Chapitre 3

L'interface utilisateur (views)

Dans Laravel, les "views" (vues en français) sont des fichiers qui contiennent du HTML, du CSS et du JavaScript qui décrivent l'apparence et le contenu d'une page web. Les vues sont utilisées pour séparer la présentation de la logique métier de l'application, ce qui facilite la maintenance et le développement de l'application. Laravel utilise le moteur de templates Blade pour gérer les vues. Blade permet d'insérer des variables et des expressions dans les vues, d'étendre des mises en page (layouts) pour éviter la duplication de code et de créer des sections qui peuvent être remplies avec du contenu spécifique à chaque page.

Pour afficher une vue dans une application Laravel, il faut généralement créer une méthode dans un contrôleur qui retourne la vue correspondante, puis définir une route qui associe cette méthode à une URL. Ainsi, lorsque l'utilisateur accède à l'URL correspondante, Laravel exécute le code du contrôleur pour récupérer les données nécessaires à la vue, puis affiche la vue dans le navigateur de l'utilisateur.

3.1 Création et codage d'une vue

Pour créer une nouvelle vue, il suffit de créer un nouveau fichier dans le répertoire `/resources/views` avec l'extension `.blade.php`. Il est possible d'écrire dans ces fichiers du code HTML, CSS et JS classique, mais en plus, ils acceptent du code PHP ainsi que ce qu'on appelle les directives blade (blade directives) qui sont pratiquement des raccourcies

vers les instructions PHP les plus utilisées. Les directives Blade sont des instructions spéciales qui sont mises en évidence par la syntaxe `@` suivie d'un mot-clé, suivi éventuellement d'un ou plusieurs arguments.

Les directives blade les plus utilisées incluent `@if`, `@else`, `@elseif`, `@switch`, `@case`, `@default`, `@foreach`, `@while`, `@for`, `@php`, `@include`, `@extends`, `@section`, `@yield`, et bien d'autres encore.

Après avoir créé la vue, il est possible de la retourner de n'importe quel contrôleur en exécutant la commande `return view('<nom-vue>')` ou bien, en utilisant l'instruction de routage `Route:view` comme il est montré dans la section 2.1.6.

3.2 Des vues avec des paramètres

Lorsqu'un contrôleur retourne une vue (avec la fonction 'view'), il est possible de passer un tableau contenant les données dont cette vue a besoin comme le montre l'exemple ci-dessous :

```
...
public function renderProfile() {
    $data = [
        'nom' => 'Chohra',
        'prenom' => 'Chemseddine',
        'email' => 'chemseddine.chohra@gmail.com'
    ];
    return view('profile', $data);
}
...
```

ces données peuvent être utilisée dans la vue `profile.blade.php` comme des variables PHP portant le même nom que les indice dans le tableau de données, c'est à dire que la vue va avoir les deux variables `$nom`, `$prenom` et `email`.

```
...
<h1>{{$nom}} {{$prenom}}</h1>
<h2>{{$email}}</h2>
...
```

Les deux accolades `{{}}` son une directive blade équivalente à l'insruction `echo` dans PHP. Le code précédent donnera le même résultat que :

```
...
<h1><?php echo $nom . ' ' . $prenom; ?></h1>
<h2><?php echo $email; ?></h2>
...
```

Il est également possible d'utiliser la méthode `with` pour passer des paramètres au lieu de passer un tableau de variable. Le code du contrôleur précédent peut être réécrit comme suit :

```
...
public function renderProfile() {
    $data = [
        'nom' => 'Chohra',
        'prenom' => 'Chemseddine',
        'email' => 'chemseddine.chohra@gmail.com'
    ];
 return view('profile')->with('nom', 'Chohra')->with('prenom', 'Chemseddine')
                       ->with('email', 'chemseddine.chohra@gmail.com');
}
...
```

3.3 Les directives Blade

Nous avons déjà expliqué que les directives "Blade" sont des instructions spéciales qu'on utilise pour manipuler le contenu et la structure de la page. Certaines directives correspondent directement à des instructions PHP comme :

— `@include` : inclut une vue partielle dans la vue en cours.

— `@if`, `@else`, `@switch` et `@case` : pour les structures conditionnelles.

— `foreach`, `for`, `while` : pour les boucles.

D'autres directives sont pratiques pour créer des templates, nous citons :

— `@yield` : utilisée pour définir un emplacement où du contenu dynamique peut être inséré. Elle est souvent utilisée pour définir un emplacement pour le contenu principal d'une page. Par exemple, vous pouvez utiliser la commande `@yield('content')` pour définir un emplacement pour le contenu principal de votre page.

— `@extend` : utilisée pour étendre un layout Blade existant et pour y ajouter des sections définies avec `@yield`. Elle est souvent utilisée pour réutiliser des layouts communs entre plusieurs pages. Par exemple, vous pouvez utiliser la commande `@extend('layouts.app')` pour étendre un layout nommé `app.blade.php` placé dans le répertoire `layouts`.

— `@section` : utilisée pour définir le contenu qui sera inséré dans un emplacement `@yield`. Elle est souvent utilisée pour définir le contenu principal d'une page ou pour définir des sections de contenu réutilisables. Par exemple, vous pouvez utiliser la commande `@section('content')` pour définir le contenu principal de votre page qui étend un layout qui utilisait la commande `@section('content')`.

3.4 Les composants (components)

les composants sont des éléments réutilisables qui permettent de créer des parties de vues. Les composants sont généralement utilisés pour les éléments d'interface utilisateur tels que

les formulaires, les menus, les boutons, les alertes ... etc.

Pour créer un composant dans Laravel, utiliser la commande suivante :

```
php artisan make:component Alert
```

Cette commande va créer un fichier `alert.blade.php` dans le répertoire `/resources/views/components`. La même commande va également créer un fichier `Alert.php` dans le répertoire `app/View/component`.

Initialement, le fichier `alert.blade.php` contient un seul `div` alors que le fichier `Alert.php` contient le code suivant :

```
<?php
namespace App\View\Components;
use Illuminate\View\Component;
class Alert extends Component
{
    public function __construct() {
    }
    public function render() {
        return view('components.alert');
    }
}
```

Le constructeur de la classe s'occupe de l'initialisation des paramètres du composant, alors que la méthode `render` retourne la vue qui correspondante.

3.4.1 Afficher les composants

Pour afficher un composant dans une vue, il suffit d'utiliser une balise avec le nom du composant ayant le préfix `x-`. Par exemple, pour afficher le composant `alert` défini dans l'exemple précédent, nous utilisons la balise :

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

```
...
```

3.4.2 Passer des paramètres à des composants

Il est possible de passer des paramètres aux composants en utilisant une syntaxe similaire aux attributs HTML. La valeurs littérales peuvent être passées directemet, par contre les variables (ou expressions) PHP doivent passer à travers des attributs ayant le préfixe `:`. L'exemple suivant montre le passage des paramètres dans les deux cas.

```
...
@php($message = "Le message a bien été envoyé")
<x-alert type="success" :message="$message"/>
...
```

Tous les paramètres passés au composant doivent être définis dans les paramètres du constructeur dans le fichier contenant la classe correspondante. Dans cet exemple, nous allons définir une alerte (en utilisant les classes prédifinies dans le framework Bootstrap [1]). Le fichier `alert.blade.php` doit contenir le code suivant :

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

Les paramètre `$message` et `$type` sont passés à travers le constructeur de la classe `Alert` dans le fichier `Alert.php`

```
<?php
    namespace App\View\Components;
    use Illuminate\View\Component;
    class Alert extends Component {
        public $type;
        public $message;
        public function __construct($type, $message) {
```

1. https://getbootstrap.com/

```
        $this->type = $type;
        $this->message = $message;
    }
    public function render() {
        return view('components.alert');
    }
}
```

Les paramètres `$type` et `$message` dans ce cas sont passés automatiquement par la fonction `render` à la vue `alert.blade.php`.

3.4.3 Affichage conditionnel

Il est parfois pratique de conditionner l'affichage d'un composant pour éviter d'inclure plusieurs tests dans la vue qui pourrait lui faire appel. Dans le cas de notre exemple, il est possible de conditionner l'affichage de l'alerte par la présence d'un message. Dans ce cas, nous allons déclarer la méthode `shouldRender` dans la classe `Alert`. Le composant est affiché seulement lorsque cette méthode retourne `true`.

```
<?php
...
class Alert extends Component {
    ...
    public function shouldRender() {
        return strlen($this->message) > 0;
    }
    ...
}
```

3.4.4 Composants anonymes

Les composants anonymes fournissent un mécanisme de gestion d'un composant via un seul fichier. Cependant, les composants anonymes utilisent un seul fichier de vue et n'ont pas de classe associée. Pour définir un composant anonyme, vous n'avez besoin que de placer un modèle Blade dans votre répertoire resources/views/components. Par exemple, en supposant que vous ayez défini un composant à `resources/views/components/alert.blade.php`, vous pouvez simplement le rendre comme ceci :

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

Il est aussi possible de créer un composant anonyme à partir du CLI (artisan) en ajoutant l'option `--view` à la commande permettant de création de composant. La commande devient :

```
php artisan make:component alert --view
```

Finalement, vu que les composants anonymes n'ont pas de classe associée, leurs paramètres ne peuvent pas être passés à travers la directive Blade `@props` comme le montre l'exemple suivant :

```
@props(['type', 'message'])
<div class="alert alert-{{$type}}">
    {{$message}}
</div>
```