Course - Laravel Framework

Middleware

Middleware cung cấp một cơ chế thuận tiện để kiểm tra và lọc các HTTP request gửi đến ứng dụng của bạn.

Tags: middleware, laravel

Giới thiệu

Ví dụ: Laravel có một *middleware* dùng để xác minh người dùng ứng dụng của bạn đã được xác thực hay không. Nếu người dùng không được xác thực, thì *middleware* sẽ chuyển hướng người dùng đến màn hình đăng nhập của ứng dụng. Tuy nhiên, nếu người dùng được xác thực, *middleware* sẽ cho phép người dùng truy cập tiếp vào ứng dụng.

Bên cạnh *middleware* auth, còn có các *middleware* khác được viết để thực hiện các công việc khác. Ví dụ, *middleware* logging sẽ ghi chép lại tất cả các yêu cầu gửi đến ứng dụng. Có rất nhiều *middleware* bên trong framework Laravel, như *middleware* auth và *bảo mật CSRF*. Hầu hết các *middleware* được đặt vào trong thư mục *app/Http/Middleware*.

Tạo Middleware

Để tạo ra một *middleware* mới, sử dụng lệnh Artisan `make:middleware`:

```
php artisan make:middleware EnsureTokenIsValid
```

Lệnh này sẽ đặt một class `EnsureTokenIsValid` mới trong thư mục *app/Http/Middleware* của bạn. Trong *middleware* này, chúng ta sẽ chỉ cho phép truy cập vào route nếu token đầu vào đã được cung cấp trước đó khớp với một giá trị cụ thể nào đó. Còn nếu không, thì chúng ta sẽ chuyển người dùng trở lại URI trang chủ:

```
<?php

namespace App\Http\Middleware;

use Closure;

class EnsureTokenIsValid
{
  /**
   * Handle an incoming request.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \Closure  $next
   * @return mixed
```

```
     */
    public function handle($request, Closure $next)
    {
        if ($request->input('token') !== 'my-secret-token')
        {
            return redirect('home');
        }
        return $next($request);
    }
}
```

Như bạn có thể thấy, nếu token đầu vào không khớp với mã bí mật của ví dụ, *middleware* sẽ chuyển HTTP đến trang *home*; còn ngược lại, thì request sẽ được chuyển tiếp vào ứng dụng. Để cho request vào tiếp ứng dụng (cho phép *middleware* "pass"), bạn nên gọi lệnh gọi callback trong biến `$next` với biến `$request`.

Tốt nhất bạn nên hình dung *middleware* như một loạt các yêu cầu mà các HTTP request phải vượt qua trước khi chúng truy cập vào ứng dụng của bạn. Từng yêu cầu này có thể kiểm tra các request và thậm chí có thể từ chối hoàn toàn các request này.

> Tất cả *middleware* đều sẽ được trả lại qua service container, vì vậy bạn có thể khai báo kiểu của bất kỳ class thư viện nào mà bạn cần trong constructor của một *middleware* nào đó.

Thực thi middleware và phản hồi sau đó

Một *middleware* có thể thực hiện các tasks trước hoặc sau khi đưa request vào bên trong ứng dụng. Ví dụ: *middleware* sau sẽ thực hiện một vài task trước khi request được ứng dụng xử lý:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
```

```
  public function handle($request, Closure $next)
  {
    // Perform action
    return $next($request);
  }
}
```

Tuy nhiên, một *middleware* cũng sẽ thực hiện task riêng của nó ngay sau khi ứng dụng đã xử lý request:

```
<?php

namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
  public function handle($request, Closure $next)
  {
    $response = $next($request);

    // Perform action
    return $response;
  }
}
```

Đăng ký Middleware

Tổng bộ Middleware

Nếu bạn muốn một *middleware* nào đó chạy bên dưới mọi request được gửi đến ứng dụng, thì hãy bố trí class của *middleware* đó vào trong thuộc tính `$middleware` của tập tin class *app/Http/Kernel.php*.

Đưa middleware vào các route

Nếu bạn muốn đưa *middleware* vào trong các route, thì trước hết bạn nên đưa *middleware* đó vào trong tệp *app/Http/Kernel.php* của ứng dụng. Mặc định, thuộc tính `$routeMiddleware` của class này chứa các đối tượng *middleware* đã có sẵn trong Laravel. Bạn có thể thêm *middleware* của riêng mình vào danh sách này và gán cho nó một key do bạn chọn:

```
// Within App\Http\Kernel class...

protected $routeMiddleware = [
  'auth' => \App\Http\Middleware\Authenticate::class,
  'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
  'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
  'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
  'can' => \Illuminate\Auth\Middleware\Authorize::class,
  'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
  'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
  'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
  'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
];
```

Một khi *middleware* đã được khai báo trong HTTP kernel, bạn có thể sử dụng phương thức `middleware` để đưa *middleware* bạn muốn vào trong một route nào đó do bạn chọn, ví dụ:

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

Bạn có thể đưa nhiều *middleware* vào trong route bằng cách truyền một loạt các tên *middleware* mà bạn muốn vào phương thức `middleware` như ví dụ sau đây:

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

Trong lúc truyền *middleware*, bạn cũng có thể phát biểu tên đầy đủ của class *middleware*, ví

dụ:

```
use App\Http\Middleware\EnsureTokenIsValid;
Route::get('/profile', function () {
  //
})->middleware(EnsureTokenIsValid::class);
```

Loại bỏ middleware

Khi đưa *middleware* vào trong một nhóm route, đôi khi bạn có thể cần ngăn chặn một vài *middleware* nào đó được áp dụng cho một route nào đó trong nhóm. Bạn có thể thực hiện điều này bằng phương thức `withoutMiddleware`:

```
use App\Http\Middleware\EnsureTokenIsValid;

Route::middleware([EnsureTokenIsValid::class])->group(function () {
  Route::get('/', function () {
    //
  });

  Route::get('/profile', function () {
    //
  })->withoutMiddleware([EnsureTokenIsValid::class]);
});
```

Bạn cũng có thể loại trừ một nhóm *middleware* nào đó ra khỏi toàn bộ nhóm route bằng cách khai báo như sau:

```
use App\Http\Middleware\EnsureTokenIsValid;

Route::withoutMiddleware([EnsureTokenIsValid::class])->group(function () {
  Route::get('/profile', function () {
    //
  });
});
```

Phương thức `withoutMiddleware` chỉ có thể loại bỏ *middleware* trên route và không áp dụng cho toàn bộ tổng bộ *middleware* bên dưới Laravel.

Nhóm middleware

Đôi khi bạn có thể muốn tạo nhóm một số *middleware* dưới một tên khóa duy nhất để đưa chúng vào các route dễ dàng hơn. Bạn có thể thực hiện điều này bằng cách sử dụng thuộc tính `$middlewareGroups` của HTTP kernel bên trong ứng dụng.

Ngoài ra, Laravel còn có thêm các nhóm *middleware web* và *api*, nó chứa các *middleware* phổ biến mà bạn có thể muốn áp dụng cho các route *web* và route *api* của ứng dụng. Hãy ghi nhớ, các nhóm *middleware* này được *service provider* `App\Providers\RouteServiceProvider` của ứng dụng áp dụng tự động và tương ứng với các route bên trong tập tin route *web* và route *api*:

```
/**
 * The application's route middleware groups.
 *
 * @var array
 */
protected $middlewareGroups = [
  'web' => [
    \App\Http\Middleware\EncryptCookies::class,
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
    \Illuminate\Session\Middleware\StartSession::class,
    // \Illuminate\Session\Middleware\AuthenticateSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    \App\Http\Middleware\VerifyCsrfToken::class,
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
  ],

  'api' => [
    'throttle:api',
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
  ],
];
```

Với các middleware nhóm, nó có thể được đưa vào các route và action của controller bằng

cách sử dụng cú pháp tương tự như các middleware riêng lẻ. Tóm lại, các middleware nhóm giúp chúng ta trong cùng một lúc có thể đưa được nhiều middleware vào trong một route:

```
Route::get('/', function () {
  //
})->middleware('web');

Route::middleware(['web'])->group(function () {
  //
});
```

> Ngoài ra, các nhóm middleware *web* và *api* còn được *service provider* `App\Providers\RouteServiceProvider` áp dụng tự động và tương ứng cho các tập tin route */web.php* và tập tin route */api.php* trong ứng dụng của bạn.

Xếp thứ tự Middleware

Hiếm khi, bạn cần các *middleware* của mình thực thi theo một thứ tự cụ thể, vì không có cách nào can thiệp được thứ tự của chúng khi chúng được đưa cho route. Trong trường hợp này, bạn có thể mô tả mức độ ưu tiên của *middleware* bằng cách sử dụng thuộc tính `$middlewarePinent` của tập tin *app/Http/Kernel.php* trong ứng dụng. Thuộc tính này có thể không có sẵn trong HTTP kernel của ứng dụng. Nếu như vậy, bạn có thể sao chép khai báo cho nó từ đoạn code ở bên dưới:

```
/**
 * The priority-sorted list of middleware.
 *
 * This forces non-global middleware to always be in the given order.
 *
 * @var string[]
 */
protected $middlewarePriority = [
  \Illuminate\Cookie\Middleware\EncryptCookies::class,
  \Illuminate\Session\Middleware\StartSession::class,
  \Illuminate\View\Middleware\ShareErrorsFromSession::class,
  \Illuminate\Contracts\Auth\Middleware\AuthenticatesRequests::class,
```

```
    \Illuminate\Routing\Middleware\ThrottleRequests::class,
    \Illuminate\Routing\Middleware\ThrottleRequestsWithRedis::class,
    \Illuminate\Session\Middleware\AuthenticateSession::class,
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
    \Illuminate\Auth\Middleware\Authorize::class,
];
```

Tham số middleware

Middleware cũng có thể nhận các tham số bổ sung. Ví dụ: nếu ứng dụng của bạn cần xác minh rằng người dùng được xác thực có `role` nào đó trước khi thực hiện một hành động cụ thể nào đó, bạn có thể tạo middleware `EnsureUserHasRole` nhận tên của role làm đối số bổ sung.

Các tham số bổ sung của *middleware* sẽ được truyền vào *middleware* sau biến `$next`:

```
<?php

namespace App\Http\Middleware;
use Closure;

class EnsureUserHasRole
{
    /**
     * Handle the incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @param  string  $role
     * @return mixed
     */
    public function handle($request, Closure $next, $role)
    {
        if (! $request->user()->hasRole($role)) {
            // Redirect...
        }
        return $next($request);
```

```
    }
}
```

Các giá trị của tham số *middleware* có thể được truyền vào trong khi khai báo route bằng cách tách biệt tên middleware với các tham số bằng dấu hai chấm ":". Nếu có nhiều tham số, thì các tham số này phải được phân cách bằng dấu phẩy ",". Ví dụ:

```
Route::put('/post/{id}', function ($id) {
    //
})->middleware('role:editor');
```

Hậu kỳ của middleware

Đôi khi *middleware* có thể cần thực hiện một số công việc sau khi HTTP response được gửi đến trình duyệt. Nếu khai báo phương thức `terminate` cho *middleware* và máy chủ web đang sử dụng FastCGI, thì phương thức này sẽ được gọi tự động sau khi HTTP response đã gửi đến trình duyệt:

```
<?php

namespace Illuminate\Session\Middleware;

use Closure;

class TerminatingMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        return $next($request);
```

```
    }

    /**
     * Handle tasks after the response has been sent to the browser.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Illuminate\Http\Response  $response
     * @return void
     */
    public function terminate($request, $response)
    {
        // ...
    }
}
```

Phương thức `terminate` sẽ nhận được cả HTTP request và HTTP response. Một khi bạn đã tạo hậu kỳ cho một *middleware*, thì bạn nên thêm nó vào danh sách route hoặc tổng bộ *middleware* trong tập tin *app/Http/Kernel.php*.

Khi gọi phương thức `terminate` của *middleware*, Laravel sẽ trả lại một phiên bản mới của *middleware* từ *service container*. Nếu bạn muốn sử dụng cùng phiên bản *middleware* khi phương thức `handle` và `terminate` được gọi, hãy đăng ký *middleware* với *service container* đó bằng phương thức `singleton` của *service container*. Thông thường, thì điều này được thực hiện bằng phương thức `register` của `AppServiceProvider`:

```
use App\Http\Middleware\TerminatingMiddleware;

/**
 * Register any application services.
 *
 * @return void
 */
public function register()
{
    $this->app->singleton(TerminatingMiddleware::class);
}
```