



# Framework PHP : LARAVEL

## Les vues

### Mouhamed DIOP - Ingénieur de Conception en Informatique (ESP)

# Objectifs spécifiques

---

**A l'issue de cette séquence, l'étudiant devrait :**

- Savoir ce que représente une vue sous Laravel
- Pouvoir créer des vues
- Pouvoir associer des routes aux vues créées
- Pouvoir passer des données à une vue
- Pouvoir faire hériter des vues à partir d'autres



# Qu'est-ce qu'une vue ?

---

## Les vues contiennent le code HTML fourni par une application

- Elles permettent de séparer le code lié à la logique applicative de celui concernant la présentation (l'affichage)
- Elles sont stockées dans le répertoire **resources/views** et porte généralement l'extension **.blade.php**
- Ces fichiers peuvent contenir aussi bien du HTML que du PHP, en plus d'autres codes spécifiques
- Il est possible pour une vue de porter d'autres extensions
  - **.html** si elle ne contient que du HTML (possible depuis la version 5.8.18 de Laravel)
  - **.php** si elle n'utilise pas de fonctionnalités spécifiques au **moteur de templates Blade** que nous verrons juste après



# Qu'est-ce qu'une vue ?

Dans sa version la plus simple, une vue peut juste correspondre à un fichier HTML

- Elle peut dans ce cas aussi bien porter l'extension **.html** que **.blade.php**



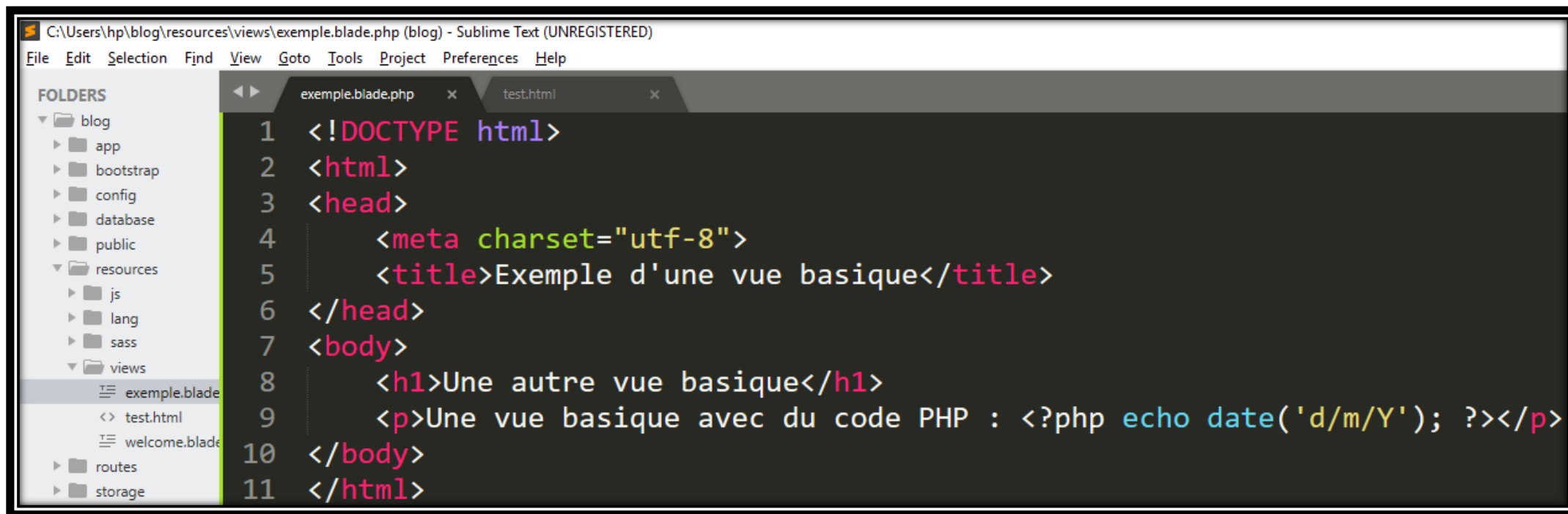
The screenshot shows a Sublime Text editor window with the title bar 'C:\Users\hp\blog\resources\views\exemple.blade.php (blog) - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. On the left, a 'FOLDERS' sidebar shows a tree structure: blog (expanded) contains app, bootstrap, config, database, public, resources (expanded), and views (expanded). Under 'resources', there are folders for js, lang, sass, and views. Under 'views', there are files 'exemple.blade.php' and 'welcome.blade.php', and folders for routes, storage, and tests. The main editor area shows the content of 'exemple.blade.php' with line numbers 1 through 11. The code is a basic HTML template:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <title>Exemple d'une vue basique</title>
6 </head>
7 <body>
8     <h1>Vue basique</h1>
9     <p>Je suis l'exemple vivant d'une vue basique</p>
10 </body>
11 </html>
```

# Qu'est-ce qu'une vue ?

## Une vue peut aussi contenir du code PHP

- On peut ainsi y incorporer du contenu dynamique
- Elle ressemble dans ce cas aux fichiers PHP classiques



The screenshot shows a Sublime Text editor window titled "C:\Users\hp\blog\resources\views\exemple.blade.php (blog) - Sublime Text (UNREGISTERED)". The editor displays the content of "exemple.blade.php" with line numbers 1 through 11. The code is a Blade template that outputs an HTML page. It includes a DOCTYPE declaration, a head section with a meta charset and a title, and a body section with a heading and a paragraph containing a PHP echo statement. The left sidebar shows a file explorer with a tree structure of the project files.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <title>Exemple d'une vue basique</title>
6 </head>
7 <body>
8     <h1>Une autre vue basique</h1>
9     <p>Une vue basique avec du code PHP : <?php echo date('d/m/Y'); ?></p>
10 </body>
11 </html>
```

# Les templates Blade

---

**Blade est un **moteur de templates** simple (mais puissant) fourni avec Laravel**

- Il permet de créer des vues avec une syntaxe épurée
- Il n'empêche pas l'utilisation directe de code PHP dans les vues
- Les vues contenant des instructions Blade doivent porter l'extension **.blade.php**
- Toutes les vues Blade sont traduites en PHP avant d'être mises en cache
  - La cache n'est mise à jour que si la vue correspondante est modifiée
- Ce système de templating est optimisé et ne surcharge donc pas une application



# Les templates Blade : affichage

**Pour afficher une variable, il suffit de le mettre entre accolades**

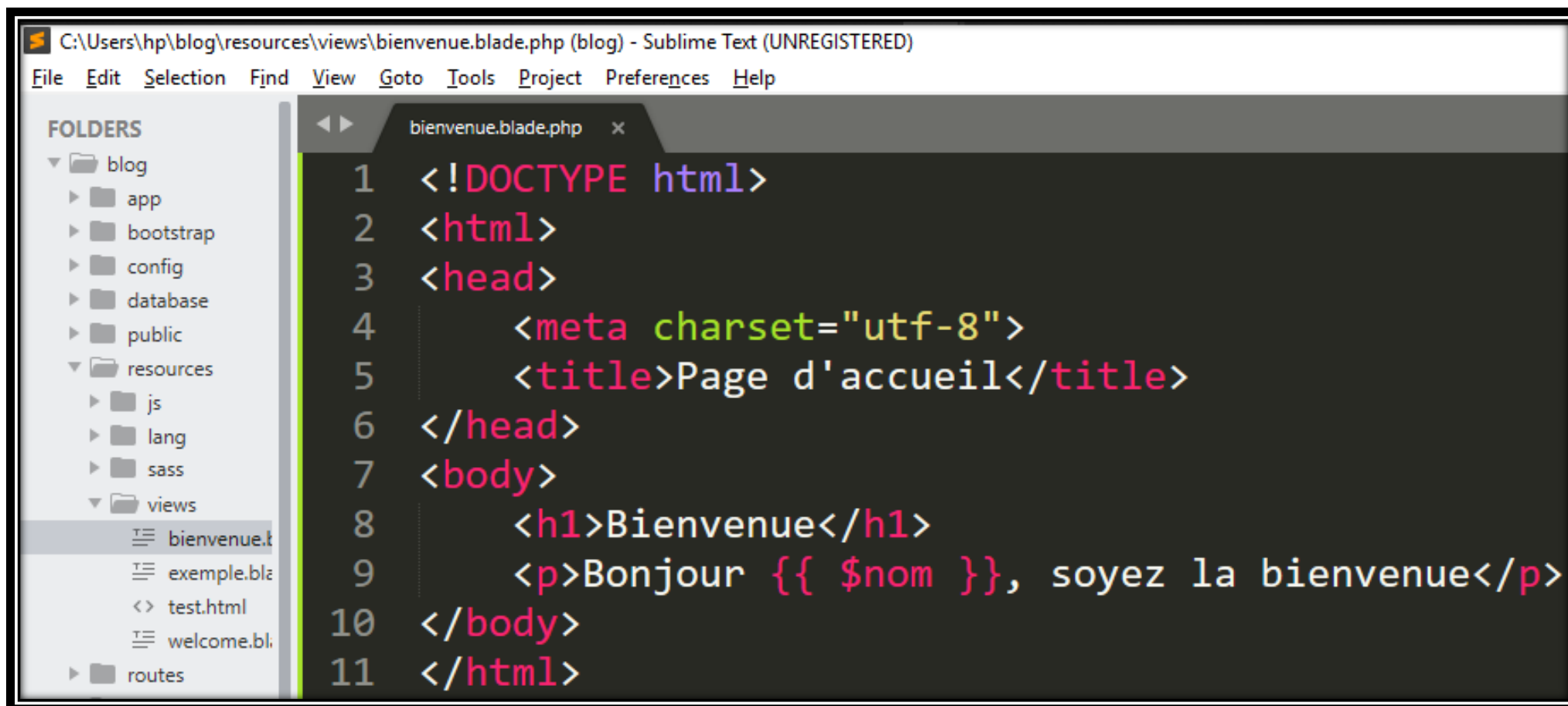
- L'expression `{{ $nom }}` équivaut à `<?php echo htmlspecialchars($nom); ?>`
  - L'affichage est sécurisé. Elle convertit les caractères spéciaux en entités HTML afin d'éviter les failles XSS
- Au besoin, il est possible de désactiver cette conversion en utilisant une variante de la syntaxe d'affichage
  - L'expression `{!! $nom !!}` équivaut à `<?php echo $nom; ?>`
  - Prendre les précautions nécessaires à son utilisation





# Les templates Blade : affichage

## Exemple d'affichage d'un message de bienvenue



The screenshot shows a Sublime Text editor window with the title bar 'C:\Users\hp\blog\resources\views\bienvenue.blade.php (blog) - Sublime Text (UNREGISTERED)'. The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. On the left, a 'FOLDERS' sidebar lists the project structure: 'blog' (expanded) containing 'app', 'bootstrap', 'config', 'database', 'public', 'resources' (expanded), 'js', 'lang', 'sass', and 'views' (expanded). Under 'views', several files are listed: 'bienvenue.bl', 'exemple.bl', 'test.html', 'welcome.bl', and 'routes'. The main editor area displays the content of 'bienvenue.blade.php' with line numbers 1 through 11. The code is a Blade template that outputs an HTML page with a welcome message.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <meta charset="utf-8">
5     <title>Page d'accueil</title>
6 </head>
7 <body>
8     <h1>Bienvenue</h1>
9     <p>Bonjour {{ $nom }}, soyez la bienvenue</p>
10 </body>
11 </html>
```



# Les templates Blade : structures conditionnelles

Il est possible de construire des structures conditionnelles en utilisant les directives **@if**, **@elseif**, **@else** et **@endif**

```
<html>
<head>
    <title>Mes messages</title>
</head>
<body>
    @if (count($messages) === 1)
        Vous avez un nouveau message !
    @elseif (count($messages) > 1)
        Vous avez plusieurs messages !
    @else
        Vous n'avez pas de message !
    @endif
</body>
</html>
```

# Les templates Blade : structures conditionnelles

Pour plus de commodités, Blade fournit également une directive **@unless**, l'équivalent de **if not ...**

```
<html>
<head>
    <title>Authentication</title>
</head>
<body>
    @unless (Auth::check())
        Vous n'êtes pas connecté
    @endunless
</body>
</html>
```

# Les templates Blade : structures conditionnelles

Les instructions **switch** peuvent être construites en utilisant les directives **@switch**, **@case**, **@break**, **@default** et **@endswitch**

```
<html>
<head>
    <title>Switch</title>
</head>
<body>
    @switch($i)
        @case(1)
            Premier cas...
            @break
        @case(2)
            Second cas...
            @break
        @default
            Cas par défaut...
    @endswitch
</body>
</html>
```

# Les templates Blade : structures itératives

Blade fournit également des directives simples pour travailler avec les boucles PHP

- Ces directives fonctionnent de la même manière que leurs homologues sous PHP

```
<html>
<head>
  <title>Boucles</title>
</head>
<body>
  @for ($i = 0; $i < 10; $i++)
    La valeur courante est {{ $i }}
  @endfor

  @foreach ($users as $user)
    <p>Utilisateur N°{{ $user->id }}</p>
  @endforeach
</body>
</html>
```

```
<html>
<head>
  <title>Boucles</title>
</head>
<body>
  @forelse ($users as $user)
    <li>{{ $user->name }}</li>
  @empty
    <p>Aucun utilisateur</p>
  @endforelse

  @while (true)
    <p>Boucle infinie</p>
  @endwhile
</body>
</html>
```

# Les templates Blade : commentaire et code PHP

---

## Blade permet également de définir des commentaires dans les vues

- Ces commentaires ne sont toutefois pas inclus dans le HTML renvoyé par l'application  
`{{-- ceci est un commentaire qui ne sera pas envoyé avec le HTML --}}`

## Il est également possible d'inclure directement du code PHP dans une vue

- Blade fournit la directive `@php...@endphp`

`@php`

`echo date('d/m/Y') ;`

`@endphp`



# Héritage de templates

---

## Blade permet de dériver une vue à partir d'une autre

- Certaines parties d'un site sont présentes dans toutes les pages
  - Entête
  - Menus
  - Etc.
- Étant donné que la plupart des applications Web conservent la même disposition générale sur plusieurs pages, il est pratique de définir cette disposition en tant que vue unique
- Cette vue unique va servir de **canevas principal** (ou **modèle** ou **template**) sur lequel on va se baser pour la création d'autres vues



# Héritage de templates

## Exemple de canevas principal

```
{{-- Emplacement: resources/views/layout.blade.php --}}  
<html>  
<head>  
    <title>@yield('title')</title>  
</head>  
<body>  
    <div id="entete">Entête de la page</div>  
    <div id="menu">Menu de la page</div>  
    <div id="contenu">  
        @yield('contenu')  
    </div>  
</body>  
</html>
```





# Héritage de templates

---

## Le canevas principal

- C'est une vue qui décrit de manière générale ce à quoi ressemblent les pages
- Les parties qui changent d'une page à une autre sont marquées à travers la directive `@yield`. Ce dernier prend en paramètre le nom choisi pour marquer la partie
- Ces parties marquées seront définies par la vue dérivée du canevas principal (vue enfant)
- La directive `@yield` peut prendre comme second paramètre la valeur par défaut (valeur prise si elle n'est pas défini par la vue enfant)
  - **Exemple** : `@yield('title', 'Titre par défaut')` ;



# Héritage de templates

## Exemple d'une vue héritant du canevas principal

- Pour faire hériter la vue *connexion* de notre canevas principal (dénommé *layout*), il faut utiliser la directive **@extends** et lui donner comme paramètre le nom du canevas principal (sans l'extension)
- La directive **@section** permet de donner du contenu aux parties préalablement marquées avec la directive **@yield**. Elle dispose de deux syntaxes d'utilisation (illustration ci-contre)

```
{{-- Emplacement: resources/views/connexion.blade.php --}}  
@extends('layout')  
@section('title', 'Connexion')  
@section('contenu')  
    Contenu de la page de connexion  
@endsection
```



# Héritage de templates

La directive **@section** peut aussi être utilisée au niveau du canevas principal

- Pour pré-remplir une partie et ainsi permettre à la vue enfant d'y ajouter du contenu spécifique (ou le remplacer)
- Il est suivi dans ce cas de la directive **@show** et non pas de **@endsection**

```
{{{-- Emplacement: resources/views/layout.blade.php --}}
<html>
<head>
  <title>@yield('titre', 'Accueil')</title>
</head>
<body>
  <div id="entete">Entête de la page</div>
  @section('menu')
    <div id="menu">Menu principal de la page</div>
  @show
  <div id="contenu">
    @yield('contenu')
  </div>
</body>
</html>
```

# Héritage de templates

## Exemple d'une vue héritant du canevas principal précédent

```
{{-- Emplacement: resources/views/connexion.blade.php --}}
@extends('layout')
@section('titre', 'Connexion')
@section('menu')
    @parent {{-- on inclut d'abord le menu principal --}}
    Contenu du menu secondaire {{-- puis le menu secondaire --}}
@endsection
@section('contenu')
    Contenu de la page de connexion
@endsection
```



# Héritage de templates

---

**Il est possible de faire référence à une vue se trouvant dans un répertoire différent**

- Admettons que nous avons deux vues :
  - L'une se trouvant dans `resources/views/login.blade.php`
  - L'autre se trouvant dans `resources/views/layouts/main.blade.php`
- Si nous voulons que la vue **login** hérite de **main**, la directive `@extends` deviendra :
  - `@extends('layouts.main')` ;
  - Le point (.) est utilisé ici comme séparateur bien vrai que l'utilisation d'un slash (/) demeure possible



# Dénomination des vues

---

**Bien que ce n'est pas obligatoire, il est recommandé de**

- Garder une certaine cohérence entre les URL et les noms des vues qui leur sont associées
  - Par exemple, pour l'accès à l'URL `/login`, nommer la vue correspondante `login.blade.php` au lieu de `connexion.blade.php`
- Donner des noms simples et expressifs
  - Eviter de mettre des caractères spéciaux dans les noms des vues
  - Eviter de mettre des noms qui n'ont rien à voir avec ce que fait la vue



# Définir une route pour une vue

Il est possible de faire appel à une vue grâce au routage préalablement abordé

- Par exemple, pour afficher une vue de connexion à l'utilisateur quand il accède à l'URL <http://monsite.test/login>, il faudra :
  - Créer la vue `resources/views/login.blade.php`
  - Créer une route web permettant d'accéder à cette vue
- Pour la création de la route
  - Ouvrir le fichier `routes/web.php`
  - Ajouter une nouvelle route en utilisant l'une ou l'autre des syntaxes suivantes

```
// Syntaxe si on veut effectuer un
// traitement avant de retourner la vue
Route::get('/login', function () {
    return view('connexion');
});
```

```
// Syntaxe si on veut retourner
// la vue sans aucun autre traitement
Route::view('/login', 'connexion');
```



# Passage de données à une vue

## Il est possible de transmettre des données à une vue

- Il suffit de les passer comme dernier paramètre quand on appelle la vue
- Les paramètres sont envoyés à travers un tableau associatif
  - Plusieurs paramètres peuvent être envoyés en même temps

```
// Une variable $name sera accessible  
// à la vue welcome. Elle vaut 'Ndiaye'  
Route::view('/welcome', 'welcome', ['name' => 'Ndiaye']);
```

```
// Deux variables ($date et $profil)  
// seront accessibles la vue accueil  
Route::get('/', function () {  
    $today = date('d/m/Y');  
    return view('accueil', [  
        'date' => $today,  
        'profil' => 'visiteur'  
    ]);  
});
```

# Passage de données à une vue

## D'autres syntaxes de passage de données existent

```
// Une variable $utilisateurs sera accessible à
// la vue list se trouvant dans le dossier user
Route::get('/users', function () {
    $utilisateurs = [
        'Modou Fall',
        'Khady Sarr',
        'Marcel Diatta',
        'Bintou Mané'
    ];

    return view('user.list', [
        'utilisateurs' => $utilisateurs
    ]);
});

// L'une ou l'autre de ces syntaxes pourrait être utilisées
return view('user.list', compact('utilisateurs'));
return view('user.list')->with('utilisateurs', $utilisateurs);
return view('user.list')->withUtilisateurs($utilisateurs);
```

# Liens utiles

---

## Pour aller plus loin...

- <https://laravel.com/docs/6.x/views>
- <https://laravel.com/docs/6.x/routing#view-routes>
- <https://laravel.com/docs/6.x/blade>
- <https://laracasts.com/series/laravel-from-scratch-2018/episodes/4>

