Middleware o filtros

Los componentes llamados *Middleware* son un mecanismo proporcionado por Laravel para **filtrar las peticiones HTTP** que se realizan a una aplicación. Un filtro o *middleware* se define como una clase PHP almacenada en un fichero dentro de la carpeta `app/Http/Middleware`. Cada *middleware* se encargará de aplicar un tipo concreto de filtro y de decidir que realizar con la petición realizada: permitir su ejecución, dar un error o redireccionar a otra página en caso de no permitirla.

Laravel incluye varios filtros por defecto, uno de ellos es el encargado de realizar la autenticación de los usuarios. Este filtro lo podemos aplicar sobre una ruta, un conjunto de rutas o sobre un controlador en concreto. Este *middleware* se encargará de filtrar las peticiones a dichas rutas: en caso de estar logueado y tener permisos de acceso le permitirá continuar con la petición, y en caso de no estar autenticado lo redireccionará al formulario de login.

Laravel incluye *middleware* para gestionar la autenticación, el modo mantenimiento, la protección contra CSRF, y algunos mas. Todos estos filtros los podemos encontrar en la carpeta `app/Http/Middleware`, los cuales los podemos modificar o ampliar su funcionalidad. Pero además de estos podemos crear nuestros propios *Middleware* como veremos a continuación.

Definir un nuevo *Middleware*

Para crear un nuevo *Middleware* podemos utilizar el comando de Artisan:

```
php artisan make:middleware MyMiddleware
```

Este comando creará la clase `MyMiddleware` dentro de la carpeta `app/Http/Middleware` con el siguiente contenido por defecto:

```php
<?php

namespace App\Http\Middleware;

use Closure;

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

El código generado por Artisan ya viene preparado para que podamos escribir directamente la implementación del filtro a realizar dentro de la función `handle`. Como podemos ver, esta función solo incluye el valor de retorno con una llamada a `return $next($request);`, que lo que hace es continuar con la petición y ejecutar el método que tiene que procesarla. Como entrada la función `handle` recibe dos parámetros:

- `$request`: En la cual nos vienen todos los parámetros de entrada de la peticion.
- `$next`: El método o función que tiene que procesar la petición.

Por ejemplo podríamos crear un filtro que redirija al home si el usuario tiene menos de 18 años y en otro caso que le permita acceder a la ruta:

```php
public function handle($request, Closure $next)
{
    if ($request->input('age') < 18) {
        return redirect('home');
    }

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

Como hemos dicho antes, podemos hacer tres cosas con una petición:

- Si todo es correcto permitir que la petición continúe devolviendo: `return $next($request);`
- Realizar una redirección a otra ruta para no permitir el acceso con: `return`

`redirect('home');`

- Lanzar una excepción o llamar al método `abort` para mostrar una página de error: `abort(403, 'Unauthorized action.');`

Middleware antes o después de la petición

Para hacer que el código de un *Middleware* se ejecute antes o después de la petición HTTP simplemente tenemos que poner nuestro código antes o después de la llamada a `$next($request);`. Por ejemplo, el siguiente _Middleware realizaría la acción **antes** de la petición:

```
public function handle($request, Closure $next)
{
    // Código a ejecutar antes de la petición

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

Mientras que el siguiente *Middleware* ejecutaría el código **después** de la petición:

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

    // Código a ejecutar después de la petición

    return $response;
}
```

Uso de *Middleware*

De momento hemos visto para que vale y como se define un *Middleware*, en esta sección veremos como utilizarlos. Laravel permite la utilización de *Middleware* de tres formas distintas: global, asociado a rutas o grupos de rutas, o asociado a un controlador o a un método de un controlador. En los tres casos será necesario registrar primero el *Middleware* en la clase `app/Http/Kernel.php`.

Middleware global

Para hacer que un *Middleware* se ejecute con **todas** las peticiones HTTP realizadas a una aplicación simplemente lo tenemos que registrar en el array `$middleware` definido en la clase `app/Http/Kernel.php`. Por ejemplo:

```
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \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,
    \App\Http\Middleware\MyMiddleware::class,
];
```

En este ejemplo hemos registrado la clase *MyMiddleware* al final del array. Si queremos que nuestro *middleware* se ejecute antes que otro filtro simplemente tendremos que colocarlo antes en la posición del array.

Middleware asociado a rutas

En el caso de querer que nuestro *middleware* se ejecute solo cuando se llame a una ruta o a un grupo de rutas también tendremos que registrarlo en el fichero `app/Http/Kernel.php`, pero en el array `$routeMiddleware`. Al añadirlo a este array además tendremos que asignarle un nombre o clave, que será el que después utilizaremos asociarlo con una ruta.

En primer lugar añadimos nuestro filtro al array y le asignamos el nombre "`es_mayor_de_edad`":

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'es_mayor_de_edad' => \App\Http\Middleware\MyMiddleware::class,
];
```

Una vez registrado nuestro *middleware* ya lo podemos utilizar en el fichero de rutas `app/Http/routes.php` mediante la clave o nombre asignado, por ejemplo:

```
Route::get('dashboard', ['middleware' => 'es_mayor_de_edad', function () {
    //...
}]);
```

En el ejemplo anterior hemos asignado el *middleware* con clave `es_mayor_de_edad` a la ruta `dashboard`. Como se puede ver se utiliza un array como segundo parámetro, en el cual indicamos el *middleware* y la acción. Si la petición supera el filtro entonces se ejecutara la función asociada.

Para asociar un filtro con una ruta que utiliza un método de un controlador se realizaría de la misma manera pero indicando la acción mediante la clave " `uses` ":

```
Route::get('profile', [
    'middleware' => 'auth',
    'uses' => 'UserController@showProfile'
]);
```

Si queremos asociar varios *middleware* con una ruta simplemente tenemos que añadir un array con las claves. Los filtros se ejecutarán en el orden indicado en dicho array:

```
Route::get('dashboard', ['middleware' => ['auth', 'es_mayor_de_edad'], function () {
    //...
}]);
```

Laravel también permite asociar los filtros con las rutas usando el método `middleware()` sobre la definición de la ruta de la forma:

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

// O sobre un controlador:
Route::get('profile', 'UserController@showProfile')->middleware('auth');
```

Middleware dentro de controladores

También es posible indicar el *middleware* a utilizar desde dentro de un controlador. En este caso los filtros también tendrán que estar registrador en el array `$routeMiddleware` del fichero `app/Http/Kernel.php`. Para utilizarlos se recomienda realizar la asignación en el constructor del controlador y asignar los filtros usando su clave mediante el método `middleware`. Podremos indicar que se filtren todos los métodos, solo algunos, o todos excepto los indicados, por ejemplo:

```
class UserController extends Controller
{
    /**
     * Instantiate a new UserController instance.
     *
     * @return void
     */
    public function __construct()
    {
        // Filtrar todos los métodos
        $this->middleware('auth');

        // Filtrar solo estos métodos...
        $this->middleware('log', ['only' => ['fooAction', 'barAction']]);

        // Filtrar todos los métodos excepto...
        $this->middleware('subscribed', ['except' => ['fooAction', 'barAction']]);
    }
}
```

Revisar los filtros asignados

Al crear una aplicación Web es importante asegurarse de que todas las rutas definidas son correctas y que las partes privadas realmente están protegidas. Para esto Laravel incluye el siguiente método de Artisan:

```
php artisan route:list
```

Este método muestra una tabla con todas las rutas, métodos y acciones. Ademas para cada ruta indica los filtros asociados, tanto si están definidos desde el fichero de rutas como **desde dentro de un controlador**. Por lo tanto es muy útil para comprobar que todas las rutas y filtros que hemos definido se hayan creado correctamente.

Paso de parámetros

Un *Middleware* también puede recibir parámetros. Por ejemplo, podemos crear un filtro para comprobar si el usuario logueado tiene un determinado rol indicado por parámetro. Para esto lo primero que tenemos que hacer es añadir un tercer parámetro a la función `handle` del *Middleware*:

```php
<?php

namespace App\Http\Middleware;

use Closure;

class RoleMiddleware
{
    /**
     * Run the request filter.
     *
     * @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)) {
            // No tiene el rol esperado!
        }

        return $next($request);
    }

}
```

En el código anterior de ejemplo se ha añadido el tercer parámetro `$role` a la función. Si nuestro filtro necesita recibir más parámetros simplemente tendríamos que añadirlos de la misma forma a esta función.

Para pasar un parámetro a un *middleware* en la definición de una ruta lo tendremos que añadir a continuación del nombre del filtro separado por dos puntos, por ejemplo:

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

Si tenemos que pasar más de un parámetro al filtro los separaremos por comas, por ejemplo: `role:editor,admin`.