

```
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 03 Contrôleurs

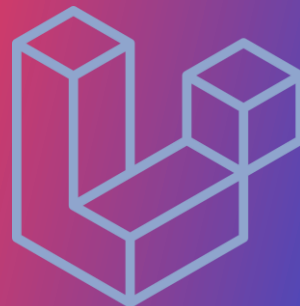


Au programme dans ce module

- ❑ Chapitre 1 : A quoi ressemble un contrôleur ?
 - Un ensemble d'actions
 - Récupérer une variable `$_GET` ou `$_POST` ou URL
 - Renvoyer une vue
 - Retour en arrière et redirections
 - Place à la pratique
 - Pour aller plus vite

- ❑ Chapitre 2 : Contrôleur avec vue (WEB)
 - Une vue toute simple
 - Des données dans les vues
 - Place à la pratique

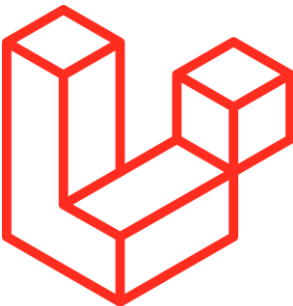
- ❑ Chapitre 3 : Contrôleur sans vue (API)
 - Une réponse simple
 - Un petit essai

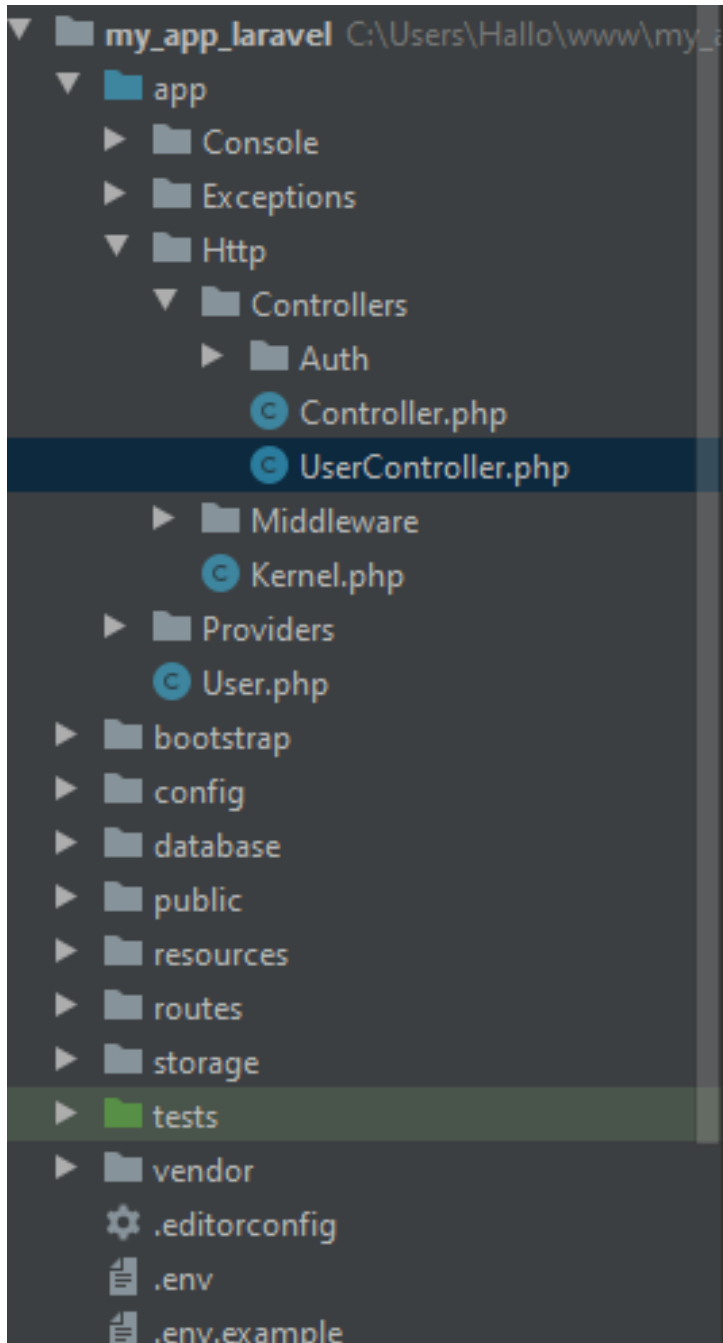


A quoi ressemble un contrôleur ?



Chapitre 1





```
3 namespace App\Http\Controllers;
4
5 use App\User;
6 use Illuminate\Http\Request;
7
8 class UserController extends Controller
9 {
10     /**
11      * @return \Illuminate\View\View
12      */
13     public function index()
14     {
15         return view('users.index');
16     }
17
18     /**
19      * @param Request $request
20      * @param int $id
21      * @return \Illuminate\View\View
22      */
23     public function show(Request $request, $id)
24     {
25         $filters = $request->all();
26         $user = User::find($id);
27
28         return view('users.show')->with([
29             'user' => $user,
30         ]);
31     }
32 }
```



A quoi ressemble un contrôleur ?

Un ensemble d'actions 1/2

```
public function index()  
{  
    return view('users.index');  
}
```

Des méthodes simples
pour afficher une vue.

Des méthodes complexes
pour récupérer et afficher
des données précises.

```
public function show(Request $request, $id)  
{  
    $filters = $request->all();  
    $user = User::find($id);  
  
    return view('users.show')->with([  
        'user' => $user,  
    ]);  
}
```



A quoi ressemble un contrôleur ?

Un ensemble d'actions 2/2

```
public function update(Request $request, $id)
{
    $inputs = $request->all();
    $user = User::find($id);

    return $this->updateOneUser($user, $inputs['update']);
}
```

Des méthodes pour
répondre à des actions
(create, update, delete, ...).

Des méthodes protégées
pour structurer et simplifier
les méthodes publiques.

```
protected function updateOneUser($user, $update)
{
    // Update code

    return back();
}
```



A quoi ressemble un contrôleur ?

Récupérer une variable \$_GET ou \$_POST ou URL

```
use Illuminate\Http\Request;
```

```
public function show(Request $request, $id = NULL)
{
    echo 'User ID : ' . $id; // NULL par défaut
    echo 'Variable : ' . $request->my_variable_front; // NULL par défaut
}
```

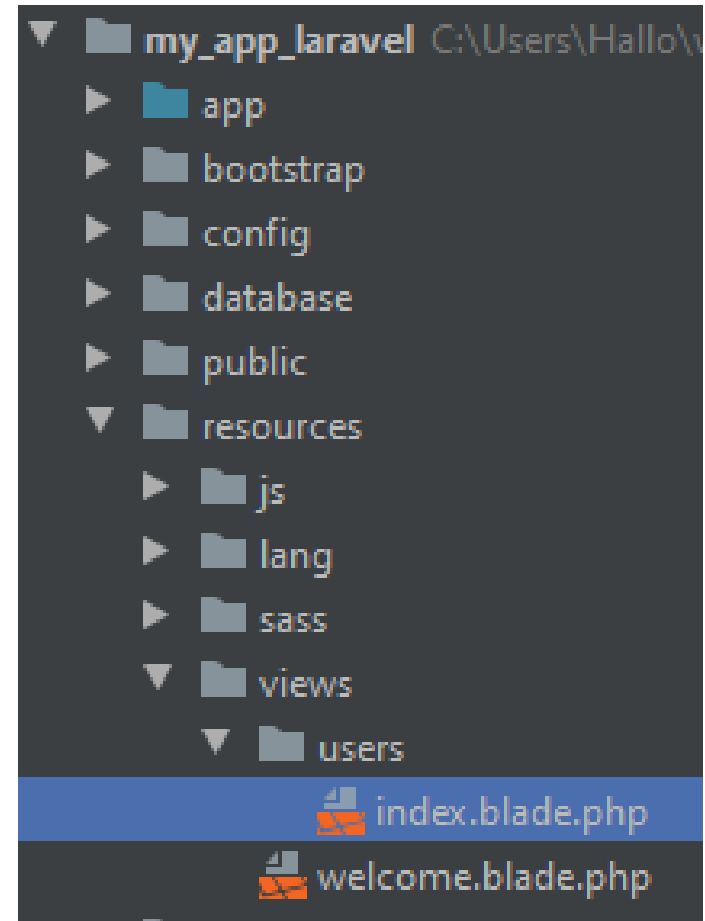


A quoi ressemble un contrôleur ?

Renvoyer une vue

```
public function index()
{
    return view('users.index');
}
```

- Le helper « view » permet de générer une vue.
- Le premier paramètre est le nom de la vue à partir de « resources/views/ ».
- Le séparateur de dossier n'est pas un / ou \ mais un simple point (anciennement pour faciliter la conversion selon que le système soit Windows ou UNIX).



A quoi ressemble un contrôleur ?

Retour en arrière et redirections

```
public function update(Request $request, $id)
{
    // Update code

    return back();
}
```

Le helper « back » redirige la requête à la requête précédant l'appel de « update() ».
En général, à la requête qui a affiché la vue d'origine (Cela dépend du routage mis en place).

```
public function update(Request $request, $id)
{
    // Update code

    return redirect()->route('ROUTE_NAME'); // Nom donné dans le fichier de routes.
}
```

Le helper « redirect » redirige la requête vers l'url en paramètre ou vers une route nommée.



Place à la pratique

- ❖ Créez les routes « index » (GET), « show » (GET), « store » (POST), « update » (PUT) et « delete » (DELETE) pour un futur contrôleur « ProductsController ». Nommez chaque route en prenant en compte le nom du contrôleur comme appris au cours précédent. Les routes « show », « update » et « delete » doivent avoir une variable d'url (pour identifier le produit ciblé par l'action).
- ❖ Créez le contrôleur « ProductsController » et ajoutez les méthodes « index », « show », « store », « update » et « delete » qui correspondent aux routes.
- ❖ Si besoin, adaptez les paramètres des méthodes du contrôleur selon les routes et leur paramètre d'url.
- ❖ Dans la méthode « index », testez les helpers suivants : « view() » (avec la vue « welcome » déjà existante), « back() », « redirect() » et « route() ».
- ❖ Y a-t-il une différence entre « redirect() ->route() » et « redirect(route()) » ?
- ❖ Cherchez l'utilité du helper « response() » dans la documentation et comment l'utiliser.
- ❖ Essayez de trouver et de renvoyer une réponse de type JSON.
- ❖ Dans la méthode « index », envoyez une variable GET (rappel: URL?var=value) en modifiant vous-même l'url du navigateur et affichez-la avec un « dd() ». (Attention à ne pas confondre avec les variables d'url que vous avez découvert récemment).
- ❖ En utilisant Postman, envoyez 3 champs key-value en POST à votre route « store ». Et affichez-les avec un dd().



A quoi ressemble un contrôleur ?

Pour aller plus vite

```
$ php artisan make:controller HomeController --resource  
Controller created successfully.
```

Cette commande crée un contrôleur seul que l'on a choisi de nommer « HomeController » avec des méthodes préconstruites. Essayez la.

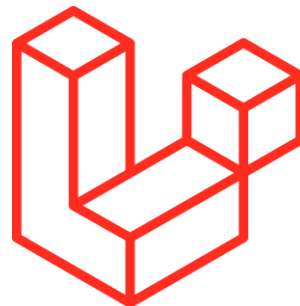
→ Pour rappel, la commande « *php artisan* » s'utilise à la racine de l'app



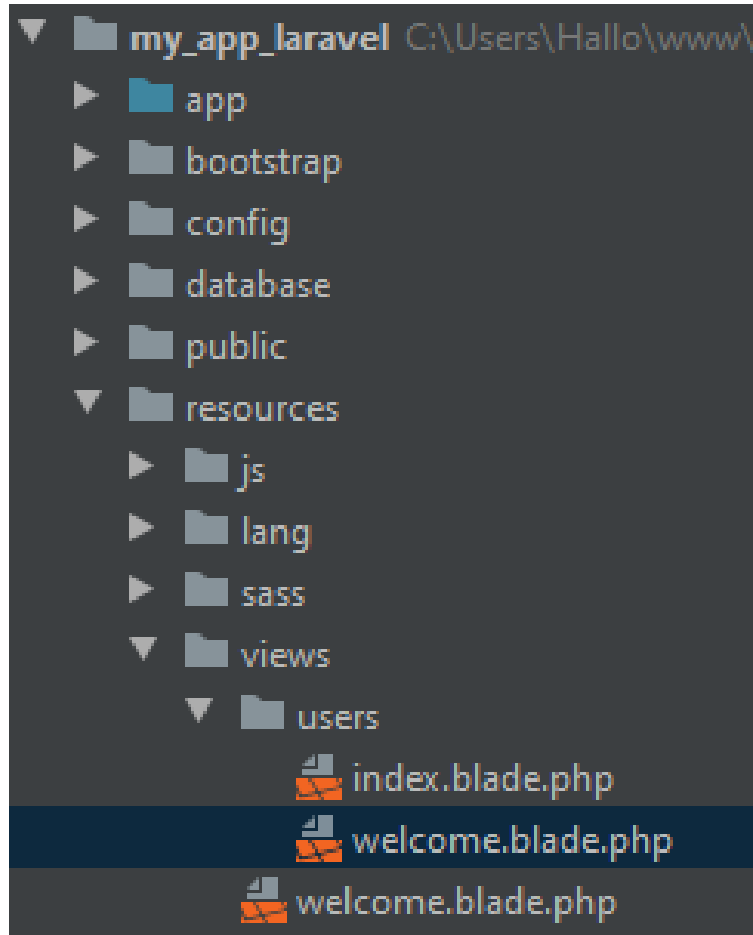
Contrôleur avec vue (WEB)



Chapitre 2



Une vue toute simple



Les vues se trouvent dans le dossier « resources/views/ »


```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Laravel</title>
</head>
<body>
    Page d'accueil des utilisateurs.
    Date : <?php echo date("j/m/Y"); ?>
</body>
</html>
```



Des données dans les vues

```
public function show($id = NULL)
{
    $user = User::find($id);

    return view('users.show', [
        'user' => $user,
    ]);
}
```



```
show.blade.php
1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="utf-8">
5      <title>Laravel</title>
6  </head>
7  <body>
8      @if(isset($user))
9          L'utilisateur s'appel : {{ $user->name }}
10     @else
11         L'utilisateur n'existe pas.
12     @endif
13 </body>
14 </html>
```

→ Laravel Blade permet d'insérer des variables plus rapidement avec

`{{ $var }}`
`{{ method() }}`



Place à la pratique

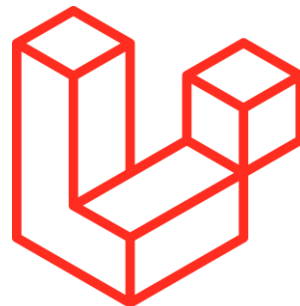
- ❖ Créez une vue « **products/create.blade.php** » avec le code html minimal. Cette vue sera retourné par la méthode « **create** » du contrôleur **ProductsController**.
- ❖ Créez un formulaire POST sur la page « **create** » avec les champs suivants : **label**, **price_ht**, **buying_price** et **description**. La cible du formulaire sera la méthode « **store** » du contrôleur « **ProductsController** » dans laquelle vous devrez afficher les données avec un dd(). Après vérification de la réception des données, commentez votre dd() et redirigez l'utilisateur sur la page du formulaire sans utiliser response() ou redirect().
- ❖ Créez une vue « **products/edit.blade.php** » qui sera affiché par la méthode « **edit** » de **ProductsController**. Dans la méthode, créez 2 variables (l'une contenant 'lorem' et l'autre 'ipsum'). Envoyez les 2 variables à la vue d'édition et affichez-les dans cette dernière avec des accolades Blade.
- ❖ Ajouter un lien de redirection dans la vue « **products.edit** » vers la page de création de produit. Ajoutez en un autre vers la route « **index** » des produits.



Contrôleur sans vue (API)



Chapitre 3



Une réponse simple

Une API peut avoir plusieurs formes et définitions selon le contexte.
Pour faire simple, nous parleront ici simplement d'**échanges sans VUE**.

```
public function show($id)
{
    $user = User::find($id);

    return response()->json([
        'user' => $user->toArray(),
    ]);
}
```

C'est la forme la plus simple d'échange. Néanmoins, il faut choisir la forme de la réponse.

Ici nous répondons en **JSON**.

L'API peut servir en WEB pour répondre à une requête AJAX ou pour des échanges simples avec d'autres services comme une application mobile.



Un petit essai

- ❖ Utilisez Postman pour envoyer 2 strings, 1 nombre entier, 1 nombre à virgule et 1 booléen à une nouvelle méthode « **data** » (POST) d'un nouveau contrôleur « **ApiController** » qui retournera (répondra) directement ces variables mais au format JSON.



```
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

