Laravel cung cấp các phương pháp khác nhau để xác nhận dữ liệu gửi đến ứng dụng . Mặc định , Base Controller class của Laravel sử dụng một ValidatesRequests trail cung cấp một phương pháp thuận lợi để xác nhận yêu cầu HTTP đến với nhiều rule xác nhận mạnh mẽ .

Validation Quick Start

Để tìm hiểu về các tính năng xác nhận mạnh mẽ của Laravel, chúng ta hãy xem xét một ví dụ hoàn chỉnh về validate form và hiển thị các thông báo lỗi lại cho người dùng.

Defining The Routes

Đầu tiên, giả sử ta có route đã được định nghĩa trong file app/Http/routes.php

// Display a form to create a blog post...

Route::get('post/create', 'PostController@create');

// Store a new blog post...

Route::post('post', 'PostController@store');

Tất nhiên, route GET sẽ hiển thị form cho người dùng tạo ra một bài blog mới, trong khi các route POST sẽ lưu trữ các bài blog mới này trong cơ sở dữ liệu.

### Creating The Controller

Tiếp theo, chúng ta hãy nhìn vào một controller đơn giản để xử lý các route. Và giờ chúng ta sẽ tạo ra các phương thức store trống:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Controllers\Controller;

class PostController extends Controller

{

/\*\*

\* Show the form to create a new blog post.

\*

\* @return Response

\*/

public function create()

{

return view('post.create');

}

/\*\*

\* Store a new blog post.

\*

\* @param Request $request

\* @return Response

\*/

public function store(Request $request)

{

// Validate and store the blog post...

}

}

### Writing The Validation Logic

Bây giờ chúng ta viết phương thức store để validate các bài blog mới. Nếu bạn kiểm tra basecontroller class của ứng dụng (App \ Http \ Controller \ Controller) , bạn sẽ thấy rằng class sử dụng một ValidataRequest trait. Trait này cung cấp một phương thức validate thuận tiện cho mọi controller.

Phương thức validate nhận một yêu cầu HTTP đến và tập hợp các validation rules. Nếu các validation rule được thông qua, code của bạn sẽ chạy bình thường; Tuy nhiên, nếu validate thất bại, một ngoại lệ được tạo ra và phản ứng lỗi thích hợp sẽ tự động được gửi lại cho người dùng. Trường hợp nếu là một HTTP request truyền thống, một phản hồi chuyển hướng sẽ được tạo ra, trong khi một phản hồi JSON sẽ được gửi cho các AJAX request.

public function store(Request $request)

{

$this->validate($request, [

'title' => 'required|unique:posts|max:255',

'body' => 'required',

]);

// The blog post is valid, store in database...

}

Ta chỉ cần pass HTTP request đến và các validation rule. Nếu validate thất bại, một phản hồi tự động tạo ra. Còn nếu thành công, code chạy bình thường

**A Note On Nested Attributes**

Nếu HTTP request chứa các tham số lồng,bạn có thể chỉ định chúng trong các rule bằng việc sử dụng dấu “.” :

$this->validate($request, [

'title' => 'required|unique:posts|max:255',

'author.name' => 'required',

'author.description' => 'required',

]);

### Displaying The Validation Errors

Những request không pass các rule, laravel chuyển hướng người dùng đến trang trước đó, mọi error sẽ được flash vào session.

Các lỗi không được ràng buộc để hiển thị lên route GET. Vì laravel sẽ kiểm tra errortrong dữ liệu session và tự động liên kết đến view nếu có sẵn. Các biến $errors là một instance của Illuminate\Support\MessageBag.

Note: Các biến $error được gắn với view bằng Illuminate\View\Middleware\ShareErrorsFromSession middleware được cung cấp ở nhóm web middleware. Khi các middleware được apply, một biến $error sẽ là biến trong view, cho phép bạn sử dụng chúng một cách thuận tiện.

Trong ví dụ, người dùng được chuyển hướng tới method create của controller khi validate thất bại, cho phép ta hiện lỗi lên view

<h1>Create Post</h1>

@if (count($errors) > 0)

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

**Customizing The Flashed Error Format**

Nếu bạn muốn tùy chỉnh format của các validation error được flash vào session khi validate thất bại, thì override formaValidationErrors trong base controller, và cần phải import class Illuminate\Contracts\Validation\Validator ở đầu file

<?php

namespace App\Http\Controllers;

use Illuminate\Foundation\Bus\DispatchesJobs;

use Illuminate\Contracts\Validation\Validator;

use Illuminate\Routing\Controller as BaseController;

use Illuminate\Foundation\Validation\ValidatesRequests;

abstract class Controller extends BaseController

{

use DispatchesJobs, ValidatesRequests;

/\*\*

\* {@inheritdoc}

\*/

protected function formatValidationErrors(Validator $validator)

{

return $validator->errors()->all();

}

}

AJAX Requests & Validation

Trong ví dụ này, ta sử dụng form truyền thống để gửi dữ liệu cho ứng dụng. Tuy nhiên, nhiều ứng dụng sử dụng các AJAX request. Khi sử dụng phương pháp validate trong một AJAX request , Laravel sẽ không tạo ra một phản hồi chuyển hướng. Thay vào đó, Laravel tạo ra một phản hồi JSON chứa tất cả các lỗi xác nhận. Phản hồi JSON này sẽ được gửi với một mã trạng thái HTTP 422.

Validating Arrays

Việc validate mảng form input không khó. Vd, email phải là unique, thì làm như sau:

$validator = Validator::make($request->all(), [

'person.\*.email' => 'email|unique:users'

]);

Tương tự, ta cũng dùng kí tự “\*” khi xác định các validation message ở file language, tạo sự dễ dàng để sử dụng 1 validation message cho mảng

'custom' => [

'person.\*.email' => [

'unique' => 'Each person must have a unique e-mail address',

]

],

[N](https://laravel.com/docs/5.2/validation" \l "other-validation-approaches)hững Cách validation khác

Tạo thủ công

Nếu bạn không muốn sử dụng các phương thức validate của ValidatesRequests trait, bạn có thể tạo ra một thể xác nhận bằng tay bằng cách sử dụng Validator facade. Các phương thức trên facade tạo ra một trường hợp xác nhận mới:

<?php

namespace App\Http\Controllers;

use Validator;

use Illuminate\Http\Request;

use App\Http\Controllers\Controller;

class PostController extends Controller

{

/\*\*

\* Store a new blog post.

\*

\* @param Request $request

\* @return Response

\*/

public function store(Request $request)

{

$validator = Validator::make($request->all(), [

'title' => 'required|unique:posts|max:255',

'body' => 'required',

]);

if ($validator->fails()) {

return redirect('post/create')

->withErrors($validator)

->withInput();

}

// Store the blog post...

}

}

Đối số đầu tiên được truyền cho method make là các dữ liệu được validate. Đối số thứ hai là các validation rule được apply cho các dữ liệu.

Sau khi kiểm tra nếu yêu cầu không được thông qua xác nhận, bạn có thể sử dụng phương thức withErrors để flash các thông báo lỗi để lên session. Khi sử dụng phương pháp này, biến $error sẽ tự động được chia sẻ đến view của bạn sau khi chuyển hướng, cho phép bạn dễ dàng hiển thị chúng với người sử dụng. Phương thức withErrors nhận một validate, một MessageBag, hoặc một mảng PHP.

**Named Error Bags**

Nếu bạn có nhiều form trên một trang duy nhất, bạn có thể muốn đặt tên cho các MessageBag lỗi, cho phép bạn truy xuất các thông báo lỗi cho một form cụ thể. Đơn giản chỉ cần pass một tên như là đối số thứ hai đến withErrors:

return redirect('register')

->withErrors($validator, 'login');

Ta có thể truy cập đến MessageBog được đặt tên từ biến error

{{ $errors->login->first('email') }}

#### After Validation Hook

Validator cũng cho phép bạn đính kèm callbacks được chạy sau khi xác nhận đã hoàn tất. Điều này cho phép bạn dễ dàng thực hiện xác nhận hơn và thậm chí thêm các thông báo lỗi vào message collection. Để bắt đầu, sử dụng phương thức after làm ví dụ:

$validator = Validator::make(...);

$validator->after(function($validator) {

if ($this->somethingElseIsInvalid()) {

$validator->errors()->add('field', 'Something is wrong with this field!');

}

});

if ($validator->fails()) {

//

}

### Form Request Validation

Đối với kịch bản xác nhận phức tạp hơn, bạn có thể muốn tạo ra một form request. Form request là các class request tùy chỉnh chứa xác nhận logic. Để tạo ra một form request cl, sử dụng: make:request

php artisan make:request StoreBlogPostRequest

Class được tạo ra sẽ được đặt trong đường dẫn app\Http\Requests. Ta hãy add thêm vài rule vào

public function rules()

{

return [

'title' => 'required|unique:posts|max:255',

'body' => 'required',

];

}

Vậy làm thế nào các validation rule được hiển thị? Tất cả bạn cần làm là nhập-gợi ý các yêu cầu trên các phương thức của controller. Các form request đến được xác nhận trước khi các controller method được gọi, có nghĩa là bạn không cần phải để lộn xộn controller của bạn với bất kỳ validation logic:

/\*\*

\* Store the incoming blog post.

\*

\* @param StoreBlogPostRequest $request

\* @return Response

\*/

public function store(StoreBlogPostRequest $request)

{

// The incoming request is valid...

}

#### Authorizing Form Requests

**form request class có chứa một phương thức authrize. Trong phương thức này, bạn có thể kiểm tra xem người dùng xác thực thực sự có quyền cập nhật một resource nhất định. Ví dụ, nếu người dùng đang cố gắng cập nhật một bài viết trên blog comment, để họ thực sự sở hữu bình luận đó? Ví dụ như:**

**/\*\***

\* Determine if the user is authorized to make this request.

\*

\* @return bool

\*/

public function authorize()

{

$commentId = $this->route('comment');

return Comment::where('id', $commentId)

->where('user\_id', Auth::id())->exists();

}

**Chú ý: việc gọi phương thức route trong ví dụ trên. Phương pháp này cho phép bạn có quyền truy cập vào các thông số URI xác định trên các route được gọi, như tham số {comment} trong ví dụ dưới đây:**

**Route::post('comment/{comment}');**

**Nếu phương thức authorize trả về false, một phản hồi HTTP với một mã trạng thái 403 sẽ tự động được đáp lại và controller method của bạn sẽ không được thực thi.**

**nếu bạn muốn có authorization logic trong app khác, chỉ cần return true ở method**

**/\*\***

\* Determine if the user is authorized to make this request.

\*

\* @return bool

\*/

public function authorize()

{

return true;

}

#### **Customizing The Flashed Error Format**

**Nếu bạn muốn tùy chỉnh định dạng của các lỗi xác nhận đã được flash vào session khi xác nhận không thành, ghi đè formatErrors lên các yêu cầu gốc của bạn (**App\Http\Requests\Request**). Import Illuminate\Contracts\Validation\Validator class ở phía trên cùng của file:**

**/\*\***

\* {@inheritdoc}

\*/

protected function formatErrors(Validator $validator)

{

return $validator->errors()->all();

}

#### **Customizing The Error Messages**

**Bạn có thể tùy chỉnh các thông báo lỗi được sử dụng bởi các form request bằng cách ghi đè các phương thức mesages . Phương thức này sẽ trả về một mảng các cặp thuộc tính / rule và thông báo lỗi tương ứng của chúng :**

**/\*\***

\* Get the error messages for the defined validation rules.

\*

\* @return array

\*/

public function messages()

{

return [

'title.required' => 'A title is required',

'body.required' => 'A message is required',

];

}

## **[Working With Error Messages](https://laravel.com/docs/5.2/validation" \l "working-with-error-messages)**

**Sau khi gọi phương thức error, bạn sẽ nhận một instance** Illuminate\Support\MessageBag, chứa nhiều phương thức thuận tiền cho thao tác với lỗi

#### Retrieving The First Error Message For A Field

Để lấy lại thông báo lỗi đầu tiên của 1 field, sử dụng phương thức first

$messages = $validator->errors();

echo $messages->first('email');

#### Retrieving All Error Messages For A Field

Nếu muốn lấy tất cả thông báo lỗi của 1 field, sử dụng phương thức get:

foreach ($messages->get('email') as $message) {

//

}

#### Retrieving All Error Messages For All Fields

Lấy lại mảng thông báo lỗi cho mọi field, sử dụng all:

foreach ($messages->all() as $message) {

//

}

#### Xác định lỗi có tồn tại hay ko

if ($messages->has('email')) {

//

}

#### Lấy lại 1 lỗi với định 1 dạng

echo $messages->first('email', '<p>:message</p>');

*Lấy lại mọi lỗi với 1 định dạng:*

foreach ($messages->all('<li>:message</li>') as $message) {

//

}

### Custom Error Messages

Nếu cần thiết, bạn có thể sử dụng các thông báo lỗi tùy chỉnh để xác nhận thay vì mặc định. Có một số cách để xác định các thông báo tùy chỉnh . Đầu tiên, bạn có thể pass các thông báo tùy chỉnh như là đối số thứ ba các phương thức Validator :: make

$messages = [

'required' => 'The :attribute field is required.',

];

$validator = Validator::make($input, $rules, $messages);

Trong ví dụ này,: attribute place-holder sẽ được thay thế bằng tên thực tế của các trường được validate. Bạn cũng có thể sử dụng place-holder nơi đông khác trong các tin nhắn xác nhận. Ví dụ như:

$messages = [

'same' => 'The :attribute and :other must match.',

'size' => 'The :attribute must be exactly :size.',

'between' => 'The :attribute must be between :min - :max.',

'in' => 'The :attribute must be one of the following types: :values',

];

#### Specifying A Custom Message For A Given Attribute

Đôi khi bạn có thể chỉ định một thông báo lỗi tùy chỉnh chỉ trong một field cụ thể. Bạn có thể làm như vậy bằng cách sử dụng ".". Đầu tiên, Chỉ định tên của thuộc tính , tiếp theo là các quy tắc:

$messages = [

'email.required' => 'We need to know your e-mail address!',

];

#### Specifying Custom Messages In Language Files

Trong nhiều trường hợp, bạn có thể muốn chỉ định thuộc tính custom message cụ thể trong một tập tin ngôn ngữ thay vì pass chúng trực tiếp vào Validator. Để làm như vậy, add message của bạn đến custom array trong tập tin ngôn ngữ resources/lang/xx/validation.php.