Développer en back-end

Manipulation des vues avec Blade

Formatrice : Elidrissi Asmae

OFPPT
La Voie de l'Avenir

Plan du cour

Le moteur Blade

Le moteur de gabarit Blade :

- Blade :
  - est le moteur de template utilisé par Laravel.
  - Son but est de permettre d’utiliser du php sur notre vue mais d’une manière assez particulière.
- Pour créer un fichier qui utilise le moteur de template Blade il vous faut ajouter l’extension “.blade.php”.
- Selon l’architecture des projets Laravel, les vues se situent sous le répertoire resources/views.

Les variables

Les variables : Afficher des données

- La première fonctionnalité la plus basique de Blade est l'affichage d'une simple variable:

Exemple : Si je transmets à ma vue la variable $maVariable = 'Hello World !' Dans ma vue {{ $maVaribale }} affichera 'Hello World' .

Il existe une variante de ces accolades pour vous permettre de ne pas échapper les caractères html.

```
$mavariable = '<h1>Mon Titre</h1>';
{{ $maVariable }}    <!-- donnera : '<h1>Mon Titre</h1>' -->
{!! $maVariable !!}   <!-- donnera : 'Mon Titre' dans une balise HTML h1 -->
```

Les variables : Passer des variables aux vues

- Vous pouvez afficher les données transmises à vos vues Blade en entourant la variable d'accolades. Par exemple, avec la route suivante :

```
Route::get('/', function () {
return view('welcome', ['name' => 'ALLAOUI']);
});
```

Vous pouvez afficher le contenu de la variable name comme ceci :

```
Hello, {{ $name }}.
```

Note: Par défaut, les instructions Blade {{ }} sont automatiquement envoyées via la fonction htmlspecialchars de PHP pour empêcher les attaques XSS. Si vous ne souhaitez pas que vos données soient échappées, vous pouvez utiliser la syntaxe suivante : **Hello, {!! $name !!}.**

Les directives

Les directives : Conditions

Blade vous permet de manipuler des données comme le ferait le php, il vous est donc possible d'utiliser les différentes structures de contrôle PHP existantes. À la différence que leur écriture diffèrent un tout petit peu.

Pour mettre en place une condition : @if / @elseif / @else, @switch, @isset, @empty#

```
<?php $animal = 'cheval'; ?>
@if ( $animal === 'chien' )
          <p>L'animal est un chien.</p>
@elseif ( $animal === 'chat' )
          <p>L'animal n'est pas un chien.</p>
@else
          <p>L'animal n'est ni un chat ni un chien.</p>
@endif
```

```
<p>L'animal n'est ni un chat ni un chien.</p>
```

Les directives : Conditions

En plus des directives conditionnelles déjà évoquées, les directives **@isset** et **@empty** peuvent être utilisées comme raccourcis pratiques pour leurs fonctions PHP respectives :

@isset

```
<?php $produit = 'costume'; ?>
@isset($produit)
            <p>Le produit existe</p>
@endisset
```

Donnera :

```
<p>Le produit existe</p>
```

@empty

```
<?php $produit = ''; ?>
@empty($produit)
            <p>Le produit n'existe pas.</p>
@endempty
```

Donnera :

```
<p>Le produit existe</p>
```

Les directives : Boucles

Les instructions Switch peuvent être construites à l'aide des **@switch** , **@case** , **@break** , **@default** et **@endswitch** :

```
<?php $age = 19; ?>

    @switch($age)
         @case( $age < 18 )
             <p>La personne est mineure.</p>
             @break
        @case( $age > 18 )
             <p>La personne est majeure.</p>
             @break
        @default
             <p>valeur par défaut.</p>
        @endswitch
```

Donnera :

```
<p>La personne est majeure.</p>
```

Les directives : Boucles

@while

```
$i = 1;

@while ($i < 3)
        <p>$i est égal à {{ $i ++ }}</p>
@endwhile
```

Donnera :

```
<p>$i est égal à 1</p>
<p>$i est égal à 2</p>
```

@foreach

```
$letters = ['a', 'b', 'c’];

@foreach ( $letters as $letter )
        <p>Lettre : {{ $letter }}</p>
@endforeach
```

Donnera :

```
<p>Lettre : a</p>
<p>Lettre : b</p>
<p>Lettre : c</p>
```

Les directives : Boucles

@for

```
$numbers = [1, 2, 3];

@for ($i = 0; $i < count($numbers); $i++)

        <p>Nombre : {{ $numbers[$i] }}</p>

@endfor
```

Donnera :

```
<p>Nombre : 1</p>
<p>Nombre : 2</p>
<p>Nombre : 3</p>
```

Les directives : Boucles

Le **@forelse** est un foreach qui vous permet de retourner ce que vous souhaitez si le tableau est vide. Cela vous économisera l'ajout d'une condition if

```
$animals = ['chien', 'chat', 'cheval'];
@forelse ($animals as $animal)
        <li>{{ $animal }}</li>
@empty
        <p>Aucun animal existant.</p>
@endforelse
```

Donnera :

```
<li>chien</li>
<li>chat</li>
<li>cheval</li>
```

Les directives : Boucles

Lorsque vous utilisez des boucles, vous pouvez également terminer la boucle ou sauter l'itération en cours :

```
$days = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']; ?>
@foreach ($days as $day)
     @if ($day == 'mardi’)
          @continue
     @endif
     <p>{{ $day }}</p>
     @if ($day == 'jeudi’)
          @break
     @endif
@endforeach
```

Donnera :

```
<p>lundi</p>
<p>mercredi</p>
<p>jeudi</p>
```

Vous pouvez également inclure la condition avec la déclaration de la directive sur une seule ligne :

```
@foreach ($days as $day)
     @continue($day == 'mardi’)
     <p>{{ $day }}</p>
     @break($day == 'jeudi')
@endforeach
```

Les directives : Boucles

Lors de la boucle, une variable **$loop** sera disponible à l'intérieur de votre boucle. Cette variable donne accès à des informations utiles telles que l'index de la boucle actuelle et s'il s'agit de la première ou de la dernière itération de la boucle :

```
$days = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']; ?>
@foreach ($days as $day)
@if ($loop->first)
<p>C'est le premier jour de la semaine : {{ $day }}</p>
@endif
@if ($loop->last)
<p>C'est le dernier jour de la semaine : {{ $day }}</p>
@endif
@endforeach
```

Donnera :

```
<p>C'est le premier jour de la semaine : lundi</p>
<p>C'est le dernier jour de la semaine : dimanche</p>
```

Les directives : Boucles

La variable **$loop** contient également une variété d'autres propriétés utiles :

Propriété	Description
$loop->index	L'index de l'itération actuelle de la boucle (commence à 0).
$loop->iteration	L'itération actuelle de la boucle (commence à 1).
$loop->remaining	Le nombre d'itérations restantes dans la boucle.
$loop->count	Le nombre total d'éléments dans le tableau en cours d'itération.
$loop->first	Indique si c'est la première itération de la boucle.
$loop->last	Indique si c'est la dernière itération de la boucle.
$loop->even	Indique si c'est une itération paire dans la boucle.
$loop->odd	Indique si c'est une itération impaire dans la boucle.
$loop->depth	Le niveau d'imbrication de la boucle actuelle.
$loop->parent	Dans une boucle imbriquée, représente la variable de boucle du parent.

Les directives :

Les commentaires:

```
{{-- This comment will not be present in the rendered HTML --}}
```

Code PHP :

Dans certaines situations, il est utile d'intégrer du code PHP dans vos vues. Vous pouvez utiliser la directive Blade **@php** pour exécuter un bloc de PHP simple dans votre modèle :

```
@php
//Ceci est de PHP
@endphp
```

Les directives :

Plus de directives :

- **`@auth() ... @endauth :`** bloc exécuté si il y a une personne authentifiée.
  - **`@auth('admin') ... @endauth :`** bloc exécuté si il y a une personne avec le rôle 'admin' authentifiée.
- **@guest … @endguest**

Mise en forme des vues

Mise en forme des vues

La plupart des applications Web conservent la même disposition générale sur différentes pages.

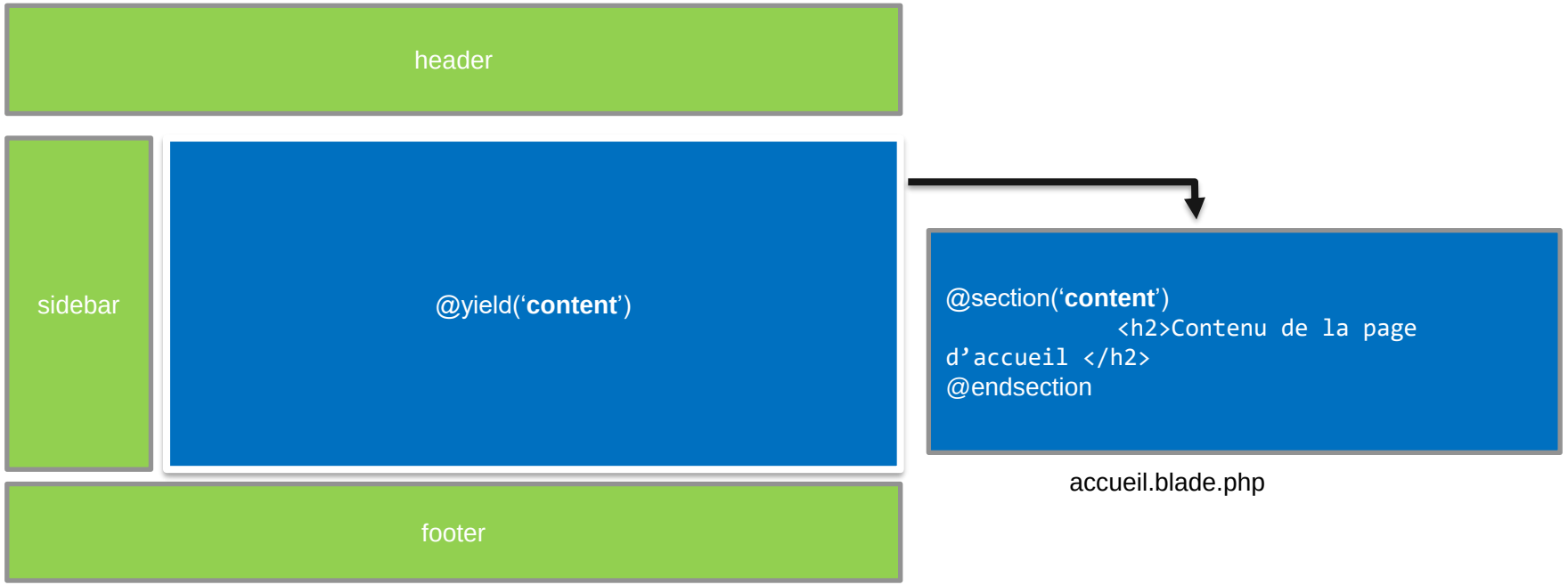
Il serait incroyablement fastidieux et difficile de maintenir notre application si nous devions répéter l'intégralité de la mise en page HTML dans chaque vue que nous créons.

Laravel propose deux méthodes pourront faciliter la mise en forme des différentes pages de l'application :

1. Mise en page en utilisant **l'héritage des modèles**
2. Mise en page en utilisant **les composants Blade**;

Mise en forme des vues :

« l'héritage de modèle » était le principal moyen de créer des applications avant l'introduction des composants . Supposant que toutes les pages d'une application utilisent la même structure: header, sidebar et footer. Ainsi, il est pratique de définir cette disposition comme une seule vue Blade :



Mise en forme des vues :

Exemple : Créer un modèle « master.blade.php » :

```
<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            Le menu principal
        @show

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

@section permet de définir une section

@show désigne la fin d'une section et déclenche son affichage (ici la chaîne Le menu principal.).

@yield désigne un espace qui sera remplacé par une section.

Mise en forme des vues :

Exemple : Hériter d'un modèle :

```
@extends('master')
@section('title', 'Titre de la page')
@section('sidebar')
    @parent
    <p>Paragraphe ajouter en plus dans le menu principal.</p>
@endsection
@section('content')
    <p>This is my body content.</p>
@endsection
```

@extends indique que cette page a pour base le modèle indiquée (master).

Contenu de la section 'sidebar' du parent

@endsection indique la fin d'une section.

Mise en forme des vues : Les composants

Il est possible de définir des parties de page que l'on va pouvoir **utiliser dans plusieurs pages** de l'application en créant des composants (***component***).

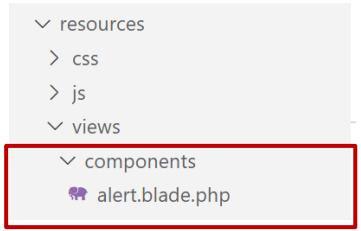
Pour créer un composant basé sur une classe, vous pouvez utiliser la commande **make:component** de Artisan

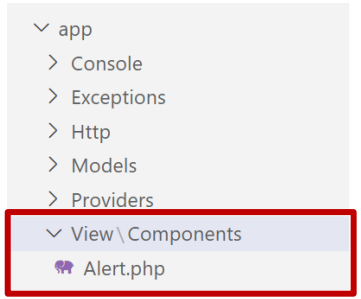
php artisan make:component **nom-composant**

Exemple: php artisan **make:component A**lert

Suite à l'exécution de cette commande, deux fichiers ont été crée :

- *la vue alert.blade.php sous views/components*
- *et la classe Alert.php sous app/View/Component*





Mise en forme des vues :

Le fichier alert.blade.php (Vue)

C'est ici que tu écris le code HTML du composant.

Exemple de contenu :

```
<div class="alert alert-{{ $type }}">
                {{ $slot }}
</div>
```

- **$type** : Une variable passée au composant.
- **$slot** : L'endroit où le contenu sera inséré dans le composant.

Le fichier alert.php

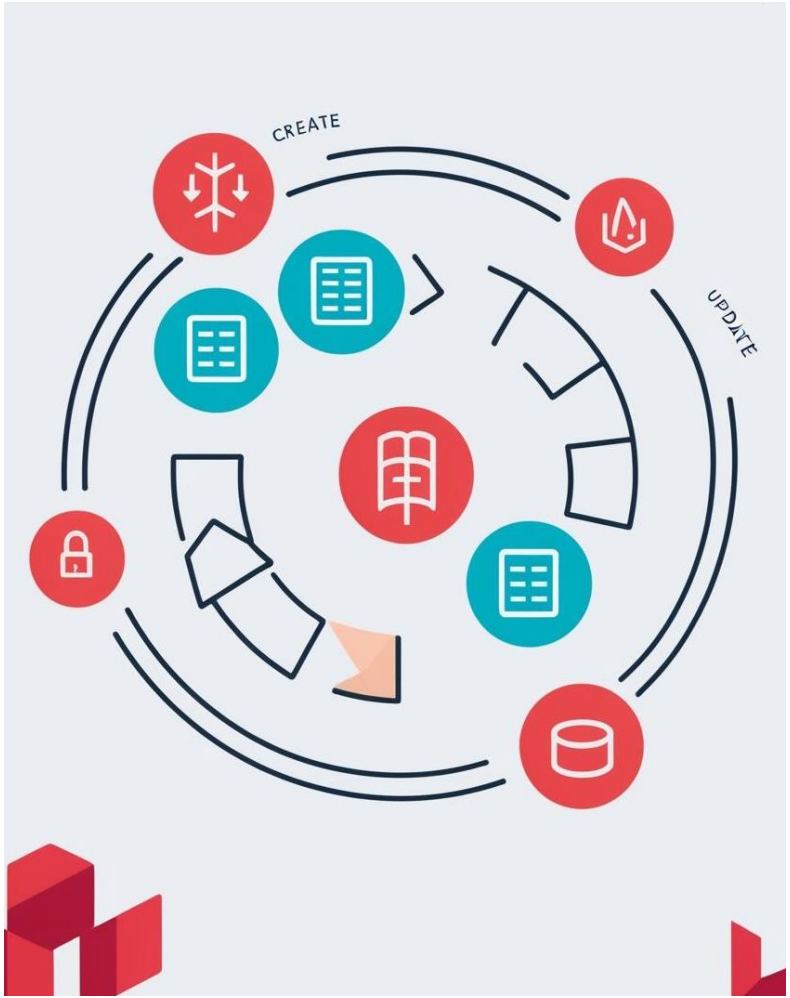
```
namespace App\View\Components;
use Illuminate\View\Component;

class Alert extends Component {
    public $type;
    // Le constructeur permet de passer des données au composant
    public function __construct($type = 'info') {
        $this->type = $type;
    }
// Renvoie la vue du composant
    public function render() {
        return view('components.alert');
    }
}
```

Mise en forme des vues : Les composants

3. L’utilisation de ce composant par la vue appelante sera désormais ainsi:

```
<x-alert type="success">
        C'est une alerte de succès !
</x-alert>
```

- <x-alert> : Utilise le composant.
- type="success" : Passe une valeur à la variable $type.
- Le texte "C'est une alerte de succès !" est placé dans $slot.



Bibliothèques de mise en forme

Installation des bibliothèques de mise en forme

Laravel UI est une bibliothèque officielle qui propose des composants d'interface utilisateur sélectifs ou prédéfinis. Le package laravel/ui est fourni avec l'échafaudage de connexion et d'enregistrement pour les mises en page React, Vue, jQuery et Bootstrap.

Exécutez la commande pour installer Laravel/UI:

```
composer require laravel/ui
```

Installer Bootstrap dans Laravel :

```
php artisan ui bootstrap
```

QCM

TP

À la prochaine !