Manipulation de Mailer

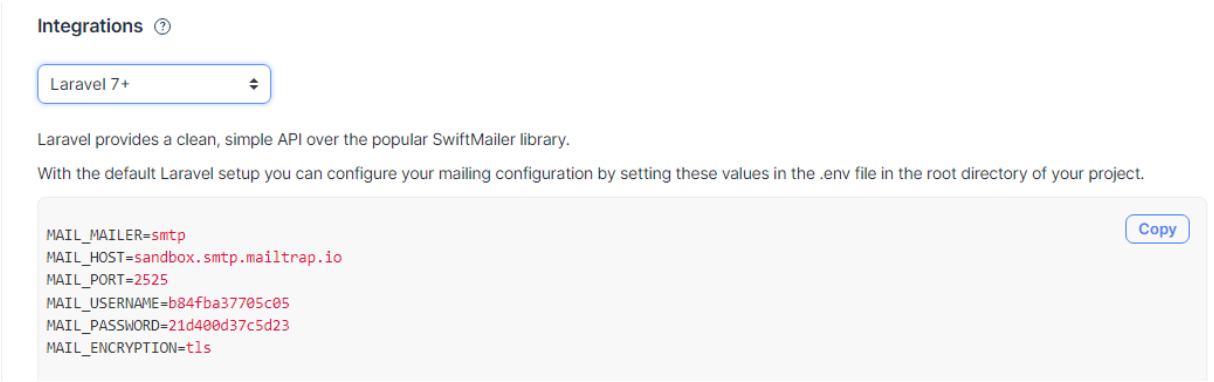
Introduction :

L'envoi d'emails ne doit pas être compliqué. Laravel fournit une API d'envoi d'emails simple et propre, alimentée par le composant populaire Symfony Mailer. Laravel et Symfony Mailer fournissent des pilotes pour l'envoi d'emails via SMTP, Mailgun, Postmark, Amazon SES, et sendmail, vous permettant de commencer rapidement à envoyer des emails via un service local ou basé sur le cloud de votre choix.

Étape 1 – Créer un compte Mailtrap pour envoyer des emails en phase développement

Lorsque vous développez une application qui envoie des courriels, vous ne voulez probablement pas envoyer des courriels à des adresses électroniques réelles. Laravel propose plusieurs façons de "désactiver" l'envoi d'emails pendant le développement local.

Vous pouvez également utiliser un service comme Mailtrap et le pilote smtp pour envoyer vos messages électroniques à une boîte aux lettres "factice" où vous pourrez les consulter avec un véritable client de messagerie. Cette approche présente l'avantage de vous permettre d'inspecter les messages finaux dans le visualisateur de messages de Mailtrap.



Il est ainsi facile de renseigner le fichier **.env** :

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=b84fba37705c05
MAIL_PASSWORD=21d400d37c5d23
MAIL_ENCRYPTION=tls
```

Étape 2 - Après avoir modifié le fichier **.env**, exécutez les deux commandes ci-dessous pour vider le cache et redémarrer le serveur Laravel.

```
php artisan config:cache
```

//config/mail.php

```
  'from' => [
        'address' => env('MAIL_FROM_ADDRESS', 'aminefste@gmail.com'),
        'name' => env('MAIL_FROM_NAME', 'Mail for you'),
    ],
```

Étape 2 - Créer la classe Mailable

Dans cette étape, nous allons créer la classe **Mailable**, qui sera utilisée pour envoyer des courriels. La classe Mailable est responsable de l'envoi des emails en utilisant un mailer qui est configuré dans le fichier **config/mail.php.** En fait, Laravel fournit déjà une commande artisan qui nous permet de créer un modèle de base.

```
php artisan make:mail SendEmail
```

```
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Queue\SerializesModels;
```

```php
class SendMail extends Mailable
{
    use Queueable, SerializesModels;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public $name;
    public function __construct($name)
    {
        $this->name=$name;
    }

    /**
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
    {
        return new Envelope(
            subject: 'notification',
        );
    }

    /**
     * Get the message content definition.
     *
     * @return \Illuminate\Mail\Mailables\Content
     */
    public function content()
    {
        return new Content(
            view: 'mail.mailView',
        );
    }

    /**
     * Get the attachments for the message.
     *
     * @return array
     */
    public function attachments()
    {
        return [];
    }
```

```
}
```

Étape 3 -Pièces jointes

Pour ajouter des pièces jointes à un courrier électronique, vous devez les ajouter au tableau renvoyé par la méthode **attachments** du message. Tout d'abord, vous pouvez ajouter une pièce jointe en fournissant un chemin d'accès au fichier à la méthode fromPath fournie par la classe **Attachment** :

```
 use Illuminate\Mail\Mailables\Attachment;

public function attachments()
    {
        return [
            Attachment::fromStorageDisk('local','example.txt'),
            ];
    }
```

Étape 3 – Créer la vue de l'email

Dans la vue **resources\views\mail\mailView.blade.php**

Copiez le code suivant :

```
<h1>Hi, {{ $name }}</h1>
<p>Sending Mail from Laravel.</p>
```

Étape 4 – Ajouter des routes

Ajoutez les lignes suivantes dans **routes/web.php**. Ici on veut tester l'email.

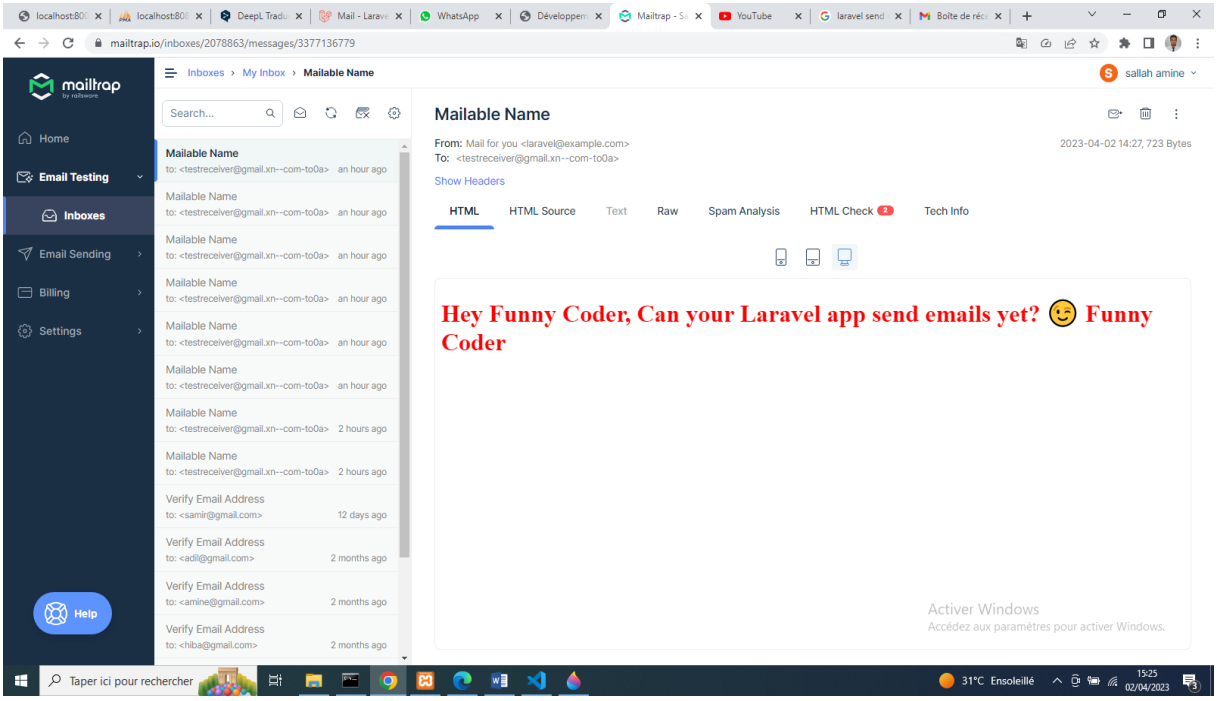
```
Route::get('/sendemail', function() {
    $name = "Funny Coder";

//The email sending is done using the to method on the Mail facade
    Mail::to('testreceiver@gmail.com'')->send(new SendEmail($name));
});
```

Étape 5 - Visitez l'URL suivante pour tester le courrier électronique de base.

```
http://localhost:8000/sendemail
```

Étape 6 - Vous pouvez voir le résultat suivant dans le serveur **Mailtrap**.



Exercice : Envoyer des emails dans Laravel en utilisant le serveur SMTP de Gmail

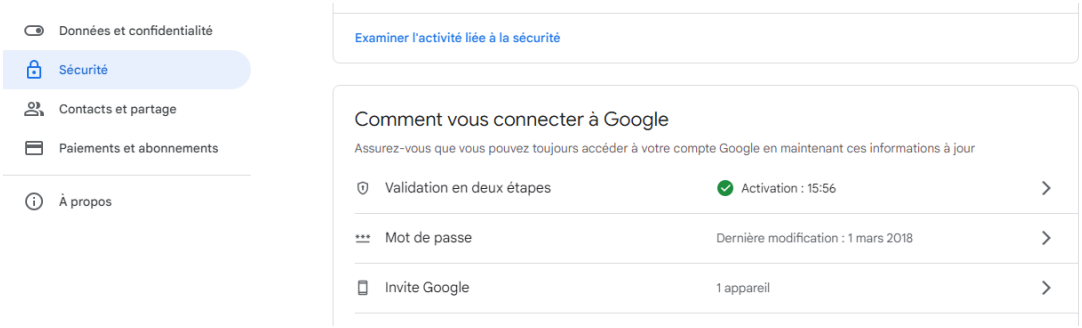
La plupart des gens préfèrent utiliser leur compte Gmail par défaut comme serveur SMTP pour leur application Laravel car il est gratuit. Dans cette partie, je vais vous montrer comment utiliser Gmail comme serveur SMTP pour envoyer vos emails dans Laravel.

Il est bon de noter que Gmail ne vous donne droit qu'à 500 emails par jour pour les comptes gratuits et 2000 emails pour les comptes payants Google Workspace, vous pouvez donc envisager de l'utiliser uniquement comme outil de test et non comme service de production.

Configuration du compte Google

Vous devez d'abord configurer certains paramètres de sécurité dans votre compte Gmail. Pour ce faire, connectez-vous à votre compte Gmail et sélectionnez "Gérer votre compte Google" dans votre profil.

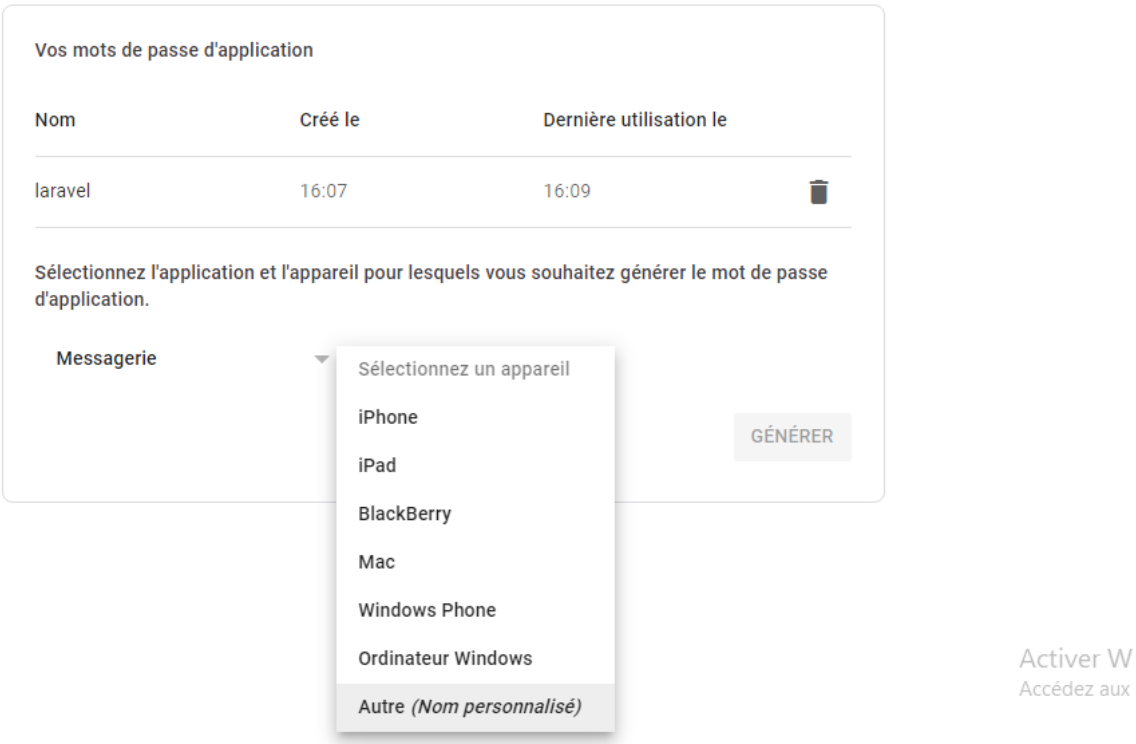
Vous devez ensuite activer **la vérification en deux étapes**



Une fois la vérification en deux étapes activée, vous aurez accès à la section du mot de passe de l'application.

Cliquez sur la section **App passwords** et créez un nouveau mot de passe pour votre application Laravel Mail.

Choisissez Messagerie et autre(Nom personnalisé)



Mise à jour des configurations de messagerie

Une fois que vous avez généré le mot de passe de votre application, vous pouvez mettre à jour vos configurations de messagerie dans le fichier **.env** pour utiliser les configurations SMTP de Gmail.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=youremail@gmail.com
MAIL_PASSWORD=azertyuiop
MAIL_ENCRYPTION=tls
```

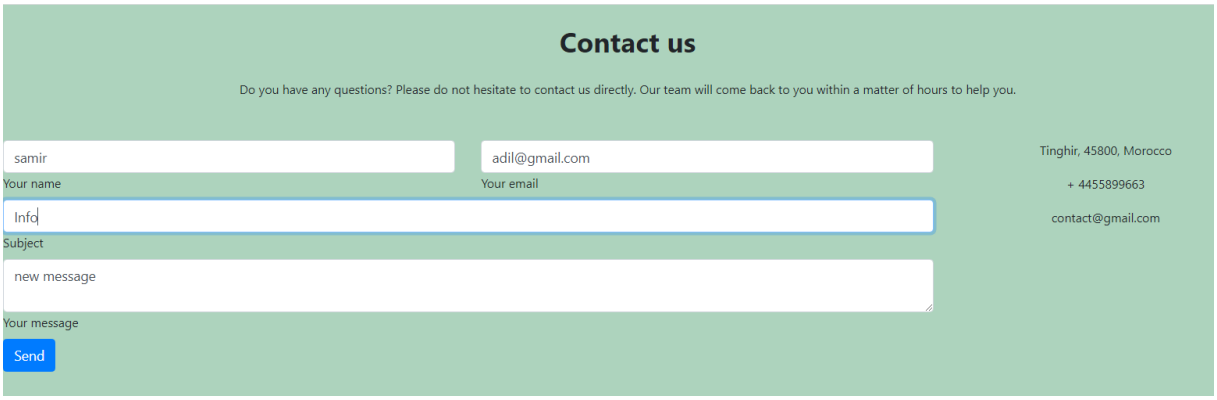
Une fois que vous avez défini les configurations, vous êtes prêt à envoyer des courriels en utilisant Gmail gratuitement. N'oubliez pas que Gmail ne permet d'envoyer que 500 courriels par jour. Si votre application doit envoyer beaucoup de courriels, vous devriez envisager d'autres services.

Nb : N’oubliez pas d’exécuter les commandes pour vider le cache :

```
php artisan config:cache
php artisan config:clear
```

Créer un formulaire de contact

Le formulaire de contact est un système qui permet aux utilisateurs d'un site web d'envoyer des mails aux administrateurs du site et vice versa.



La vue contact-form.blade.php

Observons le code complet du formulaire HTML **resources\views\mail\contact-form.blade.php** avant d'expliquer ses éléments clés. Les classes CSS « .mb-3 », « .form-label », « .form-control », « .is-invalid », ... sont du framework CSS Bootstrap 5 :

```
<!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>Document</title>
    @vite('resources/js/app.jsx')
</head>
<body>
<!--Section: Contact v.2-->
<section class="mb-4 ">

    <!--Section heading-->
```

```
    <h2 class="h1-responsive font-weight-bold text-center my-4">Contact
us</h2>
    <!--Section description-->
    <p class="text-center w-responsive mx-auto mb-5">Do you have any
questions? Please do not hesitate to contact us directly. Our team will come
back to you within
        a matter of hours to help you.</p>

    <div class="row">

        <!--Grid column-->
        <div class="col-md-9 mb-md-0 mb-5 ">
            <form id="contact-form" name="contact-form"
action="{{route('sendMail')}}" method="POST">
                    @csrf
                <!--Grid row-->
                <div class="row">

                    <!--Grid column-->
                    <div class="col-md-6">
                        <div class="md-form mb-0">
                            <input type="text" id="name" name="name"
class="form-control">
                            <label for="name" class="">Your name</label>
                        </div>
                    </div>
                    <!--Grid column-->

                    <!--Grid column-->
                    <div class="col-md-6">
                        <div class="md-form mb-0">
                            <input type="text" id="email" name="email"
class="form-control">
                            <label for="email" class="">Your email</label>
                        </div>
                    </div>
                    <!--Grid column-->

                </div>
                <!--Grid row-->

                <!--Grid row-->
                <div class="row">
                    <div class="col-md-12">
                        <div class="md-form mb-0">
                            <input type="text" id="subject" name="subject"
class="form-control">
                            <label for="subject" class="">Subject</label>
                        </div>
```

```
                    </div>
                </div>
                <!--Grid row-->

                <!--Grid row-->
                <div class="row">

                    <!--Grid column-->
                    <div class="col-md-12">

                        <div class="md-form">
                            <textarea type="text" id="msg" name="msg" rows="2"
class="form-control md-textarea"></textarea>
                            <label for="msg">Your message</label>
                        </div>

                    </div>
                </div>
                <!--Grid row-->
                 <button type="submit" class="btn btn-primary">Send</button>
            </form>

        </div>
        <!--Grid column-->

        <!--Grid column-->
        <div class="col-md-3 text-center">
            <ul class="list-unstyled mb-0">
                <li><i class="fas fa-map-marker-alt fa-2x"></i>
                    <p>Tinghir, 45800, Morocco</p>
                </li>

                <li><i class="fas fa-phone mt-4 fa-2x"></i>
                    <p>+ 4455899663</p>
                </li>

                <li><i class="fas fa-envelope mt-4 fa-2x"></i>
                    <p>contact@gmail.com</p>
                </li>
            </ul>
        </div>
        <!--Grid column-->

    </div>

</section>
<!--Section: Contact v.2-->
</body>
</html>
```

La classe Mailable et la vue du mail

Pour envoyer le mail via le formulaire de contact, nous allons passer par la classe Mailable « **ContactMail** » que nous pouvons créer en exécutant la commande *artisan* suivante :

```
php artisan make:mail ContactMail
```

Cette commande crée le fichier **app\Mail\ContactMail.php**. Créons aussi la vue **resources\views\mail\contact-mail.blade.php** pour la présentation du message.

Dans la classe **ContactMail.php**, on déclare les propriétés *public* `$name`, `$email` et `$msg` qu'on passe via le contructeur, on renvoie ensuite le sujet du message `subject("...")` et la vue **contact-mail.blade.php** via la méthode `content() et envelope()` :

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Mail\Mailables\Address;
use Illuminate\Mail\Mailables\Content;
use Illuminate\Queue\SerializesModels;
use Illuminate\Mail\Mailables\Envelope;
use Illuminate\Mail\Mailables\Attachment;
use Illuminate\Contracts\Queue\ShouldQueue;

class ContactMail extends Mailable
{
    use Queueable, SerializesModels;

    public $name, $email,$subject , $msg;
```

```php
    // On hydrate $name, $email et $msg
    public function __construct($name, $email,$subject , $msg)
    {
        $this->name = $name;
        $this->email = $email;
        $this->subject = $subject;
        $this->msg = $msg;
    }

    public function envelope()
    {
        return new Envelope(

            subject: $this->subject,
        );
    }

    /**
     * Get the message content definition.
     *
     * @return \Illuminate\Mail\Mailables\Content
     */
    public function content()
    {
        return new Content(
            view: 'mail.contact-mail '
        );
    }
}
```

Nous pouvons présenter les attributs `$name`, `$email` et `$msg` dans la vue **resources\views\mail\contact-mail.blade.php** comme ceci :

```php
<div>
    <p><strong>Name :</strong> : {{ $name }}</p>
    <p><strong>Email :</strong> : {{ $email }}</p>
    <p><strong>Message :</strong> :<br>{{ $msg }}</p>
```

```
</div>
```

La publication envoyer un mail via le serveur SMTP Google explique comment configurer le driver SMTP pour envoyer des mails dans une application Laravel.

Envoyer le mail

Dans le contrôleur **MailController App\Http\Controllers\MailController.php** :

```php
<?php

namespace App\Http\Controllers;

use App\Mail\MailableName;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Mail;


class MailController extends Controller
{
    public function sendMail(Request $request){

        //The email sending is done using the to method on the Mail facade
            Mail::to('aminefste@gmail.com')->send(new ContactMail($request->name,$request->email, $request->subject, $request->msg));
    }
    public function contact(){
     return view('mail.contact-form');
    }

}
```

Le code ci-dessus s'explique comme suit :

Si la validation réussit, on envoie le mail à une adresse e-mail :

- l'adresse de destinataire (to),
- l'envoi avec la méthode send,
- la création d'une instance de la classe ContactMail avec la transmission des données saisies.

Et si tout se passe bien le message doit arriver jusqu'à le destinataire :

Test de l’email

Quand on crée la vue pour l’email il est intéressant de voir l’aspect final avant de faire un envoi réel. Pour le faire il suffit de créer une simple route :

```
Route::get('/contact',[MailController::class,'contact']);
Route::post('/contact',[MailController::class,'sendMail'])
->name('sendMail');
```

Maintenant en utilisant l’url **localhost:8000/contact** on obtient l’aperçu directement dans le navigateur !