Développer en back-end

Les middlewares

Formatrice : Elidrissi Asmae

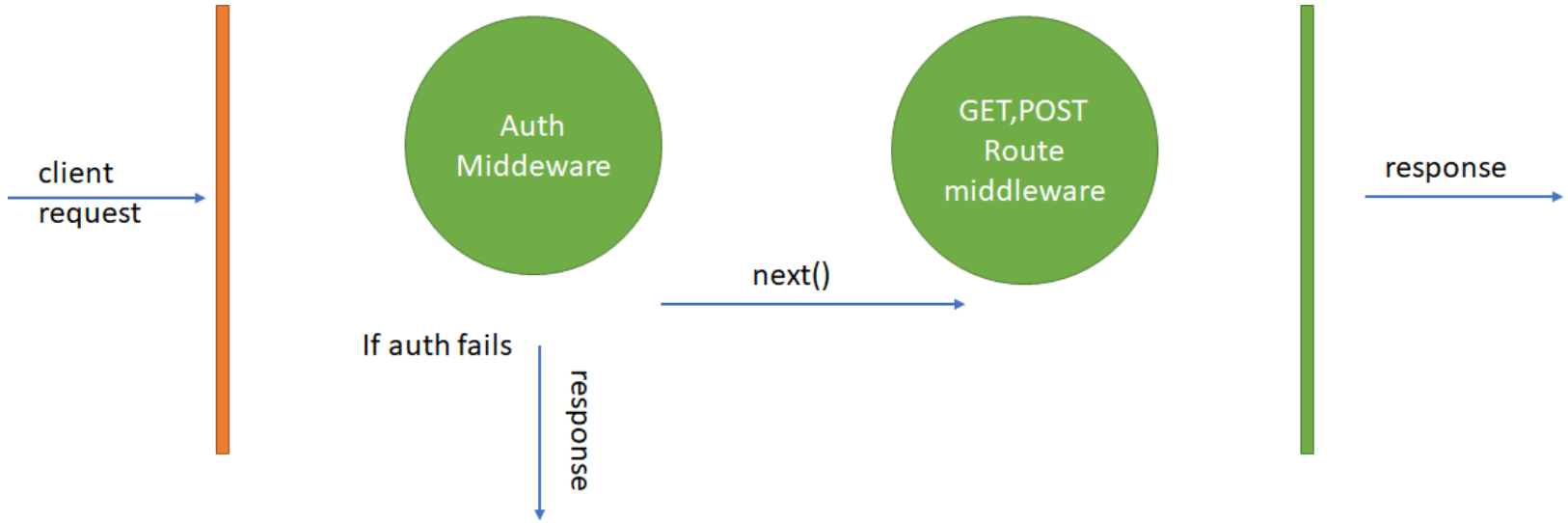
OFPPT
La Voie de l'Avenir

Plan du cour

Définition

Définition

- Un middleware (traduction textuelle : quelque chose qui est au milieu) est un mécanisme qui permet de **filtrer** les requêtes http. Autrement dit, le middleware est une **couche** placée **entre la requête http et la réponse**. Elle permet d'effectuer une opération lorsqu'une condition est rencontrée.
- Plusieurs middlewares sont inclus dans le framework Laravel, notamment des middlewares d'authentification et la protection CSRF. Tous ces middlewares sont situés dans le répertoire app/Http/Middleware.
- Par exemple, le middleware **auth** permet de vérifier si un usager est authentifié avant de donner l'accès à une route donnée. S'il ne l'est pas, l'usager sera redirigé à une page d'authentification. Tout ceci sans que vous ayez à encombrer vos contrôleurs par une tonne de code.



Création et utilisation d’un middleware

Création et utilisation d'un middleware

Définition :

Pour créer un nouveau middleware, utilisez la commande Artisan ***make:middleware*** :

Exemple:

```
php artisan make:middleware CheckAge
```

Cette commande placera une nouvelle classe « CheckAge » dans le répertoire ***app/Http/Middleware***.

Dans cet exemple, on désire ne pas autoriser les personnes ayant moins de 18 ans d'accéder à une page secrète;

On ajoute la logique de traitement dans la méthode **handle** du middleware crée:

- Si l'age est inférieure à 18, on redirige l'utilisateur à la page précédente avec un message stocké en « error ».
- Sinon, la requête sera transmise à l'application pour terminer la demande.

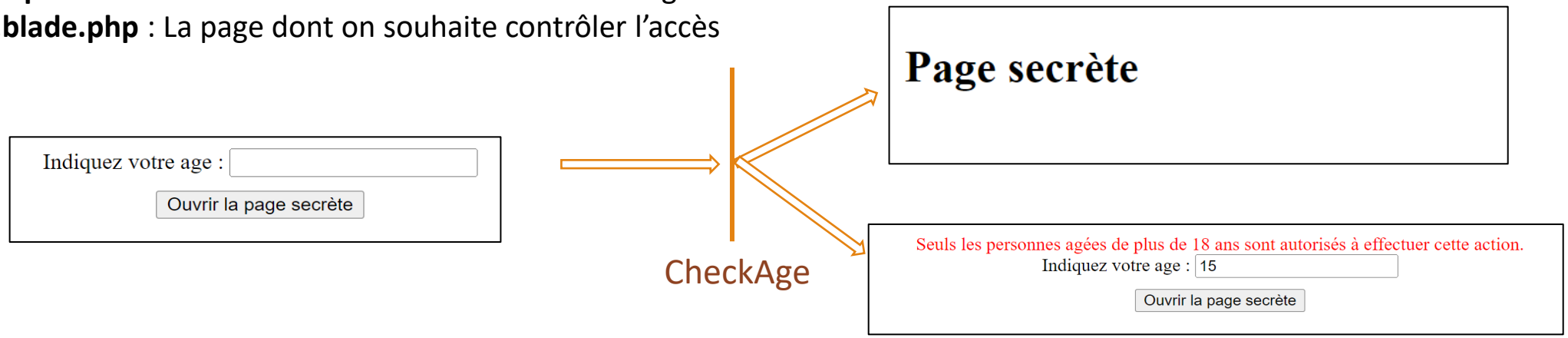
```
class CheckAge
{

   public function handle(Request $request, Closure
$next): Response
    {
        if($request->has("age") && $request->age<18)
            return redirect()->back()->with('error',
  'Seuls les personnes âgées de plus de 18 ans sont
  autorisés à effectuer cette action.')->withInput();


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

Création et utilisation d'un middleware

Utilisation:

Soient deux vues:

- **Home.blade.php** où on demande à l'utilisateur de donner son age
- **Page_secrète.blade.php** : La page dont on souhaite contrôler l'accès



On peut exploiter le middleware "CheckAge" en l'affectant à la route affichant la vue page_secrete :

```
Route::view("home", "home");
Route::view("page_secrete", "page_secrete")
    ->name("page_secrete")
    ->middleware(CheckAge::class);
```

On peut même affecter à une route plusieurs middelwares à la fois, en écrivant :

```
Route::get('/', function () {
// ...
})->middleware([First::class, Second::class]);
```

Création et utilisation d'un middleware

un middleware peut effectuer des tâches avant ou après avoir transmis la requête plus profondément dans l'application.

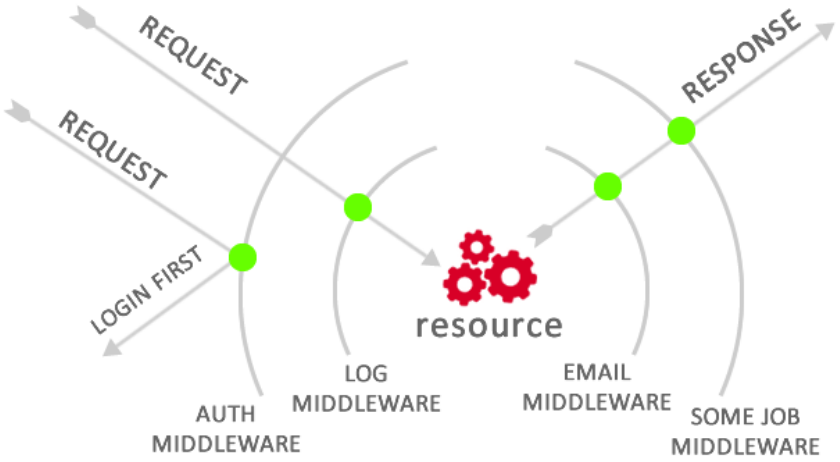
Exemples:

Avant la transmission de la requête :

```
class BeforeMiddleware
{
     public function handle(Request $request, Closure
     $next): Response{

          // Perform action

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

Après la transmission de la requête :

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

          // Perform action

          return $response;
    }
}
```



Affectation à un groupe de routes

Affectation à un groupe de routes

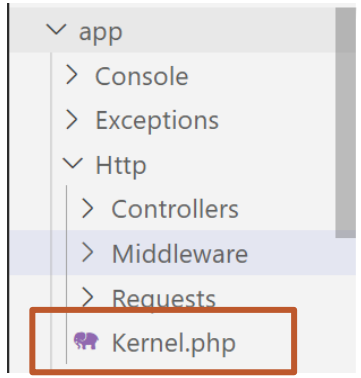
- Pour affecter un middleware à toutes les routes d'un groupe, vous pouvez utiliser la méthode middleware avant de définir le groupe.
- Les middelwares sont exécutés dans l'ordre dans lequel ils sont répertoriés dans le tableau :

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

    Route::get('/user/profile', function () {
    // Uses first & second middleware...
    });
});
```

Attribuer un alia à un middleware

Attribuer un alia à un middleware

- Laravel donne la possibilité d'attribuer des alias (surnoms) aux middlewares dans le fichier ***app/Http/Kernel.php*** de l'application.
- Par défaut, la propriété ***$middlewareAliases*** de cette classe contient des entrées pour les middlewares inclus dans Laravel.
- On peut ajouter un middleware à cette liste et lui attribuer un alias de notre choix :




```
protected $middlewareAliases = [
        '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,
        'age'=> \App\Http\Middleware\CheckAge::class
    ];
```

Ainsi, on peut utiliser l'alias lors de l'affectation du middleware aux routes :

```
Route::view("page_secrete", "page_secrete")
    ->name("page_secrete")
    ->middleware('age');
```

Middleware global

Middleware global

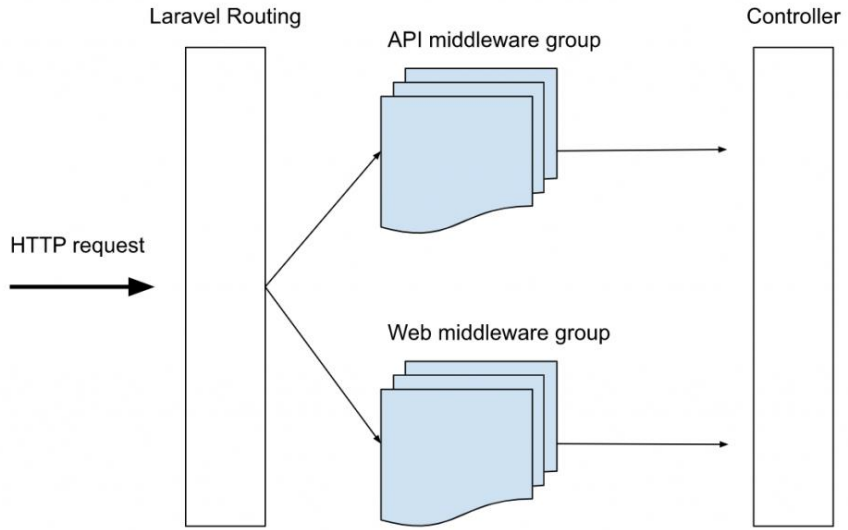
Si on souhaite qu'un middleware s'exécute lors de **chaque requête HTTP** adressée à l'application, il suffit de répertorier sa classe dans la propriété **$middleware** de la classe **app/Http/Kernel.php.**

```
protected $middleware = [
    \App\Http\Middleware\TrustProxies::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,
    \App\Http\Middleware\MaClasseMiddelware::class
];
```

Middleware de groupe

- Parfois, on souhaite regrouper plusieurs middlewares sous une seule clé pour faciliter leur affectation aux routes. On peut accomplir cela en utilisant la propriété **$middlewareGroups** de la classe **app/Http/Kernel.php**.
- Laravel comprend des groupes prédéfinis ***web*** et ***api*** qui contiennent des middleware communs qu'on peut appliquer à vos routes Web et API.

```php
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,
        \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```



les groupes de middlewares facilitent l'affectation simultanée de plusieurs middlewares à une route :

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

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

Middleware avec paramètre

Le middleware peut également recevoir des paramètres supplémentaires.
Exemple :

- Si on veut créer un middleware qui vérifie si la demande provient d'une adresse IP de confiance ou non.
- Le middleware devrait autoriser les demandes provenant d'une liste d'adresses IP de confiance et refuser les demandes provenant de toute autre adresse IP.

1. Créez un nouveau middleware en utilisant la commande artisan :

   php artisan make:middleware TrustIpMiddleware

2. Ajoutez le code suivant à la méthode ***handle*** de la classe ***TrustIpMiddleware***

```
public function handle($request, Closure $next, ...$trustedIps) {
        $clientIp = $request->getClientIp();
        if (!in_array($clientIp, $trustedIps))
        {
          return response('Non autorisé', 401);
        }
        return $next($request);
    }
```

*Les trois points (appelés "splat operator" ou "variadic parameter") dans la signature de la méthode **handle** de ce middleware permettent de recevoir un nombre variable d'arguments dans la méthode*

Middleware avec paramètre

3. On peut utiliser le middleware dans nos routes ou groupes de routes:

   a. ***Exemple 1:*** pour autoriser les demandes uniquement depuis l'adresse IP **192.168.1.100**, on peut ajouter le middleware à une route de cette manière :

```
Route::get('/protege1', function () {
        return 'Ceci est une ressource protégée.';
    })->middleware('trustip:192.168.1.100');
```

   b. ***Exemple 2:*** pour autoriser les demandes provenantes depuis l'adresse IP **192.168.1.100 et 192.168.1.82**, on peut écrire :

```
Route::get('/protege2', function () {
        return 'Ceci est une ressource protégée.';
    })->middleware('trustip:192.168.1.100, 192.168.1.82’);
```

Notez:

trustip*’ est un alia du middleware :* ***TrustIpMiddleware***

À la prochaine !