

# Création des template Blade:

## 1.Introduction :

Le **Blade** est un puissant moteur de templating dans un framework Laravel. Le blade permet d'utiliser le moteur de templating facilement, et il rend l'écriture de la syntaxe très simple. Le moteur de templating blade fournit sa propre structure comme les instructions conditionnelles et les boucles. Pour créer un modèle de blade, il suffit de créer un fichier de vue et de l'enregistrer avec une extension **.blade.php** au lieu de **.php**. Les modèles de blade sont stockés dans le répertoire **/resources/view**. Le principal avantage de l'utilisation du modèle de blade est que nous pouvons créer le modèle principal, qui peut être étendu par d'autres fichiers.

## 2.Affichage des données :

Si vous souhaitez imprimer la valeur d'une variable, il vous suffit de la placer entre des accolades.

### Syntaxe

```
{{ $variable }} ;
```

Dans le modèle de blade, nous n'avons pas besoin d'écrire le code php  
<?php echo \$variable ; ?>

## 3.Opérateur ternaire :

Dans le modèle de blade, la syntaxe de l'opérateur ternaire peut s'écrire comme suit :

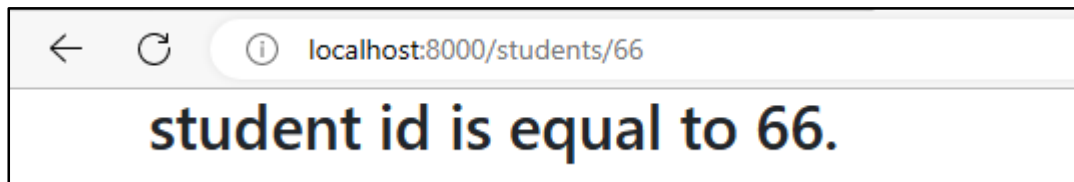
```
{{ isset($mytitle) ? $mytitle : "empty variable" }}
```

## 4. Instructions de contrôle

Le moteur de templating Blade fournit également les instructions de contrôle dans Laravel ainsi que des raccourcis pour les instructions de contrôle.

```
@if ($student->id==66)
    student id is equal to 66.
@else
    student id is not equal to 66.
@endif
```

//output



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

```
@isset($mytitle)
is defined and is not null...
@endisset

@empty($mytitlep)
$mytitlep is empty
@endempty
```

## 5. Boucles Blade

Le moteur de modélisation blade fournit des boucles telles que les directives **@for**, **@endfor**, **@foreach**, **@endforeach**, **@while** et **@endwhile**.

test.blade.php

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while(($i)<5)
    javapoint
    {{ $i++ }}
@endwhile
```

//output



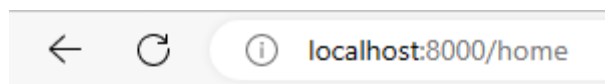
Lorsque vous utilisez des boucles, vous pouvez également sauter l'itération en cours ou terminer la boucle à l'aide des directives `@continue` et `@break` :

```
@foreach ($users as $user)
    @if ($user->id == 1)
        @continue
    @endif

    <li>{{ $user->name }}</li>

    @if ($user->id == 5)
        @break
    @endif
@endforeach
```

//output



## Home page

- 2
- 3
- 4
- 5

### La variable Loop

Lors de l'itération d'une boucle **foreach**, une variable **\$loop** sera disponible à l'intérieur de votre boucle. Cette variable permet d'accéder à certaines informations utiles, comme l'index actuel de la boucle et le fait qu'il s'agisse de la première ou de la dernière itération de la boucle :

```
@foreach ($users as $user)
    @if ($loop->first)
        This is the first iteration.
    @endif
```

```
@if ($loop->last)
    This is the last iteration.
@endif

<p>This is user {{ $user->id }}</p>
@endforeach
```

## 6.Classes conditionnelles

La directive `@class` compile de manière conditionnelle une chaîne de classes CSS. La directive accepte un tableau de classes où la clé du tableau contient la ou les classes que vous souhaitez ajouter, tandis que la valeur est une expression booléenne. Si l'élément du tableau a une clé numérique, il sera toujours inclus dans la liste des classes rendues :

Si nous voulons montrer l'utilisateur actif en vert et l'utilisateur inactif en rouge, nous pouvons utiliser la directive `@if @endif`.

```
@php
$active = true;
@endphp
@if ($active)
<span class="p-2 text-success">user</span>
@else
<span class="p-2 text-danger">user</span>
@endif
```

Afficher l'utilisateur actif en vert et l'utilisateur inactif en rouge  
Directive blade `@class`.

```
<span
@class([
    'p-2',
    'p-2 text-success' => $active,
    'p-2 text-danger' => !$active,
])>user
</span>
```

## 7. PHP brut

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

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

Si vous n'avez besoin d'écrire qu'une seule instruction PHP, vous pouvez inclure l'instruction dans la directive **@php** :

```
@php($counter = 1)
```

## 8. 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

## Activité

- **Étape 1:** Créez un nouveau dossier «layouts» dans le répertoire `/resources/views/`.
- **Étape 2:** Créez un nouveau fichier «**master.blade.php**» dans le répertoire `/resources/views/layouts/`.
- **Étape 3:** Copiez le code suivant dans le fichier “**master.blade.php**” que nous avons créé.

```
<html>
<head>
<title>App Name - @yield('title')</title>
</head>
<body>
@section('sidebar')
This is the master sidebar.
@show
<div class="container">
@yield('content')
</div>
</body>
</html>
```

- Ici, dans le modèle principal ci-dessus –
  - `@yield ('title')` est utilisé pour afficher la valeur du titre
  - `@section ('sidebar')` est utilisée pour définir une section nommée **sidebar**
  - `@show` est utilisé pour afficher le contenu d’une section
  - `@yield ('content')` est utilisé pour afficher le contenu du contenu
- **Extension de la mise en page principale**
  - Nous allons maintenant vous montrer comment étendre la mise en page principale que nous venons de créer.
  - **Étape 1:** Créez un nouveau fichier de vue **page.blade.php** dans `/resources/views/`
  - **Étape 2:** Copiez le code suivant dans le fichier **page.blade.php**

```
<!-- path: resources/views/page.blade.php -->
@extends('layouts.master')
@section('title', 'Page Title')
@section('sidebar')
@parent
<p>This is appended to the master sidebar.</p>
@endsection
@section('content')
<p>This is my body content.</p>
```

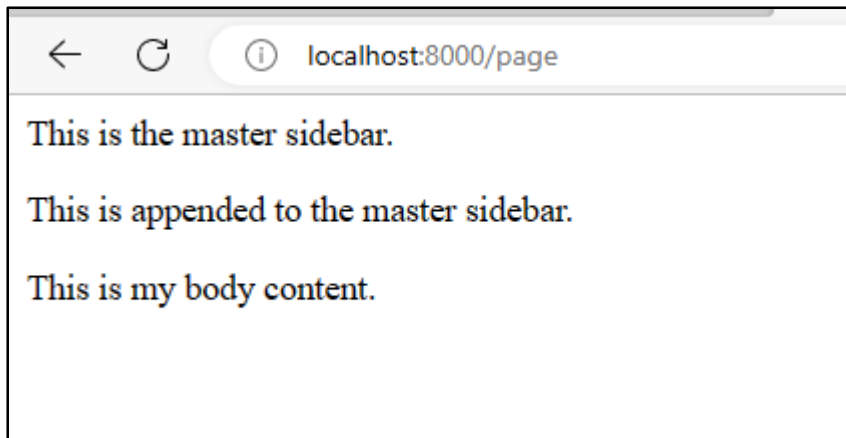
## @endsection

- Ici, dans la page ci-dessus
  - **@extends** ('layouts.master') étend la mise en page principale
  - **@section** ('title', 'Page Title') définit la valeur de la section de titre.
  - **@section** ('sidebar') définit une section de barre latérale dans la page enfant de la disposition principale
  - **@parent** affiche le contenu de la section de barre latérale, définie dans la disposition principale.
  - Ceci est ajouté à la barre latérale principale.
- **@endsection** termine la section de la barre latérale
- **@section** ('content') définit la section de contenu
- **@endsection** termine la section de contenu

- **Étape 3:** Ouvrez `routes/web.php` et configurez l'itinéraire comme ci-dessous:

```
Route::get('page', function(){  
    return view('page');  
});
```

- **Étape 4:** Ouvrez maintenant l'URL suivante dans le navigateur pour voir la sortie.
- <http://localhost:8000/page>

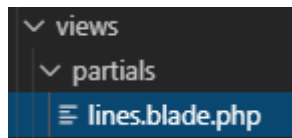


## 9. L'inclusion

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).

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





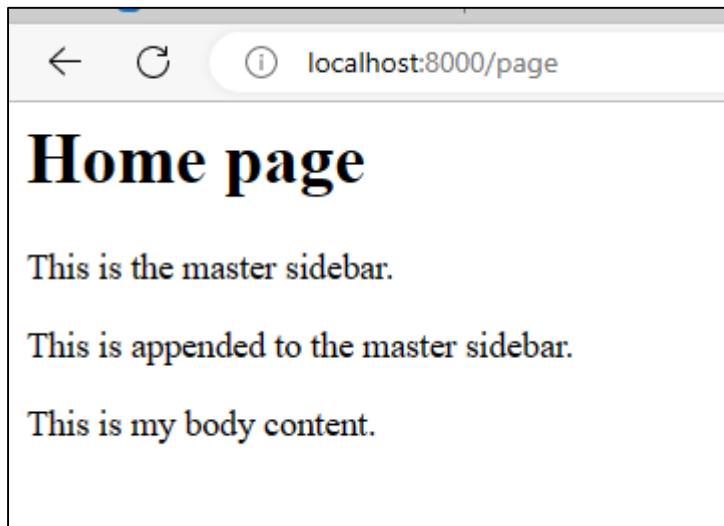
Copiez le code suivant dans le fichier `lines.blade.php`

```
<h1>Home page</h1>
```

Dans le code de la vue `page`, vous rajoutez la ligne suivante :

```
@include('partials.lines')
```

//output



## 10. Erreurs de validation

La directive `@error` peut être utilisée pour vérifier rapidement si des messages d'erreur de validation existent pour un attribut donné. Dans une directive `@error`, vous pouvez faire écho à la variable `$message` pour afficher le message d'erreur :

```
<!-- /resources/views/post/create.blade.php -->
```

```
<label for="title">Post Title</label>
```

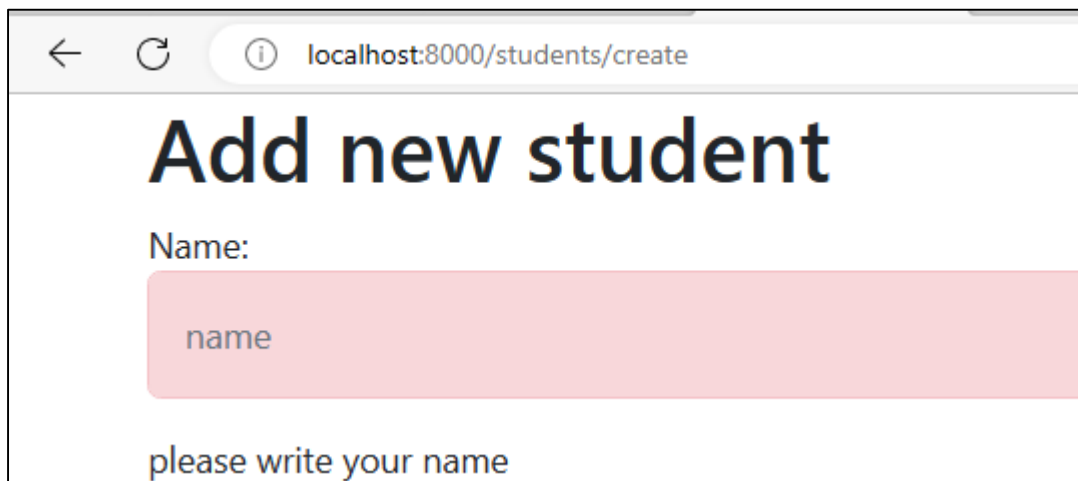
```
<input id="title"  
      type="text"
```

```
class=" @error('title') is-invalid @else is-valid @enderror">
```

```
@error('title')
```

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

```
@enderror
```



A screenshot of a web browser window. The address bar shows 'localhost:8000/students/create'. The page has a heading 'Add new student'. Below the heading is a label 'Name:' followed by a text input field. The input field has a light red background and the text 'name' inside it. Below the input field is a message 'please write your name'.