



## **LARAVEL FRAMEWORK**

### **BÀI 7: VALIDATION & SEND MAIL**

#### **PHẦN 1: VALIDATION**

- ◎ Sử dụng sử dụng Validation trong Laravel
- ◎ Gửi mail trong Laravel





## Phần I: Validation



Tổng quan Validation



Kiểm tra với hàm validate



Hiện thông báo lỗi



Tùy chỉnh các chuỗi giá trị báo lỗi



Hiện lỗi cho từng theo field



Tạo request class



Định nghĩa rules và authorize



Định nghĩa báo lỗi thân thiện



Dùng request class trong controller



Tạo rule kiểm tra mới trong Laravel



Sử dụng rule mới tạo



Test rule mới tạo



## Phần II: Gửi Mail



Send mail trong laravel



Cấu hình sử dụng mailgun



Tạo view chứa nội dung mail



Tạo mail mailable class



Tạo view chứa nội dung mail



Thực hiện gửi mail



Truyền data vào nội dung mail



Các cách gửi mail khác



- ❑ Validation là hoạt động xác định sự chính xác ý nghĩa của dữ liệu đầu vào. Thường dữ liệu đến từ các form.
- ❑ Khi thực hiện validate, bạn đưa ra các quy tắc để kiểm tra dữ liệu có đúng không. Ví dụ: điểm phải là số từ 1 đến 10, email phải đúng dạng, cmdn phải có 9 hoặc 10 ký số ...
- ❑ Laravel hỗ trợ nhiều cách để kiểm tra dữ liệu đầu vào và hiện các thông báo lỗi cho user...

**Whoops! Please correct errors and try again!**

- The first name field is required.
- The last name field is required.
- The email field is required.
- The mobilenno field is required.
- The password field is required.
- The confirm password field is required.

<p><b>First Name:</b></p> <input type="text" value="Enter First Name"/> <p>The first name field is required.</p>	<p><b>Last Name:</b></p> <input type="text" value="Enter Last Name"/> <p>The last name field is required.</p>
<p><b>Email:</b></p> <input type="text" value="Enter Email"/> <p>The email field is required.</p>	<p><b>Phone No:</b></p> <input type="text" value="Enter Mobile No"/>
<p><b>Password:</b></p> <input type="password" value="Enter Password"/> <p>The password field is required.</p>	<p><b>Confirm Password:</b></p> <input type="password" value="Enter Confirm Passowrd"/> <p>The confirm password field is required.</p>

- ❑ Giả sử có 2 routes và controller như sau

```
use App\Http\Controllers\SvController;  
Route::get("/sv",[SvController::class,'sv']);  
Route::post("/sv",[SvController::class,'sv_store'])  
->name('sv_store');
```

- ❑ Controllers/SvController.php có 2 action

```
<?php  
namespace App\Http\Controllers;  
use Illuminate\Http\Request;  
class SvController extends Controller {  
    public function sv(){  
        return view("formnhapsv");  
    }  
    function sv_store(Request $request){  
  
    }  
}
```

## ❑ views/formnhapsv.blade.php

```
<link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css">
<form method="post" class="p-3 border border-primary col-6 m-auto"
    action="{{route('sv_store')}}" > @csrf
<div class="mb-3">
    <label>Họ tên</label><input value="{{old('ht')}}" class="form-control"
name="ht">
</div>
<div class="mb-3">
    <label>Tuổi</label><input value="{{old('tuoi')}}" class="form-control"
name="tuoi">
</div>
<div class="mb-3">
    <label>Ngàysinh</label><input value="{{old('ns')}}" class="form-control"
name="ns">
</div>
<div class="mb-3">
    <label>CMND</label><input value="{{old('cmnd')}}" class="form-control"
name="cmnd">
</div>
<div class="mb-3">
    <label>Email</label> <input value="{{old('em')}}" class="form-control"
name="em">
</div>
<div class="mb-3">
    <button type="submit" class="btn btn-primary w-25">Xử lý</button>
</div>
</form>
```

❑ Chạy project và xem thử <http://localhost:8000/sv>

Họ tên

Tuổi

Ngàysinh

CMND

Email

Xử lý

Vấn đề: Trước khi lưu vào db, cần kiểm tra dữ liệu trong form xem có đúng các quy tắc:

- ❑ **Họ tên:** phải nhập, dài 2 đến 30 ký tự
- ❑ **Tuổi:** phải nhập số nguyên từ 16 đến 80
- ❑ **Ngày sinh:** phải nhập kiểu dd/mm/yyyy
- ❑ **CMND:** phải nhập từ 9 đến 10 ký số
- ❑ **Email:** phải nhập dạng email, tận cùng là @fpt.edu.vn

Để kiểm tra dữ liệu như thế , chúng ta sẽ sử dụng chức năng validation trong Laravel

- ❑ Để kiểm tra, dùng method **validate()** của đối tượng \$request.
- ❑ Các rules viết trong **validate()** để kiểm tra dữ liệu. Mỗi rule có key là name của form field, còn value là 1 string hoặc array.
- ❑ Nếu **validate()** kiểm tra fail thì 1 exception được tạo ra và báo lỗi sẽ trả về cho user, nếu pass thì code vẫn tiếp tục chạy

```
public function sv_store(Request $request) {  
    $request->validate([  
        'hoten' => ['required', 'min:3', 'max:20'],  
        'tuoi' => 'required|integer|min:16|max:100',  
        'ngaysinh' => ['required', 'date'],  
        'cmnd' => 'digits_between:9,10',  
        'email' => 'email|ends_with:@fpt.edu.vn'  
    ]  
    );  
    //echo "Code xử lý";  
}
```

- ❑ Muốn dừng kiểm tra ngay khi gặp lỗi thì dùng **bail** trong rule:

```
$request->validate([  
    'tuoi' => 'bail|required|integer|min:16|max:100',  
]);
```



- ❑ Khi có lỗi, Laravel lưu các thông báo lỗi vào mảng **\$errors**. và redirect trở lại trang web gốc
- ❑ Trong ví dụ trên, Laravel sẽ redirect trở lại route sv khi validation fail. Vì vậy bạn có thể cho hiện lỗi trong view formnhapsv

```
@if ($errors->any())  
<div class="alert alert-danger">  
    <ul>  
        @foreach ($errors->all() as $error)  
            <li>{{ $error }}</li>  
        @endforeach  
    </ul>  
</div>  
@endif
```

- The hoten field is required.
- The tuoi field is required.
- The ngaysinh field is required.
- The cmdn must be between 9 and 10 digits.

Họ tên

Tuổi

- ❑ Các chuỗi báo lỗi được Laravel định nghĩa trong file **lang/en/validation.php**. Có thể chỉnh các giá trị nếu muốn

```
return [  
    'accepted' => 'The :attribute must be accepted.',  
    'accepted_if' => 'The :attribute must be accepted when :other is :val',  
    'active_url' => 'The :attribute is not a valid URL.',  
    'after' => 'The :attribute must be a date after :date.',  
    'after_or_equal' => 'The :attribute must be a date after or equal to',  
    'alpha' => 'The :attribute must only contain letters.',  
    'alpha_dash' => 'The :attribute must only contain letters, numbers, d',  
    'alpha_num' => 'The :attribute must only contain letters and numbers.',  
    'array' => 'The :attribute must be an array.',  
    'before' => 'The :attribute must be a date before :date.',  
    'before_or_equal' => 'The :attribute must be a date before or equal t',  
    'between' => [  
        'numeric' => 'The :attribute must be between :min and :max.',  
        'file' => 'The :attribute must be between :min and :max kilobytes',  
        'string' => 'The :attribute must be between :min and :max charact'
```

- ❑ Trong blade view, dùng hàm **@error()** để check lỗi cho từng field và hiện lỗi với biến **\$message** :

```
<label>Họ tên</label>
<input value="{{old('ht')}}" class="form-control"
name="ht">
    @error('hoten')
        {{ $message }}
    @enderror
```

- ❑ Lưu ý:
  - Thường hiện lỗi cho 1 field sẽ hiện ngay tại vị trí của field
  - Hàm old trong blade giúp hiện lại giá trị trong field mà user đã nhập.

- ❑ Request class dùng để chỉnh hoạt động kiểm tra, báo lỗi của Laravel
- ❑ Để tạo request class, sử dụng lệnh make:request của artisan :  
**php artisan make:request RuleNhapSV**
- ❑ Lệnh trên sẽ tạo class trong folder app/Http/Requests. Trong class có sẵn 2 hàm **authorize()** và **rules()**.
- ❑ Hàm **authorize()** dùng để cho/cấm request class (return true/false)
- ❑ Hàm **rules()**: nơi khai báo các quy tắc kiểm tra dữ liệu.
- ❑ Ngoài ra có thể định nghĩa 2 hàm nữa là **message()** và **attributes()**

```
class RuleNhapSV extends FormRequest {  
    public function authorize(){  
        return false;  
    }  
    public function rules() {  
        return [  
            //  
        ];  
    }  
}
```

- ❑ Trong hàm rules, khai báo các quy tắc kiểm tra:

```
public function rules() {  
    return [  
        'ht' => ['required', 'min:3', 'max:20'],  
        'tuoi' => 'required|integer|min:16|max:100',  
        'ns' => ['required', 'regex:/\d{1,2}\d{1,2}\d{4}/'],  
        'cmd' => 'digits_between:9,10',  
        'em' => 'email|ends_with:@fpt.edu.vn'  
    ];  
}
```

- ❑ Hàm authorize() trả về true/false sẽ cho/cấm request

```
public function authorize() { return true; }
```

- ❑ Hàm messages() để định nghĩa lỗi thân thiện với người dùng

```
public function messages() {  
    return [  
        'ht.required' => 'Phải nhập họ tên chứ',  
        'ht.min' => 'Họ tên ngắn quá vậy',  
        'ns.required' => 'Nhập ngày sinh nữa',  
        'tuoi.required' => 'Nhập :attribute vào đi',  
        'cmd.digits_between' => 'CMND nhập 9 hoặc 10 ký tự'  
    ];  
}
```

- ❑ B1. Ở đầu controller: **use App\Http\Requests\RuleNhapSV;**
- ❑ B2: Trong action, chỉ định tham số có kiểu là request class đã tạo:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests\RuleNhapSV;
class SvController extends Controller {
    public function sv(){ return view("formnhapsv"); }
    public function sv_store(RuleNhapSV $request) {
        // "Code xử lý";
    }
}
```

- ❑ Submit form, sẽ thấy kết quả

- Phải nhập họ tên chứ
- Nhập Tuổi vào đi
- Nhập ngày sinh nữa
- Số chứng minh nhân dân nhập 9 hoặc 10 ký tự
- The em must be a valid email address.
- The em must end with one of the following: @fpt.edu.vn.

Họ tên

Tuổi

- ❑ Sử dụng lệnh `make:rule` của artisan để tạo rule mới trong `php artisan make:rule Chuhoa`
- ❑ File rule mới trong `App\Rules\chuhua.php`. Trong đó có sẵn 2 hàm **`passes()`** - trả về true/false và **`message()`** – trả về chuỗi .
- ❑ Hàm **`passes()`** là nơi viết code định nghĩa dữ liệu như thế nào là pass. Hàm **`message()`** là thông báo khi dữ liệu không pass

```
<?php
namespace App\Rules;
use Illuminate\Contracts\Validation\Rule;
class Chuhoa implements Rule {
    public function __construct(){ }
    public function passes($attribute, $value){
        return strtoupper($value) === $value;
    }
    public function message() {
        return 'Thuộc tính :attribute phải là chữ hoa!';
    }
}
```

- ❑ Dùng lệnh **use App\Rules\Chuhua;**
- ❑ Tại vị trí khai báo các rule (trong controller hoặc mail rule), tạo instance cho rule bằng từ khóa new

```
use Illuminate\Foundation\Http\FormRequest;

use App\Rules\Chuhua;

class RuleNhapSV extends FormRequest {
    public function authorize(){ return true; }
    public function rules() {
        return [
            'ht' => ['required','min:3','max:20', new Chuhua],
            'tuoi' => 'required|integer|min:16|max:100',
        ];
    }
}
```

- Thuộc tính Họ tên phải là chữ hoa bạn ơi!
- Nhập Tuổi vào đi
- Nhập ngày sinh nữa
- Số chứng minh nhân dân nhập 9 hoặc 10 ký tự
- The em must be a valid email address.
- The em must end with one of the following: @fpt.edu.vn.

Họ tên

asd

Tuổi





DEMO

Demo xác thực dữ liệu form login





## **LARAVEL FRAMEWORK**

### **BÀI 7: VALIDATION & SEND MAIL**

#### **PHẦN 1: SEND MAIL**

- ❑ Trong Laravel có thể gửi mail theo nhiều cách (driver) khác nhau như SMTP, Mailgun, Postmark, Amazon SES ...
- ❑ Có thể gửi các transaction mail (mail giao dịch) như Postmark hoặc các mail với số lượng lớn (marketing, thông báo khách hàng) như Amazon SES...
- ❑ Các driver dựa trên API như Mailgun , Postmark thường đơn giản và nhanh hơn SMTP server, nên dùng các driver này.
- ❑ Sau đây là các bước hướng dẫn sử dụng mailgun, bao gồm
  1. Đăng ký mailgun và lấy domain, api key
  2. Cấu hình sử dụng mailgun
  3. Cài gói Symfony Mailer
  4. Tạo view chứa nội dung mail
  5. Tạo mailable class
  6. Thực hiện gửi mail
  7. Truyền data vào nội dung mail

Để dùng driver Mailgun:

1. Đăng ký tài khoản trên <https://mailgun.com> . Xong vào Sending ➔ Domains ➔ API key để lấy Private API key

mailgun by sinch

Sending

Upgrade Feedback ? N

Statistics Beta

Sending

Overview

Domains

Logs

Analytics

Domains

Search all domains

Name

Accepted(Mar) Delivered(Mar) Bounced(Mar) Opened(Mar) Clicked

★ 🇺🇸 ✓ sandbox88a20d49299e42a58067feb3fe... 9

33.33% 0.00% 0.00% 0.00%

Copy domain để khai báo trong .env

API security

API keys

Private API key

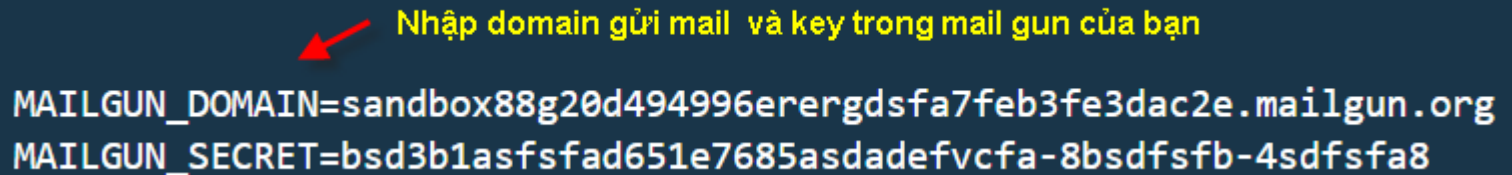
\*\*\*\*\*a70a8

Public verification key

Copy key để khai báo trong .env

2. Cấu hình sử dụng mailgun gồm 2 thông số cần khai báo: domain và private key:

Mở file .env khai báo domain gửi mail và Private API key



Nhập domain gửi mail và key trong mail gun của bạn

```
MAILGUN_DOMAIN=sandbox88g20d494996erergdsfa7feb3fe3dac2e.mailgun.org  
MAILGUN_SECRET=bsd3b1asfsfad651e7685asdadevcfa-8bsdfsfb-4sdfsfa8
```

3. Cài gói Symfony Mailer

**composer require symfony/mailgun-mailer symfony/http-client**

## 4. Tạo view chứa nội dung mail

- ❑ Tạo views/guimail.blade.php và soạn nội dung cho mail:

```
<b>Chào bạn</b>  
<p>  
    <i>Khỏe không bạn? Chúc an lành!  
    Chúc thành công</i>  
</p>
```

- ❑ Trong view nội dung mail, bạn có thể hiện giá trị các biến nếu khi nạp view có truyền data vào

5. Mailable class là 1 template tạo ra để khai báo thông số cho mail. Lưu trong app/Mail. Tạo 1 mailable class như sau

```
php artisan make:mail GuiEmail
```

File mới tạo có hàm **build**, nơi đây bạn gọi method view() để nạp view nội dung mail, from(), to(), subject(), attach()...

```
public function build() {  
    return $this  
        ->from("diachin@guoigui.com", "Tên người gửi")  
        ->to("diachi@nguoinhan.com")  
        ->subject(("Tiêu đề thư"))  
        -> attach(public_path() ."/hinh1.jpg") //đính kèm file  
        ->view('guimail'); // nạp view nội dung mail  
}
```

## 6. Thực hiện gửi mail: gọi hàm send

- ❑ Trong routes/web.php, sử dụng driver mailgun và gọi hàm send để thực hiện gửi mail.

```
use App\Mail\GuiEmail;  
Route::get("/guimail", function(){  
    Mail::mailer('mailgun')->send(new GuiEmail());  
});
```

- ❑ Bạn có thể gọi hàm send như như trên từ trong action của controller, nếu muốn



3. Truyền data: Có thể truyền data vào mailable sử dụng. Mailable lưu data vào biến public và view nội dung mail có thể truy xuất

```
use App\Mail\GuiEmail;
Route::get("/guimail", function(){
    $diem = ["toan"=>9, "van"=>7];
    Mail::mailer('mailgun')->send(new GuiEmail($diem));
});
```

Truyền data cho mailable

```
class GuiEmail extends Mailable {
    use Queueable, SerializesModels;
    public $diem = null;
    public function __construct($data) {
        $this->diem = $data;
    }
    public function build() {
        return $this
            ->from("postmaster@sandbox88a20d49299e42a58067feb3fed9ac2e.mailgun.org", "Long")
            ->to("longnv@gmail.com")
            ->subject("Thư test")
            ->attach(public_path() . "/hinh1.jpg") //đính kèm file
            ->view('guimail'); // nạp view nội dung mail
    }
}
```

- ❑ Trong views/GuiMail.blade.php , bạn có thể cho hiện các biến truyền vào

```
<p><b>Chào bạn! </b> <i>Sau đây là điểm của bạn: </i></p>  
<p>Điểm Toán : {{$diem['toan']}}</p>  
<p>Điểm Văn : {{$diem['van']}}</p>
```

- ❑ Ngoài mailgun, Laravel còn hỗ trợ nhiều cách gửi mail khác như smtp, postmark...
- ❑ Bạn có thể tham khảo thêm tại đây  
<https://laravel.com/docs/8.x/mail#sending-mail>



# DEMO

-Demo xác thực số điện thoại  
VietNam bắt đầu "+84"





## Phần I: Validation



Tổng quan Validation



Kiểm tra với hàm validate



Hiện thông báo lỗi



Tùy chỉnh các chuỗi giá trị báo lỗi



Hiện lỗi cho từng theo field



Tạo request class



Định nghĩa rules và authorize



Định nghĩa báo lỗi thân thiện



Dùng request class trong controller



Tạo rule kiểm tra mới trong Laravel



Sử dụng rule mới tạo



Test rule mới tạo



## Phần II: Gửi Mail



Send mail trong laravel



Cấu hình sử dụng mailgun



Tạo view chứa nội dung mail



Tạo mail mailable class



Tạo view chứa nội dung mail



Thực hiện gửi mail



Truyền data vào nội dung mail



Các cách gửi mail khác





**Cảm ơn**