Bài 19
Validation

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

Mục tiêu

- Triển khai được validation sử dụng form request validation
- Tạo được các validator tuỳ biến
- Hiển thị được các thông báo tuỳ biến

Validation

Giới thiệu, Bắt đầu nhanh validation

Form Request Validation, Tự tạo validators

Làm việc với nội dung lỗi, Những quy định validation có sẵn

Thêm quy định có điều kiện ,Validating mảng

Tùy biến quy định validation

Giới thiệu

- Laravel cung cấp một vài cách tiếp cận để validate dữ liệu đến ứng dụng của bạn.
- Mặc định, class base controller của Laravel sử dụng ValidatesRequests trait cung cấp phương thức khá thuận tiện cho việc validate HTTP request đến với đa dạng quy định validation.

Bắt đầu nhanh validation

- Xác định routes
- Tạo mới Controller
- Viết logic validation
- Hiển thị lỗi validation

Bắt đầu nhanh validation

- Để học tính năng validation của Laravel, Hãy xem một ví dụ hoàn chỉnh validate một form và hiển thị nội dung lỗi trả về cho người dùng.

Xác định routes

- Đầu tiên, giả sử chúng ta có route được định nghĩa trong routes/web.php :

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


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

- Tất nhiên, phương thức GET route sẽ hiển thị một form cho người dùng tạo mới một bài viết, trong khi phương thức POST route sẽ lưu bài viết đấy vào cơ sở dữ liệu

Tạo Controller

- Tiếp theo, tạo một controller đơn giản xử lý các routes.Bây giờ, chúng ta sẽ để phương thức đấy store rỗng:

```php
<?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...
    }
}
```

Viết logic validation

- Bây giờ chúng ta đã sẵn sàng viết logic vào phương thức store để validate tạo mới bài viết.
- Nếu bạn kiểm tra class base controller (App\Http\Controllers\Controller) của Laravel, bạn sẽ thấy class sử dụng một ValidatesRequests trait, nó cung cấp một phương thức validate cho tất cả controllers.
- Phương thức validate chấp nhận một HTTP request đến và đặt quy định validation.

Viết logic validation

- Nếu quy định validationthành công, code của bạn sẽ thực thi bình thường; tuy nhiên, nếu validation thất bại, mội exception sẽ được ném và tích hợp lỗi response sẽ được tự động gửi cho người dùng.
- Trong trường hợp là HTTP request, một response chuyển trang sẽ được tạo ra, trong khi một JSON response sẽ được gửi cho AJAX requests.

Viết logic validation

- Để có thể hiểu rõ hơn về phương thức validate , hãy quay lại phương thức store :

```
/**
 * Store a new blog post.
 *
 * @param    Request  $request
 * @return  Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

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

Viết logic validation

- Như bạn có thể thấy, chúng ta có thể truyền qua HTTP request đến và yêu cầu quy định validation vào phương thức validate .
- Một lần nữa, nếu validation thất bại, Một proper response sẽ tự động được tạo ra.
- Nếu validation thành công, controller sẽ được thực thi bình thường

Dừng khi validation thất bại

- Thình thoảng bạn muốn quy định validation trong một thuộc tính sau khi validation đầu tiên thất bại.
- Để làm việc đó, gán quy định bail cho thuộc tính:

```
$this->validate($request, [
    'title' => 'bail|required|unique:posts|max:255',
    'body' => 'required',
]);
```

- Trong ví dụ này, nếu quy định required trên thuộc tính title thất bại, quy định unique sẽ không cần kiểm tra.
- Quy định sẽ validate trong thứ tự mà nó được gán.

Chú ý thuộc tính lồng nhau

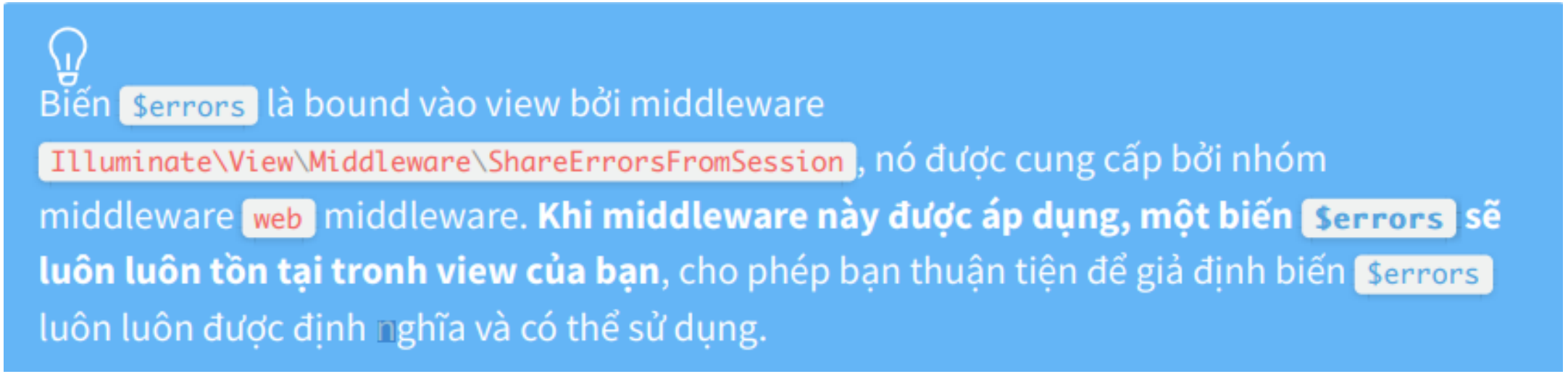
- Nếu HTTP request chứa tham số "lồng nhau", bạn có thể chỉ định chúng trong quy định validate bằng cách sử dụng cú pháp "dấu chấm":

```
$this->validate($request, [
    'title' => 'required|unique:posts|max:255',
    'author.name' => 'required',
    'author.description' => 'required',
]);
```

Hiển thị validation lỗi

- Cái gì sẽ xảy ra khi có một tham số request gửi đến không thành không với quy định validation? Như đã đề cập ở trước, Laravel sẽ tự động chuyển trang lại cho người dùng về trang trước đó.
- Ngoài ra, tất cả các lỗi validation sẽ tự động flashed vào session. Một lần nữa, chú ý rằng chúng ta sẽ không có một cách rõ ràng bind nội dung lỗi vào view của GET route.
- Bởi vì Laravel sẽ tự động kiểm tra lỗi trong dữ liệu session, và tự động bind chúng vào view nếu chúng tồn tại.
- Biến $errors sẽ là một thể hiện của Illuminate\Support\MessageBag . Để biết thêm chi tiết về nó, có thể xem tại đây.

Hiển thị validation lỗi



- Vì vậy, trong ví dụ trên, người dùng sẽ chuyển trang đến phương thức create của controller khi validation thất bại, cho phép chúng ta hiển thị nội dung lỗi trên view:

Hiển thị validation lỗi

```blade
<!-- /resources/views/post/create.blade.php -->

<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

<!-- Create Post Form -->
```

Tùy biến định dạng lỗi Flashed

- Giả sử bạn muốn tùy chỉnh nội dung lỗi của validation được flashed vào session khi validation thất bại, ghi đè phương thức formatValidationErrors trong base controller.
- Đừng quên import class Illuminate\Contracts\Validation\Validator ở trên đầu file file:

Tùy biến định dạng lỗi Flashed

```php
<?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, chúng ta sử dụng form để gửi dữ liệu vào ứng dụng.
- Tuy nhiên, nhiều ứng dụng sử dụng AJAX requests. Khi sử dụng phương thức validate trong AJAX request, Laravel sẽ không tạo ra redirect response.
- Thay vì, Laravel tạo một JSON response chứa tất cả lỗi validation. JSON response này sẽ được gửi với mã 422 HTTP status.

Form Request Validation

- Tạo Form Requests
- Authorizing Form Requests
- Tùy biến định dạng lỗi
- Tùy biến nội dung lỗi

Tạo Form Requests

- Với những trường hợp validation phức tạp, bạn có thể tạo một "form request".
- Form requests là tùy chỉnh class request chứa logic validation.
- Để tạo class form request, sử dụng lệnh make:request Artisan CLI:

```
php artisan make:request StoreBlogPost
```

Tạo Form Requests

- Class được tạo sẽ nằm ở thư mục app/Http/Requests directory. Nếu thư mục đó không tồn tại, nó sẽ được tạo khi bạn chạy lệnh make:request . Chúng ta sẽ thêm một vài quy định validation vào trong phương thức rules :

```php
/**
 * Get the validation rules that apply to the request.
 *
 * @return  array
 */
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

Tạo Form Requests

- Bạn đánh giá thế nào về quy định validation? tất cả bạn cần làm là type-hint request vào trong phương thức controller.
- Form request được validated trước khi phương thức controller được gọi, nghĩa là bạn không cần viết một mớ hỗn độn logic trong controller:

```
/**
 * Store the incoming blog post.
 *
 * @param    StoreBlogPost  $request
 * @return  Response
 */
public function store(StoreBlogPost $request)
{
    // The incoming request is valid...
}
```

Tạo Form Requests

- Nếu validation thất bại, một chuyển trang response sẽ được tạo ra để gửi cho lại người dùng đến trang trước.
- Ngoài ra lỗi sẽ được flashed vào session, vì vậy chúng ta có thể hiển thị nó.
- Nếu request là AJAX request, một HTTP response với mã 422 status sẽ được trả về cho người dùng gồm JSON representation chứa lỗi validation.

Authorizing Form Requests

- Class form request ngoài ra còn chứa một phương thức authorize .
- Bên trong phương thức, bạn có thể xác thực người dùng thực sự đã có quyền cập nhật dữ liệu.
- Ví dụ, nếu một người dùng cố gắng cập nhật comment của một một bài viết, họ thật sự sở hữu comment đấy?

Authorizing Form Requests

```php
/**
 * Determine if the user is authorized to make this request.
 *
 * @return  bool
 */
public function authorize()
{
    $comment = Comment::find($this->route('comment'));

    return $comment && $this->user()->can('update', $comment);
}
```

Authorizing Form Requests

- Khi tất cả các form requests kế thừa từ class base Laravel request, chúng ta có thể sử dụng phương thức user để truy cập xác thức người dùng.
- Ngoài ra cũng cần gọi phương thức route trong ví dụ trên. Phương thức này cho phép bạn truy cập đến tham số của URI được định nghĩa trong route được gọi, như tham số {comment} trong ví dụ trên:

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

Authorizing Form Requests

- Nếu phương thức authorize trả về false , một HTTP response với mã 403 status sẽ được tự động trả về và phương thức controller sẽ không được thực hiện.
- Nếu bạn có kế hoạch cho phép logic trong một phần khác ứng dung của bạn, đơn giản trả về true từ phương thức authorize :

```php
/**
 * Determine if the user is authorized to make this request.
 *
 * @return  bool
 */
public function authorize()
{
    return true;
}
```

Tùy biến định dạng lỗi

- Nếu bạn muốn tùy biến định dạng lỗi validation được flashed vào session khi validation thất bại, ghi đè phương thức formatErrors trong base request (App\Http\Requests\Request).
- Đừng quên import class Illuminate\Contracts\Validation\Validator class ở trên đầu file:

```
/**
 * {@inheritdoc}
 */
protected function formatErrors(Validator $validator)
{
    return $validator->errors()->all();
}
```

Tùy biến nội dung lỗi

- Bạn có thể muốn tùy biến lỗi dung lỗi bằng cách sử dụng bởi form request bằng cách ghi đè phương thức messages .
- Phương thức này trả về một mảng các cặp thuộc tính / quy định tương ứng với nội dung lỗi:

```php
/**
 * 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',
    ];
}
```

Tự tạo validator

- Redirection tự động
- Named Error Bags
- Afer Validation Hook

Tự tạo validator

- Nếu bạn không muốn sử dụng phương thức ValidatesRequests trait's validate , bạn có thể tự tạo một thể hiện validator instance bằng Validator facade.
- Phương thức make trong facade sinh ra một thể hiện mới validator:

```php
<?php

namespace App\Http\Controllers;

use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
```

Tự tạo validator

```php
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...
    }
}
```

Tự tạo validator

- Đối số đầu tiên truyền vào phương thức make là dữ liệu cần validation.
- Đối số thứ hai là mảng quy định validation được áp dụng vào dữ liệu.
- Sau khi kiểm tra request validation không thành công, bạn có thể dùng phương thức withErrors để flash nội dung lỗi vào session.

Tự tạo validator

- Khi sử dụng phương thức này, Biến $errors sẽ tự động được gửi đến views sau khi chuyển trang, cho phép bạn dễ dàng hiển thị chúng cho người dùng.
- Phương thức withErrors chấp nhận một validator, MessageBag , hoặc một array PHP.

Redirection tự động

- Nếu bạn muốn tạo mới một thể hiện validator những vẫn tự động chuyển trang bởi ValidatesRequest trait, bạn có thể gọi phương thức validate trong một thể hiện hiện tại validator.
- Nếu validation thất bạn, người dùng sẽ tự động được chuyển trang hoặc trong trường hợp là một AJAX request, một JSON response sẽ được trả về:

```
Validator::make($request->all(), [
    'title' => 'required|unique:posts|max:255',
    'body' => 'required',
])->validate();
```

Named Error Bags

- Nếu bạn có nhiều form trên một trang, bạn có thể sử dụng phương thức MessageBag , cho phép bạn nhận nội dung lỗi từ form cụ thể.
- Đơn giản chỉ là truyền thêm một tham số thứ hai của phương thức withErrors :

```
return redirect('register')
            ->withErrors($validator, 'login');
```

Named Error Bags

- Bạn cũng có thể lấy một thể hiện MessageBag từ biến $errors:

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

Afer Validation Hook

- Ngoài ra validator còn cho phép bạn thêm callbacks để chạy sau khi validation thành công.
- Điều này cho phép bạn dễ dàng thực hiện các validation và thêm nội dung lỗi cho message collection.
- Để bắt đầu, sử dụng phương thức after trong một thể hiện validator:

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

$validator->after(function($validator) {
    if ($this->somethingElseIsInvalid()) {
        $validator->errors()->add('field', 'Something is wrong with this field!');
    }
});

if ($validator->fails()) {
    //
}
```

Làm việc với nội dung lỗi

- Sau khi gọi phương thức errors trong một thể hiện Validator , bạn sẽ nhận được một thể hiện Illuminate\Support\MessageBag , sẽ có một số phương thức làm việc với nội dung lỗi.
- Biến $errors sẽ tự động được tạo cho tất cả các view, ngoài ra nó cũng là một thể hiện của class MessageBag .

Nhận về nội dung lỗi đầu tiên của một trường

- Để nhận về lỗi đầu tiên của một trường, sử dụng phương thức first :

```php
$errors = $validator->errors();

echo $errors->first('email');
```

Nhận về tất cả nội dung lỗi của một trường

- Nếu bạn cần nhận một mảng nội dung của tất cả lỗi của một trường, sử dụng phương thức get :

- ```
  foreach ($errors->get('email') as $message) {
      //
  }
  ```
  g của form, bạn có thể nhận nảng bằng cách sử dụng ký tự * :

```
foreach ($errors->get('attachments.*') as $message) {
    //
}
```
```

Nhận về tất cả các lỗi của tất cả các trường

- Để nhận một mảng tất cả các nội dung của tất cả các trường, sử dụng phương thức all :

```
foreach ($errors->all() as $message) {
    //
}
```

Xác định nội dung của một trường có tồn tại

- Phương thức has có thể sử dụng để xác định bất kỳ nội dung lỗi tồn tại của một trường:

```
if ($errors->has('email')) {
    //
}
```

Tùy biến nội dung

- Nếu bạn cần, bạn có thể tùy biến nội dung lỗi cho thể hiện validation mặc định.
- Có một vài cách để làm việc này.
- Đầu tiên, bạn có thể truyền tùy biến nội dung như là tham số thứ ba của hàm Validator::make :

```
$messages = [
    'required' => 'The :attribute field is required.',
];

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

Tùy biến nội dung

- Trong ví dụ trên, thuộc tính :attribute place-holdersẽ thay thế bởi tên thực tế của trường validation.
- Ngoài ra bạn có thể sử dụng place-holders trong nội dung validation. Ví dụ

```
$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',
];
```

Tùy biến nội dung của thuộc tính cụ thể

- Thỉnh thoảng bạn có thể tùy biến nội dung lỗi chỉ duy nhất một trường. Bạn có thể sử dụng "dấu chấm".
- Chỉ định tên của thuộc tính đầu tiên, theo bởi quy định:

```
$messages = [
    'email.required' => 'We need to know your e-mail address!',
];
```

Tùy biến nội dung trong file ngôn ngữ

- Trong hầu hết các trương hợp, bạn có thể tùy biến nội dung trong một file ngôn ngữ thay vì truyền chúng trực tiếp vào phương thức Validator .
- Để làm điều này, bạn thêm nội dung vào mảng custom trong file ngôn ngữ resources/lang/xx/validation.php .

```
'custom' => [
    'email' => [
        'required' => 'We need to know your e-mail address!',
    ],
],
```

Tùy biến thuộc tính trong file ngôn ngữ

- Nếu bạn muốn thuộc tính :attribute phần nội dung validation sẽ được thay đổi bởi tên thuộc tính tùy chỉnh, bạn có thể tùy biến trong mảng attributes của file ngôn ngữ resources/lang/xx/validation.php :

```
'attributes' => [
    'email' => 'email address',
],
```

Những quy định validation có sẵn

- Bên dưới là danh sách những quy định có sẵn và hàm của nó:

Accepted
Active URL
After (Date)
Alpha
Alpha Dash
Alpha Numeric
Array
Before (Date)
Between
Boolean
Confirmed
Date
Date Format
Different
Digits
Digits Between
Dimensions (Image Files)
Distinct
E-Mail
Exists (Database)
File
Filled
Image (File)
In
In Array
Integer
IP Address
JSON
Max
MIME Types
Min
Nullable
Not In
Numeric
Present
Regular Expression
Required
Required If
Required Unless
Required With

Những quy định validation có sẵn

- Required With All
- Required Without
- Required Without All
- Same
- Size
- String
- Timezone
- Unique (Database)
- URL

Những quy định validation có sẵn

- accepted
  - Giá trị phải là yes, on, 1, or true. Rất hữu dụng cho việc chấp nhận "Terms of Service".
- active_url
  - Giá trị phải là URL theo hàm checkdnsrr của PHP.
- afer:date
  - Giá trị phải là một ngày sau ngày đã cho. Giá trị ngày phải hợp lệ theo hàm strtotime của PHP:

```
'start_date' => 'required|date|after:tomorrow'
```

Những quy định validation có sẵn

- Thay vì truyền giá trị ngày vào một chuỗi vào hàm strtotime, bạn có thể chỉ định một trường khác để so sánh ngày:

```
'finish_date' => 'required|date|after:start_date'
```

- alp
  - (
- alpha_dash
  - Giá trị phải là chữ cái hoặc chữ số, gồm cả dấu gạch ngang và dấu gạch dưới.
- alpha_num
  - Giá trị phải là chữ số.

Những quy định validation có sẵn

- array
  - Giá trị phải là một array PHP.
- before:date
  - Giá trị phải là ngày trước ngày đã cho. Giá trị ngày sẽ được truyền vào hàm strtotime của PHP
- between:min,max
  - Giá trị phải nằm trong khoảng min and max. Chuỗi, số, và file là giống kiểu size với nhau.
- Boolean
  - Giá trị phải là kiểu boolean có thể là true , false , 1 , 0 , "1" , và "0"

Những quy định validation có sẵn

- confirmed
  - Giá trị phải khớp với trường foo_confirmation . Ví dụ, nếu trường là password , thì giá trị password_confirmation phải khớp với trương mật khẩu.
- date
  - Giá trị phải là ngày hợp lệ theo hàm strtotime của PHP.
- date_format:format
  - Giá trị phải giống format truyền vào. Định dạng phải hợp lệ với hàm date_parse_from_format của PHP. Bạn nên sử dụng either date hoặc date_format khi validate một trường.

Những quy định validation có sẵn

- diferent:field
  - Giá trị phải khác giá trị của field.
- digits:value
  - Giá trị phải là numeric và phải chính xác độ dài là value.
- digits_between:min,max
  - Giá trị phải có độ dài nằm trong khoảng min and max.

Những quy định validation có sẵn

- dimensions
  - Giá trị phải là một ảnh có kích thước giống rule's parameters:
  - Tồn tại một số thuộc tính: min_width, max_width, min_height, max_height, width, height, ratio
  - A
```
'avatar' => 'dimensions:min_width=100,min_height=200'
```
ợc xác định như 3/2 hoặc 1.5 :

```
'avatar' => 'dimensions:ratio=3/2'
```

Những quy định validation có sẵn

- distinct
  - Khi làm việc với mảng, Mảng phải không có giá trị lặp lại.

- email
  - ```
    'foo.*.id' => 'distinct'
    ```
- exists:table,column
  - Giá trị phải có trong bảng cơ sở dữ liệu.
    ```

Những quy định validation có sẵn

- Basic Usage Of Exists Rule

- Specifying A Custom Column Name

```
'state' => 'exists:states'
```

  - Thỉnh thoảng, bạn cần kiểm tra kết nối database sử dụng cho exists query. Bạn có thể làm điều này bằng cách thêm "dấu chấm" vào trước tên kết nối:

```
'state' => 'exists:states,abbreviation'
```

```
'email' => 'exists:connection.staff,email'
```

Những quy định validation có sẵn

- Nếu bạn muốn tùy biến thực thi query , bạn có thể sử dụng class Rule để định nghĩa quy định. Trong ví dụ này, chúng ta chỉ định quy tắc validation như là một mảng thay vì sử dụng ký tự | :

```php
use Illuminate\Validation\Rule;

Validator::make($data, [
    'email' => [
        'required',
        Rule::exists('staff')->where(function ($query) {
            $query->where('account_id', 1);
        }),
    ],
]);
```

Những quy định validation có sẵn

- file
  - Giá trị phải là một file tải lên thành công.
- filled
  - Giá trị không được phép trống.
- image
  - Giá trị phải là ảnh có định dạng (jpeg, png, bmp, gif, or svg)
- in:foo,bar,...
  - Giá trị phải thuộc danh sách các giá trị.

Những quy định validation có sẵn

- in_array:anotherfield
  - Giá trị phải tồn tại trong giá trị của anotherfield's.
- integer
  - Giá trị phải là kiểu integer.
- ip
  - Giá trị phải là địa chỉ IP.
- json
  - Giá trị phải là một string JSON.

Những quy định validation có sẵn

- max:value
  - Giá trị phải nhỏ hơn hoặc bằng value. Chuỗi, số, và file là kiểu giống size với nhau.
- mimetypes:text/plain,...
  - Giá trị phải khớp với MIME types:

```
'video' => 'mimetypes:video/avi,video/mpeg,video/quicktime'
```

  - Xác định MIME type của file upload, nội dung file sẽ được đọc framework sẽ đoán MIME type, có thể sẽ khác MIME type của người dùng.
- min
  - G ensions.

Những quy định validation có sẵn

- Basic Usage Of MIME Rule
  - Mặc dù bạn chỉ cần xác định extensions, thực ra quy định validates này lại là validate MIME type file bằng các đọc nội dung và đoán MIME type.
  - `'photo' => 'mimes:jpeg,bmp,png'`
    s có thể tìm thấy ở: http://svn.apache.org/repos/asf/httpd/httpd/trunk/docs/conf/mime.types
- min:value
  - Giá trị phải nhỏ hơn value. Chuỗi, số, và file là giống size với nhau.
- nullable
  - Giá trị có thể null . Nó rất hữu dụng khi validate string hoặc integer chứa giá trị null .

Những quy định validation có sẵn

- not_in:foo,bar,...
  - Giá trị phải không thuộc danh sách giá trị.
- numeric
  - Giá trị phải là số.
- Present
  - Giá trị hiện tại phải xuất hiện trong input nhưng thể được trống.
- regex:pattern
  - Giá trị phải khớp với regular expression.

Những quy định validation có sẵn

- Note: Khi sử dụng regex pattern, nó cần được xác định quy định trong mảng thay vì sử dụng pipe delimiter, đặc biệt nếu regular expression chứa pipe ký tự.
- required
  - Giá trị phải xuất hiện trong input và không được phép trống.
  - Một trường được coi là "empty" nếu một trong số điều kiện dưới đây đúng:
    - Giá trị là null .
    - Giá trị là một chuỗi rỗng.
    - Giá trị là mảng rỗng hoặc object Countable rỗng.
    - Giá trị là file upload không có đường dẫn.

Những quy định validation có sẵn

- required_if:anotherfield,value,...
  - Giá trị phải xuất hiện và không được trống nếu trường anotherfield bằng bất kỳ value.
- required_unless:anotherfield,value,...
  - Giá trị phải xuất hiện và không được phép trống trừ khi trường anotherfield bằng bất kỳ value.
- required_with:foo,bar,...
  - Giá trị phải xuất hiện và không được trống only if bất kỳ một trường khác xác định xuất hiện.
- required_with_all:foo,bar,...
  - Giá trị phải xuất hiện và không được trống only if tất cả các trường khác xác định xuất hiện.

Những quy định validation có sẵn

- required_without:foo,bar,...
  - Giá trị phải xuất hiện và không được trống only when bất cứ trường xác định không xuất hiện.
- required_without_all:foo,bar,...
  - The field under validation must be present and not empty only when all of the other specified fields are not present.
- same:field
  - Giá trị field phải khớp với trường này.

Những quy định validation có sẵn

- size:value
  - Giá trị phải có kích thước khớp với value. Đối với chuỗi, value tương ứng là số ký tự. Đối với só, value tương ứng là giá trị integer. Đối với mảng, size tương ứng là count phần tử của mảng. Đối với file, size tương ứng là kích thước file kiểu kilobytes.
- string
  - Giá trị phải là chuỗi. Nếu bạn muốn cho phép trường đó null , bạn có thể gán nullable vào trường đó.
- timezone
  - Giá trị phải là timezone identifier hợp lệ với hàm timezone_identifiers_list của PHP.

Những quy định validation có sẵn

- Unique:table,column,except,idColumn
  - Giá trị phải là unique trong bảng cơ sở dữ liệu.
  - Nếu tên column không được chỉ định, trường name sẽ được sử dụng.
- Specifying A Custom Column Name:

```
'email' => 'unique:users,email_address'
```

Những quy định validation có sẵn

- Tùy biến kết nối cơ sở dữ liệu
  - Thỉnh thoảng, có thể bạn muốn tủy chỉnh kết nối query cơ sở dữ liệu bởi Validator. Như ở trên, cài đặt unique:users như một quy định validation sẽ sử dụng kết nối mặc định database để query đến cơ sở dữ liệu.
  - Để ghi đè nó, xác định kết nối và tên bảng sử dụng "dấu chấm":

```
'email' => 'unique:connection.users,email_address'
```

- Validate Uniquebỏ qua ID:
  - T a unique. Ví dụ, cân nhắc "cập nhật hồ sơ" sẽ bao gồm name, địa chỉ e-mail, và địa điểm của người dùng.

Những quy định validation có sẵn

- Tất nhiên, bạn sẽ muốn xác định email là unique.
- Tuy nhiên, nếu người dùng chỉ thay đổi tên và không thay đổi email, bạn không muốn validation lỗi được ném ra bởi vì người dùng đó đã sử dụng cái email đấy rồi.
- Chỉ dẫn validator bỏ qua ID của người dùng, chúng ta sử dụng class Rule định nghĩa quy tắc.
- Trong ví dụ này, chúng ta sẽ chỉ định quy tắc validation như một mảng thay thế sử dụng ký tự để phân cách | quy định:

```
use Illuminate\Validation\Rule;

Validator::make($data, [
    'email' => [
        'required',
        Rule::unique('users')->ignore($user->id),
    ],
]);
```

Những quy định validation có sẵn

- Nếu bản user của bạn có a primary key không phải là id , bạn có thể chỉ định name của cột khi gọi phương thức ignore :

- Thêm điều kiện bổ sung:
  - Bạn cũng có thể thêm query chứa tùy chỉnh query bằng cách sử dụng phương thức where . Ví dụ, chúng ta thêm một hạn chế để kiểm tra account_id là 1 :

- url
  - G

```
'email' => Rule::unique('users')->where(function ($query) {
    $query->where('account_id', 1);
})
```

Thêm quy định có điều kiện

- Validating khi xuất hiện
- Thêm quy định có điều kiện

Validating khi xuất hiện

- Trong một số trường hợp, bạn có thể muốn chạy validation kiểm tra lại trường only nếu trường đó xuất hiện trong mảng input.
- Để nhanh chóng làm điều này, thêm sometimes vào trong danh sách quy tắc rule:

```
$v = Validator::make($data, [
    'email' => 'sometimes|required|email',
]);
```

- T validated nếu nó xuất h

Thêm quy định có điều kiện

- Thỉnh thoảng bạn muốn thêm quy định trong logic. Ví dụ, bạn có thể muốn yêu cầu một trường chỉ nếu trường khác có giá trị lớn hơn 100. Hoặc, Bạn muốn 2 trường có giá trị chỉ khi trường khác xuất hiện. Để làm việc đó không có gì khó khăn cả. Đầu tiên, tạo một thể hiện Validator với static rules sẽ không bao giờ thay đổi:

```
$v = Validator::make($data, [
    'email' => 'required|email',
    'games' => 'required|numeric',
]);
```

Thêm quy định có điều kiện

- Giả sử bây giờ ứng dụng web của bạn là sưu tầm game.Nếu một người sưu tầm game đăng ký ứng dụng của bạn và họ có nhỏ hơn 100 games, chúng ta muốn họ giải thích tại sao chọ có quá nhiều game. Ví dụ, có thể họ chạy một shop bán game, hoặc có thể họ thích sư tầm. Để có thể yêu cầu này, chúng ta có thể sử dụng phương thức sometimes trong thể hiện Validator

```
$v->sometimes('reason', 'required|max:500', function($input) {
    return $input->games >= 100;
});
```

Thêm quy định có điều kiện

- Tham số thứ nhất truyền vào phương thức sometimes là tên của trường chúng ta muốn validate.Tham số thứ hai là quy định chúng ta muốn thêm. Nếu truyền Closure như là tham số thứ ba trả về true , quy định sẽ được thêm. Phương thức này làm cho việc thêm quy định validate phức tạp trở lên dễ dàng hơn, ngay cả khi bạn muốn thêm nhiều validate cho nhiều trường:

```
$v->sometimes(['reason', 'cost'], 'required', function($input) {
    return $input->games >= 100;
});
```

Tham số `$input` truyền vào trong `Closure` là một thể hiện của `Illuminate\Support\Fluent` và bạn có thể truy cập input và file.

Validating mảng

- Validating mảng các trường của form không có gì khó khăn. Ví dụ, để validate mỗi email trong mảng trường input là unique

```
$validator = Validator::make($request->all(), [
    'person.*.email' => 'email|unique:users',
    'person.*.first_name' => 'required_with:person.*.last_name',
]);
```

Validating mảng

- Tương tự như vậy, bạn có thể sử dụng ký tự * khi muốn chỉ định nội dung validation trong file ngôn ngữ, làm cho việc dễ dàng sử dụng một file nội dung validate cho mảng:

```
'custom' => [
    'person.*.email' => [
        'unique' => 'Each person must have a unique e-mail address',
    ]
],
```

Tùy biến quy định validation

- Laravel cung cấp một số quy định validation rất hữu ích; tuy nhiên, có thể bạn muốn tạo validate bởi chính bạn.
- Một phương thức đăng ký tùy biến quy tắc validationlà sử dụng phương thức extend trong Validator facade.
- Chúng ta sẽ sử dụng nó trong một service provider để đăng ký tùy biến quy tắc validation:

Tùy biến quy định validation

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\Validator;
```

```php
class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return  void
     */
    public function boot()
    {
        Validator::extend('foo', function($attribute, $value, $parameters, $validator) {
            return $value == 'foo';
        });
    }

    /**
     * Register the service provider.
     *
     * @return  void
     */
    public function register()
    {
        //
    }
}
```

Tùy biến quy định validation

- Tùy biến validator Closure nhận bốn đối số: tên của $attribute được validate, giá trị $value của thuộc tính, một mảng quy định $parameters , và một thể hiện Validator . Bạn cũng có thể truyền một class và method vào phương thức extend thay vì một Closure:

```
Validator::extend('foo', 'FooValidator@validate');
```

Định nghĩa nội dung lỗi

- Bạn có thể định nghĩa một nội dung lỗi cho quy định tùy biến của bạn. Bạn có thể làm như vậy hoặc một mảng nội dung tùy biến nội dung inline hoặc thêm vào validation file ngôn ngữ. Nội dung này sẽ được đặt ở trên đầu của mảng, không ở bên trong mảng custom ,nó chỉ dành cho những nội dung lỗi attribute-specific:

```
"foo" => "Your input was invalid!",

"accepted" => "The :attribute must be accepted.",

// The rest of the validation error messages...
```

Định nghĩa nội dung lỗi

- Khi bạn tùy biến quy định validation, thỉnh thảng bạn cần định nghĩa tùy chỉnh place-holder thay thế nội dung lỗi. Bạn cũng có thể tạo một Validator như miểu tả ở trên sau đó gọi phương thức replacer trong Validator facade. Bạn có thể sử dụng trong phương thức boot của service provider:

```php
/**
 * Bootstrap any application services.
 *
 * @return  void
 */
public function boot()
{
    Validator::extend(...);

    Validator::replacer('foo', function($message, $attribute, $rule, $parameters) {
        return str_replace(...);
    });
}
```

Implicit Extensions

- Mặc định, khi một thuộc tính đã được validated là không xuất hiện hoặc chứa một giá trị rỗng như định nghĩa bởi quy tắc required , quy tắc validation thường, bao gồm cả phần extensions, là không hoạt động. Ví dụ, quy định unique sẽ không hoạt động lần nữa nếu giá trị null :

```
$rules = ['name' => 'unique'];

$input = ['name' => null];

Validator::make($input, $rules)->passes(); // true
```

Implicit Extensions

- Đối với quy tắc validate hoạt động ngay cả khi thuộc tính là rỗng, quy định phải ngụ ý rằng các thuộc tính là bắt buộc. Như tạo một "implicit" extension, sử dụng phương thức Validator::extendImplicit() :

```
Validator::extendImplicit('foo', function($attribute, $value, $parameters, $validator) {
    return $value == 'foo';
});
```

Một "implicit" extension chỉ *implies* ngụ ý là các thuộc tính là bắt buộc. Cho dù nó thực sự invalidates thuộc tính là lỗi hoặc rộng là phụ thuộc vào bạn.

Tóm tắt bài học

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: ***Session and Cookie***