

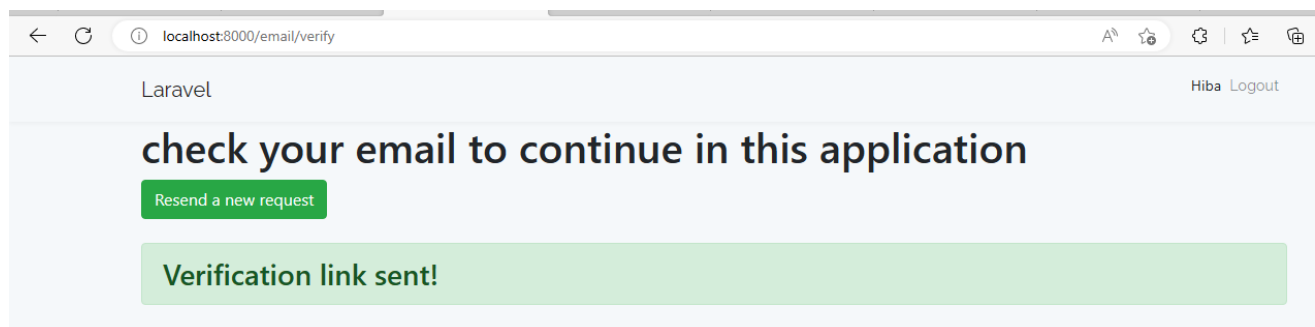
Laravel : Vérification personnalisée par courriel avec code d'activation (Email verification)

Laravel fournit un système de vérification d'email par défaut dans son système d'authentification. Nous pouvons l'utiliser pour créer un système de vérification d'email avant la confirmation d'enregistrement.

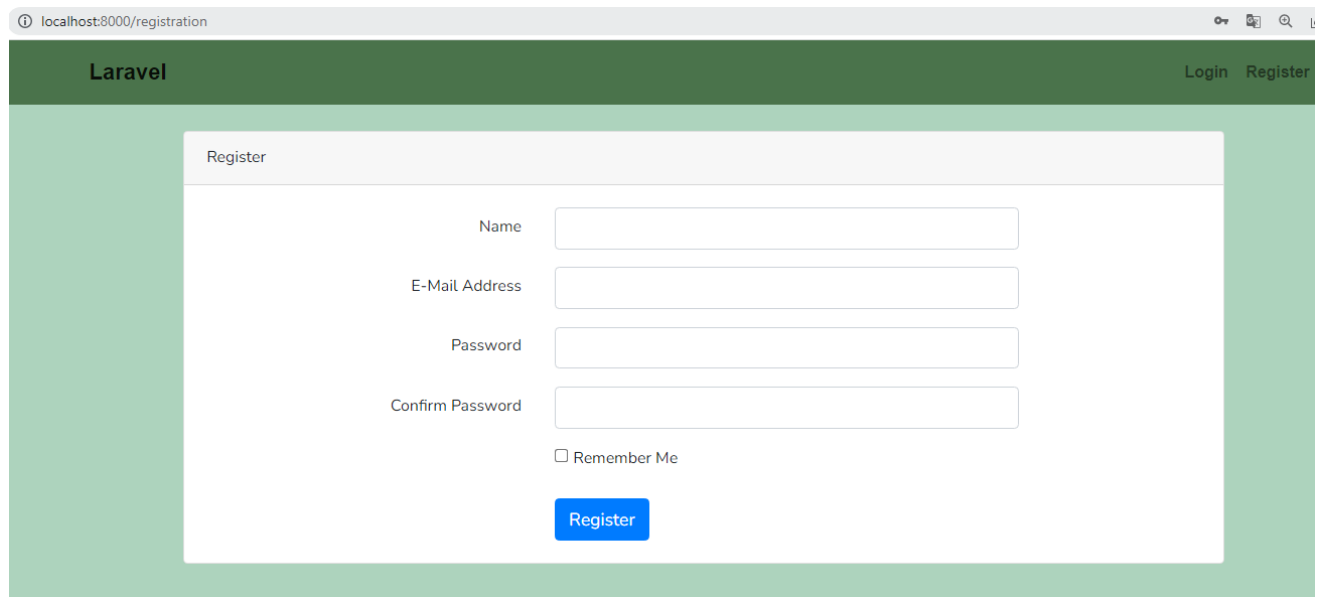
Mais dans cet exemple, je vais créer un modèle de vérification d'email personnalisé laravel. Ainsi, vous apprendrez la vérification d'email personnalisée de Laravel à partir de ce TP avec des conseils étape par étape.

Voir l'aperçu des images ci-dessous :

Email verification notify page:



Register Page:



localhost:8000/registration

Laravel Login Register

Register

Name

E-Mail Address

Password

Confirm Password

☐ Remember Me

Register

Modèle d'email avec code d'activation:

Verify your adress email

From: Mail for you <aminefste@gmail.com>
To: <user10@gmail.com>

[Show Headers](#)

HTML

HTML Source

Text

Raw

Spam Analysis

HTML Check 2

Tech Info



Click in this link below to Verify your adress email :

[link](#)

Etape 1: S'inscrire à mailtrap

Enregistrez-vous sur **mailtrap** pour envoyer des emails de test. Et obtenez les informations d'identification dans la boîte de réception de démonstration.

My Inbox

SMTP
Settings

Email
Address

Auto
Forward

Integrations ?

Laravel 7+



Laravel provides a clean, simple API over the popular SwiftMailer.

With the default Laravel setup you can configure your mailer in the `.env` file in the root directory of your project.

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

Ensuite, nous devons ajouter la configuration de **Mailer** dans le fichier `.env`.

`.env`

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

Exécutons ensuite les commandes suivantes :

- Arrêtez le serveur `ctrl+c`
- `php artisan config:cache`
- `php artisan serve`

Etape 2 : Créer une table de migration et le modèle UserVerify

Exécutez la commande :

```
Php artisan make :model UserVerify -m
```

Nous allons créer une table **users_verify**:

// **App\Models\UserVerify.php**

```
namespace App\Models;

use App\Models\User;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class UserVerify extends Model
{
    use HasFactory;
    protected $table = "users_verify";
    protected $fillable = [
        'user_id',
        'token',
    ];
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

// **database/migrations/xxxx_xx_xx_xxx_create_user_verifies_table.php**

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
```

```

{
    Schema::create('users_verify', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('user_id');
        $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
        $table->string('token');
        $table->timestamps();
    });
}

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

```

Etape 3 : Configurer le modèle User

// **App\Models\User.php**

```

<?php

namespace App\Models;

use App\Models\Car;
use App\Models\UserVerify;
use Illuminate\Notifications\Notifiable;
use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    public function cars(){
        return $this->hasMany(Car::class);
    }

    public function userverify(){
        return $this->hasOne(UserVerify::class);
    }
}

```

```

protected $fillable = [
    'name',
    'email',
    'password',
];

protected $hidden = [
    'password',
    'remember_token',
];

protected $casts = [
    'email_verified_at' => 'datetime',
];
}

```

Etape 3 : Créer des routes

Dans cette étape, nous devons créer une route personnalisée pour la vérification de l'email. Ouvrez donc votre fichier **routes/web.php** et ajoutez la route suivante.

//routes/web.php

```

//Routes Auth

Route::group(['middleware'=>'guest'],function(){
Route::get('/',function(){
    return redirect()->route('login');
});
Route::get('login', [AuthController::class, 'index']->name('login'));
Route::post('post-login', [AuthController::class, 'postLogin']->name('login.post'));
Route::get('registration', [AuthController::class, 'registration']->name('register'));
Route::post('post-registration', [AuthController::class, 'postRegistration']->name('register.post'));

});
Route::get('logout', [AuthController::class, 'logout']->name('logout')->middleware('auth'));

```

```
//Routes for email verification
Route::get('account/verify/{token}', [AuthController::class, 'verifyAccount'])->
>middleware('auth')->name('user.verify');
Route::get('/email/verify', function () {
    return view('auth.verify-email');
})->middleware('auth')->name('verification.notice');
Route::post('/email/verification-notification',[AuthController::class,
'sendAnotherVerification'])
->middleware('auth')->name('sendAnotherVerification');
//Route Car
Route::middleware(['auth','verifiedEmail'])->group(function () {
    Route::resource('cars', CarController::class);
});
```

Etape 3: Ajouter deux actions au contrôleur AuthController

Dans cette étape, nous devons mettre à jour le code des méthodes **postRegistration()** et ajoutons les actions **verifyAccount(\$token)** et **sendAnotherVerification()**. Copions-les comme suit :

app/Http/Controllers/Auth/AuthController.php

```
<?php

namespace App\Http\Controllers;

use Carbon\Carbon;
use App\Models\User;
use App\Models\UserVerify;
use Illuminate\Support\Str;
use Illuminate\Http\Request;
use App\Mail\SendVerificationMail;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Mail;
use Illuminate\Auth\Events\Registered;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Facades\Validator;

class AuthController extends Controller
{
    public function index()
    {
```

```

        return view('auth.login');
    }

    public function registration()
    {
        return view('auth.registration');
    }

    public function postLogin(Request $request)
    {
        $rules = [
            'email' => ['required', 'email'],
            'password' => ['required', 'min:6'],
        ];
        $messages=[
            'email.email'=>'email not valid'
        ];
        // dd($request->has('remember')); return false if unchecked
        $validateForm=Validator::make($request->all(),$rules,$messages);
        if($validateForm->fails())
            return redirect()->back()->withErrors($validateForm)->onlyInput('email');

        if (Auth::attempt(array('email'=>$request->email,'password'=>$request->password),$request->has('remember'))) {
            $request->session()->regenerate();

            return redirect()->intended('cars');
        }

        return redirect("login")->withSuccess('Oppes! You have entered invalid credentials')->onlyInput('email');
    }

    public function postRegistration(Request $request)
    {
        $rules=[
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'password' => 'required|min:6',
            'confirmPassword' => 'required|same:password',
        ];

        $validateData=Validator::make($request->all(),$rules);
        if($validateData->fails())
            return redirect()->back()->withErrors($validateData)->withInput();
        $data=[

```



```

        "name"=>$request->name,
        "password"=>$request->password,
        "email"=>$request->email,
    ];
    $user=$this->create($data);
    // dd($user);
    // event(new Registered($user));
    $token=Str::random(50);
    UserVerify::create([
        'user_id'=>$user->id,
        'token'=>$token
    ]);
    Mail::to($request->email)->queue(new SendVerificationMail($token));
    Auth::attempt(array('email'=>$data['email'],'password'=>$data['password']));
    return redirect("/cars")->withSuccess('Great! You have Successfully
loggedin');
}

public function create(array $data)
{

    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password'])
    ]);

}

public function logout() {

    Session::flush();

    Auth::logout();

    return Redirect('login');
}
public function verifyAccount($token){

    $userVerify=Auth::user()->userverify()->first();

    if($userVerify->token==$token)
    {
        $user=User::find(Auth::user()->id);
        $user->email_verified_at=Carbon::now();
        $user->save();
        return redirect()->route('cars.index');
    }
}

```

```

        else
            return redirect()->route('verification.notice');

    }

    public function sendAnotherVerification(){

        $userVerify=Auth::user()->userverify()->first();
        Mail::to(Auth::user()->email)->queue(new SendVerificationMail($userVerify->token));
        return back()->with('message', 'Verification link sent!');
    }
}

```

Etape 4: Créer les vues en Blade

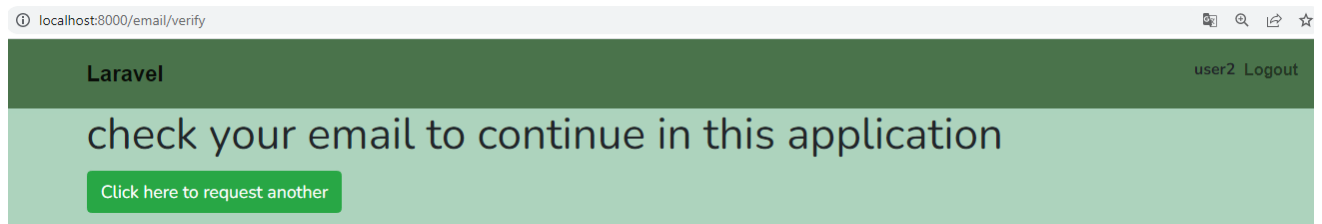
Ici, nous devons créer une vue de blade pour `verify-email`. Une vue demandant à l'utilisateur de cliquer sur le lien de vérification de l'adresse électronique qui lui a été envoyé par Laravel après son inscription. Cette vue sera affichée aux utilisateurs lorsqu'ils essaieront d'accéder à d'autres parties de l'application sans avoir vérifié leur adresse électronique au préalable :

resources/views/auth/verify-email.blade.php

```

@extends('layouts.layout')
@section('content')
    <div class="container">
        <h1>check your email to continue in this application</h1>
        <form action="{{ route('sendAnotherVerification') }}" method="POST">
            @csrf
            @method('POST')
            <button class="btn btn-success" type="submit">Click here to request
another</button>
        </form> <br>
        @if (Session::has('message'))
            <h3 class="alert alert-success"> {{ Session::get('message') }} </h3>
        @endif
    </div>
@endsection

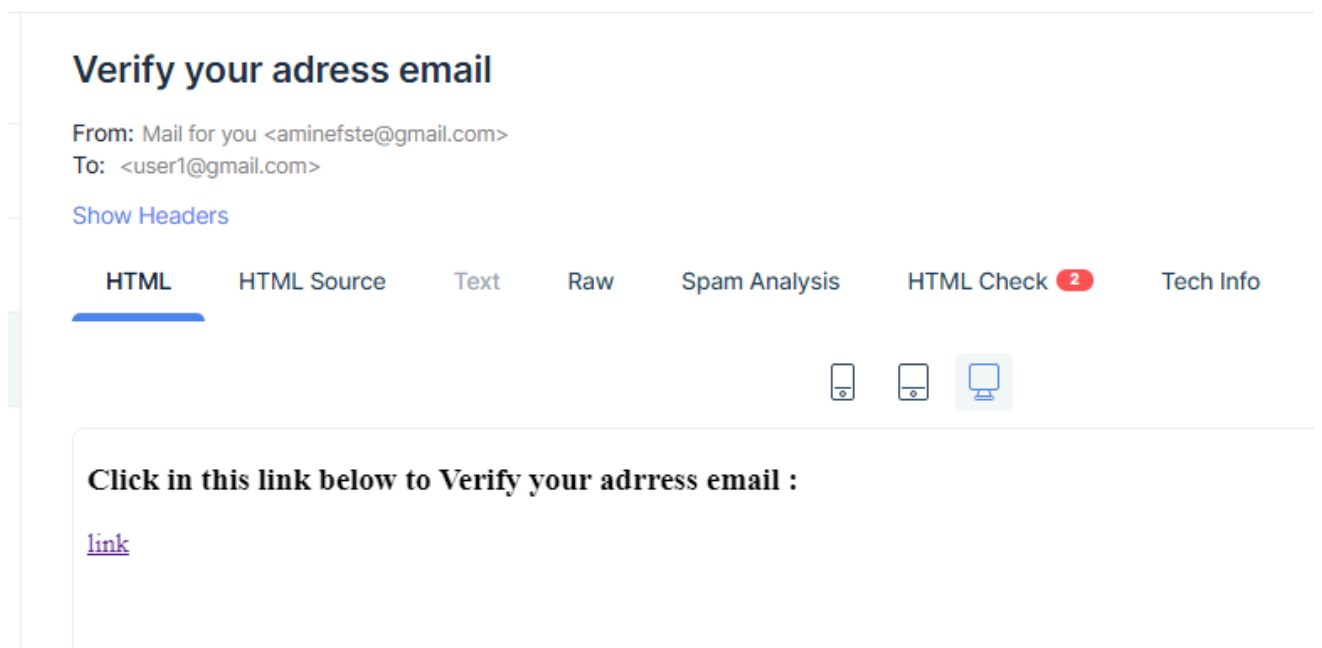
```



Ici, nous devons créer des fichiers de blade pour l'email uniquement (la vue qui s'affiche dans la boîte email).

resources/views/mail/verifymailview.blade.php

```
<!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>
</head>
<body>
  <h3>Click in this link below to Verify your adress email :</h3>
  <a href="{{URL::temporarySignedRoute('user.verify',now()-
>addMinutes(30),['token' => $token])}}">link
  </a>
</body>
</html>
```



Etape 5 : Création d'un Middleware pour vérifier les emails

Ici, nous devons créer un middleware pour vérifier si l'email de l'utilisateur est vérifié. Nous allons donc le créer comme ci-dessous :

```
php artisan make:middleware VerifiedEmailMiddleware
```

app/Http/Middleware/ VerifiedEmailMiddleware.php

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class VerifiedEmailMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        if(Auth::user()->email_verified_at)
            return $next($request);
        else
            return redirect()->route('verification.notice');
    }
}
```

Puis enregistrer le middleware :

//app/Http/Kernel.php

```
protected $routeMiddleware = [

    ....

    'verifiedEmail' =>\App\Http\Middleware\VerifiedEmailMiddleware::class,

];
```

Etape 6 : Création de 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 SendVerificationMail
```

//app/Mail/SendVerificationMail.php

```
<?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;

class SendVerificationMail extends Mailable
{
    use Queueable, SerializesModels;

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

    /**
     * Get the message envelope.
     *
     * @return \Illuminate\Mail\Mailables\Envelope
     */
    public function envelope()
    {
        return new Envelope(
            subject: 'Verify your adress email',

```

```
    );  
}  
  
/**  
 * Get the message content definition.  
 *  
 * @return \Illuminate\Mail\Mailables\Content  
 */  
public function content()  
{  
    return new Content(  
        view: 'mail.verifyemailview',  
    );  
}  
  
/**  
 * Get the attachments for the message.  
 *  
 * @return array  
 */  
public function attachments()  
{  
    return [];  
}  
}
```