

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;  
21  
22 while (again) {  
23     iN = -1;  
24     again = false;  
25     getline(cin, sInput);  
26     system("cls");  
27     stringstream(sInput) >> dblTemp;  
28     iLength = sInput.length();  
29     if (iLength < 4) {  
30         again = true;  
31         continue;  
32     } else if (sInput[iLength - 3] != ' ') {  
33         again = true;  
34         continue;  
35     } while (++iN < iLength) {  
36         if (isdigit(sInput[iN])) {  
37             continue;  
38         } else if (iN == (iLength - 3)) {  
39             continue;  
40         }  
41     }  
42 }
```

LARAVEL

—

Module 05

Middlewares et Providers



Au programme dans ce module

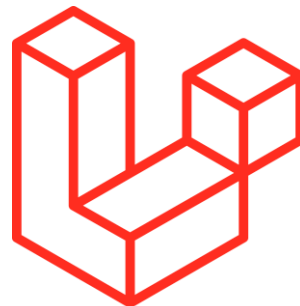
- ❑ Chapitre 1 : Les Middlewares
 - Leur fonctions
 - Middleware d'authentification
 - Place à la pratique
- ❑ Chapitre 2 : Les Providers
 - Leur fonctions
 - Place à la pratique



Les Middlewares



Chapitre 1



Leurs utilités 1/2

→ Un middleware sert à effectuer des actions répétitives, principalement, avant que la requête soit transmise au méthode du Contrôleur.

```
protected $middlewareGroups = [  
    'web' => [  
        \App\Http\Middleware\EncryptCookies::class,  
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,  
        \Illuminate\Session\Middleware\StartSession::class,  
        \Illuminate\Session\Middleware\AuthenticateSession::class,  
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,  
        \App\Http\Middleware\VerifyCsrfToken::class,  
        \Illuminate\Routing\Middleware\SubstituteBindings::class,  
    ],  
  
    'api' => [  
        'throttle:60,1',  
        \Illuminate\Routing\Middleware\SubstituteBindings::class,  
    ],  
];
```

```
protected function mapWebRoutes()  
{  
    Route::middleware('web')  
        ->namespace($this->namespace)  
        ->group(base_path('routes/web.php'));  
}
```

RouteServiceProvider.php

```
▼ Http  
▶ Controllers  
▶ Middleware  
Ⓢ Kernel.php
```

→ Il existe des groupes de Middlewares appeler lors du chargement des fichiers de Routes.



Leurs utilités 2/2

→ Dans le cycle de vie d'une requête, les middlewares peuvent se place dans les Route, entre la Route et le Contrôleur ou dans le Contrôleur.

```
// Route avec authentification obligatoire
Route::get('user/{id}', 'UserController@home')->middleware('auth');
```

→ Depuis la route

```
class UserController extends Controller
{
    /**
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth')->except('welcome');
    }
}
```

→ Depuis le contrôleur

```
$this->middleware('auth')->only(['home', 'show', 'update']);
```



Middleware d'authentification

- Laravel appelle automatiquement la méthode « handle() » lorsqu'il s'agit d'un middleware.
- Un middleware n'est pas différent d'une autre classe mais défini par l'endroit où il est appelé.

```
class Authenticate extends Middleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request $request
     * @param  \Closure $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if (Auth::guard('user')->guest()) {
            return response('Unauthorized 401.');
```

→ La présence des 2 premiers paramètres est obligatoire. C'est Laravel qui les remplit.

→ C'est par cette méthode que le cycle de vie de la requête continue.



Place à la pratique

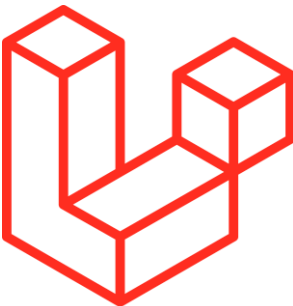
- ❖ Cherchez comment avec Laravel on peut savoir si une requête est en JSON ou en AJAX avec les méthodes de la classe Request.
- ❖ Créez un middleware (avec la commande : **php artisan make:middleware MIDDLEWARE_NAME**). Ce dernier devra bloquer toutes les requêtes JSON et AJAX.
- ❖ Cherchez comment utiliser le helper « **abort()** ».
- ❖ Utilisez le helper « **abort()** » comme réponse si la requête est transmise en JSON ou AJAX.
- ❖ Créez un alias du Middleware dans « kernel.php ».
- ❖ Créez une route avec la fonction `middleware` qui utilisera l'alias du middleware.
- ❖ Pour tester la requête, utilisez Postman pour envoyer une requête JSON (application/json) à la route utilisant le Middleware que vous venez de créer.



Les Providers

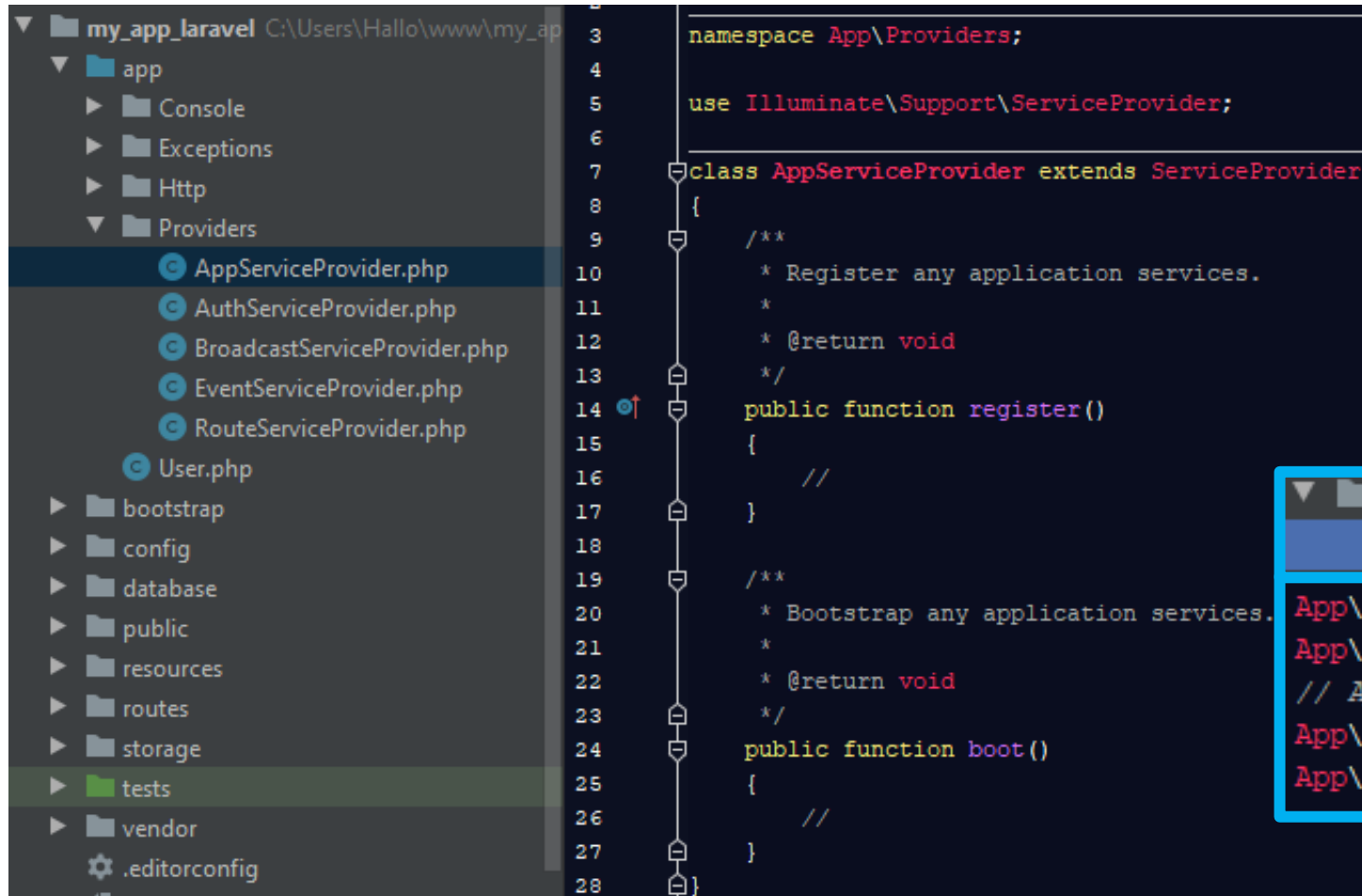


Chapitre 2



Leur fonctions

→ Laravel appelle automatiquement la méthode « boot() » lorsqu'il s'agit d'un Providers.



```
3 namespace App\Providers;
4
5 use Illuminate\Support\ServiceProvider;
6
7 class AppServiceProvider extends ServiceProvider
8 {
9     /**
10      * Register any application services.
11      *
12      * @return void
13      */
14     public function register()
15     {
16         //
17     }
18
19     /**
20      * Bootstrap any application services.
21      *
22      * @return void
23      */
24     public function boot()
25     {
26         //
27     }
28 }
```

→ Un Providers est défini par extension de la classe « ServiceProvider »

→ Les Providers sont enregistrés dans « config/app.php » dans le tableau PHP « providers »



```
App\Providers\AppServiceProvider::class,
App\Providers\AuthServiceProvider::class,
// App\Providers\BroadcastServiceProvider::class,
App\Providers\EventServiceProvider::class,
App\Providers\RouteServiceProvider::class,
```



Les Providers

```
/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    //Blade
    $this->bladeDirective();
}

/**
 * Add Blade directives
 */
protected function bladeDirective()
{
    // Debug
    Blade::directive('d', function ($expression) {
        return "<?php d($expression); ?>";
    });
    Blade::directive('dd', function ($expression) {
        return "<?php dd($expression); ?>";
    });
    Blade::directive('var_dump', function ($expression) {
        return "<?php var_dump($expression); ?>";
    });

    // Config
    Blade::directive('config', function ($expression) {
        return "<?php echo config($expression); ?>";
    });

    // Route
    Blade::directive('route', function ($expression) {
        return "<?php echo route($expression); ?>";
    });
}
```

```
// Route
Blade::directive('route', function ($expression) {
    return "<?php echo route($expression); ?>";
});
```

- A ce stade, AppServiceProvider sert simplement à exécuter du code au démarrage de l'application, comme l'ajout de directive Blade, la configuration de variables d'environnement PHP ou encore la langue choisie depuis les headers de la requête.
- Il n'y a pas de limite à l'utilisation de AppServiceProvider.
- Il est néanmoins préférable de créer d'autres ServiceProvider pour répondre à une tâche précise, comme l'ajout de directive Blade.



Place à la pratique

- ❖ Créez un nouveau ServiceProvider « BladeDirectiveServiceProvider » pour ajouter de nouvelles directives Blade.
- ❖ Placez-vous dans BladeDirectiveServiceProvider et en vous inspirant des exemples du cours :
 - ❖ Créez une directive/commande Blade pour utiliser le helper « route() » dans une vue sous la forme « @route() ».
 - ❖ Créez une directive/commande Blade pour utiliser le helper « asset() » dans une vue sous la forme « @asset() ».
 - ❖ Créez une directive/commande Blade pour utiliser le helper « dd() » dans une vue sous la forme « @debug() ».
 - ❖ Créez une directive/commande Blade pour utiliser le helper « mix() » dans une vue sous la forme « @mix() ».
- ❖ Testez ces 3 nouvelles directives dans l'une de vos vues.



```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;  
21  
22 while (again) {  
23     iN = -1;  
24     again = false;  
25     getline(cin, sInput);  
26     system("cls");  
27     stringstream(sInput) >> dblTemp;  
28     iLength = sInput.length();  
29     if (iLength < 4) {  
30         again = true;  
31         continue;  
32     } else if (sInput[iLength - 3] != '.') {  
33         again = true;  
34         continue;  
35     } while (++iN < iLength) {  
36         if (isdigit(sInput[iN])) {  
37             continue;  
38         } else if (iN == (iLength - 3)) {  
39             continue;  
40         }  
41     }  
42     // ...  
43 }
```

Fin du module

