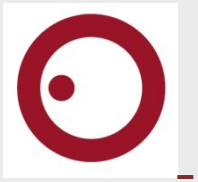

# Laravel

## *Laravel E-mail, Notification and Queues*

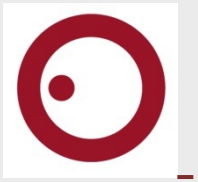
*Marco Monteiro*



# Contributors



- Author(s):
  - Marco Monteiro (marco.monteiro@ipleiria.pt)

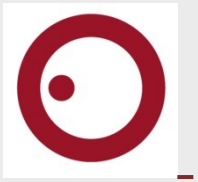


# Summary



1. E-Mail services
2. Notifications
3. Mailables
4. Queues
5. References

# 1 – E-MAIL SERVICES

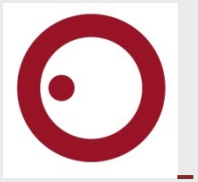


# E-Mail on Laravel



- Laravel provides an API to send E-Mail based on the SwiftMailer library
- Can use drivers for SMTP, Mailgun, Postmark, Amazon SES and sendmail
- Driver and e-mail host configuration can be defined on "config/mail.php" and ".env" file. Example:

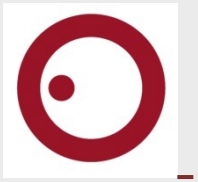
```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=5532ac8…
MAIL_PASSWORD=7acd7f…
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=noreply@mysite.test
MAIL_FROM_NAME="${APP_NAME}"
```



# Sending e-mail during development



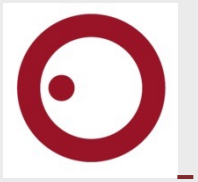
- When developing an application that sends email, you probably don't want to send emails to real email addresses.
- Ways to avoid sending a real email during local development.
  - **Log Driver** - all email messages are written to log files for inspection
  - **Universal To** - all email messages are sent to a universal recipient
  - **Mailtrap** – **recommended approach** - use the Mailtrap service (or similar), that sends all your email messages to a dummy mailbox where you may view them in a true email client



# Sending e-mail during development



- **Log Driver** - all email messages are written to log files for inspection
  - Change the mail driver from "smtp" to "log"
  - To view the messages logs on **storage/logs** folder

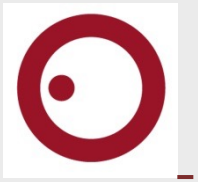


# Sending e-mail during development



- **Universal To** - all email messages are sent to a universal recipient
  - Change the "**to**" option in the **config/mail.php** file

```
'to' => [
    'address' => 'example@example.com',
    'name' => 'Example'
],
```

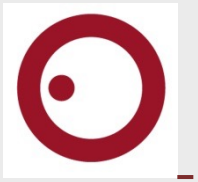


# Sending e-mail during development



- **Mailtrap** – use the Mailtrap service, that sends all your email messages to a dummy mailbox where you may view them in a true email client
- Mailtrap service: https://mailtrap.io/
  - Register via `https://mailtrap.io/register/signup`
  - **Access your inboxes** via `https://mailtrap.io/inboxes`
  - Store the SMTP credentials somewhere safe. Use them to configure the e-mail service (.env file)

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=5532ac8…
MAIL_PASSWORD=7acd7f…
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=noreply@mysite.test
MAIL_FROM_NAME="${APP_NAME}"
```

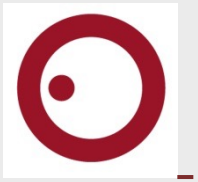


# Sending e-mail



- Laravel sends e-mail:
  - Automatically
    - Examples: when user completes registration if application required email verification; when user requests to reset his password; etc.
  - With **notifications**
    - We can create notifications for different channels – one of the supported channel is the e-mail. Notification e-mail are very simple to create – includes a pre-built mail message with an adjustable message template (appropriate for simple messages)
  - With **Mailable** classes
    - We have full control of the mail message. We can create custom email templates with blade views

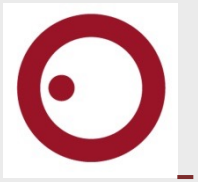
# 2 – NOTIFICATIONS



# Notifications



- Laravel provides support for sending notifications across a variety of delivery channels, including email, SMS (via Vonage, formerly known as Nexmo), Slack and other channels built by the community
- Notifications should be short, informational messages that notify users of something that occurred in your application.
  - For example: "Invoice Paid"; "Order Shipped"; etc.
- A notification can be delivered on multiple channels (e.g. email and SMS).



# Notifications



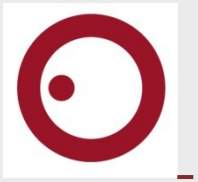
- Each notification is represented by a single class that is typically stored in `app/Notifications` folder
- To create a Notification class:

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

- Notification class extends from `Illuminate\Notifications\Notification`

```
…
use Illuminate\Notifications\Notification;

class InvoicePaid extends Notification
{
    …
```

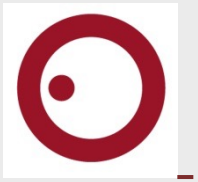


# Notification Class



- Use constructor arguments to add data related to the notification (e.g. the $invoice that was paid)

```
class InvoicePaid extends Notification
{
    private $invoice;
    public function __construct($invoice)
    {
        $this->invoice = $invoice;
    }
…
```



# Notification Class

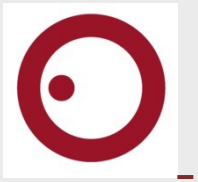


- Supported channels are specified by **via** method

```
public function via($notifiable)
{
    return ['mail'];
}
```

- For each supported channel, add a message building method, such as toMail, toSms, toDatabase, etc… On this method we can customize the message content for that specific channel

```
public function toMail($notifiable)
{
   return (new MailMessage)
      ->line('The introduction to the notification.')
      ->action('Notification Action', url('/'))
      ->line('Thank you for using our application!');
}
```



# Notification Class



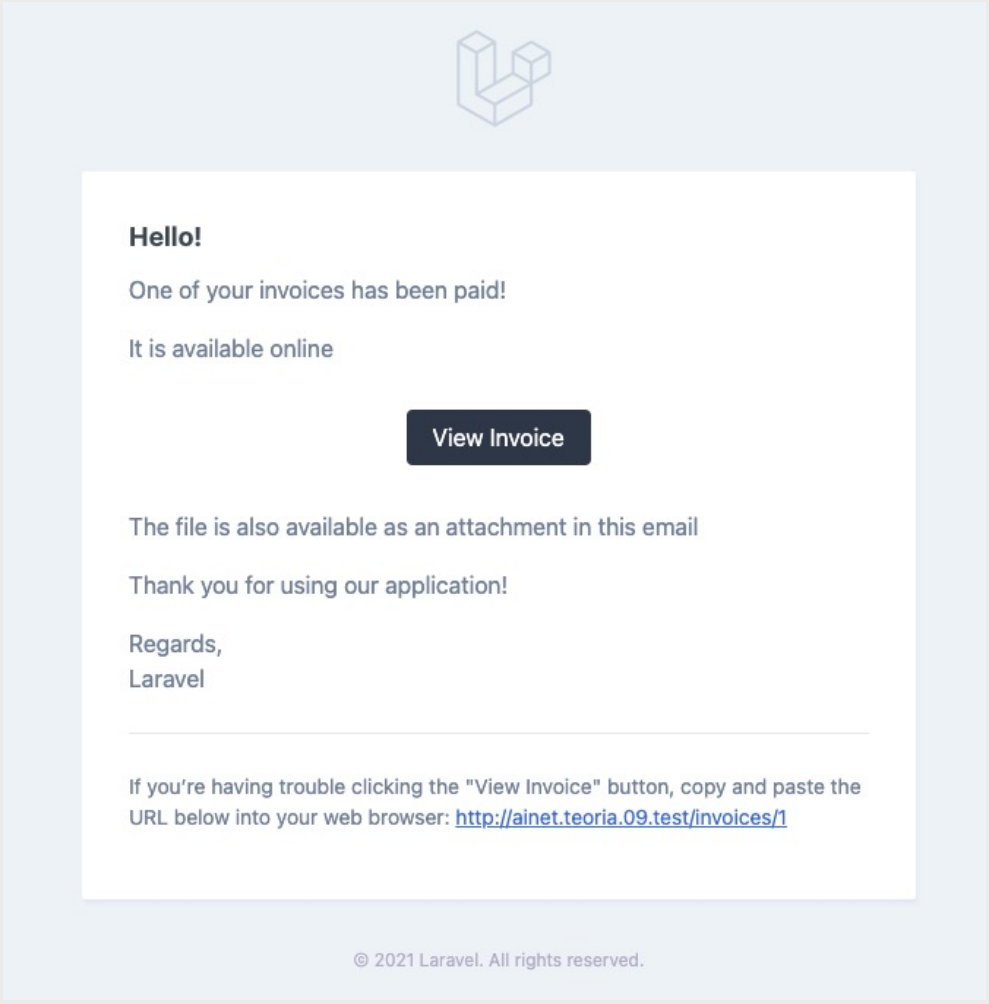
- Example of "**toMail**" method

```
public function toMail($notifiable)
    {
        $url = url('/invoice/' . $this->invoice->id);
        $path = storage_path('app/invoices/' .
                             $this->invoice->fileName);
        return (new MailMessage)
            ->greeting('Hello!')
            ->line('One of your invoices has been paid!')
            ->line('It is available online')
            ->action('View Invoice', $url)
            ->line('The file is also available as an ' .
                        'attachment in this email')
            ->line('Thank you for using our application!')
            ->attach($path);
    }
```

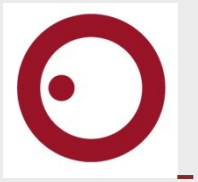
# Notification



- Previous example mail message:




*Mail message will also include an attachment*



# Send e-mail with notifications

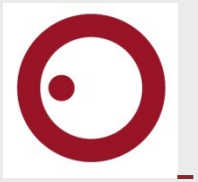


1. With the **User model** (or any other class that include the Notifiable trait)
   - The User class must include the Notifiable trait

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

   - To send the e-mail:

```
use App\Notifications\InvoicePaid;
…
$user = … // some User
$user->notify(new InvoicePaid($invoice));
```



# Send e-mail with notifications



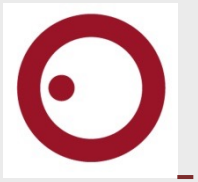
2. With the **Notification** facade class

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

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

**$users** is an array with the list of users that the e-mail is sent to

# 3 – MAILABLES



# Mailable Class

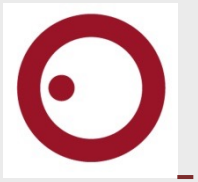


- With Laravel, each type of email sent by your application is represented as a "**Mailable**" class
  - These classes are stored in the `app/Mail` folder
  - We can also use a Mailable class to send email with notifications
  - To create a Mailable class:

```
php artisan make:mail OrderShipped
```

  - Mailable class extends from `Illuminate\Mail\Mailable`

```
…
use Illuminate\Mail\Mailable;
…
class OrderShipped extends Mailable  …
```

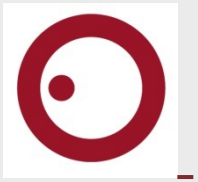


# Mailable Class



- Use constructor arguments to add data related to the mailable (e.g. the $order that was shipped)

```
class OrderShipped extends Mailable
{
    private $order;
    public function __construct($order)
    {
        $this->order = $order;
    }
…
```

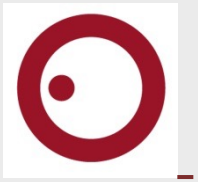


# Mailable Class



- Mailable represents the e-mail message and includes information about the delivery, presentation and content ("from", "subject", "attach", "view", etc.)
- Mailable class' configuration is done in the **build** method. Example:

```
public function build()
{
   return $this->from('noreply@mysite.com')
      ->subject('Your order has shipped')
      ->view('emails.orders.shipped')
      ->with([
                'orderId' => $this->order->id,
                'orderName' => $this->order->name,
                'orderPrice' => $this->order->price,
            ])
      ->attachFromStorage('orders/' . $this->order->filename);
 }
```

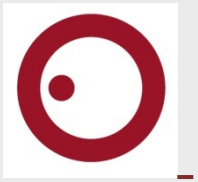


# Mailable View



- Mailable Views are defined as a "normal" blade view

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Mail Message</title>
</head>
<body>
    <h1>Order # {{$orderId}} has been shipped</h1>
    <h3>Order name: {{$orderName}}</h3>
    <h3>Total price: {{$orderPrice}}</h3>
    <p>To access the order receipt click
    <a href="{{route('invoices.show', ['id' => $orderId]) }}">here</a>
    </p>
</body>
</html>
```



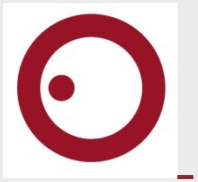
# Mailable View



- Mailable Views can also use Markdown templates (combine blade components and markdown syntax)
- Markdown mailable messages allow you to take advantage of the pre-built templates and components of mail notifications in your mailables.

- Check documentation:

`https://laravel.com/docs/mail#markdown-mailables`



# Send E-Mail with Mailable

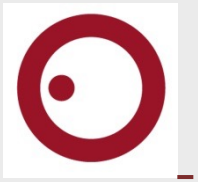


- E-Mail is sent with the **Mail** facade
  - The mail to send is an instance of the Mailable

```
use Illuminate\Support\Facades\Mail;
…
$user =   // User to send the e-mail to
Mail::to($user)
    ->send(new OrderShipped($order));
```

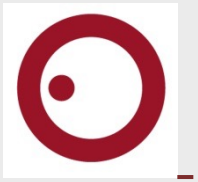
```
use Illuminate\Support\Facades\Mail;
…
$user =   // User to send the e-mail to
Mail::to($user)
    ->cc($moreUsers)
    ->bcc($evenMoreUsers)
    ->send(new OrderShipped($order));
```

- OrderShipped is a **Mailable** class





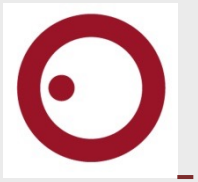
# 4 – QUEUES



# Queues



- Some tasks, such as parsing and storing an uploaded CSV file, sending one or several e-mails, take too long to perform during a typical web request
- For these, we can create queued jobs that may be processed in the background
- By moving time intensive tasks to a queue, your application can respond to web requests with blazing speed and provide a better user experience
- We can send e-mails (with notifications or mailables) using queues.



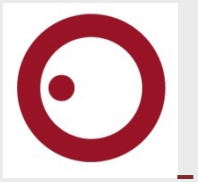
# Send e-mail with notifications - Queues



- Notification can be queued by adding the **ShouldQueue** interface and **Queueable** trait to the Notification class

```
…
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;

class InvoicePaid extends Notification implements ShouldQueue
{
    use Queueable;
    …
}
```



# Send e-mail with notifications - Queues



- After defining the notification as queuable, we can send the e-mail notification as usual

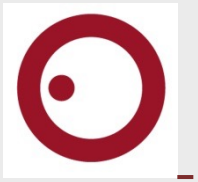
```
use App\Notifications\InvoicePaid;
…
$user = … // some User
$user->notify(new InvoicePaid($invoice));
```

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

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

- We can also specify a delay when sending the e-mail notification using a queue

```
$delay = now()->addMinutes(10);
$user->notify((new InvoicePaid($invoice))->delay($delay));
```



# Send e-mail with Mailable - Queues



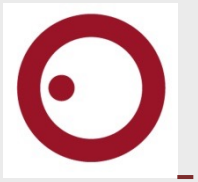
- Mailable can be queued by adding the Queueable trait to the Mailable class
  - By default, Mailable classes are already created with that trait

```
class OrderShipped extends Mailable
{
    use Queueable, SerializesModels;
     …
```

- Then, when sending the e-mail with the Mail facade, replace the method send with **queue** or **later**

```
Mail::to($user)
    ->queue(new OrderShipped($order));
```

```
Mail::to($user)
    ->later(now()->addMinutes(10), new OrderShipped($order));
```



# Queue

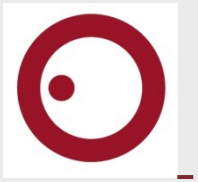


- Previous code (to send e-mail through a queue) only works if the application is connected to a queue service
- Laravel provides a unified queueing API to connect to different queue backends
  - Drivers for  Amazon SQS, Redis, or a relational database
  - Driver and connection to the queue backend can be defined on "config/queue.php" and ".env" file. Example:

```
QUEUE_CONNECTION=database
```

- Note: to use database queue driver, we must create the table "jobs" to hold the jobs - "failed_jobs" holds the jobs that have failed
- This requires us to execute:

```
php artisan queue:table
php artisan migrate
```



# Queue Worker



- After the queue is configured, we must run the Queue Worker, with the command:

```
php artisan queue:work
```

  - *Once the previous command has started, it will continue to run until it is manually stopped, or you close your terminal.*

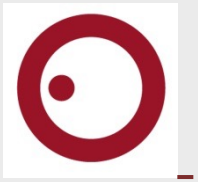
- The queue worker runs at **background** and guarantees that the new jobs as processed as they are pushed onto the queue.

- *If using the "database" queue driver, open the table "jobs" on the database and view the data when a new job is pushed on to the queue*

- Check documentation:

`https://laravel.com/docs/queues#running-the-queue-worker`

# 5 – REFERENCES



# References



- Official Documentation
  - https://laravel.com/docs/mail
  - https://laravel.com/docs/notifications
  - https://laravel.com/docs/queues
- Mailtrap - Email Sandbox Service
  - https://mailtrap.io
- SwiftMailer
  - https://swiftmailer.symfony.com