là một ví dụ về email được tạo bởi mail channel:



Chú ý: Khi gửi notification qua mail, hãy cấu hình tùy chọn `name` trong tệp cấu hình `config/app.php` của bạn. Giá trị này sẽ được sử dụng trong header và footer của thông điệp mail notification của bạn.

Tùy chọn khác trong việc định dạng mail notification

Thay vì xác định "dòng" văn bản trong notification class, thì bạn có thể sử dụng phương thức `view` để chỉ định template tùy chỉnh mà sẽ được sử dụng trong việc hiển thị notification email:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)->view(
    'emails.name', ['invoice' => $this->invoice]
  );
}
```

Bạn có thể chỉ định hình thức văn bản thuần túy cho mail bằng cách truyền tên của view resource đang cần làm phần tử thứ hai của mảng mà sẽ được cấp cho phương thức `view` như sau:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)->view(
    ['emails.name.html', 'emails.name.plain'],
    ['invoice' => $this->invoice]
  );
}
```

Thông điệp lỗi

Một số notification sẽ thông tin cho người dùng về các lỗi, chẳng hạn như thanh toán hóa đơn không thành công. Bạn có thể chỉ ra một thông điệp mail nào đó liên quan đến lỗi bằng cách gọi phương thức `error` khi xây dựng thông điệp của mình. Khi sử dụng phương thức `error` trên một thông điệp mail nào đó, nút gọi action sẽ có màu đỏ thay vì màu đen:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Message
 */
public function toMail($notifiable)
{
  return (new MailMessage)
    ->error()
    ->subject('Notification Subject')
    ->line('...');
}
```

Điều chỉnh người gửi - sender

Theo mặc định, địa chỉ người gửi của email sẽ được khai báo trong tập tin cấu hình *config/mail.php*. Tuy nhiên, bạn có thể khai báo địa chỉ người gửi cho một notification cụ thể bằng cách sử dụng phương thức `from`:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)
    ->from('barrett@example.com', 'Barrett Blair')
```

```
            ->line('...');
    }
```

Điều chỉnh người nhận

Khi gửi notification qua `mail` channel, hệ thống notification sẽ tự động tìm kiếm thuộc tính `email` trên đối tượng cần thông báo của bạn. Bạn có thể tùy chỉnh địa chỉ email nào sẽ được sử dụng để gửi notification đi bằng cách xác định phương thức `routeNotificationForMail` trên đối tượng cần thông báo:

```
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * Route notifications for the mail channel.
     *
     * @param  \Illuminate\Notifications\Notification  $notification
     * @return array|string
     */
    public function routeNotificationForMail($notification)
    {
        // Return email address only...
        return $this->email_address;

        // Return email address and name...
        return [$this->email_address => $this->name];
    }
}
```

Điều chỉnh tiêu đề

Theo mặc định, tiêu đề của email là tên class của notification được định dạng thành "Title Case". Vì vậy, nếu notification class của bạn có tên là `InvoicePaid`, thì tiêu đề của email sẽ là "Invoice Paid". Nếu bạn muốn chỉ định một tiêu đề thay thế cái mặc định cho notification của mình, thì bạn có thể dùng phương thức `subject` khi xây dựng thông điệp của mình:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)
    ->subject('Notification Subject')
    ->line('...');
}
```

Điều chỉnh Mailer

Theo mặc định, mail thông báo sẽ được gửi bằng cách sử dụng mailer mặc định được cấu hình trong tập tin *config/mail.php*. Tuy nhiên, bạn có thể chỉ định một mailer khác trong thời gian chạy (runtime) bằng cách gọi phương thức `mailer` khi tạo thông điệp cho thông báo của bạn:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)
    ->mailer('postmark')
```

```
        ->line('...');
}
```

Điều chỉnh các template

Bạn có thể sửa đổi template HTML và văn bản thuần túy được sử dụng bởi thông báo mail bằng cách publish tài nguyên của gói notification. Sau khi chạy lệnh Artisan này, các template thông báo mail sẽ được đặt trong thư mục `resources/views/vendor/notifications`:

```
php artisan vendor:publish --tag=laravel-notifications
```

Đính kèm tập tin

Để thêm tập tin đính kèm vào mail thông báo, hãy sử dụng phương thức `attach` trong khi xây dựng thông báo của bạn. Phương thức `attach` chấp nhận đường dẫn tuyệt đối của tập tin đính kèm làm đối số thứ nhất của nó:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
    return (new MailMessage)
        ->greeting('Hello!')
        ->attach('/path/to/file');
}
```

Trong khi đính kèm tập tin vào một thông điệp, bạn cũng có thể chỉ định tên hiển thị và/hoặc thêm kiểu MIME bằng cách truyền một mảng làm đối số thứ hai cho phương thức `attach`:

```
/**
```

```
     * Get the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        return (new MailMessage)
            ->greeting('Hello!')
            ->attach('/path/to/file', [
                'as' => 'name.pdf',
                'mime' => 'application/pdf',
            ]);
    }
```

Không giống như đính kèm tập tin trong các đối tượng mail, bạn không thể đính kèm tập tin trực tiếp từ đĩa lưu trữ bằng cách sử dụng `attachFromStorage`. Bạn nên sử dụng phương thức `attach` với một đường dẫn tuyệt đối đến tập tin trên đĩa lưu trữ. Ngoài ra, bạn có thể trả về một mail từ phương thức `toMail`:

```
use App\Mail\InvoicePaid as InvoicePaidMailable;

/**
     * Get the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return Mailable
     */
    public function toMail($notifiable)
    {
        return (new InvoicePaidMailable($this->invoice))
            ->to($notifiable->email)
            ->attachFromStorage('/path/to/file');
    }
```

Đính kèm dữ liệu thô

Phương thức `attachData` có thể được sử dụng để đính kèm một chuỗi byte thô dưới dạng tập tin đính kèm. Khi gọi phương thức `attachData`, bạn nên cung cấp tên tập tin sẽ được gán cho tập tin đính kèm:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
{
  return (new MailMessage)
    ->greeting('Hello!')
    ->attachData($this->pdf, 'name.pdf', [
      'mime' => 'application/pdf',
    ]);
}
```

Sử dụng Mail

Nếu cần, bạn có thể trả về một đối tượng mail đầy đủ từ phương thức `toMail` trong notification của bạn. Khi trả về một `Mailable` thay vì `MailMessage`, bạn sẽ cần chỉ định người nhận thư bằng cách sử dụng phương thức `to` của đối tượng mail:

```
use App\Mail\InvoicePaid as InvoicePaidMailable;


/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return Mailable
 */
public function toMail($notifiable)
{
  return (new InvoicePaidMailable($this->invoice))
```

```
        ->to($notifiable->email);
}
```

Mailable & Notification theo yêu cầu

Nếu bạn đang gửi notification theo yêu cầu, thì phiên bản `$notifiable` được cung cấp cho phương thức `toMail` sẽ là một phiên bản của `Illuminate\Notifications\AnonymousNotifiable`, cung cấp một phương thức `routeNotificationFor` có thể được sử dụng để truy xuất địa chỉ email mà notification theo yêu cầu nên được gửi đến:

```
use App\Mail\InvoicePaid as InvoicePaidMailable;
use Illuminate\Notifications\AnonymousNotifiable;

/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return Mailable
 */
public function toMail($notifiable)
{
  $address = $notifiable instanceof AnonymousNotifiable
    ? $notifiable->routeNotificationFor('mail')
    : $notifiable->email;

  return (new InvoicePaidMailable($this->invoice))
    ->to($address);
}
```

Xem trước mail thông báo

Khi thiết kế một template cho mail thông báo, thật tiện lợi khi xem trước nhanh chóng thông điệp của thư được hiển thị ra trên trình duyệt của bạn giống như một template *Blade* điển hình. Vì lý do này, Laravel cho phép bạn trả lại bất kỳ mail nào được tạo bởi mail thông báo trực tiếp từ controller hoặc hàm nóng của route. Khi một `MailMessage` được trả lại, nó sẽ được hiển thị và hiển thị trong trình duyệt, cho phép bạn nhanh chóng xem trước thiết kế của nó mà không cần gửi nó đến một địa chỉ email thực tế:

```php
use App\Models\Invoice;
use App\Notifications\InvoicePaid;

Route::get('/notification', function () {
  $invoice = Invoice::find(1);

  return (new InvoicePaid($invoice))
    ->toMail($invoice->user);
});
```

Mail thông báo markdown

Mail thông báo markdown cho phép bạn tận dụng các template mail thông báo được tạo sẵn, đồng thời cho phép bạn tự do viết thông điệp dài hơn và tùy ý. Vì các thông điệp được viết bằng Markdown, nên Laravel có thể hiển thị các template HTML, đáp ứng cho các nội dung thông điệp trong khi vừa tự động tạo một bản văn bản thuần túy.

Tạo thông điệp

Để tạo thông điệp với template Markdown tương ứng, bạn có thể sử dụng tùy chọn `--markdown` của lệnh Artisan `make:notification`:

```
php artisan make:notification InvoicePaid --markdown=mail.invoice.paid
```

Giống như tất cả các thông báo mail khác, các thông báo sử dụng Markdown phải khai báo phương thức `toMail` trên notification class của chúng. Tuy nhiên, thay vì sử dụng các phương thức `line` và `action` để tạo thông điệp, hãy sử dụng phương thức `markdown` để chỉ định tên của Markdown template sẽ được sử dụng. Một mảng dữ liệu sẽ được cung cấp vào template có thể được truyền làm đối số thứ hai cho phương thức:

```php
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\MailMessage
 */
public function toMail($notifiable)
```

```
{
  $url = url('/invoice/'.$this->invoice->id);

  return (new MailMessage)
    ->subject('Invoice Paid')
    ->markdown('mail.invoice.paid', ['url' => $url]);
}
```

Button component

Button component hiển thị một link button được căn giữa. Component này chấp nhận hai đối số, một url và một màu tùy chọn. Các màu được hỗ trợ là primary, green và red. Bạn có thể thêm nhiều button component vào notification theo ý bạn muốn:

```
@component('mail::button', ['url' => $url, 'color' => 'green'])
View Invoice
@endcomponent
```

Panel component

Component kiểu panel sẽ hiển thị khối văn bản nào đó trong một panel có màu nền hơi khác so với phần còn lại của mail thông báo. Điều này cho phép bạn thu hút sự chú ý của người đọc vào một khối văn bản cần nhấn mạnh:

```
@component('mail::panel')
This is the panel content.
@endcomponent
```

Table component

Component kiểu table cho phép bạn chuyển đổi Markdown table thành một table HTML. Component này chấp nhận Markdown table làm nội dung của nó. Căn chỉnh cột table được hỗ trợ bằng cách sử dụng cú pháp căn chỉnh Markdown table:

```
@component('mail::table')
```

```
| Laravel       | Table         | Example  |
| ------------- |:-------------:| --------:|
| Col 2 is      | Centered      | $10      |
| Col 3 is      | Right-Aligned | $20      |
@endcomponent
```

Tùy chỉnh các component

Bạn có thể xuất tất cả các component kiểu Markdown sang ứng dụng của riêng mình để tùy chỉnh. Để xuất các component, hãy sử dụng lệnh Artisan `vendor:publish` cho tag `laravel-mail`:

```
php artisan vendor:publish --tag=laravel-mail
```

Lệnh này sẽ publish các component Markdown vào thư mục *resources/views/vendor/mail*. Thư mục mail sẽ chứa một thư mục *html* và một thư mục *text*, mỗi thư mục chứa các đại diện tương ứng cho component của nó. Bạn có thể tùy ý điều chỉnh các component này theo ý bạn muốn.

Tùy chỉnh CSS

Sau khi xuất các component, thì thư mục *resources/views/vendor/mail/html/themes* sẽ chứa tập tin *default.css*. Bạn có thể tùy chỉnh CSS trong tập tin này và các style của bạn sẽ tự động được xếp trong hàng trong HTML của mail thông báo Markdown của bạn.

Nếu bạn muốn xây dựng một giao diện hoàn toàn mới cho các Markdown component, thì bạn có thể đặt một tập tin CSS trong thư mục *html/themes*. Sau khi đặt tên và lưu tập tin CSS, hãy cập nhật tùy chọn `theme` trong tập tin cấu hình `mail` khớp với tên của theme mới của bạn.

Để tùy chỉnh theme cho một mail thông báo riêng lẻ, bạn có thể gọi phương thức `theme` trong khi xây dựng mail thông báo. Phương thức `theme` chấp nhận tên của theme sẽ được sử dụng khi gửi mail thông báo:

```
/**
 * Get the mail representation of the notification.
 *
 * @param  mixed  $notifiable
```

```
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        return (new MailMessage)
            ->theme('invoice')
            ->subject('Invoice Paid')
            ->markdown('mail.invoice.paid', ['url' => $url]);
    }
```

Cơ sở dữ liệu của thông báo

Điều kiện yêu cầu

Trong CSDL có lưu trữ được thông tin mail thông báo trong một bảng CSDL. Bảng này sẽ chứa thông tin như loại thông báo cũng như cấu trúc dữ liệu JSON mô tả các thông số cần thiết cho một mail thông báo.

Bạn có thể truy vấn bảng này để hiển thị các thông báo trong giao diện người dùng của ứng dụng. Tuy nhiên, trước khi bạn có thể làm điều đó, bạn sẽ cần tạo một bảng CSDL để chứa các thông số của mail thông báo. Bạn có thể sử dụng lệnh Artisan notification:table để tạo migration với schema bảng phù hợp:

```
php artisan notifications:table
```

```
php artisan migrate
```

Định dạng CSDL thông báo

Nếu thông báo được lưu trữ trong bảng CSDL, bạn nên khai báo phương thức `toDatabase` hoặc `toArray` trên notification class. Phương thức này sẽ nhận dữ liệu `$notifiable` và sẽ trả về một mảng PHP thuần túy. Mảng trả về sẽ được mã hóa dưới dạng JSON và được lưu trữ trong cột dữ liệu của bảng thông báo của bạn. Hãy xem một ví dụ về phương thức `toArray`:

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

```
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function toArray($notifiable)
    {
      return [
        'invoice_id' => $this->invoice->id,
        'amount' => $this->invoice->amount,
      ];
    }
```

Chọn `toDatabase` hay `toArray`

Phương thức `toArray` cũng được kênh quảng bá sử dụng để xác định dữ liệu nào sẽ phát tới giao diện người dùng hỗ trợ JavaScript của bạn. Nếu bạn muốn có hai mảng khác nhau cho CSDL và kênh quảng bá, thì bạn nên chọn phương thức `toDatabase` thay vì phương thức `toArray`.

Truy cập thông báo

Sau khi mail thông báo được lưu trữ trong cơ sở dữ liệu, bạn sẽ cần truy cập đến chúng. Trait `Illuminate\Notifications\Notifiable` có trên mô hình `App\Models\User` mặc định của Laravel, chứa các Eloquent của `notifications` mà sẽ trả về các đối tượng thông báo. Để tải notification, bạn có thể tìm dữ liệu giống như mọi eloquent khác. Theo mặc định, các thông báo sẽ được sắp xếp theo dấu thời gian `created_at` và thứ tự ưu tiên cho các thông báo được tạo gần đây nhất:

```
$user = App\Models\User::find(1);

foreach ($user->notifications as $notification) {
  echo $notification->type;
}
```

Nếu bạn chỉ muốn truy xuất các thông báo "unread", bạn có thể sử dụng relationship `unreadNotifications`. Các thông báo này sẽ được sắp xếp theo dấu thời gian `created_at` với các thông báo gần đây nhất được xếp ở phía đầu bảng:

```php
$user = App\Models\User::find(1);

foreach ($user->unreadNotifications as $notification) {
  echo $notification->type;
}
```

> **Chú ý:** Để truy cập thông báo từ ứng dụng JavaScript của bạn, bạn nên controller thông báo cho ứng dụng của mình, controller này sẽ trả về thông báo cần tìm, chẳng hạn như thông báo liên quan đến người dùng hiện tại. Sau đó, bạn có thể thực hiện một HTTP request tới URL của controller đó từ ứng dụng JavaScript của bạn.

Đánh dấu thông báo là read

Thông thường, bạn sẽ muốn đánh dấu một thông báo là "read" khi người dùng đã xem qua nó. Trait `Illuminate\Notifications\Notifiable` có một phương thức `markAsRead`, nó sẽ cập nhật cột `read_at` trên record của cơ sở dữ liệu mail thông báo:

```php
$user = App\Models\User::find(1);

foreach ($user->unreadNotifications as $notification) {
  $notification->markAsRead();
}
```

Tuy nhiên, thay vì lặp qua từng mail thông báo, bạn có thể sử dụng phương thức `markAsRead` trực tiếp trên tập hợp các mail thông báo:

```php
$user->unreadNotifications->markAsRead();
```

Bạn cũng có thể sử dụng truy vấn mass-update để đánh dấu tất cả các notifications là read mà không cần truy xuất chúng từ cơ sở dữ liệu:

```php
$user = App\Models\User::find(1);

$user->unreadNotifications()->update(['read_at' => now()]);
```

Bạn có thể xóa các thông báo hoàn toàn khỏi bảng CSDL:

```
$user->notifications()->delete();
```

Truyền phát thông báo

Điều kiện yêu cầu

Trước khi phát thông báo, bạn nên cấu hình và làm quen với các service truyền phát event của Laravel. Truyền phát event có một cách tương tác với các event Laravel phía máy chủ từ giao diện người dùng có hỗ trợ JavaScript của bạn.

Định dạng thông báo truyền phát

Channel thông báo sẽ truyến phát các thông báo bằng cách sử dụng các service truyền phát event của Laravel, cho phép ứng dụng JavaScript của bạn nhận thông báo theo thời gian thực. Nếu một thông báo hỗ trợ truyền phát, bạn có thể khai báo phương thức `toBroadcast` trên notification class. Phương thức này sẽ nhận dữ liệu `$notifiable` và sẽ trả về một đối tượng `BroadcastMessage`. Nếu phương thức `toBroadcast` không tồn tại, phương thức `toArray` sẽ được sử dụng để thu thập dữ liệu cần được phát sóng. Dữ liệu trả về sẽ được mã hóa dưới dạng JSON và phát tới giao diện người dùng hỗ trợ JavaScript của bạn. Hãy xem một ví dụ về phương thức `toBroadcast`:

```
use Illuminate\Notifications\Messages\BroadcastMessage;

/**
 * Get the broadcastable representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return BroadcastMessage
 */
public function toBroadcast($notifiable)
{
  return new BroadcastMessage([
    'invoice_id' => $this->invoice->id,
    'amount' => $this->invoice->amount,
```

```
    ]);
}
```

Cấu hình hàng chờ để truyền phát

Tất cả các thông báo này cũng có thể được xếp hàng chờ để phát đi. Nếu bạn muốn cấu hình kết nối hàng chờ hoặc tên hàng chờ được sử dụng để xếp hàng chuẩn bị phát đi, bạn có thể sử dụng các phương thức `onConnection` và `onQueue` của `BroadcastMessage`:

```
return (new BroadcastMessage($data))
    ->onConnection('sqs')
    ->onQueue('broadcasts');
```

Tùy chỉnh loại thông báo

Ngoài dữ liệu bạn chỉ định, tất cả các thông báo truyền phát cũng có trường loại chứa tên lớp đầy đủ của thông báo. Nếu bạn muốn tùy chỉnh cấu hình `type`, bạn có thể khai báo phương thức `broadcastType` trên lớp thông báo:

```
use Illuminate\Notifications\Messages\BroadcastMessage;

/**
 * Get the type of the notification being broadcast.
 *
 * @return string
 */
public function broadcastType()
{
    return 'broadcast.message';
}
```

Theo dõi thông báo

Thông báo sẽ phát trên kênh riêng tư được định dạng theo cú pháp `{notifiable}.{id}`. Vì vậy, nếu bạn đang gửi thông báo đến đối tượng `App\Models\User` có `ID` là `1`, thông báo sẽ được phát trên kênh riêng tư như `App.Models.User.1`. Khi sử dụng Laravel Echo,

bạn có thể dễ dàng theo dõi thông báo trên một kênh nào đó bằng phương thức `notification`:

```
Echo.private('App.Models.User.' + userId)
  .notification((notification) => {
    console.log(notification.type);
  });
```

Tùy chỉnh kênh thông báo

Nếu bạn muốn tùy chỉnh kênh mà thông báo được phát trên đó, bạn có thể khai báo phương thức `acceptBroadcastNotificationsOn` trên thực thể thông báo:

```
<?php

namespace App\Models;

use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
  use Notifiable;

  /**
   * The channels the user receives notification broadcasts on.
   *
   * @return string
   */
  public function receivesBroadcastNotificationsOn()
  {
    return 'users.'.$this->id;
  }
}
```

Thông báo qua SMS

Điều kiện yêu cầu

Gửi thông báo SMS trong Laravel được hỗ trợ bởi package Vonage (trước đây gọi là Nexmo). Trước khi có thể gửi thông báo qua Vonage, bạn cần cài đặt package `laravel/nexmo-notification-channel` và `nexmo/laravel` với Composer.

```
composer require laravel/nexmo-notification-channel nexmo/laravel
```

Package `nexmo/laravel` có tập tin cấu hình của riêng nó. Tuy nhiên, bạn không bắt buộc phải xuất tập tin cấu hình này sang ứng dụng của riêng mình. Bạn có thể chỉ cần sử dụng các biến môi trường `NEXMO_KEY` và `NEXMO_SECRET` để cài đặt khóa công khai và khóa bí mật cho package Vonage của mình.

Tiếp theo, bạn sẽ cần thêm mục nhập cấu hình nexmo vào *config/services.php* của mình.

```
'nexmo' => [
  'sms_from' => '15556666666',
],
```

Tùy chọn `sms_from` chính là số điện thoại mà tin nhắn SMS của bạn sẽ được gửi từ đó. Bạn nên tạo số điện thoại cho ứng dụng của mình trong bảng điều khiển Vonage.

Định dạng thông báo SMS

Nếu thông báo hỗ trợ gửi dưới dạng SMS, bạn nên khai báo phương thức toNexmo trên notification class. Phương thức này sẽ nhận dữ liệu $notifiable và sẽ trả về một đối tượng `Illuminate\Notifications\Messages\NexmoMessage`:

```
/**
 * Get the Vonage / SMS representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\NexmoMessage
 */
public function toNexmo($notifiable)
{
```

```
    return (new NexmoMessage)
        ->content('Your SMS message content');
}
```

Nội dung Unicode

Nếu tin nhắn SMS của bạn chứa các ký tự unicode, bạn nên gọi phương thức `unicode` khi tạo đối tượng `NexmoMessage`:

```
/**
 * Get the Vonage / SMS representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\NexmoMessage
 */
public function toNexmo($notifiable)
{
    return (new NexmoMessage)
        ->content('Your unicode message')
        ->unicode();
}
```

Định dạng thông báo Shortcode

Laravel cũng hỗ trợ gửi thông báo shortcode, là các mẫu tin nhắn pre-defined trong tài khoản Vonage của bạn. Để gửi thông báo SMS shortcode, bạn nên khai báo phương thức `toShortcode` trên notification class của mình. Từ bên trong phương thức này, bạn có thể trả về một mảng chỉ định loại thông báo (`alert`, `2fa` hoặc `marketing`) cũng như các giá trị tùy chỉnh sẽ được điền vào template:

```
/**
 * Get the Vonage / Shortcode representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return array
 */
```

```
public function toShortcode($notifiable)
{
  return [
    'type' => 'alert',
    'custom' => [
      'code' => 'ABC123',
    ],
  ];
}
```

Giống như routing Thông báo SMS, bạn nên triển khai phương thức `routeNotificationForShortcode` trên mô hình thông báo của mình.

Tùy chỉnh số điện thoại "From"

Nếu bạn muốn gửi một số thông báo từ một số điện thoại khác với số điện thoại đã được chỉ định trong tập tin *config/services.php* của mình, bạn có thể gọi phương thức `from` trên đối tượng `NexmoMessage`:

```
/**
 * Get the Vonage / SMS representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return NexmoMessage
 */
public function toNexmo($notifiable)
{
  return (new NexmoMessage)
    ->content('Your SMS message content')
    ->from('15554443333');
}
```

Thêm tham chiếu khách hàng

Nếu bạn muốn theo dõi chi phí cho mỗi người dùng, nhóm hoặc khách hàng, bạn có thể thêm "client reference" vào thông báo. Vonage sẽ cho phép bạn tạo báo cáo bằng cách sử

dụng tham chiếu khách hàng (client reference) này để bạn có thể hiểu rõ hơn về việc sử dụng SMS của một khách hàng cụ thể. Client reference có thể là bất kỳ chuỗi nào có tối đa 40 ký tự:

```
/**
 * Get the Vonage / SMS representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return NexmoMessage
 */
public function toNexmo($notifiable)
{
  return (new NexmoMessage)
    ->clientReference((string) $notifiable->id)
    ->content('Your SMS message content');
}
```

Routing thông báo SMS

Để routing thông báo Vonage đến số điện thoại thích hợp, hãy khai báo phương thức `routeNotificationForNexmo` trên đối tượng của bạn như sau:

```
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
  use Notifiable;

  /**
   * Route notifications for the Nexmo channel.
   *
   * @param  \Illuminate\Notifications\Notification  $notification
```

```
     * @return string
     */
    public function routeNotificationForNexmo($notification)
    {
        return $this->phone_number;
    }
}
```

Thông báo Slack

Điều kiện yêu cầu

Trước khi bạn có thể gửi thông báo qua Slack, bạn phải cài đặt kênh thông báo Slack qua Composer:

```
composer require laravel/slack-notification-channel
```

Bạn cũng sẽ cần tạo ứng dụng Slack cho nhóm của mình. Sau khi tạo ứng dụng, bạn nên định cấu hình "Incoming Webhook" cho workspace. Sau đó, Slack sẽ cung cấp cho bạn một URL webhook mà bạn có thể sử dụng khi routing thông báo Slack.

Định dạng thông báo Slack

Nếu thông báo hỗ trợ được gửi dưới dạng tin nhắn Slack, bạn nên xác định phương thức `toSlack` trên notification class. Phương thức này sẽ nhận dữ liệu `$notifiable` và sẽ trả về một đối tượng `Illuminate\Notifications\Messages\SlackMessage`. Thông điệp Slack có thể chứa nội dung văn bản cũng như "attachment" định dạng văn bản bổ sung hoặc một mảng các trường dữ liệu. Hãy xem một ví dụ cơ bản về `toSlack`:

```
/**
 * Get the Slack representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\SlackMessage
 */
public function toSlack($notifiable)
{
```

```
    return (new SlackMessage)
        ->content('One of your invoices has been paid!');
}
```

Tập tin đính kèm với Slack

Bạn cũng có thể thêm "attachment" vào thông điệp Slack. Tập tin đính kèm cung cấp các tùy chọn định dạng phong phú hơn so với tin nhắn văn bản đơn giản. Trong ví dụ này, chúng ta sẽ gửi thông báo lỗi về một ngoại lệ đã xảy ra trong một ứng dụng, bao gồm một liên kết để xem thêm chi tiết về ngoại lệ:

```
/**
 * Get the Slack representation of the notification.
 *
 * @param  mixed  $notifiable
 * @return \Illuminate\Notifications\Messages\SlackMessage
 */
public function toSlack($notifiable)
{
    $url = url('/exceptions/'.$this->exception->id);

    return (new SlackMessage)
        ->error()
        ->content('Whoops! Something went wrong.')
        ->attachment(function ($attachment) use ($url) {
            $attachment->title('Exception: File Not Found', $url)
                ->content('File [background.jpg] was not found.');
        });
}
```

Attachment cũng cho phép bạn chỉ định một mảng dữ liệu sẽ được hiển thị cho người dùng. Dữ liệu đã cho sẽ được trình bày dưới dạng bảng để dễ đọc:

```
/**
 * Get the Slack representation of the notification.
 *
```

```
     * @param  mixed  $notifiable
     * @return SlackMessage
     */
    public function toSlack($notifiable)
    {
      $url = url('/invoices/'.$this->invoice->id);

      return (new SlackMessage)
        ->success()
        ->content('One of your invoices has been paid!')
        ->attachment(function ($attachment) use ($url) {
          $attachment->title('Invoice 1322', $url)
            ->fields([
              'Title' => 'Server Expenses',
              'Amount' => '$1,234',
              'Via' => 'American Express',
              'Was Overdue' => ':-1:',
          ]);
        });
    }
```

Markdown trong nội dung tập tin đính kèm

Nếu một số trường tập tin đính kèm của bạn chứa Markdown, bạn có thể sử dụng phương thức markdown để chỉ dẫn Slack phân tích cú pháp và hiển thị các trường tập tin đính kèm đã cho dưới dạng văn bản có định dạng Markdown. Các giá trị được phương thức này chấp nhận là: pretext, text và/hoặc các trường dữ liệu. Để biết thêm thông tin về định dạng tập tin đính kèm Slack, hãy xem tài liệu API Slack:

```
    /**
     * Get the Slack representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return SlackMessage
     */
    public function toSlack($notifiable)
    {
```

```
  $url = url('/exceptions/'.$this->exception->id);

  return (new SlackMessage)
    ->error()
    ->content('Whoops! Something went wrong.')
    ->attachment(function ($attachment) use ($url) {
      $attachment->title('Exception: File Not Found', $url)
        ->content('File [background.jpg] was *not found*.')
        ->markdown(['text']);
    });
}
```

Routing thông báo Slack

Để routing thông báo Slack đến nhóm và kênh Slack thích hợp, hãy khai báo phương thức `routeNotificationForSlack` trên dữ liệu notifiable của bạn. Điều này sẽ trả về webhook URL mà thông báo sẽ được gửi đến. Webhook URL có thể được tạo bằng cách thêm dịch vụ "Incoming Webhook" vào nhóm Slack của bạn:

```
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
  use Notifiable;

  /**
   * Route notifications for the Slack channel.
   *
   * @param  \Illuminate\Notifications\Notification  $notification
   * @return string
   */
  public function routeNotificationForSlack($notification)
```

```
    {
        return 'https://hooks.slack.com/services/...';
    }
}
```

Bản địa hóa thông báo

Laravel cho phép bạn gửi thông báo bằng ngôn ngữ khác với ngôn ngữ hiện tại của HTTP request và thậm chí sẽ ghi nhớ ngôn ngữ này nếu thông báo được xếp hàng chờ.

Để thực hiện điều này, class `Illuminate\Notifications\Notification` cung cấp một phương pháp bản địa để đặt ngôn ngữ mong muốn. Ứng dụng sẽ thay đổi thành ngôn ngữ này khi thông báo đang được đánh giá và sau đó hoàn nguyên về ngôn ngữ trước đó khi quá trình đánh giá hoàn tất:

```
$user->notify((new InvoicePaid($invoice))->locale('es'));
```

Bản địa hóa cho nhiều mục cũng có thể đạt được thông qua facade `Notification`:

```
Notification::locale('es')->send(
    $users, new InvoicePaid($invoice)
);
```

Ngôn ngữ ưa thích của người dùng

Đôi khi, các ứng dụng cũng muốn lưu trữ ngôn ngữ ưa thích của mỗi người dùng. Bằng cách triển khai contract `HasLocalePreference` trên mô hình thông báo của bạn, bạn có thể chỉ dẫn Laravel sử dụng ngôn ngữ được lưu trữ trước đó khi gửi thông báo:

```
use Illuminate\Contracts\Translation\HasLocalePreference;

class User extends Model implements HasLocalePreference
{
    /**
     * Get the user's preferred locale.
     *
```

```
     * @return string
     */
    public function preferredLocale()
    {
        return $this->locale;
    }
}
```

Khi bạn đã triển khai interface, Laravel sẽ tự động sử dụng ngôn ngữ ưa thích khi gửi thông báo và mail tới mô hình. Do đó, không cần gọi phương thức `locale` khi sử dụng interface này:

```
$user->notify(new InvoicePaid($invoice));
```

Các event trong thông báo mail

Event gửi thông báo

Khi một thông báo đang được gửi đi, event `Illuminate\Notifications\Events\NotificationSending` sẽ gọi lên bởi hệ thống thông báo. Event này chứa cả đối tượng thông báo và dữ liệu thông báo. Bạn có thể đăng ký chương trình theo dõi cho event này trong `EventServiceProvider` của ứng dụng của bạn:

```
/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
    'Illuminate\Notifications\Events\NotificationSending' => [
        'App\Listeners\CheckNotificationStatus',
    ],
];
```

Thông báo sẽ không được gửi nếu một chương trình theo dõi cho event `NotificationSending` trả về `false` từ phương thức `handle` của nó:

```php
use Illuminate\Notifications\Events\NotificationSending;

/**
 * Handle the event.
 *
 * @param  \Illuminate\Notifications\Events\NotificationSending  $event
 * @return void
 */
public function handle(NotificationSending $event)
{
  return false;
}
```

Bên trong chương trình theo dõi event, bạn có thể truy cập các thuộc tính notifiable, notification và channel về event để tìm hiểu thêm về người nhận thông báo hoặc chính thông báo:

```php
/**
 * Handle the event.
 *
 * @param  \Illuminate\Notifications\Events\NotificationSending  $event
 * @return void
 */
public function handle(NotificationSending $event)
{
  // $event->channel
  // $event->notifiable
  // $event->notification
}
```

Sự kiện đã gửi thông báo

Khi một thông báo được gửi đi, event `Illuminate\Notifications\Events\NotificationSent` sẽ được gửi đi bởi hệ thống thông báo. Event này sẽ chứa cả đối tượng thông báo và cả dữ liệu thông báo. Bạn có thể đăng ký chương trình theo dõi cho event này trong `EventServiceProvider` của mình:

```php
/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
  'Illuminate\Notifications\Events\NotificationSent' => [
    'App\Listeners\LogNotification',
  ],
];
```

> Sau khi đăng ký chương trình theo dõi trong `EventServiceProvider` của bạn, thì hãy sử dụng lệnh Artisan `event:create` để tạo nhanh các class cho chương trình theo dõi.

Bên trong chương trình theo dõi event, bạn có thể truy cập các thuộc tính notifiable, notification, channel và response về event để tìm hiểu thêm về người nhận thông báo hoặc chính thông báo:

```php
/**
 * Handle the event.
 *
 * @param  \Illuminate\Notifications\Events\NotificationSent  $event
 * @return void
 */
public function handle(NotificationSent $event)
{
  // $event->channel
  // $event->notifiable
  // $event->notification
  // $event->response
}
```

Các channel tùy chỉnh

Laravel cung cấp một số channel thông báo, nhưng bạn có thể muốn viết driver của riêng

mình để gửi thông báo qua các channel khác. Laravel đã làm cho nó trở nên đơn giản. Để bắt đầu, hãy tạo một class có chứa một phương thức send. Phương thức sẽ nhận được hai đối số: `$notifiable` và `$notification`.

Bên trong phương thức send, bạn có thể gọi các phương thức trên thông báo để truy xuất đối tượng tin nhắn mà kênh của bạn hiểu và sau đó gửi thông báo đến đối tượng `$notifiable` theo cách bạn muốn:

```php
<?php

namespace App\Notifications;

use Illuminate\Notifications\Notification;

class VoiceChannel
{
  /**
   * Send the given notification.
   *
   * @param  mixed  $notifiable
   * @param  \Illuminate\Notifications\Notification  $notification
   * @return void
   */
  public function send($notifiable, Notification $notification)
  {
    $message = $notification->toVoice($notifiable);

    // Send notification to the $notifiable instance...
  }
}
```

Khi class của channel thông báo của bạn đã được tạo ra, bạn có thể trả lại tên class từ phương thức `via` của bất kỳ thông báo nào của bạn. Trong ví dụ này, phương thức `toVoice` trong thông báo của bạn có thể trả về bất kỳ đối tượng nào bạn chọn để đại diện cho tin nhắn thoại. Ví dụ: bạn có thể tạo ra class `VoiceMessage` của riêng mình để đại diện cho các thông báo sau:

```php
<?php

namespace App\Notifications;

use App\Notifications\Messages\VoiceMessage;
use App\Notifications\VoiceChannel;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Notification;

class InvoicePaid extends Notification
{
  use Queueable;

  /**
   * Get the notification channels.
   *
   * @param  mixed  $notifiable
   * @return array|string
   */
  public function via($notifiable)
  {
    return [VoiceChannel::class];
  }

  /**
   * Get the voice representation of the notification.
   *
   * @param  mixed  $notifiable
   * @return VoiceMessage
   */
  public function toVoice($notifiable)
  {
    // ...
  }
}
```