MIDDLEWARE

1. Giới thiệu (Introduction)

Middleware cung cấp một cơ chế thuận tiện để kiểm tra và lọc các HTTP requests đến ứng dụng. Ví dụ, trong Laravel có middleware `Authenticate` dùng để xác minh người dùng ứng dụng đã được xác thực. Nếu người dùng không được xác thực, middleware sẽ chuyển hướng người dùng đến màn hình đăng nhập ứng dụng. Tuy nhiên, nếu người dùng được xác thực, middleware sẽ cho phép thực hiện yêu cầu vào ứng dụng.

Middleware có thể được viết để thực hiện nhiều tác vụ khác nhau bên cạnh xác thực. Ví dụ, một middleware ghi nhật ký có thể ghi lại tất cả các yêu cầu đến ứng dụng. Có một số middleware được định nghĩa sẵn Laravel, bao gồm middleware để xác thực và CSRF protection. Tất cả các middleware này đều nằm trong thư mục `app/Http/Middleware`.

2. Defining Middleware (Khai báo middleware)

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

Cú pháp:

```
php artisan make:middleware MiddlewareName
```

Trong đó:

- `MiddlewareName` là tên của middleware muốn tạo.

Lệnh này sẽ tạo một lớp `MiddlewareName` mới trong thư mucr `app/Http/Middleware`.

Ví dụ:

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

Câu lệnh trên sẽ tạo một file là `app/Http/Middleware/EnsureTokenIsValid.php`. File sẽ có nội dung như sau:

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class EnsureTokenIsValid
{
```

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

Trong đó:

- `return $next($request)` là đoạn code cho phép request tiếp tục thực thi.

Ví dụ, chỉ cho phép truy cập vào route nếu `token` input được cung cấp khớp với một giá trị được chỉ định. Nếu không, sẽ chuyển hướng người dùng trở lại `home` URI:

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

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

Lúc này request gửi đến nếu không có input token hoặc input token không phải là `'my-secret-token'` thì request sẽ được redirect về URL có đường dẫn (path) `/home`.

Middleware & Responses

Tất nhiên, một middleware có thể thực hiện các tác vụ trước hoặc sau khi chuyển yêu cầu đến ứng dụng. Ví dụ: middleware sau sẽ thực hiện một số tác vụ trước khi yêu cầu 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, middleware này sẽ thực hiện nhiệm vụ của nó sau khi ứng dụng xử lý yêu cầu:

```
<?php

namespace App\Http\Middleware;

use Closure;

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

        // Perform action

        return $response;
    }
}
```

3. Đăng ký middlware (Registering Middleware)

Global Middleware

Nếu muốn middleware chạy trong mọi HTTP request đến ứng dụng, hãy liệt kê lớp middleware trong thuộc tính `$middleware` của lớp `app/Http/Kernel.php`.

Assigning Middleware To Routes

Nếu muốn gán middleware cho các route cụ thể, trước tiên phải gán cho middleware một key trong file `app/Http/Kernel.php` của ứng dụng. Theo mặc định, thuộc tính `$routeMiddleware` của lớp này chứa các entries cho middleware đi kèm với Laravel. Có thể thêm middleware của riêng mình vào danh sách này và gán cho nó một key được 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,
];
```

Khi middleware đã được khai báo trong HTTP kernel, có thể sử dụng phương thức `middleware` để gán middleware cho một route:

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

Có thể gán nhiều middleware cho route bằng truyền một mảng các tên middleware vào phương thức `middleware`:

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

Khi gán middleware, cũng có thể truyền tên lớp:

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

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

Khi gán middleware cho một nhóm các route, đôi khi có thể cần ngăn middleware được áp dụng cho một route riêng lẻ trong nhóm. Có thể thực hiện 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]);
});
```

Middleware Groups

Đôi khi nhóm một số middleware dưới một khóa duy nhất để gán chúng cho các route dễ dàng hơn. Có thể thực hiện điều này bằng cách sử dụng thuộc tính `middlewareGroups` của HTTP kernel.

Ngoài ra, Laravel đi kèm với các nhóm middleware `web` và `api` chứa middleware phổ biến cho các route web và API. Các nhóm middleware này được tự động áp dụng bởi `App\Providers\RouteServiceProvider` của ứng dụng cho các route trong file route `web` và `api` tương ứng:

```
/**
 * 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,
    ],
];
```

Nhóm middleware có thể được chỉ định cho các route và controller actions bằng cách sử dụng

cú pháp tương tự như middleware riêng lẻ. Một lần nữa, các nhóm middleware giúp thuận tiện hơn khi gán nhiều middleware cho một route cùng lúc:

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

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

Sorting Middleware

Đôi khi các middleware cần được gọi theo thứ tự, nhưng khi đăng ký trong route lại không có tác vụ này. Chính vì điều đó, Laravel cung cấp thuộc tính `$middlewarePriority` trong file `app/Http/Kernel.php` để sắp xếp các middleware theo thứ tự ưu tiên xử lý từ trên xuống.

```
/**
 * The priority-sorted list of middleware.
 *
 * This forces non-global middleware to always be in the given order.
 *
 * @var array
 */
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\Session\Middleware\AuthenticateSession::class,
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
    \Illuminate\Auth\Middleware\Authorize::class,
];
```

2.2.4. Tham số Middleware (Middleware Parameter)

Middleware cũng có thể nhận các tham số bổ sung. Ví dụ: nếu ứng dụng cần xác minh rằng người dùng được xác thực có "vai trò" nhất định trước khi thực hiện một hành động nhất định, có thể tạo middleware `EnsureUserHasRole` nhận tên vai trò làm đối số bổ sung.

Các tham số middleware bổ sung sẽ được chuyển đến middleware sau đối số $ next:

```
<?php
```

```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 tham số Middleware có thể được chỉ định khi định nghĩa route bằng cách tách tên Middleware và các tham số bằng `:` Nếu có nhiều tham số phải được phân cách bằng dấu phẩy:

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

Terminable Middleware

Đôi khi middleware có thể cần thực hiện một số công việc sau khi HTTP response đã được gửi đến trình duyệt. Nếu định nghĩa phương thức `terminate` trên middleware và máy chủ web có sử dụng FastCGI, phương thức `terminate` sẽ tự động được gọi sau khi response được trả về:

```php
<?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 nhận được cả request và response. Khi đã khai báo một terminable middleware, nên thêm nó vào danh sách các route hoặc global middleware trong file `app/Http/Kernel.php`.

Khi gọi method `terminate` trong middleware, Laravel sẽ resolve một middleware instance mới từ service container. Nếu muốn sử dụng một middleware instance khi các method `handle` và `terminate` được gọi thì nên đăng ký middleware với container thông qua phương thức `singleton` .

Thông thường, điều này phải được thực hiện trong phương thức `register` của `AppServiceProvider`:

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

/**
```

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

CSRF PROTECTION

1. Giới thiệu (Introduction)

CSRF là viết tắt của Cross-site request forgeries là yêu cầu giả mạo trên nhiều trang web, còn được gọi là tấn công bằng một cú nhấp chuột. Theo đó các hacker có thể giả mạo người dùng đã được xác thực gửi đi các request đến server. Laravel giúp dễ dàng bảo vệ ứng dụng khỏi các cuộc tấn công giả mạo yêu cầu (CSRF) trên nhiều trang web.

Lỗ hổng bảo mật (Vulnerability)

Ví dụ hệ thống có một route `/user/email` với phương thức `POST` request sử dụng để thay đổi địa chỉ email xác thực của người dùng. Rất có thể, route này yêu cầu input field `email` để nhập địa chỉ email mà người dùng muốn sử dụng.

Nếu không có CSRF protection, một trang web độc hại (malicious website) có thể tạo một HTML form trỏ đến route `/user/email` của ứng dụng và gửi địa chỉ email của chính họ:

```
<form action="https://your-application.com/user/email" method="POST">
    <input type="email" value="malicious-email@example.com">
</form>

<script>
    document.forms[0].submit();
</script>
```

Nếu trang web độc hại tự động submit form khi trang được tải, người dùng độc hại chỉ cần dụ một người dùng không nghi ngờ ứng dụng truy cập trang web của họ và địa chỉ email của họ sẽ được thay đổi trong ứng dụng.

Để ngăn chặn lỗ hổng này, cần kiểm tra mọi `POST`, `PUT`, `PATCH`, hoặc `DELETE` request đến để tìm giá trị secret session mà ứng dụng độc hại không thể truy cập.

2. Ngăn ngừa CSRF Requests (Preventing CSRF Requests)

Laravel tự động tạo CSRF "token" cho mỗi phiên người dùng đang hoạt động do ứng dụng quản lý. Token này được sử dụng để xác minh rằng người dùng được xác thực là người thực sự đưa ra các yêu cầu đối với ứng dụng. Vì mã thông báo này được lưu trữ trong phiên của người dùng và thay đổi mỗi khi phiên được tạo lại, ứng dụng độc hại không thể truy cập vào nó.

CSRF token của phiên hiện tại có thể được truy cập thông qua phiên của request hoặc thông qua chức năng trợ giúp `csrf_token`:

```
use Illuminate\Http\Request;

Route::get('/token', function (Request $request) {
    $token = $request->session()->token();

    $token = csrf_token();

    // ...
});
```

Khi tạo các HTML "POST", "PUT", "PATCH", or "DELETE" HTML form trong ứng dụng nên bao gồm hidden CSRF `_token` field trong form để CSRF protection middleware có thể xác thực yêu cầu. Để thuận tiện, có thể sử dụng `@csrf` Blade để tạo hidden token input field:

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

    <!-- Equivalent to... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

Middleware `App\Http\Middleware\VerifyCsrfToken`, bao gồm trong nhóm middleware web theo mặc định, sẽ tự động xác minh rằng token trong request input khớp với token được lưu trữ session. Khi hai token này khớp với nhau, thì người dùng đã xác thực là người yêu cầu.

3. CSRF Tokens & SPAs

Nếu đang xây dựng một SPA sử dụng Laravel làm API backend, nên tham khảo tài liệu Laravel Sanctum documentation để biết thông tin về cách xác thực với API và bảo vệ khỏi các lỗ hổng CSRF.

4. Excluding URIs From CSRF Protection

Khi muốn loại một tập các URI khỏi CSRF protection. Ví dụ, nếu đang sử dụng Stripe để xử lý thanh toán và sử dụng hệ thống webhook của nó, thì cần phải loại trừ các route xử lý webhook của Stripe khỏi tính năng CSRF protection vì Stripe sẽ không biết CSRF token nào cần được gửi đến route.

Thông thường, nên đặt các loại route này bên ngoài nhóm middleware `web` mà `App\Providers\RouteServiceProvider` áp dụng cho tất cả các tuyến trong file `routes/web.php`. Tuy nhiên, cũng có thể loại trừ các route bằng cách thêm các URIs vào thuộc tính `$except` của middleware `VerifyCsrfToken`:

```
<?php

namespace App\Http\Middleware;

use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;

class VerifyCsrfToken extends Middleware
{
    /**
     * The URIs that should be excluded from CSRF verification.
     *
     * @var array
     */
    protected $except = [
        'stripe/*',
        'http://example.com/foo/bar',
        'http://example.com/foo/*',
    ];
}
```

5. X-CSRF-TOKEN

Ngoài việc kiểm tra CSRF token dưới dạng tham số POST, middleware `App\Http\Middleware\VerifyCsrfToken` cũng sẽ kiểm tra `X-CSRF-TOKEN` request header. Ví dụ: có thể lưu trữ token trong thẻ HTML meta:

```
<meta name="csrf-token" content="{{ csrf_token() }}">
```

Sau đó, có thể hướng dẫn một thư viện như jQuery tự động thêm token vào tất cả các request header. Điều này cung cấp một CSRF protection đơn giản, thuận tiện cho các ứng dụng dựa trên AJAX bằng JavaScript:

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
```

```
});
```

6. X-XSRF-TOKEN

Laravel lưu trữ CSRF token hiện tại trong `XSRF-TOKEN` cookie được mã hóa đi kèm với mỗi response được tạo bởi framework. Có thể sử dụng giá trị cookie để đặt `X-XSRF-TOKEN` request header.

Cookie này chủ yếu được gửi như một sự thuận tiện cho nhà phát triển vì một sốJavaScript frameworks và thư viện, như Angular và Axios, tự động đặt giá trị của `X-XSRF-TOKEN` header trên các request cùng nguồn gốc.