Laravel middleware

Qué son los Http Middleware, una de las piezas principales del framework PHP Laravel. Cómo trabajar con Middlewares en Laravel, creando uno nuevo.

Hemos podido conocer varias de las principales capas de aplicación. Hemos preferido comenzar por describir las partes más sencillas y de las cuales estamos seguros muchos tenían nociones, como son las vistas o controladores. Del MVC nos faltan por ver los modelos, pero antes de ponernos con ellos nos vamos a detener en los middleware.

Hasta ahora hemos visto que desde el sistema de routing podemos invocar a los controladores, o incluso también a través de un "closure" a las vistas. Sin embargo no habíamos mencionado que, antes llegar el flujo de ejecución a éstos hay una capa intermedia que se ejecuta en toda solicitud: el middleware.

Los middleware en términos generales son partes del software que actúan de mediadores entre distintos actores, permanecen en medio para facilitar la interacción o comunicación entre distintas partes de un sistema. Aquí en Laravel se quedan entre medias del sistema de routing y los controladores, permitiendo hacer cosas al pasar de unos a otros, generalmente operaciones de filtrado.

Según la definición de la documentación oficial de Laravel, "HTTP middleware provee un mecanismo adecuado para filtrar solicitudes HTTP entrantes a la aplicación [...] hay diversos middleware incluidos en el frameworkLaravel, como middleware para mantenimiento, autenticación, protección CSRF mediante token, etc."

Además nosotros podemos hacer nuestros propios middleware para diversas tareas, como añadir una salida en las cabeceras de toda respuesta, realizar un log de todas las solicitudes al servidor, etc. Como puedes ver, el middleware es el sitio ideal para incluir código que quieres ejecutar al principio de toda solicitud, aunque si quieres también los puedes asociar solamente a determinadas rutas.

Archivo Kernel.php, donde se registran los middelware

Antes de ponernos a realizar nuestro propio middleware es una buena idea echar un vistazo a los que ya tenemos funcionando de manera predeterminada. Estamos seguros que ésto ayudará a aclarar el concepto de middleware en Laravel. Para ver los middleware configurados en nuestro sistema tenemos que entrar en el archivo app/Http/Kernel.php.

El Kernel es el que le dice a Laravel qué middlewares tiene que cargar. Allí encontrarás una clase que tiene registrados los middleware que se van a ejecutar en el sistema, tanto de manera global (propiedad $middleware) como para rutas particulares (propiedad $routeMiddleware).

Nota: Quizás recuerdas que en este archivo Kernel.php ya habíamos entrado anteriormente, cuando descubrimos el sistema de rutas. En el artículo Verbos en las rutas de Laravel vimos que al realizarse una ruta de tipo post se activa una comprobación de un token, solicitado para aumentar la seguridad frente CSRF (Cross-siterequestforgery o falsificación de petición desde otros sitios). Esa comprobación se realiza en el middleware "VerifyCsrfToken". En el mencionado artículo habíamos pedido comentar esa línea, para que no se pusiera en marcha ese middleware y poder comprobar si funcionaban las rutas post.

Crear un middleware

Para crear un middleware podríamos tomar como punto de partida uno de los que ya vienen por defecto en Laravel, pero realmente hay una manera más apropiada, usando el asistente "artisan".

Desde la línea de comandos, en la carpeta raíz del proyecto, podemos invocar artisan y solicitarle el comando "make:middleware" indicando a continuación el nombre del middleware que se desea crear.

```
phpartisanmake:middlewareDomingoMiddleware
```

Nuestro middleware se llama DomingoMiddleware y se encargará de hacer cosas cuando detecte que el día actual es un domingo. Ese comando de artisan genera un middleware básico en el que solo tenemos un método, que enseguida explicamos.

La localización del archivo que se ha creado, y en general de todos los middlewares en Laravel, está en app/Http/Middleware/. Allí encontrarás ahora el archivo DomingoMiddleware que como también corresponde con el nombre de la clase que implementa el middleware, tiene la primera letra en mayúscula.

Este es el código de nuestro primer middleware:

```
<?php

namespaceApp\Http\Middleware;

useClosure;

classDomingoMiddleware
{
publicfunctionhandle($request, Closure $next)
{
if(date('w')==='0'){
echo"Es domingo!";
}else{
echo"No es domingo";
}
return$next($request);
}
}
```

Sobre el código anterior queremos que te fijes en el método handle(), que es el que se ejecutará cuando se ponga en marcha este middleware. Tiene unas cuantas cosas que conviene explicar detenidamente, pero de momento solo queremos que se aprecie el if, donde se comprueba si el día de la semana actual es domingo, realizando dos acciones distintas si era o no era ese día de la semana.

Nota: Como acción de respuesta dependiendo de si es o no domingo realizamos una salida con un echo. Queremos remarcar que ese echo no tiene mucho sentido, porque no es el momento adecuado de lanzar salida al navegador, pero ahora mismo nos sirve en este middleware de prueba, para verlo en funcionamiento y recibir alguna salida como respuesta. El único motivo por tanto de realizar ese echo es para saber si se está ejecutando. Como ya sabes, la salida la generamos desde las vistas. En el middleware acciones típicas son redirigir al usuario a una página o realizar cualquier tarea de mantenimiento, etc.

Registrar el middleware de manera global

Ahora vamos a registrar el middleware para ver cómo se ejecuta en el sistema. Empezaremos registrando en modo global, para que se ejecute en todas y cada una de las solicitudes que atienda nuestra aplicación.

Esto es muy sencillo y ya lo hemos dejado entrever al principio del artículo. Se hace desde el archivo Kernel.php que está en la ruta app/Http/Kernel.php.

Hay una propiedad de la clase Kernel donde se coloca toda la lista de middleware a ejecutar de manera global;

```
protected$middleware=[

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
 // otros middleware
 // ...
        \App\Http\Middleware\DomingoMiddleware::class,
];
```

Como puedes ver, es un array en el que debemos indicar el listado de los middleware a ejecutar en cada solicitud, en el orden en el que van a ser invocados. Nosotros hemos agregado el middleware creado anteriormente en el último elemento de la lista, pero si necesitase mayor prioridad sería solo ponerlo antes en el array.

Una vez registrado el middleware de manera global podrías entrar en cualquier página de la aplicación y debería verse la salida del middleware, informando si es o no domingo.

Registrar el middleware para una ruta determinada o un controlador

Hay varias maneras de hacer este paso, desde el sistema de rutas o incluso desde los controladores, pero siempre debemos comenzar por asignar un nombre a nuestro middleware en el archivo Kernel.php. Allí, en la clase Kernel, hay una segunda propiedad

llamada $routeMiddleware, donde tenemos un array asociativo con los middlewares que deseemos usar en las rutas.

Cada elemento del array tiene un índice y ese índice será el nombre que le demos al middleware para referirnos a él desde el sistema de routing.

```
protected$routeMiddleware=[
'auth'=> \App\Http\Middleware\Authenticate::class,
 // otros middleware de rutas...
'domingo'=> \App\Http\Middleware\DomingoMiddleware::class,
];
```

Ahora podemos ejecutar el middleware a través del índice que le hemos dado en el array anterior. Lo veremos hacer mediante tres alternativas distintas:

1. Desde el sistema de routing, al definir la ruta, podemos indicar un middleware que queremos usar. Esto lo tenemos que hacer desde el registro de rutas, realizado en el archivo routes.php, mediante una sintaxis como la que puedes ver a continuación.

```
Route::get('/test',['middleware'=>'domingo',function(){
return'Probando ruta con middleware';
}]);
```

4. También desde el sistema de routing podemos generar un grupo de rutas donde se ejecute este middleware.

```
Route::group(['middleware'=>'domingo'],function(){
Route::get('/probando/ruta',function(){
//código a ejecutar cuando se produzca esa ruta y el verbo
return'get';
});

 Route::post('/probando/ruta',function(){
 //código a ejecutar cuando se produzca esa ruta y el verbo POST
 return'post';
 });
});

Así hemos indicado dos rutas donde se ejecutará ese middleware
etiquetado como "domingo".
```

15. Desde un controlador también podemos llamar a un middleware. Lo podemos hacer desde el constructor, por lo que ese middleware afectará a todas las acciones dentro del controlador.

```
 classPrimerControllerextendsController
 {
 publicfunction__construct(){
 $this->middleware('domingo');
 }
}
```

En este caso no necesitamos mencionar el middleware desde el sistema de rutas, solo lo mencionamos desde el constructor del controlador, para ejecutarse en cualquiera de las acciones declaradas en él.

Encadenar un middleware con el siguiente

Según la filosofía de los middlewares en Laravel, como ya se comentó, pueden existir varios que se ejecuten en cadena, uno detrás de otro, para cada solicitud. Por ello tenemos que asegurarnos que este encadenamiento se pueda producir. Realmente este trabajo nos lo dan ya hecho en el middleware que se genera con artisan, por lo que no debemos preocuparnos nosotros. Sin embargo queremos que se vea dónde se consigue ese procesamiento en cadena.

Echemos un vistazo al método handle() del middleware generado automáticamente:

```
publicfunctionhandle($request, Closure $next)
{
return$next($request);
}
```

Como se puede ver, en este método se devuelve con return una llamada a $next() pasándole $request como parámetro. Esa llamada es la que permite que se ejecute el siguiente middleware en la lista de middlewares globales y además enviarle la Request completa al siguiente middleware para que siga filtrándola.

Realmente no tenemos que hacer nada manualmente, porque Laravel ya le pasa en el método handle el $request y el $next como parámetro. Además, Laravel sabe si ese middleware es el último de la lista, en cuyo caso no debería haber otro encadenamiento. Simplemente sería asegurarse de dejar ese return que aparece de manera predeterminada, para que todo siga funcionando.

Conclusión

Los middlewares son una herramienta potente para realizar muchos tipos de acciones. Hemos visto las posibilidades más básicas que nos ofrece Laravel 5.

Sin embargo, las utilidades del middleware se extienden para cualquier cosa que podamos necesitar.