Module: Développer en back-end

Manipulation des vues avec Blade

Filière: Développement digital – option web full stack

Formatrice: Asmae YOUALA

ISTA AL ADARISSA

PLAN DU COURS

Introduction

1. Les structures de contrôle
2. Mise en forme des vues à l'aide des :
   - Modèles (Layout Blade)
   - Composants Blade

Introduction

- **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.**
- 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.

- Exemple :

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

Très utile par exemple si l'on veut afficher le contenu d'un article de blog édité par un wysiwyg se trouvant dans notre Base de Données.

Introduction

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 brut dans votre modèle :

```
@php
    $counter = 1;
@endphp
```

1. Les structures de contrôle

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#

@if, @elseif, @else

```
$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 <!-- donnera : -->
        <p>L'animal n'est ni un chat ni un chien.</p>
```

1. Les structures de contrôle

@switch

```
@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
```

1. Les structures de contrôle

@isset

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

<!-- donnera : -->
<p>Le produit existe</p>
```

@empty

```
$produit = ‘’;

        @empty($produit)
                <p>Le produit n'existe pas.</p>
        @endempty
<!-- donnera : -->
<p>Le produit n'existe pas.</p>
```

1. Les structures de contrôle

Pour réaliser une boucle : @for, @foreach, @forelse et @while#

```
  @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 )
        <!-- mon code exemple : -->
                <p>Lettre : {{ $letter }}</p>
        @endforeach
<!-- donnera : -->
<p>Lettre : a</p>
<p>Lettre : b</p>
<p>Lettre : c</p>
```

1. Les structures de contrôle

@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>
```

1. Les structures de contrôle

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

<!-- donnera : -->
<li>chien</li>
<li>chat</li>
<li>cheval</li>

<!-- Si le tableau $animals est vide -->
$animals = [];
<!-- Cela donnera : -->
<p>Aucun animal existant.</p>
```

1. Les structures de contrôle

Dans vos boucles vous avez également accès à une variable $**loop** qui permet de récupérer certaines informations concernant l'itération en cours dans la boucle.
Par exemple si vous voulez mettre une condition sur la première ou la dernière itération :

```
$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
```

Pour voir les diverses méthodes applicables à cette variable $loop je vous renvoie à la documentation officielle de Laravel

1. Les structures de contrôle

Les commentaires:

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

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.

2. Mise en forme des vues :

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

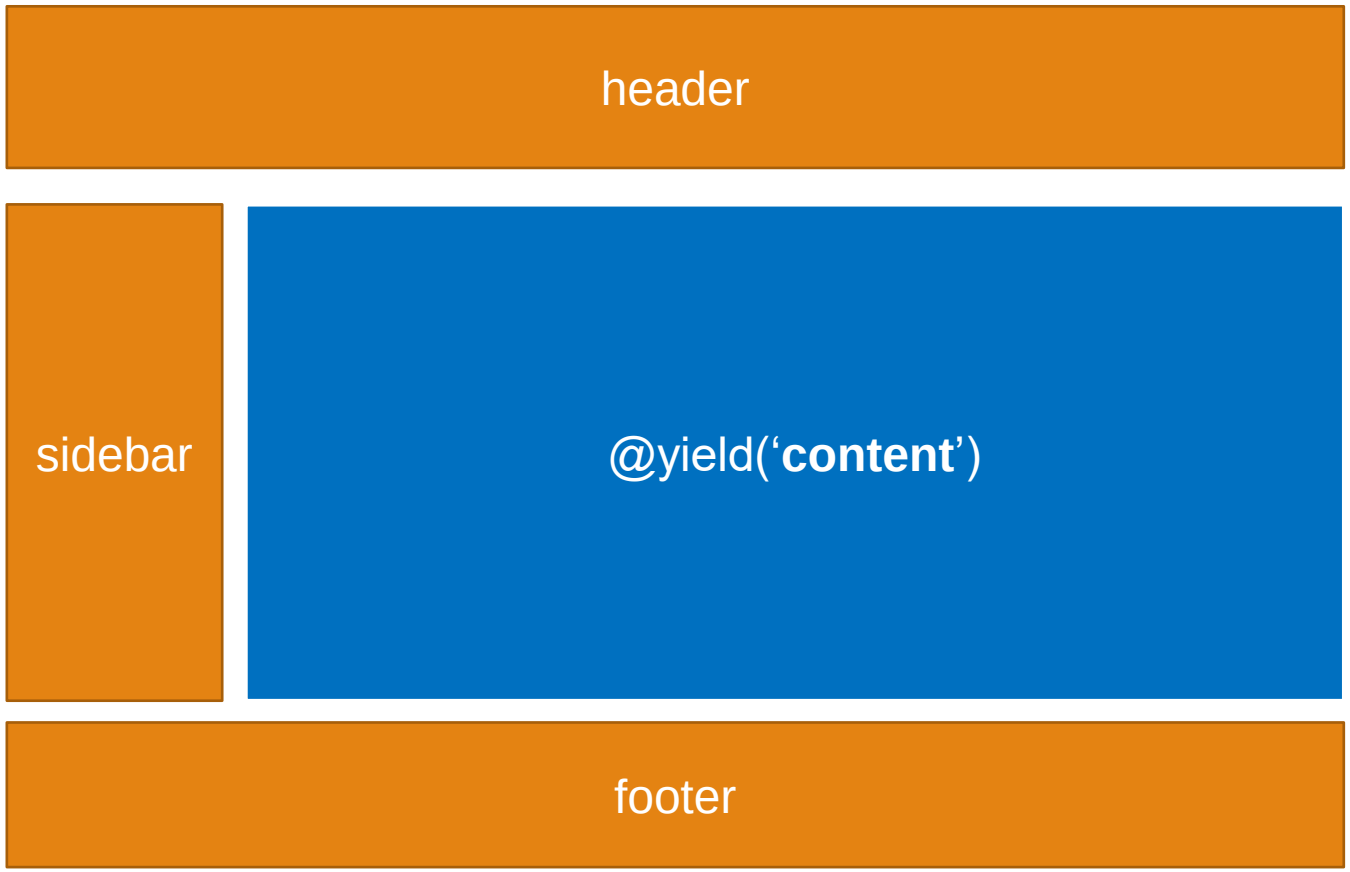
Il serait incroyablement fastidieux et difficile de maintenir notre application si nous devions répéter l'intégralité de la mise en page HTML dans chaque vue que nous créons.

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**;

2. Mise en forme des vues : Modèles (layout blade)

« 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 :



template.blade.php

```
@section('content')
    <h2>Contenu de la page
d'accueil </h2>
@endsection
```

accueil.blade.php

2. Mise en forme des vues : Modèles (layout blade)

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.

2. Mise en forme des vues : Modèles (layout blade)

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).

@parent désigne un espace qui sera remplacé par la section indiquée (sidebar) en provenance du modèle (master)

@endsection indique la fin d'une section.

2. Mise en forme des vues : Les composants blade

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***).

- Définir un composant :

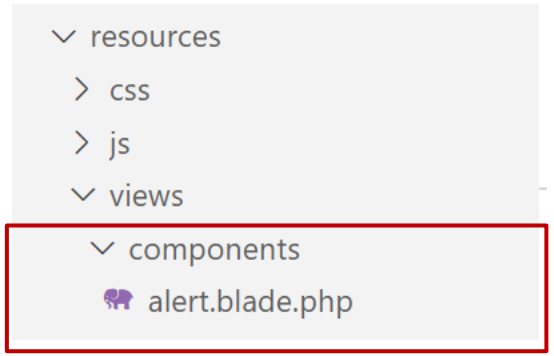
Pour créer un composant basé sur une classe, vous pouvez utiliser la commande make:component de Artisan

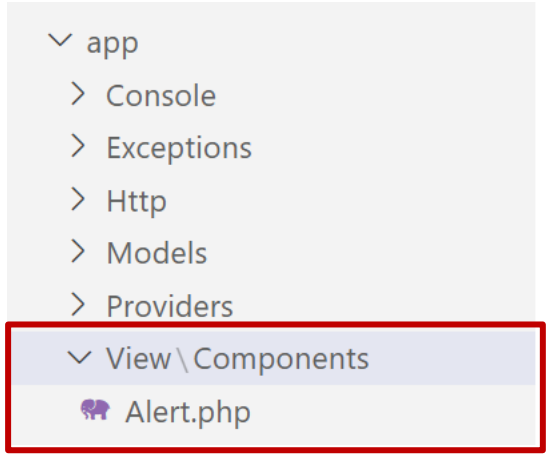
```
php artisan make:component nom-composant
```

Exemple: `php artisan make:component alert`

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*





2. Mise en forme des vues : Les composants blade

Le fichier **alert.blade.php** crée sous le répertoire views/components a l'allure suivante:

```
<div>
    <!-- Knowing is not enough; we must apply. Being willing is not enough; we must
do. - Leonardo da Vinci -->
</div>
```

On peut y ajouter le code html suivant afin de définir notre composant;

```
<div>
    <strong>Attention!</strong>
</div>
```

Utilisation du composant créé depuis une autre vue :

```
<x-alert/>
```

Il suffit de préfixer le nom du composant par « x- » et de l'appeler sous forme d'une balise.

2. Mise en forme des vues : Les composants blade

Transmission des données aux composants:

Vous pouvez transmettre des données aux composants Blade à l'aide d'attributs HTML.

- Des valeurs primitives peuvent être transmises au composant à l'aide de simples chaînes d'attributs HTML.
- Les expressions et variables PHP doivent être transmises au composant via des attributs qui utilisent le :caractère comme préfixe :

<x-alert type="error" :message="$message"/>

Exemple:

1. Modifions le fichier **Alert.php** placé sous le répertoire **app/View/Components**:

Nous souhaitons que le titre de l'alerte soit personnalisé par les vues appelantes à ce composant, c'est pour cette raison qu'on doit ajouter à la classe Alert l'attribut public **$titre** et modifier le constructeur pour pouvoir l'initialiser:

```php
class Alert extends Component
{
    public $titre;

    public function __construct($titre)
    {
        $this->titre=$titre;
    }
   public function render(): View|Closure|string
    {
        return view('components.alert');
    }
}
```

ou

```php
class Alert extends Component
{

  public function __construct(public string $titre)
    { }

  public function render(): View|Closure|string
    {
        return view('components.alert');
    }
}
```

2. Mise en forme des vues : Les composants blade

Transmission des données aux composants:

2. Modifions, par la suite, le fichier alert.blade.php pour prendre en considération le nouveau paramètre $titre :

```
<div>
    <i>{{$titre}}</i> <br>
  <strong>Attention!</strong>
</div>
```

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

```
<x-alert titre="Accès refusé"/>
```

Remarque:

S'il s'agit d'une transmission d'une variable ou d'une expression php, l'attribut HTML doit être précédé par (:)

```
<x-alert titre="Accès refusé" :message="$message"/>
```