⬅Revenir à la théorie.

Blade

1. Présentation

Pour faire nos vues, on utilise du PHP et du HTML. Cependant, des personnes trouvent quue PHP est lourd. De là est né des moteurs de templates pour palier à cela. Blade en fait partie.

Laravel possède un moteur de Template: Blade.
Il a une syntaxe très particulière qui permet de vous faire gagner du temps et de créer aussi des modèles de document.

2. Les substitutions

Par exemple une vue en php

```
<p>L'article n°<?php echo $numero ?> se trouve dans l'allée <?php echo $allee ?></p>
```

Ou pourrait utiliser la syntaxe simplifiée apparue dans PHP 5.4:

```
<p>L'article n°<?= $numero ?> se trouve dans l'allée <?= $allee ?></p>
```

Notre vue à la sauce Blade:

```
<p>L'article n°{{$numero}} se trouve dans l'allée {{$allee}}</p>
```

Comme vous le constatez, on n'a pas utilisé la directive <?php ?>. Mais directement en appelant nos variables reçues de la route: $numero et $allee.

Blade demande qu'on encadre les variables avec des doubles accolades ouvrantes et fermantes: {{$numero}} et {{$allee}}.

3. Les conditions

On utilise les directives @if, @else, @elseif et @endif.

Soit la route suivante:

```
Route::get('/article', function () {
    $numero = 23;
    $allee = '6a';
    return view('article', ['numero' => $numero, 'allee' => $allee]);
});
```

La vue 'article.blade.php':

```
<p>
    @if ($numero == 23)
        Le numéro 23 !? Excellent Article ! ❤
    @elseif ($numero == 24)
        Notre second meilleur Article {{$numero}} ! Good Choice ! 😎
    @else
        Ce n'est pas notre meilleur article ({{$numero}}) mais il est cool quand même. 😊
    @endif
</p>
<p>Vous pourrez le trouver dans l'allée {{$allee}}.</p>
```

4. Les boucles

Pour les exemples, nous utiliserons la route:

```
Route::get('/vampire', function()
{
    $armes = array('pieu','ail','crucifix','soleil');
    return view('attack', ['armes' => $armes]);
});
```

4.1 foreach

Notre view 'attack.blade.php' sans utiliser Blade:

```
<h1>Attaquons ces vampires ! </h1>
Pour attaquer ou éloigner un vampire tu auras besoin d'un des éléments suivants:<br/>
<?php
foreach ($armes as $arme) {
    echo "- $arme<br/>";
}
?>
</ul>
```

Notre view 'attack.blade.php' avec Blade:

```
<h1>Attaquons ces vampires ! </h1>
Pour attaquer ou éloigner un vampire tu auras besoin d'un des éléments suivants:<br/>
@foreach ($armes as $arme)
    - {{$arme}}<br/>
@endforeach
```

On voit la facilité de lecture avec Blade. 😀

4.2 for

```
<h1>Attaquons ces vampires ! </h1>
Pour attaquer ou éloigner un vampire tu auras besoin d'un des éléments suivants:<br/>
@for ($i=0;$i<count($armes);$i++)
    - {{$armes[$i] }}<br/>
@endfor
```

4.3 forelse

```
<h1>Attaquons ces vampires ! </h1>
<p>Pour attaquer ou éloigner un vampire tu auras besoin:</p>
@forelse ($armes as $arme)
    - {{$arme}}<br />
@empty
    Mince, je ne m'en rappelle plus ! Bonne chance ! 😅
@endforelse
```

4.4 while

```
@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

4.5 La variable de boucle $loop

Lors de boucles, on peut utiliser cette variable $loop et les propriétés suivantes:

Propriété	Description
$loop->index	L'index de l'itération courante (commence à 0)
$loop->iteration	L'itération courante (commence à 1).
$loop->remaining	Nombre d'itérations restantes.
$loop->count	Le nombre total d'éléments du tableau qui est en train d'être parcouru.
$loop->first	S'il s'agit de la première itération dans la boucle.
$loop->last	S'il s'agit de la dernière itération dans la boucle.
$loop->even	S'il s'agit d'une itération paire dans la boucle.
$loop->odd	S'il s'agit d'une itération impaire dans la boucle.
$loop->depth	The nesting level of the current loop.
$loop->parent	Dans une boucle imbriquée, la variable $loop de boucle du parent.

Exemple 1

```
@foreach ($users as $user)
    @if ($loop->first)
        Ceci est la première itération
    @endif

    @if ($loop->last)
        Ceci est la dernière itération
    @endif

    <p>C'est l'utilisateur ayant l'id {{ $user->id }}</p>
@endforeach
```

Exemple 2

```
@foreach ($users as $user)
    @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            Ceci est la première itération de la boucle parente.
        @endif
    @endforeach
@endforeach
```

5. Directive @PHP

Parfois il peut être utile d'embarquer du code PHP dans vos vues. Vous pouvez utiliser la directive Blade @php et @endphp
Exemple

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

6. Les modèles

Alors on voit que l'on peut simplifier l'écriture de certaines pages webs grâve à Blade.

Mais on peut aussi faire des modèles de documents qui pourront être utilisés par nos vues.

Nos vues fourniront les données nécessaires pour remplir le modèle.

Nous allons voir deux manières différentes de faire des modèles: Avec héritage d'un modèle ou via Component (composant)

7. Modèles avec héritage

Créons notre page modèle nommée modele.blade.php

```
<html>
    <body>
        <h1>@yield('titre')</h1>
        @section('contenu')
            <p>Contenu principal du modèle.</p>
        @show
    </body>
</html>
```

Et une page page.blade.php

```
@extends('modele')

@section('titre')
    Mon titre
@stop

@section('contenu')
    @parent
    <p>Contenu de la page utilisant le modèle</p>
@stop
```

Ca donnera comme résultat en HTML:

```
<h1>Mon titre</h1>
<p>Contenu principal du modèle.</p>
<p>Contenu de la page utilisant le modèle</p>
```

Nous allons voir maintenant un Exemple complet

- La route:

```
Route::get('/page{numero}', function ($numero) {
    return view("page$numero")->with('numero',$numero);
});
```

- modele avec JS et CSS Bootstrap: montemplate.balde.php

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/boo
    <title>@yield('titre')</title>
</head>

<body>
    <div class='container'>
        <h1>@yield('titre1')</h1>
        @section('contenu')
        @show
    </div>
    <!-- jQuery and Bootstrap Bundle (includes Popper) -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/js/bootstrap.bundle.min
</body>
</html>
```

- Page.blade.php

```
@extends('montemplate')

@section('titre')
    Super page {{$numero}}!
@stop

@section('h1')
    Je suis la page{{$numero}}
@stop

@section('contenu')
    @parent
    <p>Je suis le contenu de la page {{$numero}}. </p>
    @section('tutu')

    @show
@stop
```

- Page1.blade.php

```
@extends('page')
@section('tutu')
    <h2>Lorem Ipsum</h2>
    <p>
        Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem I
    </p>
@stop
```

- Page2.blade.php

```
@extends('page')
@section('tutu')
    <h2>Les inondations</h2>
    <p>
        Les inondations qui touchent la Belgique est une terrible catastrophe.
    </p>
@stop
```

7.1 Différences entre Yield et Section

On peut légitimement se poser la question mais quand est-ce que j'utilise @yield et @section dans mon modèle ? En effet tous les deux servent à insérer quelque chose.

Pour expliquer la différence, nous partirons du modèle nommé modele.blade.php:

```
    <h1> @yield("titre1") </h1>
    @section("contenu")
    Contenu venant du modèle.
    @show
```

7.2 Yield

Utilisez @yield dans le modèle si vous souhaitez écraser complètement les données enfants sur la mise en page principale.

Exemple de remplissage dans page.blade.php

```
@extends("modele")
@section("titre1")
    Ce texte est en h1
@stop
```

Résultat:

```
Ce titre est en h1
Contenu venant du modèle.
```

7.3 Section

Utilisez @section dans le modèle si vous souhaitez utiliser les données du modèle et les données de la page enfant ensemble avec @parent.

Exemple de remplissage dans page.blade.php

```
@extends("modele")
@section("contenu")
    @parent
    Contenu venant de la page enfant.
@show
```

Résultat:

```
Ce titre est en h1
Contenu venant du modèle.
Contenu venant de la page enfant.
```

8. Modèles avec Component

Alors ici on va créer notre modèle à partir d'un fichier nommé par exemple: modele.blade.php qui se trouvera dans le répertoire app/ressources/views/components

On va créer ce fichier modele.blade.php via php artisan avec la commande

```
php artisan make:component modele
```

Cette commande créera aussi pour vous un autre fichier app\View\Components\modele.php

Notre modele pourrait par exemple contenir:

```
<!DOCTYPE html>
<html>
    <head>
        <title>{{ $title ?? 'Todo Manager' }}</title>
    </head>
    <body>
        <h1>{{ $titreH1 }}</h1>
        <hr/>
        {{ $slot }}
    </body>
</html>
```

Utilisons maintenant notre composant dans la vue suivante:

```
<x-modele>
    <x-slot name="title">
        {{ $title }}
    </x-slot>

    <x-slot name="titreH1">
        Welcome !
    </x-slot>

    Je suis ce qui va apparaître dans $slot car je ne fais pas partie d'un 'x-slot'.
</x-modele>
```

On constate que nous avons créé une balise <x-modele> et que cette balise comporte le nom de notre composant: modele.

Si on veut garnir $title ou $titreH1 de notre modele, nous utiliserons la balise <x-slot> avec l'attribut name où l'on donne comme valeur le nom de la variable à remplacer: title ou titreH1.

La variable $slot sera remplie par tout ce qui sera affiché en dehors des balises <x-slot>.

Que va faire ce code ?

```
{{ $title ?? 'Todo Manager' }}
```

Si $title n'est pas nul alors on affiche la valeur de $title sinon on affiche la valeur 'Todo Manager'

⬅Revenir à la théorie.
