Utilisation des Middleware

1. Introduction

Le Middleware fournit un mécanisme pratique pour inspecter et filtrer les requêtes HTTP entrantes dans votre application. Par exemple, Laravel inclut un middleware qui vérifie que l'utilisateur de votre application est authentifié. Si l'utilisateur n'est pas authentifié, le middleware le redirige vers l'écran de connexion de votre application. Cependant, si l'utilisateur est authentifié, le middleware permettra à la requête de se poursuivre dans l'application.

Des middlewares supplémentaires peuvent être écrits pour effectuer une variété de tâches autres que l'authentification. Par exemple, un middleware de journalisation peut enregistrer toutes les demandes entrantes dans votre application. Plusieurs middlewares sont inclus dans le framework Laravel, notamment des middlewares pour l'authentification et la protection CSRF. Tous ces middlewares sont situés dans le répertoire **app/Http/Middleware**.

2. Définition

Pour créer un nouvel middleware, utilisez la commande

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

Cette commande va placer une nouvelle classe **EnsureTokenIsValid** dans votre répertoire **app/Http/Middleware**. Dans ce middleware, nous n'autoriserons l'accès à la route que si l'entrée du **token** fourni correspond à une valeur spécifiée. Sinon, nous redirigerons les utilisateurs vers l'URI **home** :

```
<?php

namespace App\Http\Middleware;

use Closure;

class EnsureTokenIsValid
{
    /**
     * Handle an incoming request.
```

```
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->input('token') !== 'my-secret-token') {
            return redirect()->route('welcome');
        }

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

Comme vous pouvez le voir, si le **token** donné ne correspond pas à notre **token** secret, le middleware renverra une redirection HTTP au client ; sinon, la requête sera transmise plus loin dans l'application. Pour transmettre la requête plus loin dans l'application (en permettant au middleware de "passer"), vous devez appeler la callback **$next** avec la requête **$request**.

Il est préférable d'envisager le middleware comme une série de "couches" que les requêtes HTTP doivent traverser avant d'atteindre votre application. Chaque couche peut examiner la requête et même la rejeter entièrement.

```
Route::get('students/{student}', function (Student $student) {

  return  '<h1>'.$student->name.'</h1>';

})->name('student')->middleware('token');
```

Ce middleware vérifie si le paramètre **token** passé dans l'URL correspond bien à la chaine `'my-secret-token'.` Sinon on se redirige vers la page d'accueil.

Exemple : Visitez l'URL suivante pour tester le Middleware avec des paramètres

localhost:8000/students/1?token=my-secret-token

3. Enregistrement

3.1 Middleware Globale

Si vous souhaitez qu'un middleware s'exécute lors de chaque requête HTTP adressée à votre application, répertoriez la classe middleware dans la propriété **$middleware** de votre classe **app/Http/Kernel.php**.

```
   protected $middleware = [
      // \App\Http\Middleware\TrustHosts::class,
        \App\Http\Middleware\TrustProxies::class,
        \App\Http\Middleware\EnsureTokenIsValid::class,
        \Illuminate\Http\Middleware\HandleCors::class,
        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
        \App\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    ];
```

3.2 Affectation de middleware aux routes

Si vous souhaitez affecter un **middleware** à des routes spécifiques, vous devez d'abord affecter au middleware une **clé** dans le fichier **app/Http/Kernel.php** de votre application. Par défaut, la propriété **$routeMiddleware** de cette classe contient des entrées pour le **middleware** inclus avec **Laravel**. Vous pouvez ajouter votre propre middleware à cette liste et lui attribuer une clé de votre choix :

```
    protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
        'signed' => \App\Http\Middleware\ValidateSignature::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
        'token' => \App\Http\Middleware\EnsureTokenIsValid::class,
    ];
}
```

3.3 Groupe de Middlewares

Parfois, vous souhaiterez peut-être regrouper plusieurs middlewares sous une seule clé pour faciliter leur affectation aux routes. Vous pouvez accomplir cela en utilisant la propriété **$middlewareGroups** de votre noyau HTTP.

Prêt à l'emploi, Laravel est livré avec des groupes **middlewares Web** et **API** qui contiennent des middlewares communs que vous voudrez peut-être appliquer à vos routes **Web** et **API**. N'oubliez pas que ces groupes de middleware sont automatiquement appliqués par le fournisseur de services **App\Providers\RouteServiceProvider** de votre application aux routes dans vos fichiers de route Web et API correspondants :

```
 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' => [
            //
\Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
            'throttle:api',
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
        ],
    ];
```

4. Activation du middleware

Le middleware peut être activé sur n'importe quelle route. Dans le fichier **routes/web.php**, nous pouvons utiliser la fonction **->middleware()** pour demander à Laravel d'activer le middleware :

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

Vous pouvez affecter plusieurs middlewares à la route en transmettant un tableau de noms de middlewares à la méthode middleware :

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

5. L’exclusion de middleware

Lors de l'attribution d'un **middleware** à un groupe de routes, vous devrez peut-être parfois empêcher le middleware de s’appliquer sur une route individuelle au sein du groupe. Vous pouvez accomplir cela en utilisant la méthode **withoutMiddleware**

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

La méthode **withoutMiddleware** ne peut supprimer que le middleware de route et ne s'applique pas au middleware global.

6. Paramétrage

Le **middleware** peut également recevoir des paramètres supplémentaires. Par exemple, si votre application doit vérifier que l'utilisateur authentifié a un "rôle"

donné avant d'effectuer une action donnée, vous pouvez créer un middleware **EnsureUserHasRole** qui reçoit un nom de rôle comme argument supplémentaire. Des paramètres middleware supplémentaires seront transmis au middleware après l'argument **$next** :

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class EnsureUserHasRole
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure(\Illuminate\Http\Request):
(\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse)
$next
     * @return
\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse
     */
    public function handle(Request $request, Closure
$next,$role)

    {            if ($role=="admin"){
                return $next($request);
    }
    else

     return redirect()->route('welcome');


    }
}
```

Les paramètres du **middleware** peuvent être spécifiés lors de la définition de la route en séparant le nom du middleware et les paramètres par un **:** .

Plusieurs paramètres doivent être délimités par des virgules :

```
Route::get('/profile', function () {

  return "my profile";})->middleware('role:admin') ;
```

7. Terminable Middleware

Un **middleware terminable** exécute une tâche après l'envoi de la réponse au navigateur. Ceci peut être accompli en créant un middleware avec la méthode **terminate** dans le middleware. **Le middleware terminable** doit être enregistré dans le middleware global. La méthode **terminate** recevra deux arguments : **$request** et **$response**. La méthode terminate peut être créée comme indiqué dans le code suivant.

Exemple

Etape 1 : Créer **TerminateMiddleware** en exécutant la commande suivante.

```
php artisan make:middleware TerminateMiddleware
```

Etape 2 : Copiez le code suivant dans le **TerminateMiddleware** nouvellement créé à l'adresse **app/Http/Middleware/TerminateMiddleware.php**

```
<?php

namespace App\Http\Middleware;
use Closure;

class TerminateMiddleware {
   public function handle($request, Closure $next) {
      echo "Executing statements of handle method of
TerminateMiddleware.";
      return $next($request);
   }

   public function terminate($request, $response) {
      echo "<br>Executing statements of terminate method of
TerminateMiddleware.";   }}
```

Etape 3 : Enregistrez le **TerminateMiddleware** dans le fichier **app\Http\Kernel.php**. Ajoutez la ligne surlignée en jaune dans ce fichier pour enregistrer **TerminateMiddleware.**

```
    protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
        'signed' => \App\Http\Middleware\ValidateSignature::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
         'token' => \App\Http\Middleware\EnsureTokenIsValid::class,
         'terminate'=>\App\Http\Middleware\TerminateMiddleware::class
    ];
}
```

Etape 4 : Ajoutez la ligne de code suivante dans le fichier **app/ routes/web.php**.

```
Route::get('/home', function () {

     return '<h1>home page</h1>';
})->name('welcome')->middleware('terminate');
```

Etape 5 : Visitez l'URL suivante pour tester le Terminable Middleware.

localhost:8000/home

La sortie apparaîtra comme indiqué dans l'image suivante.