ROUTING CƠ BẢN VÀ HTTP RESPONSES

1. Routing là gì?

Route thực hiện chức năng định tuyến, đường dẫn cho các HTTP request được gửi đến đúng nơi ta muốn nó đến. Nếu như khi làm một project bằng PHP thuần thì chúng ta sử dụng file htaccess dùng để cấu hình máy chủ apache, tức là khi người dùng đánh đường dẫn trên browser thì đường dẫn đó sẽ map trong file htaccess để điều hướng đến các page của trang web, nhưng trong Laravel thì Route sẽ thực hiện điều đó.

2. Rounting cơ bản

Mặc định để khai báo các route dành cho web thì khai báo trong hai file `routes/web.php` hoặc `routes/api.php`. Trong đó, nếu khai báo route trong `routes/api.php` thì các path sẽ được thêm tiền tố api ở phía trước.

Ngoài ra nếu muốn khai báo route trong file khác thì cần khai báo file đó `app/Providers/RouteServiceProvider.php`.

Cấu trúc của một Routing

Route trong Laravel hỗ trợ định nghĩa cho tất cả các HTTP request method như sau:

```
Route::METHOD('URL', ACTION);
```

Trong đó METHOD gồm:

`Route::get($uri, $callback)` //Nhận resquest với phương thức `GET`.

`Route::post($uri, $callback)` //Nhận resquest với phương thức `POST`.

`Route::put($uri, $callback)` //Nhận resquest với phương thức `PUT`.

`Route::patch($uri, $callback)` //Nhận resquest với phương thức `PATCH`.

`Route::delete($uri, $callback)` //Nhận resquest với phương thức `DELETE`.

`Route::options($uri, $callback)` //Nhận resquest với phương thức `OPTIONS`.

Hoặc nếu muốn định nghĩa một route hỗ trợ nhiều method thì dùnng phương thức `match` theo cú pháp:

Route::match(['METHOD1', 'METHOD2',...], $uri, $callback);

// Nhận resquest với các phương thức kết hợp.

Hoặc là nhận tất cả các HTTP method với phương thức any:

Route::any($uri, $callback); // Nhận resquest với tất cả các phương thức.

Nhóm các route:

Route::group('prefix'=>$uri, $callback); // Nhóm các route.

Trong các cú pháp trên:

- `$uri` là đường dẫn (path) muốn route xử lý.
- - `$callback` là một callback, callable hoặc một array chứa thông tin controller và phương thức tác động đến.

Ví dụ: Khai báo route `hello` sẽ trả về chuỗi "web -> Xin chào" khi truy cập vào path `/hello`.

```php
<?php
//Tập tin routes\web.php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\CarController;

Route::get('/hello', function () {
    return "web -> Xin chào!";
});
```

Kết quả:

Ví dụ: Khai báo route `api/xin-chao` cũng sẽ trả về chuỗi "api -> Xin chào!". Nhưng sẽ đặt trong file `routes/api.php`.

```php
<?php
//Tập tin routes\api.php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::get('/xin-chao', function () {
```

```
    return "api -> Xin chào!";
});
```

Kết quả:

Trong ví dụ trên route được viết trong file routes/api.php, nên đường dẫn (path) sẽ tự động thêm api vào trước /xin-chao, nênđường dẫn localhost:8000/api/xin-chao.

Nếu muốn đưa logic vào trong controller thì sử dụng cú pháp sau:

```
Route::RouteMethod($uri, [$ControllerName, $method]);
```

- `RouteMethod` là method action của Routes.
- `$uri` là đường dẫn muốn xử lý.
- `$ControllerName` là controller chứa logic.
- `$method` là phương thức trong controller chứa logic để xử lý cho route.

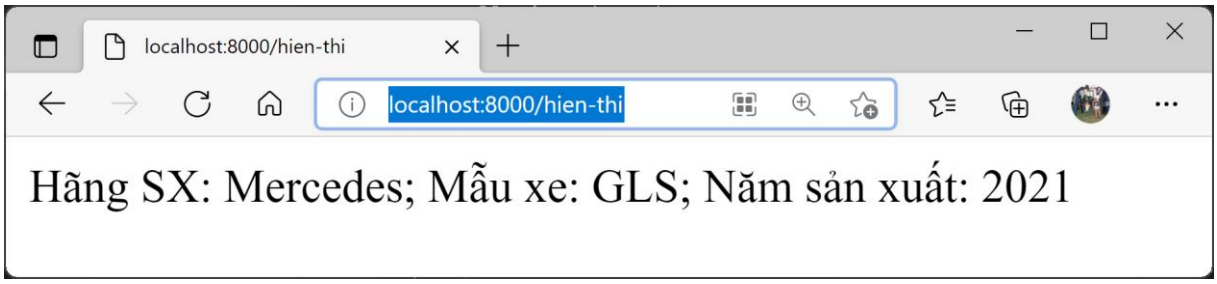
Vú dụ: Có controller tên CarController (app/Http/Controllers/CarController.php):

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Car;

class CarController extends Controller
{
    public function hien_thi()
    {
        return "Hãng SX: Mercedes; Mẫu xe: GLS; Năm sản xuất: 2021";
    }
}
```

Khi đó viết một route như sau:

```
Route::get('/hien-thi', [\App\Http\Controllers\CarController::class, 'hien_thi']);
```

Kết quả:



CSRF token trong Route

Bất kỳ forms HTML nào trỏ đến các route POST, PUT, PATCH hoặc DELETE được địnhnghĩa trong file định tuyến web phải bao gồm trường mã thông báo CSRF. Nếu không, yêu cầu sẽ bị từ chối. Có thể đọc thêm về CSRF protection trong tài liệu CSRF:

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

Route redirect

Để định nghĩa một route chuyển hướng đến URI khác, khi đó sử dụng phương thức **`Route::redirect`**. Phương pháp này cung cấp một cú pháp đơn giản thay vì khai báo một route đầy đủ hay một controller để thực hiện việc chuyển hướng.

Cú pháp:

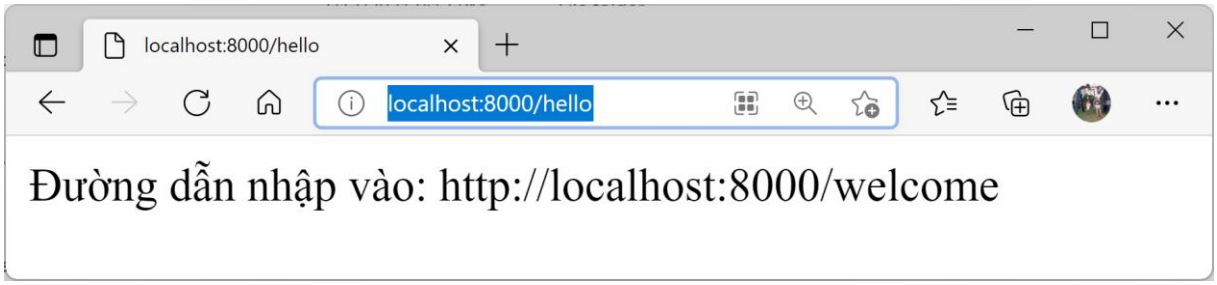
```
Route::redirect($uri, $redirectTo, $status)
```

Trong đó:

- `$uri` là đường dẫn (path) muốn xử lý.
- `$redirectTo` là đường dẫn (path) muốn chuyển hướng (redirect) đến.
- `$status` là http status mà các bạn muốn thiết lập, mặc định `$status` là 302.

Ví dụ:

```
Route::redirect('/welcome', '/hello');
```



Mặc định, `Route::redirect` trả về mã 302 nhưng Laravel cho phép bạn có thể thay đổi mã này bằng cách thêm tham số thứ ba.

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

Nếu không muốn thêm quá nhiều tham số, có thể sử dụng `Route::permanentRedirect` để trả về mã 301.

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

Lưu ý: Việc lựa chọn mã redirect phù hợp sẽ giúp cho việc SEO tốt hơn. Vậy khi nào redirect với 301, khi nào redirect với 302:

- Redirect với 301 thông báo rằng trang web này đã chuyển hướng vĩnh viễn, thường là thay đổi domain mới.
- Redirect với 302 thông báo rằng trang web này chỉ chuyển hướng tạm thời, thường là do bảo trì.

View Routes

Nếu muốn route được khai báo chỉ nhằm mục đích trả về View thì có thể sử dụng method Route::view thay vì phải định nghĩa một route đầy đủ hoặc controller.

Cú pháp:

```
Route::view($uri, $viewName, $data)
```

Trong đó:

- `$uri` là đường dẫn (path) muốn xử lý.
- `$viewName` là view muốn render.
- `$data` data muốn truyền vào view.
- Ngoài ra, còn có thể truyền thêm 2 tham số khác vào đó là `http status` và `header`.

Chú ý: Route view chỉ làm việc với HTTP request GET hoặc HEAD.

Ví dụ:

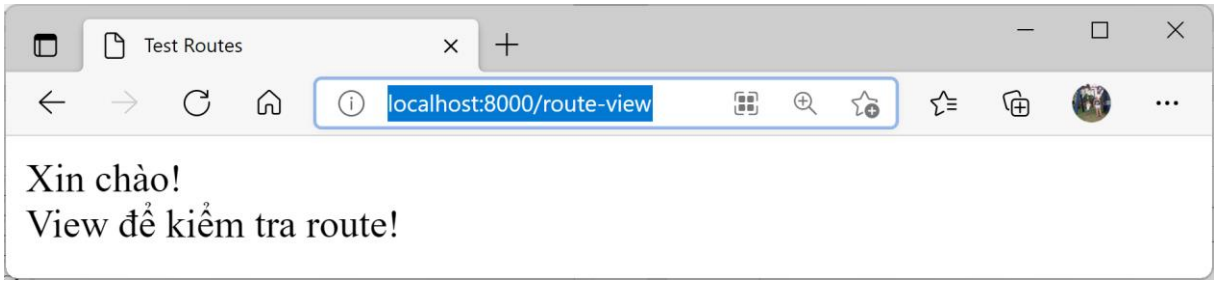
```
Route::view('/route-view', 'welcome');
```

Ở đây sử dụng một file view, có tên là resources/views/test-view.blade.php.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Test Routes</title>
    </head>
    <body>
        Xin chào!
        <br>View để kiểm tra route!
    </body>
</html>
```

Bây giờ thử truy cập đường dẫn http://localhost:8000/route-view để xem kết quả.

Kết quả:



Ví dụ: Cho tham số thứ ba trong method Route::view:

```
Route::view('/route-view', 'test-route', ['name' => 'Nguyễn Đức Duy']);
```

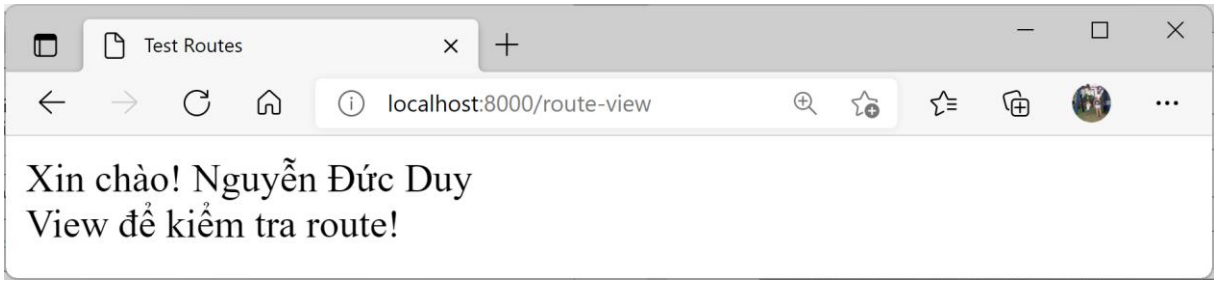
Trong route đã truyền một mảng dữ liệu vào file view resources/views/ test-view.blade.php. Bây giờ sửa code như sau: tìm đến dòng code có chứa chuỗi "Xin chào!" và chỉnh code để có thể nhận dữ liệu và in ra màn hình.

```
    <body>
        Xin chào!
        <?php
            echo $name;
        ?>
```

```
        <br>View để kiểm tra route!
    </body>
```

Ở đây, key `name` trong mảng dữ liệu của route đã được đổi thành tên biến `$name` chứa giá trị tương ứng.

Kết quả:



3. Truyền tham số trong route (Route Parameters)

URI có thể chứa cả tham số, nhưng chẳng lẽ phải khai báo từng giá trị mặc định trong routing? Trong Laravel để route có thể nhận tham số (parameter) thì chỉ cần thêm cú pháp {name} vào trong $uri với name là định danh cho parameter đó.

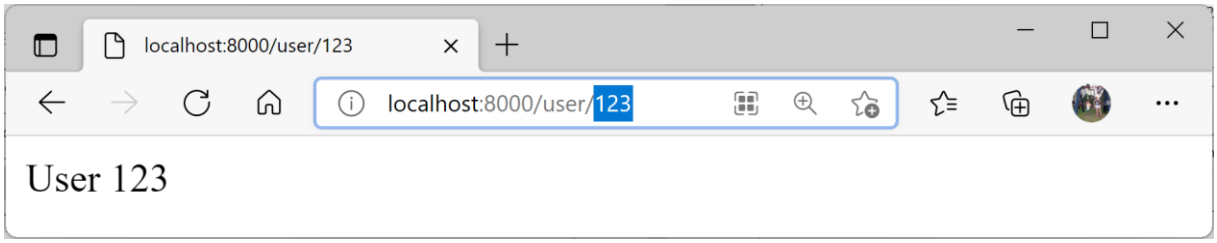
Các tham số bắt buộc (Required parameters)

Chẳng hạn có một URI /user sẽ nhận tham số id để hiển thị thông tin của user. Định nghĩa route parameter như sau:

- Để khai báo tham số trong URI, ta đặt nó nằm trong cặp {};
- Tên tham số chỉ chứa ký tự chữ cái;
- Để nhận giá trị từ tham số, ta khai báo trong Closure object/Controller method.

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

Kết quả:



Có thể thêm nhiều tham số trong cùng một URI. Định nghĩa như sau:

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

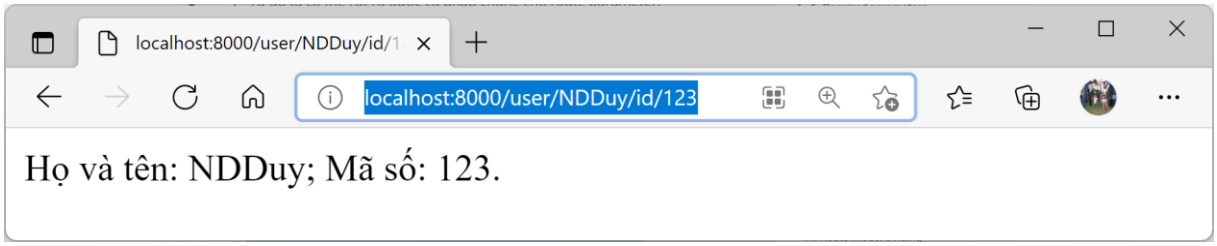
```
    return "Họ và tên: $name; Mã số: $id.";
});
```

Trong đoạn code trên ta có thể thấy $name sẽ nhận giá trị từ {name}, cũng như $id nhận giá trị từ {id}. Có thể đặt tên biến nhận giá trị khác với tên tham số khai báo trong URI.

Nhận xét:

- Thứ tự các biến nhận giá trị tham số phải theo thứ tự khái báo tham số trong URI.
- Tên tham số và tên biến nhận giá trị tham số có thể khác nhau.

Kết quả:



Parameters & Dependency Injection

Dependency Injection là một thuật ngữ, chỉ việc giải quyết các phục thuộc của một object thông qua constructor. Trong Laravel dependency injection xuất hiện ở mọi nơi và đương nhiên đối với callback trong route cũng có.

Ví dụ: Dependency Request vào route.

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

Đối với các route có parameter sẽ phải để các tham số dependency phía trước, sau đó các parameter sẽ đặt sau các dependency.

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

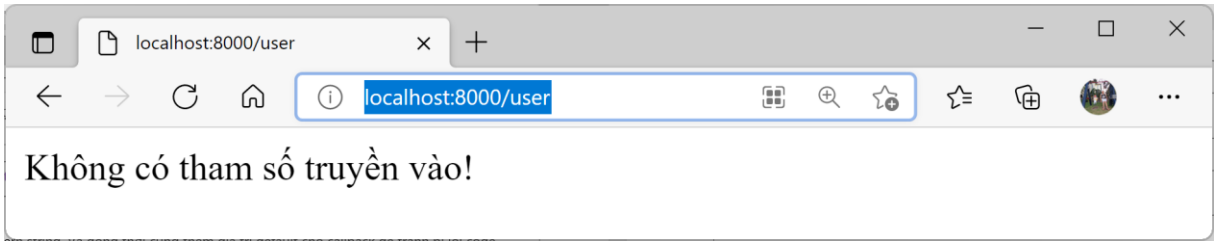
Các tham số tùy chọn (Optional paramters)

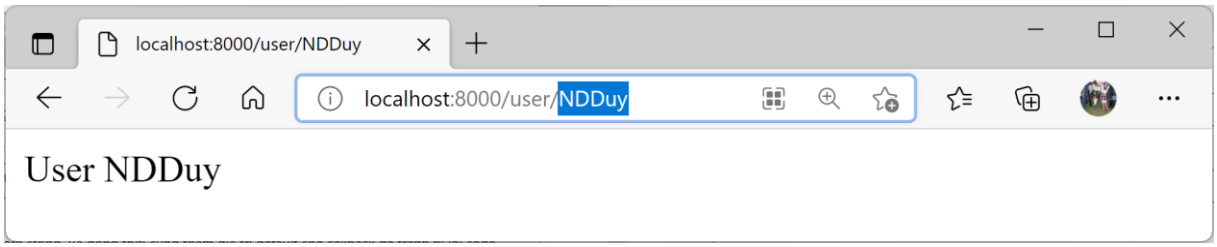
Nếu muốn truyền vào URL một parameter không bắt buộc thì chỉ cần thêm dấu ? vào sau partern string. Và đồng thời cũng thêm giá trị default cho callback để tránh bị lỗi code.

Ví dụ:

```php
Route::get('/user/{name?}', function ($name = null) {
    if ($name == null) {
        return 'Không có tham số truyền vào!';
    }

    return $name;
});
```

Kết quả:





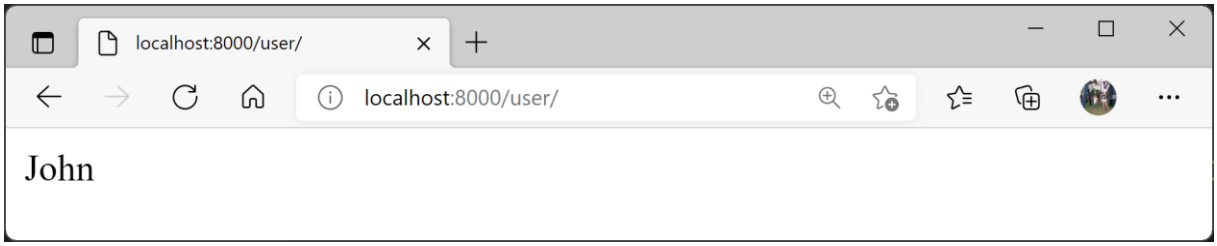
Nhận xét: Để khai báo một tham số tùy chọn (có thể có hoặc không)

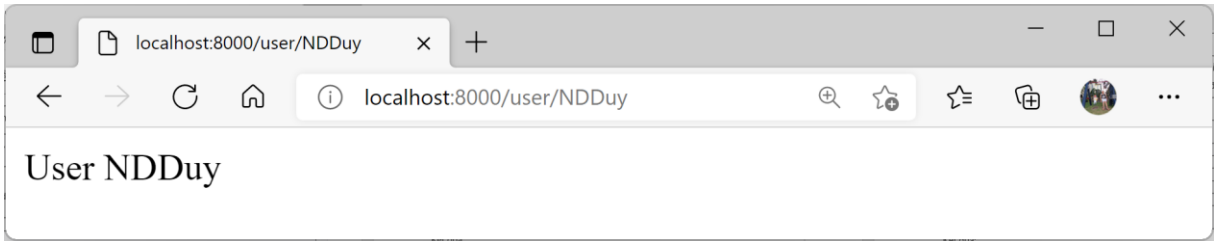
- Thêm dấu `?` sau tên tham số;
- Gán giá trị mặc định cho biến chứa giá trị tham số (có thể null).

Ví dụ:

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

Kết quả:





Ví dụ:

```
Route::get('/id/{id?}/user/{name}', function ($id = 1, $name) {
    return "Mã số: $id; Họ và tên: $name.";
});
```

Bây giờ nhập đường dẫn http://localhost:8000/id/name/NDDuy , chắc chắn sẽ báo lỗi 404.

Lưu ý:

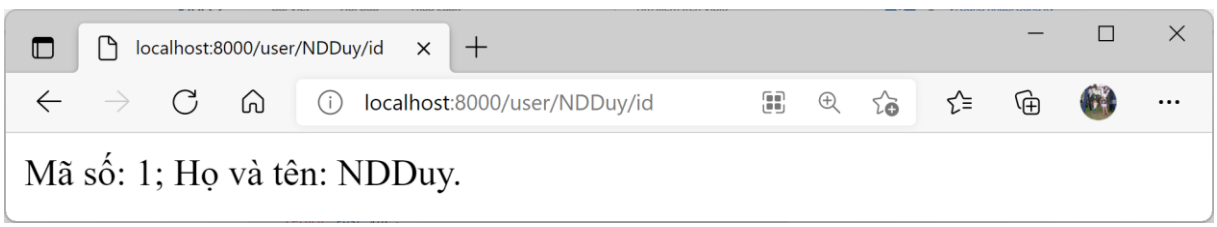
Trong route trên, nếu không truyền id cho id thì nó sẽ mặc định nhận là 1. Nhưng lưu ý, trong Laravel khi khai báo một optional parameter thì thỏa 2 điều liện:

- Phải đặt nó ở cuối URI;
- Trong một URI chỉ chứa duy nhất một optional parameter.

Ví dụ:

```
Route::get('/user/{name}/id/{id?}', function ($name, $id = 1) {
    return "Mã số: $id; Họ và tên: $name.";
});
```

Kết quả:



Regular Expression Constraints

Có thể giới hạn định dạng của các tham số route bằng cách sử dụng phương thức `where` trên một thể hiện của route. Phương thức where chấp nhận tên của tham số và biểu thức chính quy xác định cách hạn chế tham số:

Cú pháp:

```
where($name, $pattern)

// or

where([$name => $pattern])
```

Trong đó:

- `$name` là parameter name;
- `$pattern` là biểu thức regular expression.

Ví dụ:

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

Tham số thứ nhất mà where sẽ nhận là tên tham số cần ràng buộc, tham số thứ hai sẽ là chuỗi pattern do ta định nghĩa. Nếu tham số không phù hợp với ràng buộc sẽ trả về lỗi 404.

Ví dụ:

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

Kết quả:

Có thể ràng buộc nhiều tham số khác nhau trong cùng một route. Ví dụ:

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

Lúc này **`where`** sẽ nhận một mảng chứa các ràng buộc cho từng tham số.

Ngoài ra Laravel có cung cấp sẵn một số phương thức `where` kèm với regex phổ biến như sau:

- `whereNumber($name)`: tương đương với `where($name, '[0-9]+')`
- `whereAlpha($name)`: tương đương với `where($name, '[a-zA-Z]+')`
- `whereAlphaNumeric($name)`: tương đương `where($name, '[a-zA-Z0-9]+')`
- `whereUuid($name)`: tương đương với `where($name, '[\da-fA-F]{8}-[\da-fA-F]{4}-[\da-fA-F]{4}-[\da-fA-F]{4}-[\da-fA-F]{12}')`

Global Constraints

Khi muốn định nghĩa pattern cho tất cả route parameter name nào đó. Có thể sử dụng:

```
Route::pattern($name, $pattern);
```

Trong đó:

- `$name` là parameter name;
- `$pattern` là biểu thức regular expression.

Ví dụ:

```
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');
```

Lúc này tất cả các route parameter được khai báo và thực thi sau `Route::pattern` sẽ hoạt

động. Để hoạt động tốt nhất đối với tất cả các loại route thì nên khai báo trong phương thức `boot` của file `app/Providers/RouteServiceProvider.php`

Ví dụ: Tất cả các route có parameter name là id đều phải là số.

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

Khi pattern đã được xác định, nó sẽ tự động được áp dụng cho tất cả các routes sử dụng tên tham số đó:

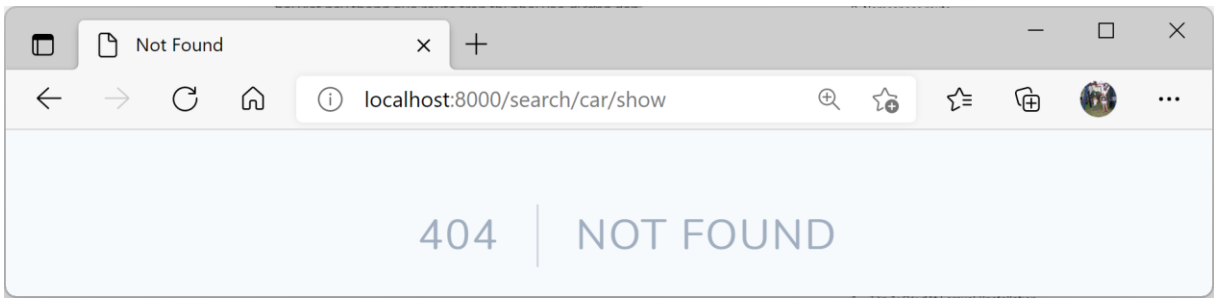
```
Route::get('/user/{id}', function ($id) {
    // Only executed if {id} is numeric...
});
```

Mã hóa '/' (Encoded forward Slashes)

Ví dụ khai báo một route để tìm kiếm như sau:

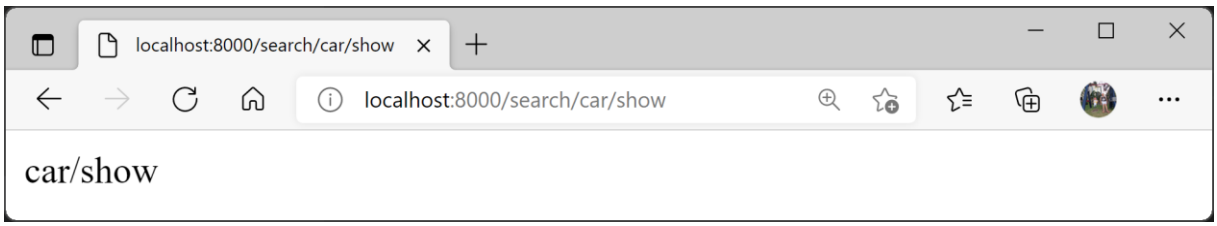
```
Route::get('search/{search}', function ($search) {
    return $search;
});
```

Giả sử tìm kiếm chuỗi "car/show:, nhập đường dẫn http://localhost:8000/search/car/show, nhưng lại báo lỗi 404.



Xảy ra lỗi trên là vì giá trị tham số `search` có chứa ký tự `/`, làm cho framework hiểu là đang

truy cập và một cấp con trong URI, chứ không là là giá trị bình thường của tham số `search`. Laravel đã khắc phục lỗi này bằng cách ràng buộc tham số chứa `/` với pattern là `.*`. Cần sửa đoạn code trên như sau:



Lưu ý: Trường hợp này được dùng khi làm chức năng tìm kiếm, nên để route này cuối cùng để tránh trường hợp trùng URI với các route khác.

4. Đặt tên cho route (Named Routes)

Thay vì phải nhớ URI của từng route, trong Laravel cho phép đặt tên cho route để tiện cho việc quản lý và generate ra route URL. Để gán tên cho route thì bạn chỉ cần sử dụng phương thức name hoặc as với cú pháp như sau:

```
name($name)

// hoặc

as($name)
```

Trong đó:

- `$name` là tên route muốn đặt.

Ví dụ: Đặt tên cho route có đường dẫn (path) là `/user/profile` tên là profile.

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

Có thể chỉ định tên route cho các hành động của controller:

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

Khi muốn đặt tiền tố (prefix) cho tất cả route trong group thì sử dụng phương thức `as` hoặc

name với cú pháp tương tự.

Ví dụ: Với phương thức as

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

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

Khi đó:

```
$url = route('user.profile'); //http://localhost:8000/user/profile
$url = route('user.setting'); //http://localhost:8000/user/setting
```

Ví dụ: Với phương thức name

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

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

Khi đó:

```
$url = route('user.profile'); //http://localhost:8000/user/profile
$url = route('user.setting'); //http://localhost:8000/user/setting
```

Generate URL sử dụng route name (Generating URLs To Named Routes)

Để tạo ra URL qua route name thì sử dụng phương thức `route()` với cú pháp sau:

```
Route($name, $parameter);
```

Trong đó:

- `$name` là tên của route;
- `$parameter` là một mảng chứa các parameter truyền vào route. Nếu route không có parameter đó thì nó sẽ trở thành URL Query String.

Ví dụ:

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

Với việc đặt tên sẽ dễ dàng lấy url từ route:

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

Khi đó giải trị $url là http://localhost:8000/profile

Cũng như chuyển hướng đến route đó:

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

Nếu route được đặt tên có chứa tham số, có thể truyền giá trị cho tham số bằng cách:

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

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

Khi đó giải trị $url là http://localhost:8000/user/1/profile

Nếu chuyển các tham số bổ sung vào mảng, các cặp key/value sẽ tự động được thêm vào chuỗi truy vấn của URL đã tạo:

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

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

Khi đó giải trị $url là http://localhost:8000/user/1/profile?photos=yes

Kiểm tra route hiện tại

Khi route đã được đặt tên, có thể kiểm tra request đang gửi tới có thuộc name nào không?

Ví dụ: Kiểm tra tên route hiện tại từ middleware:

```
public function handle($request, Closure $next)
{
    if ($request->route()->named('profile')) {
        //
    }

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

5. Nhóm route (Route group)

Các route trong cùng một nhóm sẽ được chia sẻ các thuộc tính route như namespace, middleware, tiền tố tên, tiền tố URI,...

Middleware

Middleware là một khái niệm chỉ việc cung cấp ra các bộ filter HTTP request cho ứng dụng. Ví dụ như khi người dùng tạo một POST request đến path `/user/create` thì sẽ loại bỏ các khoảng trống trong dữ liệu gửi lên.

Để gán middleware cho các route chung một nhóme, sử dụng phương thức `middleware()` với cú pháp:

```
middleware($middleware)
```

Trong đó:

- `$middleware` là một sting, array chứa middleware name.

Ví dụ:

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

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

```
    });
});
```

Subdomain Routing

Laravel cũng cung cấp nhóm sub-domain. Nhóm này có thể chỉ định tham số như một URI, cho phép giữ một phần sub-domain để sử dụng trong các route con.

Để khai báo nhóm sub-domain, bạn sử dụng method `domain` như sau:

```
domain($domain)
```

Trong đó:

- `$domain` là domain muốn xử lý.

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

Tham số `{account}` đóng vai trò như một thành phần của từng URI route con, nên thứ tự phải được đứng trước vì toàn bộ URI sẽ là `{account}.` `example.com/user/{id}`.

Lưu ý: Để đảm bảo các route sub-domain có thể hoạt động, nên khai báo các route này trước các route domain gốc, điều này sẽ ngăn không cho domain gốc ghi đè lên các sub-domain có cùng URI.

Tiền tố URI route (Route prefix)

Phương thức tiền tố (prefix) có thể được sử dụng để đặt tiền tố cho mỗi route trong nhóm với một URI nhất định. Ví dụ: Muốn đặt tiền tố cho tất cả các URI định tuyến trong nhóm với quản trị viên:

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

Tiền tố tên route (Route Name Prefixes)

Cũng như route prefix, route named prefix sẽ thêm tiền tố tên chung cho mỗi route con nằm trong nhóm.

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

Có thể thay thế ký tự ngăn cách `.` bằng ký tự khác, miễn sao phù hợp cú pháp code PHP.

Nếu có một nhóm route có cùng chung namespace, prefix, name, thì có thể gom các thuộc tính này vào `Route::group`.

```
Route::group([
    'namespace' => 'Admin',
    'prefix' => 'admin',
    'name' => 'admin.'
], function() {
    //
});
```

Lưu ý: Khi build app với routes/api.php thì các route được khai báo sẽ tự động đưa vào trong một route group có prefix là /api.

6. Route Model Binding

Khi inject một model instance theo ID nào đó vào route hoặc controller action, thông thường phải truy vấn đến model theo ID đã cho. Nhưng Laravel route model binding cung cấp một cú pháp để có thể tự động inject các model object trong route. Tức là thay vì chỉ inject ID của User rồi mới khởi tạo model thì ta sẽ inject luôn cả model object thông qua ID nhận từ tham số URI.

Binding ngầm (Implicit binding)

Laravel tự động giải quyết các mô hình Eloquent được xác định trong routes hoặc hành động của controller có tên biến trùng tên với tên tham số. Ví dụ:

```
use App\Models\User;

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

```
});
```

Kkhi truy cập đường dẫn http://localhost:8000//user/1, thì một model object sẽ được khởi tạo với ID bằng 1 từ database, sau đó inject vào route và trả về `$user->email`. Nếu không tồn tại user với ID bằng 1, thì ta sẽ nhận kết quả là lỗi 404.

Tất nhiên, ràng buộc ngầm cũng có thể xảy ra khi sử dụng các phương thức controller. Một lần nữa, hãy lưu ý rằng URI segment {user} khớp với biến $ user trong controller có chứa 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]);
}
```

Customizing The Key

Đôi khi muốn giải quyết các mô hình Eloquent bằng cách sử dụng một cột khác với id. Để làm như vậy, có thể chỉ định cột trong định nghĩa tham số định tuyến:

```
use App\Models\Post;

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

Mặc định thì Route Model Binding sẽ dùng ID để truy vấn vào database. Có thể thay đổi thiết lập này bằng cách khai báo method `getRouteKeyName` trong model muốn thay đổi.

```
/**
 * Get the route key for the model.
 *
 * @return string
 */
public function getRouteKeyName()
```

```
{
    return 'slug';
}
```

Binding rõ ràng (Explitcit binding)

Nếu muốn code trở nên rõ ràng, có thể sử dụng explitcit binding trong `RouteServiceProvider` tại `boot` bằng cách sử dụng method `Route::model`.

```
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);

    // ...
}
```

Trong đó:

- Tham số thứ nhất sẽ là tên tham số URI;
- Tham số thứ hai là class model.

Tiếp theo, định nghĩa route có chứa tham số {user}:

```
use App\Models\User;

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

Customizing The Resolution Logic

Nếu muốn sử dụng cách xử lý logic riêng, có thể sử dụng phương thức `Route::bind`. Closure object được truyền vào sẽ nhận giá trị của tham số trên URI và sẽ trả về model object cần để inject nếu thỏa mãn điều kiện đưa ra. Trong `RouteServiceProvider:`

```
public function boot()
{
    parent::boot();

    Route::bind('user', function ($value) {
        return App\User::where('name', $value)->first() ?? abort(404);
    });
}
```

Ở dòng `return App\User::where('name', $value)->first() ?? abort(404);` là xử lý logic kiểm tra xem có tồn tại user có trường `name` bằng giá trị `$value` không, nếu có thì return model object của user đó, còn không thì trả về lỗi 404.

Ngoài ra, tránh trường hợp code quá nhiều trong `RouteServiceProvider`, có thể định nghĩa xử lý logic riêng này vào model class muốn thông qua method `resolveRouteBinding`.

```
/**
 * Retrieve the model for a bound value.
 *
 * @param  mixed  $value
 * @return \Illuminate\Database\Eloquent\Model|null
 */
public function resolveRouteBinding($value)
{
    return $this->where('name', $value)->first() ?? abort(404);
}
```

7. Route dự phòng (Fallback Route)

Với fallback route, có thể thực hiện một xử lý nào đó khi không có bất kì route nào thỏa mãn với request, thường thì sử dụng để báo lỗi 404 và xử lý thêm vài công việc nào đó.

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

Lưu ý: Fallback Route phải được định nghĩa cuối cùng, sau cả các route mã hóa /.

8. Giới hạn truy cập (Rate Limiting)

Defining Rate Limiters

Laravel bao gồm các dịch vụ giới hạn tốc độ mạnh mẽ và có thể sử dụng để hạn chế lưu lượng

truy cập cho một route hoặc nhóm route nhất định. Để bắt đầu, nên xác định cấu hình giới hạn tốc độ đáp ứng nhu cầu của ứng dụng. Thông thường, điều này sẽ được thực hiện trong phương thức `configureRateLimiting` của lớp `App\Providers\RouteServiceProvider` của ứng dụng.

Rate limiters được định nghĩa bằng cách sử dụng `RateLimiter` facade của phương thức `for` . Phương thức `for` chấp nhận tên rate limiter và một hàm đóng trả về limit configuration sẽ áp dụng cho các route được gán ate limiter. Limit configuration là các instance của lớp `Illuminate\Cache\RateLimiting\Limit` . Lớp này chứa các phương thức "builder" hữu ích để có thể nhanh chóng xác định giới hạn. Tên rate limiter có thể là bất kỳ chuỗi nào:

```
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 request đến vượt quá giới hạn tốc độ đã chỉ định, response có mã trạng thái HTTP 429 sẽ tự động được trả lại bởi Laravel. Nếu muốn xác định phản hồi của riêng sẽ được trả về theo giới hạn tỷ lệ, có thể sử dụng phương thức `response` :

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

Vì các rate limiter callbacks nhận HTTP request instance, có thể xây dựng rate limit dynamically thích hợp dựa trên request hoặc authenticated user:

```
RateLimiter::for('uploads', function (Request $request) {
```

```
    return $request->user()->vipCustomer()
                ? Limit::none()
                : Limit::perMinute(100);
});
```

Segmenting Rate Limits

Đôi khi có thể muốn phân đoạn rate limits theo một số giá trị tùy ý. Ví dụ: muốn cho phép người dùng truy cập một toute nhất định 100 lần mỗi phút trên mỗi địa chỉ IP. Để thực hiện điều này, có thể sử dụng phương thức `by` khi xây dựng rate limit:

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

Để minh họa tính năng này bằng một ví dụ khác, 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());
});
```

Multiple Rate Limits

Nếu cần, có thể trả về mộtarray các rate limits cho rate limiter configuration. Mỗi rate limit sẽ được đánh giá cho route dựa trên thứ tự chúng được đặt trong array:

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

Attaching Rate Limiters To Routes

Rate limiters có thể được gắn vào các route hoặc nhóm route bằng cách sử dụng `throttle` middleware. Throttle middleware chấp nhận tên của rate limiter muốn gán cho route:

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

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

Throttling With Redis

Thông thường, `throttle` middleware được ánh xạ tới lớp `Illuminate\Routing\Middleware\ThrottleRequests`. Ánh xạ này được định nghĩa trong HTTP kernel (`App\Http\Kernel`) của ứng dụng. Tuy nhiên, nếu đang sử dụng Redis làm trình điều khiển bộ nhớ cache, có thể phải thay đổi ánh xạ này để sử dụng lớp `Illuminate\Routing\Middleware\ThrottleRequestsWithRedis`. Lớp này hiệu quả hơn trong việc quản lý limiting sửu dụng Redis:

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

9. Form Method Spoofing

HTML forms không hỗ trợ các hành động `PUT`, `PATCH`, hay `DELETE`. Vì vậy, khi định nghĩa các route `PUT`, `PATCH`, hay `DELETE` được gọi từ HTML forms, sẽ cần thêm hidden `_method` field vào form. Giá trị được gửi với `_method` field sẽ được sử dụng làm phương thức 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, có thể sử dụng chỉ thị `@method` Blade directive để tạo trường `_method` input field:

```
<form action="/example" method="POST">
```

```
    @method('PUT')
    @csrf
</form>
```

10. Truy cập route hiện tại (Accessing The Current Route)

Có thể sử dụng các phương thức `current`, `currentRouteName`, và `currentRouteAction` trong facade `Route` để truy cập thông tin về route đang xử lý:

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

$route = Route::current(); // Illuminate\Routing\Route
//Lấy ra current route dạng object (Illuminate\Routing\Route).
$name = Route::currentRouteName(); // string
//Lấy ra current route name. Nếu route không set name thì nó sẽ trả về null.
$action = Route::currentRouteAction(); // string
//Lấy ra current action của route, nếu sử dụng Controller làm action cho route.
```

11. Route Cache

Khi triển khai ứng dụng , nên tận dụng route cache của Laravel. Sử dụng route cache sẽ giảm đáng kể thời gian cần thiết để đăng ký tất cả các route của ứng dụng. Để tạo bộ đệm route cache, hãy thực hiện lệnh Artisan `route:cache`:

```
php artisan route:cache
```

Khi route đã được cache, laravel sẽ load data trong file cache ra mà không quan tâm đến file route nữa. Chính vì thế, khi có điều chỉnh gì trong route laravel cũng không nhận. Để apply lại route mới, có thể run lại lệnh cache một lần nữa. Hoặc clear route cache rồi cache lại.

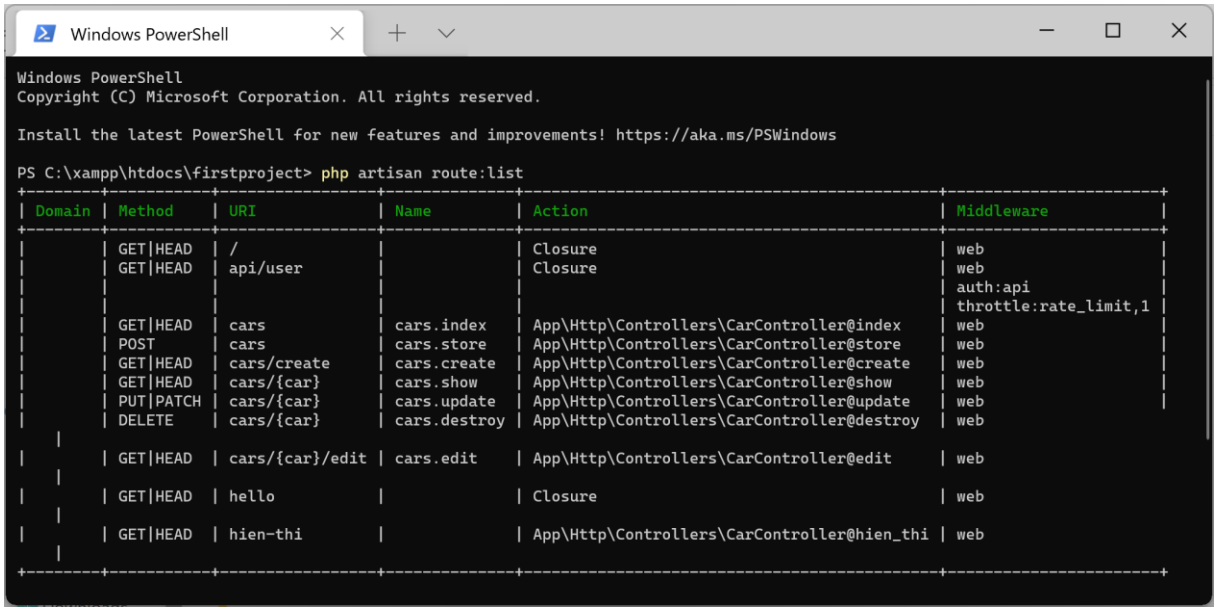
Có thể sử dụng lệnh `route:clear` để xóa bộ đệm của route:

```
php artisan route:clear
```

12. Hiển thị danh sách route

Để hiển thị ra danh sách route đã được đăng ký trong Laravel, sử dụng câu lệnh:

```
php artisan route:list
```



Ngoài ra, có thể thêm các option khác như:

- `--columns=[Column]` chọn các cột hiển thị.
- `--compact` chỉ hiển thị cột method, URI và Action.
- `--json` xuất data dưới dạng json.
- `--method=[Method]` lọc các route theo method.
- `--name[=NAME]` lọc theo route name.
- `--path[=PATH]` lọc theo path.
- `--reverse` đảo ngược thứ tự sắp xếp route.
- `--sort[=SORT]` sắp xếp route theo column. Default là sắp xếp theo URI.