Middleware

Giới thiệu

Middleware cung cấp một cơ chế thuận lợi cho việc lọc yêu cầu HTTP gửi lên cho ứng dụng của bạn.Ví dụ, Laravel bao gồm một middleware cái mà xác thực người dùng ứng dụng của bạn có đã được chứng nhận. Nếu người dùng chưa được xác thực, middleware sẽ đưa người dùng trở lại màn hình login. Tuy vậy, nếu như người dùng được xác thực,middleware sẽ cho phép yêu cầu được tiếp tục xa hơn trong ứng dụng.

Và dĩ nhiên,middleware phụ có thể được viết để thể hiện một loạt những nhiệm vụ bên cạnh việc xác thực. Một CORS middleware có nhiệm vụ thêm tiêu đề chính cho tất cả sự trả lời có thể rời ứng dụng của bạn. Một logging middleware có thể khóa toàn bộ yêu cầu đầu vào tới ứng dụng của bạn.

Có một vài middleware trong Laravel Framework, bao gồm cho việc xác thực và bảo vệ CSRF. Tất cả chúng đều được chứa trong thư mục `app/Http/Middleware`.

Defining Middleware

Để tạo một middleware mới,chúng ta sử dụng lệnh artisan `make:middleware` :

```
php artisan make:middleware CheckAge
```

Dòng lệnh này sẽ thay thế một lớp mới `CheckAge` ở trong thư mục `app/Http/Middleware`.Ở trong middleware này , chúng tôi chỉ cho phép truy cập route nếu `age` được cung cấp lớn hơn 200. Bằng không, chúng ta sẽ trở về trang `home` URI.

```
<?php

namespace App\Http\Middleware;

use Closure;

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

```
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->age <= 200) {
            return redirect('home');
        }

        return $next($request);
    }

}
```

Như bạn có thể thấy ,nếu `age` thấp hơn hoặc bằng `200`, middleware sẽ trả về HTTP trực tiếp đến cho khách ; Mặt khác ,yêu cầu sẽ được thông qua ở trong ứng dụng của bạn . Để có thể thông qua những yêu cầu xa hơn đi vào trong ứng dụng,hãy gọi hàm callback `$next` cùng với `$request`.

Tốt nhất là hãy hình dung middleware là một chuỗi các "layers" trên HTTP request cần phải đi qua trước khi đi vào trong chương trình. Mỗi layer sẽ thực hiện kiểm tra request và thậm chí có thể huỷ từ chối request hoàn toàn.

Before & After Middleware

Việc middleware chạy trước hay chạy sau một request phụ thuộc vào chính nó. Ví dụ, middleware dưới đây sẽ làm một vào tác vụ **trước khi** request được chương trình xử lý:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

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

Tuy nhiên, middleware này sẽ thực hiện việc của nó **sau khi** request được xử lý bởi chương trình: <?php

```php
namespace App\Http\Middleware;

use Closure;

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

        // Perform action

        return $response;
    }
}
```

Registering Middleware

Global Middleware

Nếu bạn muốn một middleware có thể thực thi trong mọi HTTP request tới ứng dụng của bạn, đơn giản chỉ cần thêm tên class của middleware trong thuộc tính `$middleware` của class `app/Http/Kernel.php`.

Thiết lập middleware cho route

Nếu bạn muốn gán middleware cho route cụ thể, đầu tiên bạn cần thêm middleware đấy vào trong file `app/Http/Kernel.php`. Mặc định, thuộc tính `$routeMiddleware` sẽ chứa một số class thuộc middleware của framework Laravel. Để thêm middleware của bạn, đơn giản chỉ là thêm nó vào dach sách và gán từ khóa bạn chọn. Ví dụ:

```php
// Within App\Http\Kernel Class...

protected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' =>
\Illuminate\Routing\Middleware\SubstituteBindings::class,
```

```
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'throttle' =>
\Illuminate\Routing\Middleware\ThrottleRequests::class,
];
```

Khi middleware đã được định nghĩa trong HTTP kernel, bạn có thể sử dụng phương thức `middleware`gán cho một route:

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

Ngoài ra bạn cũng có thể gán nhiều middleware cho một route:

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

Khi đã gán middleware, bạn cũng có thể sử dụng tên đầy đủ của middleware:

```
use App\Http\Middleware\CheckAge;

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

Nhóm Middleware

Thỉnh thoảng bạn muốn nhóm một vài middleware lại trong một khóa để thực hiện gán vào route dễ dàng hơn. Bạn có thể sử dụng thuộc tính `$middlewareGroups` của HTTP kernel.

Mặc định, Laravel cung cấp sắp 2 nhóm middleware `web` và `api` chứa những middleware thường sử dụng mà bạn có thể muốn áp dụng cho web UI và API routes:

```
/**
 * 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\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:60,1',
        'auth:api',
    ],
];
```

Nhóm middleware được gán vào routes và controller sử dụng cú pháp tương tự như với từng middleware. Một lần nữa, nhóm middleware làm đơn giản trong việc gán các middleware vào trong một route:

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

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

Chú ý, Nhóm `web` middleware được tự động áp dụng trong file `routes/web.php` qua `RouteServiceProvider`.

Tham số middleware

Middleware cũng có thể nhận thêm các tham số truyền vào. Ví dụ, nếu ứng dụng của bạn cần xác thực có "role" cụ thể trước khi thực hiện một thao tác nào đó, bạn có thể tạo một `CheckRole`middleware để nhận tên của role như một tham số.

Thêm các tham số middleware sẽ dược truyền vào middleware ngay sau tham số `$next` của hàm handle:


```
<?php

namespace App\Http\Middleware;

use Closure;

class CheckRole
{
    /**
     * 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);
    }

}
```

Tham số middleware có thể được khai báo trên route bằng cách phân chia tên middleware và tham số bởi dấu `:`. nhiều thao số thì phân chia bởi dấy phẩy:

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

Terminable Middleware

Thỉnh thoảng một middleware có thể cần thực hiện sau khi HTTP response đã được gửi xong cho trình duyệt. Ví dụ, "session" middleware đi kèm với Laravel cung cấp dữ liệu session cho storage sau khi response được gửi tới trình duyệt. Nếu bạn định nghĩa một `terminate` vào trong middleware, nó sẽ tự động được gọi sau khi response được gửi tới trình duyệt.

```php
<?php

namespace Illuminate\Session\Middleware;

use Closure;

class StartSession
{
    public function handle($request, Closure $next)
    {
        return $next($request);
    }

    public function terminate($request, $response)
    {
        // Store the session data...
    }
}
```


Hàm `terminate` sẽ nhận cả request và response. Khi bạn đã định nghĩa terminable middleware, bạ phải thêm nó vào trong danh sách global middleware trong HTTP kernel.

Khi gọi hàm `terminate` trong middleware, Laravel sẽ thực hiện giải quyết trường hợp mới cho middleware từ service container. Nếu bạn muốn sử dụng cùng một trường hợp khi mà hàm `handle`và hàm `terminate` dduwcoj gọi, đăng ký middleware vào trong container sử dụng hàm `singleton`.