Cours Laravel 10 – les vues

laravel.sillo.org/cours-laravel-10-les-vues/

2 mars 2023

Je vous ai déjà parlé des vues dans ce cours et de Blade et on a eu plusieurs exemples de code. Dans ce chapitre, je vais faire un peu le point et montrer des possibilités intéressantes qui n'ont pas encore été évoquées.

Pour les besoins d'illustrer ce chapitre, on va repartir de l'application des films que j'ai créée pour ce cours et dont vous pouvez récupérer la version définitive ici. Il suffit de décompresser dans un dossier et ensuite de lancer cette commande :

```
composer install
```

Il faut aussi créer une base de données et bien renseigner dans le fichier **.env** :

```
DB_DATABASE=laravel10
DB_USERNAME=root
DB_PASSWORD=
```

Ensuite, lancer les migrations et la population :

```
php artisan migrate --seed
```

Ensuite l'application doit fonctionner avec l'url **…/films**.

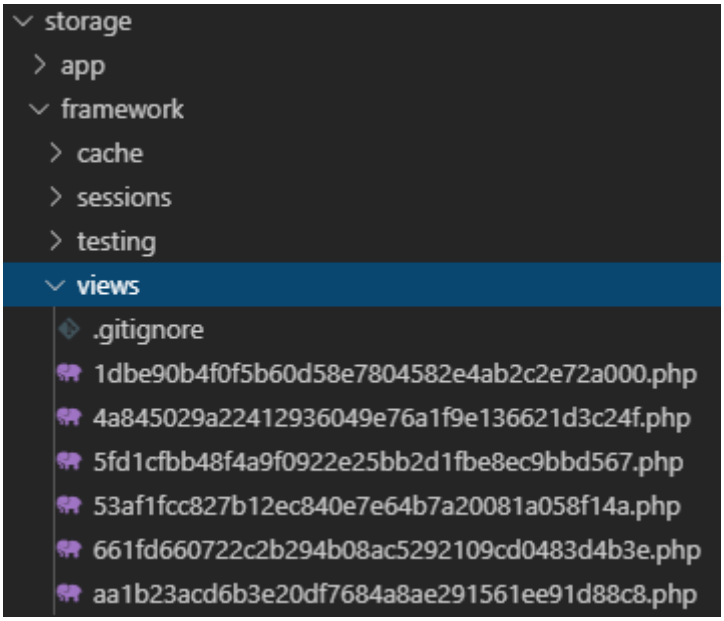
Le cache

Les vues générées par Blade sont en PHP et sont en cache jusqu'à ce qu'elles soient modifiées, ce qui assure de bonnes performances. Le cache se situe ici :

Si vous avez besoin de nettoyer ce cache, ce n'est pas la peine d'aller supprimer ces fichiers dans le dossier, il y a une commande Artisan pour ça :

```
php artisan view:clear
```

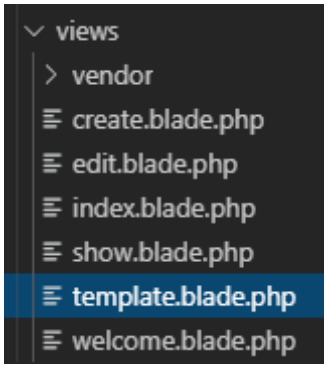
On en a besoin parfois en cours de développement lorsqu'une vue ns s'est pas régénérée correctement.



L'héritage

On a vu qu'une vue peut en étendre une autre, c'est un héritage. Ainsi pour les vues de l'application, on a un template de base :

Ce template comporte la structure globale des pages et est déclaré comme parent par les autres vues :




```
@extends('template')
```

Dans le template, on prévoit un emplacement (**@yield**) pour que les vues enfants puissent placer leur code :

```
<main class="section">
    <div class="container">
        @yield('content')
    </div>
</main>
```

Ainsi dans la vue **index.blade.php**, on utilise cet emplacement :

```
@section('content')
   // Code de la vue
@endsection
```

*On peut également déclarer une section dans le template avec **@section** lorsqu'on a du code dans le template qu'on veut soit utiliser, soit surcharger.*

L'inclusion

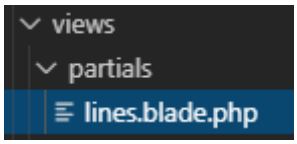
On peut faire beaucoup de choses avec l'héritage, mais il est souvent utile de pouvoir inclure une vue dans une autre, classiquement, on parle de vue partielle (partial).

Si on regarde de près la vue index, on a un tableau avec comme corps des lignes avec les films :

Aperiam minus.	Voir	Modifier	Supprimer
Blanditiis asperiores dolor.	Voir	Modifier	Supprimer
Consectetur aliquam.	Voir	Modifier	Supprimer

On peut mettre le code qui génère ces lignes dans une vue partielle :

En récupérant le code de la vue **index** :



```
@foreach($films as $film)
    <tr @if($film->deleted_at) class="has-background-grey-lighter" @endif>
        <td><strong>{{ $film->title }}</strong></td>
            <td>
                @if($film->deleted_at)
                    <form action="{{ route('films.restore', $film->id) }}"
method="post">
                        @csrf
                        @method('PUT')
                        <button class="button is-primary"
type="submit">Restaurer</button>
                    </form>
                @else
                    <a class="button is-primary" href="{{ route('films.show',
$film->id) }}">Voir</a>
                @endif
            </td>
            <td>
                @if($film->deleted_at)
                @else
                    <a class="button is-warning" href="{{ route('films.edit',
$film->id) }}">Modifier</a>
                @endif
            </td>
        <td>
            <form action="{{ route($film->deleted_at? 'films.force.destroy' :
'films.destroy', $film->id) }}" method="post">
                @csrf
                @method('DELETE')
                <button class="button is-danger" type="submit">Supprimer</button>
            </form>
        </td>
    </tr>
@endforeach
```

Et dans la vue **index**, on se contente d'inclure cette vue :

```
<tbody>
    @include('partials.lines')
</tbody>
```

*On dispose aussi des directives **@includeIf**, **@includeWhen et @includeFirst**.*

Une autre possibilité consiste à gérer l'itération à partir de la vue principale :

```
@each('partials.lines', $films, 'film')
```

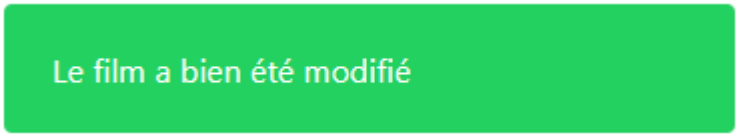
Et évidemment on ne conserve dans la vue partielle que le code inclus dans la boucle :

```
<tr @if($film->deleted_at) class="has-background-grey-lighter" @endif>
    ...
</tr>
```

Les composants

Comme alternative à l'inclusion, on dispose des composants. Voyons de quoi il s'agit.

Dans la vue **index**, on a une partie du code consacrée à la notification lorsqu'une action a bien abouti comme la modification d'un film :



Voici le code correspondant :

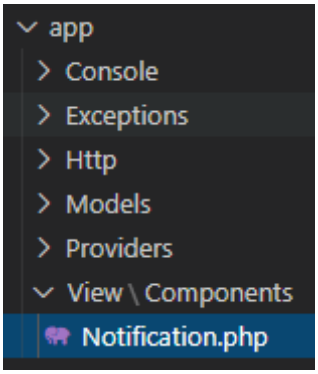
```
@if(session()->has('info'))
    <div class="notification is-success">
        {{ session('info') }}
    </div>
@endif
```

On crée le composant avec Artisan :

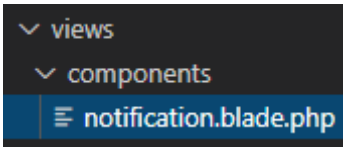
```
php artisan make:component Notification
```

On a création d'une classe :

Et d'une vue :



Pour la classe, on prévoit ce code :




```
<?php

namespace App\View\Components;

use Closure;
use Illuminate\Contracts\View\View;
use Illuminate\View\Component;

class Notification extends Component
{
    public $text;

    public function __construct($text)
    {
        $this->text = $text;
    }

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

Et pour la vue :

```
<div class="notification is-success">
    {{ $text }}
</div>
```

La variable **$text** va recevoir ce qu'on va injecter dans le composant. Voici le code modifié dans la vue **index** :

```
@if(session()->has('info'))
    <x-notification :text="session('info')"/>
@endif
```

*On peut aussi prévoir une variable **{{ $slot }}** dans le composant et injecter directement du contenu HTML.*

Les structures de contrôle

@if

La directive **@if** permet d'introduire une condition. Par exemple, dans la vue partielle **lines** qu'on a créée ci-dessus, on trouve ce code :

```
@if($film->deleted_at) class="has-background-grey-lighter" @endif
```

Ce qui permet de griser le fond de la ligne pour un film dans la corbeille :

*Les directives **@else** et **@unless** permettent de compléter la logique.*

Remarquez au passage la directive **@php** qui permet d'insérer du code PHP dans la vue.

@isset et @empty

La directive **@isset** est l'équivalent de la fonction PHP **isset()**.

La directive **@empty** est l'équivalent de la fonction PHP **empty()**.

@auth et @guest

Il est fréquent d'avoir à tester si un utilisateur est connecté et on peut écrire :

```
@if (Auth::check())
```

Mais il est plus élégant d'utiliser **@auth**. De la même manière, on dispose de **@guest**.

*Il existe aussi la directive **@switch** équivalente de la directive PHP **switch**.*

Condition personnalisée

On peut aussi se construire une condition personnalisée. On pourrait prévoir ce code dans **AppServiceProvider**:

```
public function boot(): void
{
    ...

    Blade::if('admin', function () {
        return auth()->user()->role === 'admin';
    });

    Blade::if('redac', function () {
        return auth()->user()->role === 'redac';
    });

    Blade::if('request', function ($url) {
        return request()->is($url);
    });
}
```

On crée ainsi 3 directives :

- **admin** : vérifie qu'un utilisateur est administrateur
- **redac** : vérifie qu'un utilisateur est rédacteur
- **request** : vérifie que la requête correspond à une certaine url

Du coup, on peut écrire ce genre de code :

```
@admin
  <li>
    <a href="{{ url('admin') }}">@lang('Administration')</a>
  </li>
@endadmin
@redac
  <li>
    <a href="{{ url('admin/posts') }}">@lang('Administration')</a>
  </li>
@endredac
```

Et dans une vue :

```
@request('password/reset')
  <li class="current">
    <a href="{{ request()->url() }}">@lang('Password')</a>
  </li>
@endrequest
@request('password/reset/*')
  <li class="current">
    <a href="{{ request()->url() }}">@lang('Password')</a>
  </li>
@endrequest
```

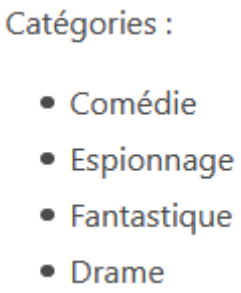
Les boucles

Pour les boucles, on dispose de **@for**, **@foreach**, **@each**, **@forelse** et **@while**. La logique est la même que les directives équivalentes de PHP. Voyons un exemple dans la vue **show** :

```
@foreach($film->categories as $category)
    <li>{{ $category->name }}</li>
@endforeach
```

C'est la partie du code qui remplit la liste des catégories quand on affiche un film :

Avec la directive **@foreach** on dispose de la variable interne **$loop** qui offre des informations sur l'index en cours. Voici ce qui est disponible :



Propriété	Description
$loop->index	L'index en cours (commence à 0)
$loop->iteration	L'itération courante (commence à 1)
$loop->remaining	Les itérations restantes
$loop->count	Le nombre total d'itérations
$loop->first	Indique si c'est le premier élément
$loop->last	Indique si c'est le dernier élément
$loop->depth	Le niveau d'imbrication actuel
$loop->parent	La variable d'itération parente si imbrication

Les composeurs de vues (view composers)

Les composeurs de vues sont des fonctions de rappel ou des classes qui sont appelées lorsqu'une vue est générée, ce qui permet de localiser du code. Voyons un exemple dans l'application.

Regardez ce code dans **AppServiceProvider** (on aurait pu créer un provider spécifique) :

```
public function boot(): void
{
    View::composer(['index', 'create', 'edit'], function ($view) {
        $view->with('categories', Category::all());
        $view->with('actors', Actor::all());
    });
}
```

On dit là que lorsqu'on affiche une des vues **index**, **create** ou **edit** il faut envoyer les variables **$categories** et **$actors**.

En résumé

Avec Blade on peut :

- utiliser l'héritage de vue,
- inclure une vue,
- utiliser un composant,
- utiliser de nombreuses directives pour organiser le code,
- utiliser des boucles,
- …

D'autre part, les composeurs de vues permettent de localiser du code facilement.