BLADE TEMPLATES

1.1. Giới thiệu

Blade template là một công cụ giúp xử lý cú pháp trong view một cách ngắn ngọn. Không giống như các Template khác trong PHP, Blade cho phép sử dụng cả PHP thuần trong đó. Tất cả các Blade Template sẽ được biên dịch về mã PHP thuần và cache lại nên về cơ bản sử dụng Blade Template cũng không làm cho ứng dụng chậm đi.

Blade views có thể được trả về từ các Routes hoặc Controller bằng cách sử dụng view helper. Tất nhiên, như đã đề cập trong phần trước về views, dữ liệu có thể được truyền đến Blade view bằng cách sử dụng đối số thứ hai của view helper:

```
Route::get('/', function () {
    return view('greeting', ['name' => 'Finn']);
});
```

1.2. Displaying Data

Để sử dụng Blade Template trong Laravel, chỉ cần tạo view với đuôi file là .blade.php, cách sử dụng giống view thông thường. Có thể hiển thị dữ liệu được truyền đến Blade views bằng cách bao biến trong dấu ngoặc nhọn.

Ví dụ:

File `resources/views/demo-blade-tempalte.blade.php`:

```
<body>
    <h1>Blade Template trong Laravel!</h1>

    <h2>Xin chao! {{ $name }}</h2>

    <h3>Hello! {!! $name !!}</h3>

    <h4>Hi! <?=$name?></h4>
</body>
```

File `routes/web.php`:

```
Route::get('/demo-blade-template', function () {
    return view('demo-blade-template', ['name' => 'Blade Template']);
});
```

Kết quả:

Cặp dấu {{ }} tượng trưng cho câu lệnh echo trong PHP kết hợp với hàm htmlspecialchars để phòng chống lỗi XSS attack. Nghĩa là sẽ không thể in ra mã HTML sử dụng cặp dấu {{ }} này được.

Rendering JSON

Đôi khi muốn truyền một mảng vào view với ý định hiển thị nó dưới dạng JSON để khởi tạo một biến JavaScript. Ví dụ:

```
<script>
    var app = <?php echo json_encode($array); ?>;
</script>
```

Tuy nhiên, thay vì gọi json_encode theo cách thủ công, có thể sử dụng lệnh `@json` Blade. Chỉ thị `@json` chấp nhận các đối số giống như hàm `json_encode` của PHP. Theo mặc định, chỉ thị `@json` gọi hàm `json_encode` với các cờ `JSON_HEX_TAG`, `JSON_HEX_APOS`, `JSON_HEX_AMP` và `JSON_HEX_QUOT`:

```
<script>
    var app = @json($array);
    var app = @json($array, JSON_PRETTY_PRINT);
</script>
```

HTML Entity Encoding

Mặc định, Blade sẽ mã hóa kép các thực thể HTML. Nếu muốn vô hiệu mã hóa kép, hãy gọi phương thức `Blade::withoutDoubleEncoding` từ phương thức `boot` của `AppServiceProvider`:

```
<?php

namespace App\Providers;
```

```
use Illuminate\Support\Facades\Blade;
use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider {
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot() {
        Blade::withoutDoubleEncoding();
    }
}
```

Displaying Unescaped Data

Mặc định, các câu lệnh Blade `{{ }}` được gửi tự động thông qua hàm htmlspecialchars của PHP để ngăn chặn các cuộc tấn công XSS. Nếu không muốn dữ liệu bị escaped, có thể sử dụng cú pháp sau:

```
Hello! {!! $name !!}
```

1.3. Blade & JavaScript Frameworks

Để Blade `không compile` đoạn code nào đó thì chỉ cần thêm ký tự `@` vào trước đoạn code.

File `resources/views/demo-blade-tempalte.blade.php`:

```
<body>
    <h1>Blade Template trong Laravel!</h1>
    <h2>Xin chao! @{{ $name }}</h2>
    <h3>Hello! @{!! $name !!}</h3>
    <h4>Hi! @<?=$name?></h4>
</body>
```

Kết quả:

Trong ví dụ trên, `@` symbol sẽ bị Blade loại bỏ. Tuy nhiên, biểu thức `{{ name }}` sẽ vẫn không bị công cụ Blade ảnh hưởng, cho phép nó được hiển thị bởi khung JavaScript framework.

`@` symbol cũng có thể được sử dụng để thoát khỏi các chỉ thị của Blade:

```
{{-- Blade template --}}

@@json()

<!-- HTML output -->

@json()
```

The `@verbatim` Directive

Nếu đang hiển thị các biến JavaScript trong một phần lớn template, có thể bọc HTML trong chỉ thị `@verbatim` để không phải đặt trước mỗi câu lệnh Blade echo bằng ký hiệu `@`:

```
@verbatim
    <div class="container">

        Hello! {{ name }}.

    </div>
@endverbatim
```

1.4. Blade Directives

Ngoài việc kế thừa template và hiển thị dữ liệu, Blade cũng cung cấp các các cấu trúc điều khiển PHP phổ biến, chẳng hạn như các câu lệnh điều kiện, lặp,....

If Statements

Có thể tạo câu lệnh `if` bằng cách sử dụng các lệnh `@if`, `@elseif`, `@else`, và `@endif` . Các chỉ thị ý nghĩa tương tự như đối với PHP thuần `if`, `else`, `elseif`, `endif`:

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

Ngoài ra Blade còn cung cấp thêm một số các directives rút gọn khác như:

@unless

Nếu logic trong `unless` trả về `false` thì code bên trong directive sẽ được thực thi:

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

@isset

Kiểm tra một biến có tồn tại hay không. Logic tương tự `if(isset())` trong PHP:

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

@empty

Kiểm tra xem một biến có tồn tại hoặc bằng `null` hay không. Logic tương tự `if(empty())` trong PHP:

```
@empty($records)
    // $records is "empty"...
@endempty
```

Authentication Directives

Các lệnh `@auth` và `@guest` có thể được sử dụng để nhanh chóng xác định xem người dùng hiện tại được xác thực (`authenticated`) hay là khách:

@auth

```
@auth
    // The user is authenticated...
@endauth
```

@guest

```
@guest
    // The user is not authenticated...
@endguest
```

Nếu cần, có thể chỉ định `authentication` cần được kiểm tra khi sử dụng chỉ thị `@auth` và `@guest`:

```
@auth('admin')
    // The user is authenticated...
@endauth

@guest('admin')
    // The user is not authenticated...
@endguest
```

Environment Directives

@production

Có thể kiểm tra xem ứng dụng có đang chạy trong môi trường `production` hay không bằng cách sử dụng lệnh **`@production`**. Nếu môi trường là `production` thì code bên trong cặp directive sẽ được thực thi:

```
@production
    // Production specific content...
@endproduction
```

@env

Hoặc, có thể xác định xem ứng dụng có đang chạy trong một môi trường cụ thể hay không bằng cách sử dụng lệnh `@env`:

```
@env('staging')
    // The application is running in "staging"...
@endenv

@env(['staging', 'production'])
    // The application is running in "staging" or "production"...
@endenv
```

Section Directives

@hasSection

Để kiểm tra xem trong template cha có tồn tại section nào không. Nếu có code bên trong cặp directive sẽ được thực thi:

```
@hasSection('navigation')
    <div class="pull-right">
        @yield('navigation')
    </div>
    <div class="clearfix"></div>
@endif
```

@sectionMissing

Đây là trường hợp ngược lại của `@hasSection`:

```
@sectionMissing('navigation')
    <div class="pull-right">
        @include('default-navigation')
    </div>
@endif
```

Switch Statements

Các câu lệnh switch có thể được xây dựng bằng cách sử dụng các lệnh `@switch`, `@case`, `@break`, `@default` và `@endswitch`:

```
@switch($i)
    @case(1)
        First case...
        @break
    @case(2)
        Second case...
        @break
    @default
        Default case...
@endswitch
```

Loops

Để sử dụng các vòng lặp trong Blade, cần thêm `@` vào trước các câu lệnh lặp trong PHP, và Blade không hỗ trợ câu lệnh `do-while`:

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

Trong đó `foresle` sẽ kiểm tra nếu array cần lặp là một mảng rỗng thì logic trong cặp directive `@empty` và `@endforelse` sẽ được thực thi.

Khi sử dụng các vòng lặp, cũng có thể kết thúc vòng lặp hoặc bỏ qua lần lặp hiện tại bằng cách sử dụng chỉ thị `@continue` và `@break`:

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

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

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

Cũng có thể bao gồm điều kiện `continue` hoặc `break` trong khai báo:

```
@foreach ($users as $user)
    @continue($user->type == 1)

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

    @break($user->number == 5)
@endforeach
```

The Loop Variable

Khi lặp, một biến `$loop` sẽ có sẵn bên trong vòng lặp. Biến này cung cấp quyền truy cập vào một số thông tin hữu ích như chỉ số vòng lặp hiện tại, đây là lần lặp đầu tiên hay lần cuối cùng,...:

```
@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
```

Nếu đang ở trong một vòng lặp lồng nhau, có thể truy cập biến `$loop` của vòng lặp cha thông qua thuộc tính `parent`:

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

Biến `$loop` cũng chứa nhiều thuộc tính hữu ích khác:

Property	Description
`$loop->index`	Lấy chỉ mục (index) của vòng lặp hiện tại (bắt đầu từ 0).
`$loop->iteration`	Lấy lần lặp hiện tại là lần lặp thứ mấy (bắt đầu từ 1).
`$loop->remaining`	Lấy số vòng lặp còn phải lặp.
`$loop->count`	Lấy số lượng item của vòng lặp.
`$loop->first`	Kiểm tra xem lần lặp hiện tại có phải lần lặp đầu tiên không?
`$loop->last`	Kiểm tra xem lần lặp hiện tại có phải lần lặp cuối cùng không?
`$loop->even`	Kiểm tra xem lần lặp này có phải chẵn không?
`$loop->odd`	Kiểm tra xem lần lặp này có phải lẻ không?
`$loop->depth`	Mức độ lồng của vòng lặp hiện tại.
`$loop->parent`	Truy xuất đến thông tin của vòng lặp cha.

Conditional Classes

Chỉ thị `@class` biên dịch có điều kiện một chuỗi `CSS class`. Chỉ thị chấp nhận một mảng các lớp trong đó khóa mảng chứa lớp hoặc các lớp muốn thêm vào, trong khi giá trị là một biểu thức boolean. Nếu phần tử mảng cómột `numeric key`, nó sẽ luôn được đưa vào danh sách lớp được hiển thị:

```
@php
    $isActive = false;
    $hasError = true;
@endphp
```

```
<span @class([
    'p-4',
    'font-bold' => $isActive,
    'text-gray-500' => ! $isActive,
    'bg-red' => $hasError,
])></span>

<span class="p-4 text-gray-500 bg-red"></span>
```

Including Subviews

Blade template cho phép include các view khác vào trong view hiện tại một cách đơn giản bằng việc sử dụng chỉ thị `@include`. Lúc này ở các view được include cũng có thể sử dụng được tất cả các biến có trong view hiện tại.

Ví dụ: Giả sử có một view `resources/views/error.blade.php` như sau:

```
<h2>Day la View được include! - {{ $error }}</h2>
```

File `resources/views/home.blade.php`:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>
    @include('error')
</body>
```

File `routes/web.php`:

```
Route::get('/home ', function () {
    return view('home', ['name' => 'View home!', 'error' => 'View error!']);
 });
```

Kết quả:

View page source:

Trong trường hợp muốn đưa thêm `data` vào trong view được include thì có thể sử dụng cú pháp sau:

```
@include('view.name', [$variableName => $data])
```

Trong đó:

- `$variableName` là tên biến chưa data muốn truyền vào sub view;
- `$data` là giá trị của `$variableName`.

File `resources/views/home.blade.php`:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>

    @include('error',['error' => 'View error!'])
</body>
```

File `routes/web.php`:

```
Route::get('/home ', function () {
    return view('home', ['name' => 'View home!']);
 });
```

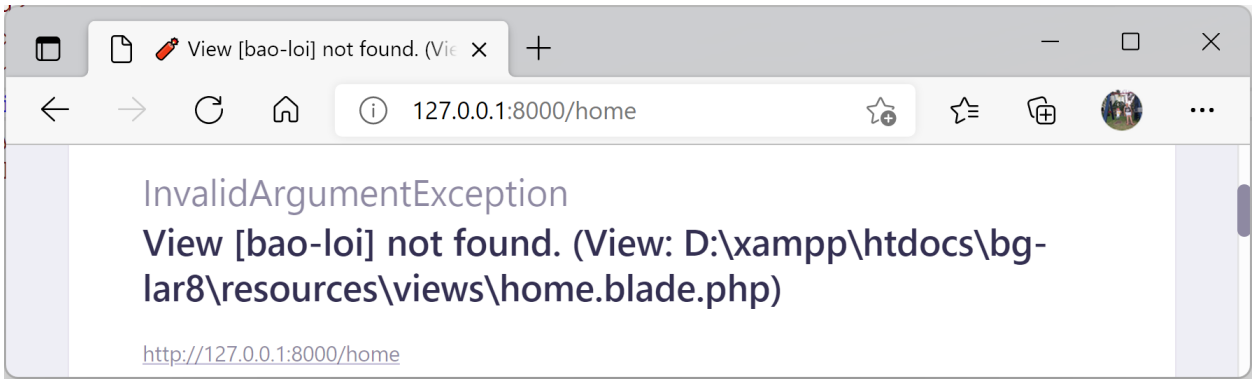
`@includeIf`

Trong Blade Template nếu `@include` một view không tồn tại thì Laravel sẽ báo lỗi. Nếu muốn bỏ qua exception đó thì có thể dùng `@includeIf`. Directive này sẽ check nếu như view tồn tại thì mới thực thi việc include view, không thì bỏ qua:

File resources/views/home.blade.php:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>
    @include('bao-loi',['error' => 'View error!'])
</body>
```

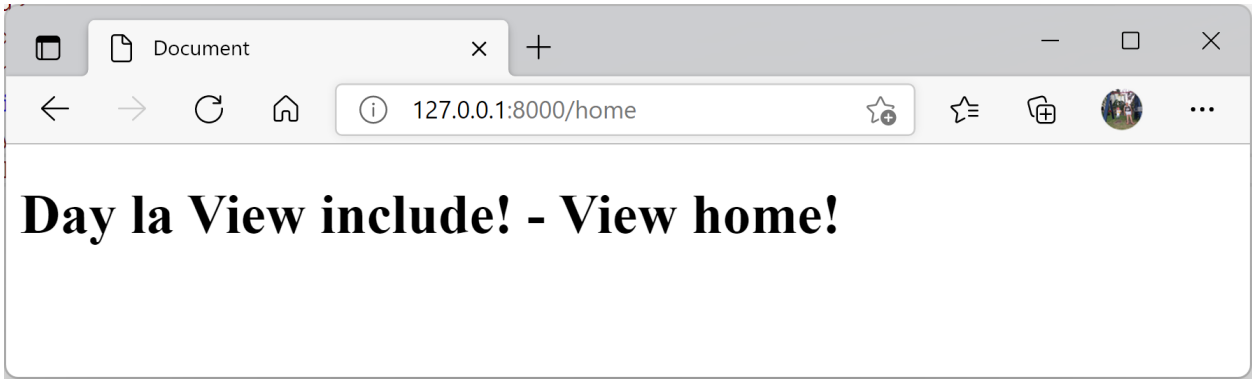
Kết quả: Khi view 'bao-loi' không tồn tại:



File resources/views/home.blade.php:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>
    @includeIf('bao-loi',['error' => 'View error!'])
</body>
```

Kết quả: Khi view 'bao-loi' không tồn tại:



@includeWhen

Kiểm tra điều kiện trước khi include view, cú pháp:

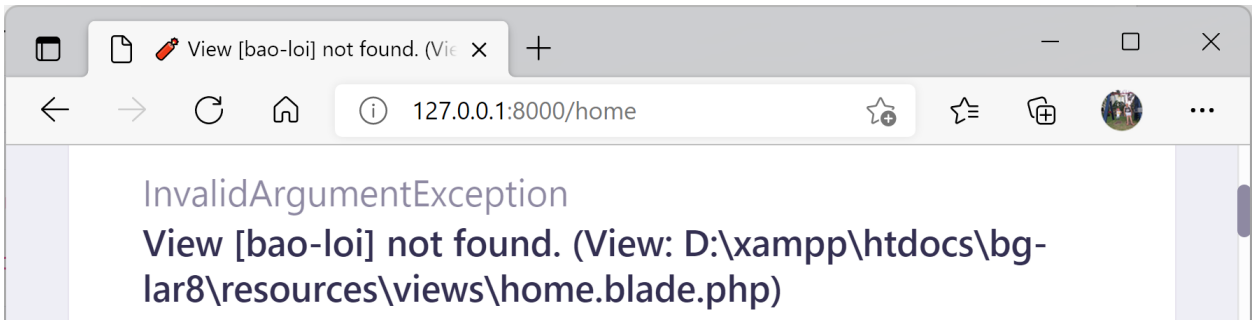
```
@includeWhen($boolean, 'view.name', $data)
```

$boolean trả về true thì view sẽ được include và ngược lại false thì view sẽ không được include.

File resources/views/home.blade.php:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>
    @includeWhen(true,'bao-loi',['error' => 'View error!'])
</body>
```

Kết quả: Khi view 'bao-loi' không tồn tại và $boolean là true:



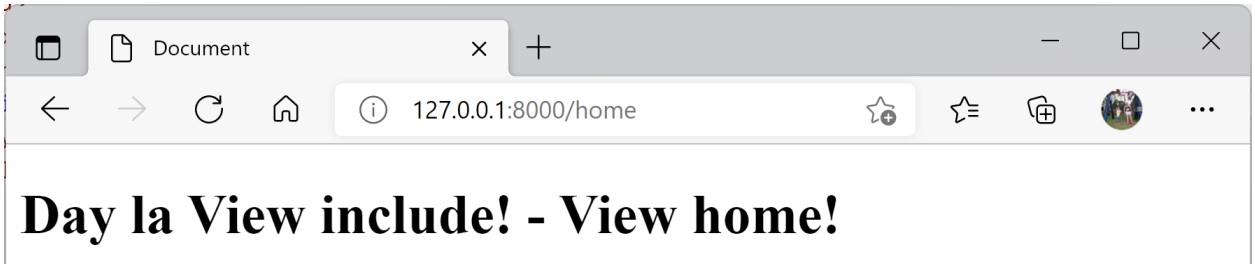

```
@includeWhen($boolean, 'view.name', $data)
```

$boolean trả về true thì view sẽ được include và ngược lại false thì view sẽ không được include.

File resources/views/home.blade.php:

```
<body>
    <h1>Day la View include! - {{ $name }}</h1>
    @includeWhen(false,'bao-loi',['error' => 'View error!'])
</body>
```

Kết quả: Khi view 'bao-loi' không tồn tại và $boolean là false:



@includeUnless

Đây là directive phủ định của @includeWhen. Nghĩa là $boolean trả về true thì view sẽ không được include và ngược lại false thì view sẽ được include. Cú pháp:

```
@includeUnless($boolean, 'view.name', $data)
```

@includeFirst

Cho phép truyền vào một danh sách view. Và nó sẽ kiểm tra xem nếu view nào tồn tại đầu tiên trong danh sách thì nó sẽ include view đó. Các view phía sau sẽ không được include nữa. Cú pháp:

```
@includeFirst(['view.name1', 'view.name2', 'view.name3',...], $data)
```

@each

Có thể kết hợp các vòng lặp và include thành một dòng với chỉ thị `@each` của Blade, với cú pháp:

```
@each('view.name', $array, 'item', 'view.empty');
```

Trong đó:

- `view.name` là view muốn include;
- `$array` là mảng dữ liệu muốn lặp;
- `item` là giá trị sẽ được gán qua mỗi lần lặp;
- `view.empty` là view sẽ được include khi `$array` rỗng (giá trị này có thể bỏ qua).

Ví dụ:

File `resources/views/post.blade.php`:

```
<h3> {{ $post['name'] }} </h3>
```

File `resources/views/home.blade.php`:

```
<body>
    <div>
        <h1>Danh sach post</h1>
        @each('post', $posts, 'post','error')
     </div>
</body>
```

File `routes/web.php`:

```
Route::get('/home', function () {
    $posts = [
        ['name' => 'Post 1'],
        ['name' => 'Post 2'],
        ['name' => 'Post 3'],
        ['name' => 'Post 4']
    ];
```

```
    return view('home', ['posts' => $posts]);
});
```

The @once Directive

`@once` directive cho phép thực thi hành động bên trong nó một lần duy nhất khi render view.

Ví dụ có 2 chỗ cùng include một đoạn code nếu như sử dụng directive `@once` thì các đoạn code phía sau once đầu tiên sẽ không được thực thi nữa.

```
@once
    @push('scripts')
        <script>
            // Your custom JavaScript...
        </script>
    @endpush
@endonce
```

Comments

Blade cũng cho phép thêm các comments trong các views. Tuy nhiên, không giống như các comments HTML:

```
{{-- This comment will not be present in the rendered HTML --}}
```