Course - Laravel Framework

Phân tuyến - Routing

Hầu hết các route Laravel căn bản chấp nhận một URI và một hàm thực thi, bằng việc cung cấp một phương thức đơn giản và ấn tượng trong việc tạo route và hoạt động sẽ không cần các tập tin cấu hình route phức tạp.

Tags: Phân tuyến - Routing

Hầu hết các route Laravel căn bản chấp nhận một URI và một hàm thực thi, bằng việc cung cấp một phương thức đơn giản và ấn tượng trong việc tạo route và hoạt động sẽ không cần các tập tin cấu hình route phức tạp.

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Các tập tin route mặc định

Tất cả các route được tạo trong các tập tin route của bạn, những tập tin này được để trong thư mục */routes* của ứng dụng. Những tập tin này sẽ được tải một cách tự động bởi `App\Providers\RouteServiceProvider` của ứng dụng của bạn. Tập tin *routes/web.php* sẽ khai báo các route giành cho giao diện web. Những route này sẽ được nhượng vào nhóm middleware `web`, thứ mà sẽ cung cấp các tính năng như session-state và lớp bảo vệ CSRF. Trong khi đó các route trong `routes\api.php` thì stateless (không có session) và sẽ được nhượng vào nhóm middleware `api`.

Đối với hầu hết các ứng dụng, bạn sẽ bắt đầu khai báo route trong tập tin *routes/web.php*, các route được khai báo trong *routes/web.php* sẽ có thể được truy cập bằng cách nhập URL của route trong trình duyệt web của bạn. Lấy ví dụ bạn có thể truy cập vào route sau bằng cách chuyển đến http://localhost/user trong trình duyệt của bạn.

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Các route được khai báo trong tập tin *routes/api.php* sẽ được lồng ghép vào trong một nhóm route bởi `RouteServiceProvider`. Bên trong nhóm này, URI sẽ tự động có tiền tố `/api`, do đó bạn không cần phải làm bằng tay cho mọi route trong tập tin này. Bạn có thể tùy chỉnh lại các tiền tố và các option route khác qua class `RouteServiceProvider` của bạn.

Phương thức router có sẵn

Router cho phép bạn đăng ký các route mà tương ứng với bất kỳ động từ HTTP nào:

```php
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Thỉnh thoảng bạn cần phải đăng ký một route mà tương ứng với nhiều động từ HTTP. Bạn sẽ có thể thực hiện điều này bằng cách sử dụng phương thức `match`. Hay, bạn sẽ thậm chí có thể đăng ký một route mà tương thích với tất cả các động từ HTTP bằng cách sử dụng phương thức `any`.

```php
Route::match(['get', 'post'], '/', function () {
    //
});

Route::any('/', function () {
    //
});
```

> Khi khai báo nhiều route mà chia sẽ cùng URI, thì các route sử dụng các phương thức `get`, `post`, `put`, `patch`, `delete`, và `options` nên được khai báo trước các route sử dụng các phương thức `any`, `match` và `redirect`. Nhằm đảm bảo request gửi đến sẽ được khớp với route đúng.

Tính năng Dependency Injection

Bạn có thể khai báo kiểu bất kỳ dependency nào được đòi hỏi bởi route của bạn trong *callback signature* của route của bạn. Các dependency được khai báo sẽ được resolve một cách tự động và được tiêm chèn vào callback bằng *service container* của Laravel. Ví dụ, bạn có thể khai báo kiểu của class `Illustrator\Http\Request` để request HTTP hiện hành được tiêm một cách tự động vào trong callback của route của bạn:

```php
use Illuminate\Http\Request;

Route::get('/users', function (Request $request) {
```

```
    // ...
});
```

Lớp bảo vệ CSRF Protection

Hãy ghi nhớ, bất kỳ form HTML nào trỏ đến các route `POST`, `PUT`, `PATCH`, và `DELETE` mà được khai báo trong các tập tin route web nên thêm một field về *CFRS token*. Nếu không, các request sẽ bị từ chối. Bạn có thể đọc thêm về lớp bảo vệ *CSRF protection* trong tài liệu *CSRF*.

```
<form method="POST" action="/profile">
    @csrf
    ...
</form>
```

Cách chuyển hướng route

Nếu bạn đang khai báo một route mà chuyển hướng sang URI khác, bạn sẽ có thể sử dụng phương thức `Route::redirect`. Phương thức này cho một đường tắt tiện nghi nhằm để bạn phải khai báo một route hay controller dài dòng để thực thi một việc chuyển hướng đơn giản:

```
Route::redirect('/here', '/there');
```

Mặc định, `Route::redirect` sẽ trả lại một mã status `302`, bạn có thể thay đổi mã status bằng cách sử dụng thêm tham số thứ ba như sau:

```
Route::redirect('/here', '/there', 301);
```

Hoặt, bạn có thể sử dụng phương thức `Route::permanent` để trả lại một mã status `302`:

```
Route::permanentRedirect('/here', '/there');
```

Chú ý: Khi bạn sử dụng tham số route trong các route chuyển hướng (redirect), các

tham số sau sẽ được dự trữ bởi Laravel và không thể sử dụng được: `destination` và `status`.

Các route hiển thị (view routes)

Nếu route của bạn chỉ cần trả lại một view, bạn có thể sử dụng phương thức `Route::view`. Giống như phương thức `redirect`, phương thức này cung cấp một đường tắt đơn giản để bạn không phải khai báo dài dòng một route hay controller. Phương thức view nhận một URI ở tham số đầu tiên và một cái tên của view ở tham số thứ hai. Ngoài ra, bạn có thể cung cấp một mảng dữ liệu để truyền vào view ở phần tham số thứ ba:

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

> **Chú ý:** Khi bạn sử dụng tham số route trong các route view, các tham số sau được dự trữ bởi Laravel và không thể sử dụng được: `view`, `data`, `status`, và `headers`.

Tham số trong route

Tham số bắt buộc

Thỉnh thoảng, bạn sẽ cần phân đoạn lại các URI bên trong route của bạn. Ví dụ, để tiếp nhận lại ID của một người dùng từ URL, thì bạn có thể thực hiện được bằng cách khai báo các tham số route như sau:

```
Route::get('/user/{id}', function ($id) {
    return 'User '.$id;
});
```

Bạn có thể khai báo bao nhiêu tham số tùy theo yêu cầu trong route của bạn:

```php
Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {
    //
});
```

Các tham số route luôn được bao quanh bởi cặp dấu ngoặt móc {}, và sẽ chứa các ký tự alphabet. Gạch dưới (_) cũng được chấp nhận bên trong các tên tham số của route. Các tham số route sẽ được tiêm chèn vào trong các callback/controller dựa trên thứ tự của chúng - tên của các tham số callback/controller của route thì không quan trọng.

Tiêm chèn các tham số và dependency

Nếu route của bạn có các thư viện dependency mà bạn muốn *service container* của Laravel tiêm chèn một cách tự động vào trong callback của route của bạn, thì bạn nên liệt kê các tham số route của bạn sau các thư viện dependency của bạn.

```php
use Illuminate\Http\Request;

Route::get('/user/{id}', function (Request $request, $id) {
    return 'User '.$id;
});
```

Các tham số tùy chọn

Đôi khi bạn sẽ cần mô tả một tham số route mà có thể không phải thường xuyên xuất hiện trong URI. Bạn có thể thực hiện bằng cách đặt một dấu chấm hỏi ? phía sau tên tham số. Hãy đảm bảo mang vào biến tương ứng của route một giá trị mặc định.

```php
Route::get('/user/{name?}', function ($name = null) {
    return $name;
});

Route::get('/user/{name?}', function ($name = 'John') {
    return $name;
});
```

Các biểu thức ràng buộc

Bạn có thể ràng buộc định dạng nào đó vào trong tham số route của bạn bằng cách sử dụng phương thức where trên một route instance, phương thức where chứa tên tham số và một biểu thức regular để định nghĩa cách mà tham số đó ràng buộc:

```
Route::get('/user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');

Route::get('/user/{id}', function ($id) {
    //
})->where('id', '[0-9]+');

Route::get('/user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Để thuận tiện có vài biểu thức regular phổ biến thường dùng kiểu phương thức giúp bạn nhanh chóng thêm các mẫu ràng buộc vào trong route của bạn.

```
Route::get('/user/{id}/{name}', function ($id, $name) {
    //
})->whereNumber('id')->whereAlpha('name');

Route::get('/user/{name}', function ($name) {
    //
})->whereAlphaNumeric('name');

Route::get('/user/{id}', function ($id) {
    //
})->whereUuid('id');
```

Nếu request được gửi đến không khớp với các ràng buộc mẫu đã cho, một phản hồi với trạng thái 404 sẽ được trả lại.

Ràng buộc tổng thể

Nếu muốn một tham số route nào đó luôn luôn được gắn với một biểu thức regular nào đó, thì bạn có thể sử dụng phương thức `pattern`. Nơi khai báo các biểu thức regular này nên là trong phương thức `boot` của class `App\Providers\RouteServiceProvider`.

```
/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
public function boot()
{
    Route::pattern('id', '[0-9]+');
}
```

Một khi biểu thức được khai báo, thì nó sẽ tự động được cập nhật cho tất cả các route đang sử dụng tên của tham số đó.

```
Route::get('/user/{id}', function ($id) {
    // Only executed if {id} is numeric...
});
```

Mã hóa forward slashes

Component route của Laravel cho phép dùng tất cả các ký tự trừ ký tự `/` trong giá trị tham số route. Bạn phải công khai ký tự `/` như là một phần trong giá trị đó bằng cách sử dụng mệnh đề điều kiện `where` với biểu thức regular.

```
Route::get('/search/{search}', function ($search) {
    return $search;
})->where('search', '.*');
```

> **Chú ý:** Mã hóa forward slashs chỉ được hỗ trợ với route sau cùng.

Đặt tên cho route

Đặt tên các route cho đi sự thuận tiện trong việc tạo URLs hoặt chuyển hướng sang một route khác nào đó. Bạn có thể khai báo một cái tên cho một route bằng phương thức `name` khi khai báo route như sau:

```
Route::get('/user/profile', function () {
    //
})->name('profile');
```

Hoặt bạn có thể sử dụng trong việc khai báo route với các action của controller như sau:

```
Route::get(
    '/user/profile',
    [UserProfileController::class, 'show']
)->name('profile');
```

> **Chứ ý:** Tên của route cần phải luôn luôn được thống nhất.

Tạo ra URL với các route có tên

Một khi bạn đã nhượng một cái tên vào một route đã có, thì bạn có thể sử dụng cái tên đó để tạo ra các URL hay chuyển hướng thông qua các hàm helper `route` hay `redirect`.

```
// Generating URLs...
$url = route('profile');

// Generating Redirects...
return redirect()->route('profile');
```

Nếu các tên của route có thêm các tham số, thì bạn có thể truyền các giá trị bằng tham số thứ hai trong hàm `route`. Những tham số sẽ tự động được chèn vào trong URL đã tạo theo đúng các vị trí được phân bổ của chúng.

```
Route::get('/user/{id}/profile', function ($id) {
    //
})->name('profile');
```

```php
$url = route('profile', ['id' => 1]);
```

Nếu bạn kèm thêm tham số trong mảng, thì các cặp khóa/giá trị đó sẽ được tự động thêm vào chuỗi truy vấn của URL được sinh ra.

```php
Route::get('/user/{id}/profile', function ($id) {
    //
})->name('profile');

$url = route('profile', ['id' => 1, 'photos' => 'yes']);

// /user/1/profile?photos=yes
```

> Thỉnh thoảng bạn sẽ muốn khai báo các giá trị mặc định cho các tham số URL, chẳng hạn như vị trí hiện tại. Thì để hoàn thành điều này, bạn có thể cần phải sử dụng phương thức `URL::default`.

Khai báo route hiện tại

Nếu bạn muốn xác định xem request hiện tại đã được khai báo route với một cái tên nào đó, bạn có thể sử dụng phương thức `named` trên một instance của Route. Ví dụ, bạn có thể kiểm tra tên route hiện tại từ một middleware route:

```php
/**
 * Handle an incoming request.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Closure  $next
 * @return mixed
 */
public function handle($request, Closure $next)
{
    if ($request->route()->named('profile')) {
        //
    }
```

```
    return $next($request);
}
```

Nhóm Route

Nhóm route cho phép share các thộc tính của route, chẳng hạn như các middleware, qua một số lượng lớn các route mà không cần phải khai báo các thuộc tính đó cho từng route riêng biệt.

Việc lồng ghép các nhóm sẽ cho phép merge các thuộc tính một cách thông minh với các nhóm cha của chúng. Middleware và mệnh đề `where` sẽ được merge trong khi tên và các tiền tố sẽ được thêm vào. Các ký hiệu namespace và slash trong tiền tố URL sẽ được tự động thêm vào ở nơi phù hợp.

Middleware

Để gắn middleware cho tất cả các route vào trong một nhóm, bạn có thể sử dụng phương thức `middleware` trước khi khai báo nhóm. Middleware được thực thi theo thứ tự mà chúng được liệt kê trong mảng.

```
Route::middleware(['first', 'second'])->group(function () {
    Route::get('/', function () {
        // Uses first & second middleware...
    });

    Route::get('/user/profile', function () {
        // Uses first & second middleware...
    });
});
```

Controllers

Nếu một nhóm route có cùng một controller, thì bạn có thể sử dụng phương thức controller để khai báo controller chung cho tất cả các route trong cùng một nhóm. Sau đó, khi khai báo các route, bạn chỉ cần cung cấp phương thức controller mà chúng gọi tới.

```php
use App\Http\Controllers\OrderController;

Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

Subdomain

Việc nhóm route còn có thể được sử dụng để xử lý các route theo subdomain. Subdomain có thể được gắn động như một tham số trong URI, cho phép bạn ghi lại subdomain nào được sử dụng trong route hay controller của bạn. Subdomain có thể được xác định bằng cách gọi phương thức domain trước khi khai báo nhóm:

```php
Route::domain('{account}.example.com')->group(function () {
    Route::get('user/{id}', function ($account, $id) {
        //
    });
});
```

> **Chú ý:** Để đảm bảo các subdomain cho các route có thể đi tới được. Bạn nên đăng ký các route với subdomain trước khi đăng ký các route với root domain. Điều này sẽ ngăn ngừa các route với root domain chồng chéo lên các route với subdomain nào mà có cùng đường dẫn URI.

Tiền tố trong route

Phương thức `prefix` có thể được sử dụng để chia tiền tố từng route thành nhóm với một URI nào đó. Ví dụ, bạn có thể muốn chia tiền tố cho các URI trong nhóm với tiền tố `admin`.

```php
Route::prefix('admin')->group(function () {
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

Chia tiền tố theo tên

Phương thức `name` có thể được sử dụng trong việc chia tiền tố cho từng route theo tên thành nhóm với một chuỗi nào đó. Ví dụ, bạn có thể muốn chia tiền tố cho tất cả các tên của các route trong một nhóm `admin`. Chuỗi được đem đi chia tiền tố sẽ được nối ngay trước với tên của route, nên phải đảm bảo là đã cung cập thêm ký tự tiếp nối (`.`) trong tiền tố cụ thể.

```
Route::name('admin.')->group(function () {
    Route::get('/users', function () {
        // Route assigned name "admin.users"...
    })->name('users');
});
```

Ràng buộc route model

Khi tiêm chèn một model ID vào trong action của controller, bạn sẽ thường truy xuất database để nhận về model tương ứng với ID đó. Gán model vào route trong Laravel là một cách thuận tiện để tự động tiêm đối tượng model vào trong route của bạn một cách trực tiếp. Ví dụ, thay vì chèn vào ID của một user, thì bạn có thể chèn vào nguyên bộ đối tượng model User mà tương đương với ID đó.

Ràng buộc ngầm

Laravel tự động resolve các model Eloquent được khai báo trong các route và các action của controller nơi mà được khai báo kiểu cho các biến có tên khớp với tên phân đoạn trên route. Ví dụ

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
    return $user->email;
});
```

Do biến `$user` được khai báo kiểu theo model Eloquent `App\Models\User` và tên biến khớp với phân đoạn URI `{user}`. Laravel sẽ tự động tiêm đối tượng model nào có ID trùng với giá trị phù hợp từ URI của request. Nếu một đối tượng model trùng khớp không được

tìm thấy trong database, một hồi đáp *404 HTTP* sẽ được tự động tạo ra.

Tất nhiên ràng buộc ngầm còn rất khả thi trong việc sử dụng các phương thức controller. Chú ý kỹ phân đoạn URI khớp với biến `$user` trong controller chứa một khai báo kiểu `App\Models\User`:

```
use App\Http\Controllers\UserController;
use App\Models\User;

// Route definition...
Route::get('/users/{user}', [UserController::class, 'show']);

// Controller method definition...
public function show(User $user)
{
    return view('user.profile', ['user' => $user]);
}
```

Các model bị xóa mềm

Nhìn chung, việc ràng buộc model ngầm sẽ không lấy được các model bị xóa mềm. Tuy nhiên, bạn có thể cấu trúc việc ràng buộc ngầm sao cho để lấy được những model này bằng cách gọi phương thức withTrashed trên khai báo route của bạn.

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
    return $user->email;
})->withTrashed();
```

Tùy biến khóa

Thỉnh thoảng bạn sẽ muốn resolve các model Eloquent bằng cách sử dụng một column nào đó khác `id`. Để thực hiện việc này, bạn có thể khai báo cột trong khai báo tham số route.

```
use App\Models\Post;
```

```
Route::get('/posts/{post:slug}', function (Post $post) {
    return $post;
});
```

Nếu bạn muốn ràng buộc model để luôn sử dụng một cột nào đó khác `id` khi nhận một class model nào đó, thì bạn có thể chồng chéo phương thức `getRouteKeyName` trên model Eloquent.

```
/**
 * Get the route key for the model.
 *
 * @return string
 */
public function getRouteKeyName()
{
    return 'slug';
}
```

Tùy biến khóa và phạm vi

Khi liên kết ngầm nhiều mô hình Eloquent trong một khai báo route duy nhất, bạn có thể muốn xác định phạm vi mô hình Eloquent thứ hai sao cho nó phải là con của mô hình Eloquent trước đó. Ví dụ: hãy xem qua khai báo này truy xuất một bài đăng trên blog bằng khóa `slug` cho một người dùng cụ thể:

```
use App\Models\Post;
use App\Models\User;

Route::get('/users/{user}/posts/{post:slug}', function (User $user, Post $post) {
    return $post;
});
```

Khi sử dụng liên kết ngầm có khóa tùy biến làm tham số route được lồng vào nhau, Laravel sẽ tự động phân phạm vi truy vấn để truy xuất mô hình lồng nhau bởi cha của nó bằng cách sử dụng các quy ước để đoán tên mối quan hệ trên model cha. Trong trường hợp này, giả

định mô hình `User` có một mối quan hệ có tên là `posts` (dạng số nhiều của tên tham số route) có thể được sử dụng để truy xuất mô hình `Post`.

Nếu muốn, bạn cũng có thể để Laravel xác định các ràng buộc "con" ngay cả khi custom key không có. Để làm như vậy, bạn có thể gọi phương thức `scopeBindings` khi khai báo route của mình:

```
use App\Models\Post;
use App\Models\User;

Route::get('/users/{user}/posts/{post}', function (User $user, Post $post) {
    return $post;
})->scopeBindings();
```

Hoặc, bạn có thể để cho toàn bộ nhóm khai báo route sử dụng liên kết phạm vi:

```
Route::scopeBindings()->group(function () {
    Route::get('/users/{user}/posts/{post}', function (User $user, Post $post) {
        return $post;
    });
});
```

Tùy biến hoạt động model bị thiếu

Thông thường, một phản hồi `HTTP 404` sẽ được tạo nếu không tìm thấy mô hình liên kết ngầm. Tuy nhiên, bạn có thể tùy chỉnh hành vi này bằng cách gọi phương thức bị thiếu khi khai báo route của bạn. Phương thức bị thiếu chấp nhận một closure sẽ được gọi đến nếu không thể tìm thấy một mô hình liên kết ngầm:

```
use App\Http\Controllers\LocationsController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;

Route::get('/locations/{location:slug}', [LocationsController::class, 'show'])
  ->name('locations.view')
  ->missing(function (Request $request) {
    return Redirect::route('locations.index');
```

```
});
```

Ràng buộc công khai

Bạn không cần phải sử dụng quy ước ngầm định của Laravel để ràng buộc mô hình. Bạn cũng có thể khai báo một cách công khai cách mà các tham số route tương thích với các mô hình. Để thiết lập một ràng buộc công khai, hãy sử dụng phương thức `model` của router để thiết lập class cho tham số đó. Bạn nên khai báo ràng buộc công khai mô hình ở phần đầu của phương thức `boot` trong class `RouteServiceProvider`.

```
use App\Models\User;
use Illuminate\Support\Facades\Route;

/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
public function boot()
{
  Route::model('user', User::class);
  // ...
}
```

Kế tiếp đó chỉ cần khai báo một route có một tham số là `{user}`.

```
use App\Models\User;

Route::get('/users/{user}', function (User $user) {
  //
});
```

Do chúng ta đã ràng buộc tất cả các tham số `{user}` với model `App\Models\User`, nên một instance của class sẽ được tiêm vào trong route. Vì vậy, ví dụ một request là *users/1* sẽ tiêm instance `User` từ database có `ID` người dùng là 1.

Nếu không một instance nào có ID đang cần tìm, thì một hồi đáp mã 404 sẽ được tự động sinh ra.

Tùy biến logic của phương thức

Nếu bạn muốn tùy biến logic cho phương thức ràng buộc mô hình của riêng mình thì bạn có thể sử dụng phương thức Route::bind. Hàm closure mà được truyền vào tham số của phương thức bind sẽ nhận giá trị của phân khúc URI và trả lại instance của class mà đã được tiêm vào trong route. Chú ý, tùy biến kiểu này nên đặt vào trong phương thức boot của class RouteServiceProvider của ứng dụng của bạn.

```
use App\Models\User;
use Illuminate\Support\Facades\Route;

/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
public function boot()
{
  Route::bind('user', function ($value) {
    return User::where('name', $value)->firstOrFail();
  });
  // ...
}
```

Ngoài ra, bạn còn có thể chèn đè lên phương thức `resolveRouteBinding` của model Eloquent của bạn. Phương thức này sẽ nhận giá trị của phân khúc URI và sẽ trả lại instance của class mà được tiêm vào trong route.

```
/**
 * Retrieve the model for a bound value.
 *
 * @param  mixed  $value
 * @param  string|null  $field
 * @return \Illuminate\Database\Eloquent\Model|null
```

```
     */
    public function resolveRouteBinding($value, $field = null)
    {
        return $this->where('name', $value)->firstOrFail();
    }
```

Nếu một route đang sử dụng ràng buộc ngầm, thì phương thức `resolveChildRouteBinding` sẽ được sử dụng để lấy ràng buộc con của mô hình cha.

```
    /**
     * Retrieve the child model for a bound value.
     *
     * @param  string  $childType
     * @param  mixed  $value
     * @param  string|null  $field
     * @return \Illuminate\Database\Eloquent\Model|null
     */
    public function resolveChildRouteBinding($childType, $value, $field)
    {
        return parent::resolveChildRouteBinding($childType, $value, $field);
    }
```

Các route Fallback

Bằng việc sử dụng phương thức `Route::fallback`, bạn có thể khai báo một route mà sẽ được thực thi khi không một route nào khác khớp với request hiện tại. Nói chung, các request không kiểm soát thường được tự động render với một trang mã `404` thông qua hàm xử lý exception trong ứng dụng của bạn. Tuy nhiên, khi bạn khai báo route `fallback` trong tập tin *routes/web.php*, tất cả các middleware của nhóm web sẽ được áp dụng vào route này. Cũng như bạn có thể thoải mái thêm bất kỳ middleware nào khác vào trong route này khi cần thiết.

```
Route::fallback(function () {
    //
});
```

Chú ý: Route `fallback` nên được thiết lập cuối cùng trong các route của ứng dụng.

Rate Limit

Laravel có các tính năng rate limit mạnh mẽ và linh hoạt mà bạn có thể ứng dụng để hạn chế lưu lượng truyền tải của một route nào đó hay một nhóm route nào đó.

Khai báo Rate Limiter

Để bắt đầu, chúng ta nên khai báo các cấu hình rate limiter mà ứng dụng của bạn đang cần. Thông thường, nó được thực hiện với phương thức `configureRateLimiting` của class `App\Providers\RouteServiceProvider` trong ứng dụng của bạn.

Rate limiter được khai báo bằng phương thức `for` của facade `RateLimiter`, phương thức `for` có tham số tên của rate limiter và hàm closure sẽ trả lại cấu hình limit mà sẽ được áp dụng cho các route đã được gắn với rate limiter. Cấu hình limit là các instance của class `Illuminate\Cache\RateLimiting\Limit`, các class này chứa các phương thức builder hữu dụng mà bạn có thể khai báo nhanh chóng các cấu hình limit của bạn. Tên của rate limiter có thể là bất cứ thứ gì mà bạn muốn.

```
use Illuminate\Cache\RateLimiting\Limit;
use Illuminate\Support\Facades\RateLimiter;

/**
 * Configure the rate limiters for the application.
 *
 * @return void
 */
protected function configureRateLimiting()
{
  RateLimiter::for('global', function (Request $request) {
    return Limit::perMinute(1000);
  });
}
```

Nếu như request gửi tới vượt quá mức giới hạn chỉ định, thì một hồi đáp mã `429` sẽ được tự động trả lại bởi Laravel. Nếu bạn muốn tự khai báo một hồi đáp riêng mà được trả lại bởi

một rate limiter, thì bạn có thể sử dụng phương thức `response`.

```
RateLimiter::for('global', function (Request $request) {
  return Limit::perMinute(1000)->response(function () {
    return response('Custom response...', 429);
  });
});
```

Do các callback của rate limiter sẽ nhận HTTP request gửi đến, nên bạn có thể xây dựng một các linh động rate limit phù hợp với request gửi đến hay quyền người dùng.

```
RateLimiter::for('uploads', function (Request $request) {
  return $request->user()->vipCustomer() ? Limit::none() : Limit::perMinute(100);
});
```

Phân đoạn các rate limit

Đôi khi bạn có thể muốn phân đoạn ratelimit theo một vài giá trị tùy ý. Ví dụ: bạn có thể muốn cho phép người dùng truy cập một route nhất định với 100 lần mỗi phút trên mỗi địa chỉ IP. Để thực hiện điều này, bạn có thể sử dụng phương pháp sau đây khi xây dựng ratelimit cho route của mình:

```
RateLimiter::for('uploads', function (Request $request) {
  return $request->user()->vipCustomer()
    ? Limit::none()
    : Limit::perMinute(100)->by($request->ip());
});
```

Hãy minh họa tính năng này bằng một ví dụ khác, chúng ta có thể giới hạn quyền truy cập vào route ở mức 100 lần mỗi phút cho mỗi ID người dùng được xác thực hoặc 10 lần mỗi phút cho mỗi địa chỉ IP đối với khách:

```
RateLimiter::for('uploads', function (Request $request) {
  return $request->user()
    ? Limit::perMinute(100)->by($request->user()->id)
    : Limit::perMinute(10)->by($request->ip());
```

```
});
```

Làm việc với nhiều RateLimit

Nếu cần, bạn có thể trả về một loạt các ratelimit cho cấu hình ratelimiter nào đó. Mỗi ratelimit sẽ tương ứng cho route dựa trên thứ tự mà chúng được đặt trong mảng:

```
RateLimiter::for('login', function (Request $request) {
  return [
    Limit::perMinute(500),
    Limit::perMinute(3)->by($request->input('email')),
  ];
});
```

Đưa Rate limiter vào các route

Các ratelimiter có thể được gắn vào các route hoặc nhóm route bằng cách sử dụng middleware *throttle*. Middleware *throttlee* chấp nhận tên của bộ ratelimit mà bạn muốn đưa cho route:

```
Route::middleware(['throttle:uploads'])->group(function () {
  Route::post('/audio', function () {
    //
  });

  Route::post('/video', function () {
    //
  });
});
```

Dùng middleware *throttle* với Redis

Thông thường, middleware *throttle* được map tới lớp `Illuminate\Routing\Middleware\ThrottleRequests`. Việc map được khai báo trong kernel HTTP của ứng dụng của bạn (*App\Http\Kernel*). Tuy nhiên, nếu bạn đang sử dụng Redis làm bộ nhớ cache cho ứng dụng, bạn có thể muốn thay đổi sang sử dụng lớp `Illuminate\Routing`

`\Middleware\ThrottleRequestsWithRedis`. Lớp này hiệu quả hơn trong việc quản lý ratelimiter bằng cách sử dụng Redis:

```
'throttle' => \Illuminate\Routing\Middleware\ThrottleRequestsWithRedis::class,
```

Giả mạo phương thức form

Các HTML form không hỗ trợ các phương thức `PUT`, `PATCH` hoặc `DELETE`. Vì vậy, khi xác định các request `PUT`, `PATCH` hoặc `DELETE` được gọi từ HTML form, bạn sẽ cần thêm field ẩn `_method` vào trong form. Giá trị được gửi vào field `_method` sẽ được sử dụng làm phương thức cho HTTP request:

```
<form action="/example" method="POST">
  <input type="hidden" name="_method" value="PUT">
  <input type="hidden" name="_token" value="{{ csrf_token() }}">
</form>
```

Để thuận tiện, bạn có thể sử dụng *directive* `@method` trong *blade* để tạo *field* `_method`:

```
<form action="/example" method="POST">
  @method('PUT')
  @csrf
</form>
```

Truy cập vào route hiện tại

Bạn có thể sử dụng các phương thức `current`, `currentRouteName` và `currentRouteAction` của Route để truy cập thông tin về route đang xử lý request được gửi đến:

```
use Illuminate\Support\Facades\Route;
$route = Route::current();            // Illuminate\Routing\Route
$name = Route::currentRouteName();    // string
$action = Route::currentRouteAction(); // string
```

Bạn có thể tham khảo tài liệu API cho cả class cơ bản của facade `Route` và các đối tượng `Route` để review tất cả các phương thức bên trong router và các class route.

Cross-Origin Resource Sharing (CORS)

Laravel có thể tự động phản hồi các CORS HTTP request có `OPTIONS` với các giá trị mà bạn định cấu hình. Tất cả cài đặt CORS có thể được cấu hình trong tập tin cấu hình *config/cors.php* trong ứng dụng của bạn. Các `OPTIONS` request sẽ tự động được xử lý bởi *HandleCors* middleware được đem vào theo mặc định trong ngăn xếp middleware tổng thể của ứng dụng. Ngăn xếp middleware tổng thể được đặt trong kernel HTTP của ứng dụng (*App\Http\Kernel*).

> Để biết thêm thông tin về CORS và các CORS headers, hãy tham khảo MDN web documentation on CORS.

Cache các route của Laravel

Khi phát hành ứng dụng của bạn, bạn nên tận dụng tính năng *route cache* của Laravel. Việc sử dụng *route cache* sẽ giảm đáng kể thời gian đăng ký tất cả các route vào trong ứng dụng của bạn. Để tạo *route cache*, hãy thực hiện lệnh Artisan `route:cache`:

```
php artisan route:cache
```

Sau khi chạy lệnh này, các tập tin *route cache* sẽ được tải theo mọi request gửi đến ứng dụng. Hãy ghi nhớ, nếu bạn thêm bất kỳ route mới nào, bạn sẽ cần tạo lại một bộ *route cache* mới. Do đó, bạn chỉ nên chạy lệnh `route:cache` trong quá trình phát hành dự án của mình.

Bạn có thể sử dụng lệnh `route:clear` để xóa đi các tập tin *route cache*.

```
php artisan route:clear
```