

# Laravel - Middleware


Inspección y filtrado de peticiones HTTP

# ¿Que es?

Se trata de un **mecanismo para inspeccionar y filtrar peticiones HTTP**. Intercepta peticiones entrantes y “pre-procesa”.

Ejemplo; Laravel incluye un “middleware” para verificar que el usuario de la aplicación se ha autenticado. Al llegar una petición, si el usuario se ha identificado, el proceso continúa con normalidad (la petición se redirige donde deba). En caso contrario, middleman redirige a la página de login.

Otro ejemplo de middleware incluido con Laravel es la directiva @CSRF, para la protección de datos provenientes de formularios.



# Crear un Middleware

Todos los middleware creados (no los incluidos por defecto) se deben guardar en: **app/Http/Middleware**.

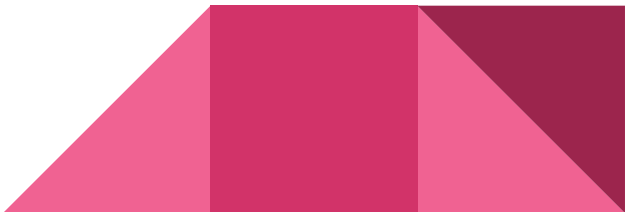
Para crear un Middleware se puede emplear la siguiente instrucción de artisan:

```
php artisan make:middleware [NombreMiddleWare]
```

Ejemplo:

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

Este comando crea una clase “EnsureTokenIsValid” en app/Http/Middleware.




# Ejemplo Middleware - EnsureTokenIsValid

```
<?php
namespace App\Http\Middleware;
use Closure; // clase anónima que permite definir funciones o callbacks sin nombre
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class EnsureTokenIsValid {
    // Controlar petición entrante. Requiere parámetros de clase Request y clase Closure (respuesta)
    // @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next

    public function handle(Request $request, Closure $next): Response {
        // código del middleware - realiza acción
        if ($request->input('token') !== 'mi-token-secreto') {
            return redirect('/home'); // middleware DETIENE EL FLUJO
        }
        // Middleware cede el paso
        return $next($request); // middleware permite que la solicitud continúe
    }
}
```



# Ejemplo Middleware - EnsureTokenIsValid

El método se llama **handle()** porque se encarga de "**manejar**" o procesar la solicitud entrante en el middleware.

Este método permite a los middleware realizar su tarea (como la validación del token en este caso) y luego decidir si la solicitud debe continuar o si debe redirigirse o rechazarse.



# Ejemplo Middleware - EnsureTokenIsValid

Continuando con el código anterior, sobre los parámetros de handle():

- **Request \$request** es una instancia de la clase *Illuminate\Http\Request*, que contiene todos los datos de la solicitud HTTP que está siendo procesada
- La función/**método \$next** representa el siguiente paso o acción a realizar en la cadena de middleware después de que el middleware actual haya ejecutado su lógica.
- **Closure** se utiliza como tipo de parámetro para la función \$next. Es una clase anónima que permite definir funciones o callbacks sin nombre.

Así pues, **\$next(\$request)** pasa la solicitud al siguiente paso (middleware o controlador).




## Ejemplo Middleware - EnsureTokenIsValid

Si el token proporcionado no coincide con “mi-token-secreto”, el middleware devolverá una redirección HTTP al cliente; de lo contrario, la solicitud se enviará más adelante en la aplicación.

Para pasar la solicitud a la aplicación (permitiendo que el middleware la "pase"), debe llamar al *callback* `$next` (tipo closure) con el `$request`.


Es mejor imaginar el **middleware como una serie de "capas" o barreras que las solicitudes HTTP deben atravesar** antes de llegar a la aplicación. **Cada capa puede examinar la solicitud e incluso rechazarla por completo.**



# Middleware con acción post-procesado

Se puede configurar que la acción del middleware se realice después del paso de la petición, enviando la petición antes y devolviendo la respuesta:

```
class AfterMiddleware {  
    public function handle(Request $request, Closure $next): Response {  
        // Middleware cede el paso, pero no termina  
        $response = $next($request);  
        // código del middleware - realiza acción  
        return $response;  
    }  
}
```





# Uso de Middleware - registrar Middleware

Hay dos formas de registrar un Middleware:

1. **Registrar globalmente:** siempre pasará automáticamente por toda petición entrante.
2. **Asignar a rutas (Route):** para revisar únicamente peticiones específicas.



# Uso de Middleware - registrar Middleware globalmente

Laravel posee un stack (listado) de middlewares global; es posible añadir middlewares generados a este listado, añadiendo sus clases en **bootstrap/app.php**. Por ejemplo, para añadir EnsureTokenIsValid:

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

```
->withMiddleware(function (Middleware $middleware) {  
    $middleware->append(EnsureTokenIsValid::class);  
})
```

El objeto `$middleware` es una instancia de `Illuminate\Foundation\Configuration\Middleware`. Es el responsable de gestionar los middlewares asignados a todas las rutas de la aplicación.

El método `append()` añade el middleware indicado al listado global de middlewares.

# Uso de Middleware - registrar Middleware a rutas

Si se desea **asignar middleware a rutas específicas**, se puede invocar el **método `middleware()`** al definir la ruta. Usando el ejemplo de `EnsureTokenIsValid`:

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

```
Route::get('/profile', function () {  
    // código de la ruta...  
})->middleware(EnsureTokenIsValid::class);
```




# Middleware con Alias

Es posible **registrar globalmente un Middleware y asignarle un Alias para poder incluirlo solo cuando sea necesario**. Para ello, al registrar globalmente, en lugar de incluirlo con `append()`, indicaremos un nombre a la clase del Middleware, dentro de un array, con **alias()**. Por ejemplo, para `EnsureTokenIsValid`, nuevamente en **bootstrap/app.php**:

```
use App\Http\Middleware\EnsureTokenIsValid;  
  
->withMiddleware(function (Middleware $middleware) {  
    $middleware->alias([ 'ensureToken' => EnsureTokenIsValid::class ]);  
})
```

Ahora, al definir una ruta con Middleware, es posible usar solo el alias:

```
Route::get('/profile', function () {  
    // código de la ruta...  
})->middleware('ensureToken');
```



# Middleware con Alias - Alias por defecto

Alias	Middleware
auth	<code>Illuminate\Auth\Middleware\Authenticate</code>
auth.basic	<code>Illuminate\Auth\Middleware\AuthenticateWithBasicAuth</code>
auth.session	<code>Illuminate\Session\Middleware\AuthenticateSession</code>
cache.headers	<code>Illuminate\Http\Middleware/SetCacheHeaders</code>
can	<code>Illuminate\Auth\Middleware\Authorize</code>
guest	<code>Illuminate\Auth\Middleware\RedirectIfAuthenticated</code>
password.confirm	<code>Illuminate\Auth\Middleware\RequirePassword</code>
precognitive	<code>Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests</code>
signed	<code>Illuminate\Routing\Middleware\ValidateSignature</code>
subscribed	<code>\Spark\Http\Middleware\VerifyBillableIsSubscribed</code>
throttle	<code>Illuminate\Routing\Middleware\ThrottleRequests</code> or <code>Illuminate\Routing\Middleware\ThrottleRequestsWithRedis</code>
verified	<code>Illuminate\Auth\Middleware\EnsureEmailIsVerified</code>

# “Terminable” Middleware

Consisten en **Middleware que ejecutan código una vez que ya se ha enviado una respuesta al navegador** del cliente.

Se definen con el **método “terminate”** en la clase del middleware.

Requieren que el servidor web utilice FastCGI.

FastCGI es un protocolo para mejorar el rendimiento de aplicaciones web al permitir una comunicación más rápida y eficiente entre el servidor web y las aplicaciones.

