Bài 16
View và Blade

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

Mục tiêu

- Trình bày được cơ chế hoạt động của view
- Đưa được dữ liệu từ controller lên view
- Trình bày được cú pháp của Blade
- Sử dụng được Blade để tạo layout
- Sử dụng được if trong Blade
- Sử dụng được vòng lặp trong Blade

View

Tạo views
Truyền dữ liệu vào views
Chia sẻ dữ liệu cho tất cả views
View Composers

View trong Laravel

- View chứa nội dung HTML và kết hợp với dữ liệu truyền vào từ Controller
- View được lưu trữ tại thư mục resources/views .

```
<!-- View stored in resources/views/greeting.blade.php -->

<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>
```

Sử dụng View

- Sử dụng View thông qua tên của file (bỏ qua phần đuôi)
- Ví dụ: Sử dụng file resources/views/greeting.blade.php:

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

View ở trong thư mục con

- Khi View được đặt trong một thư mục con của thư mục resources/views thì cần chỉ định tên của thư mục đó
- Ví dụ, với file View là resources/views/ admin/profile.blade.php thì cần chỉ định thư mục admin kèm với tên file

```
return view('admin.profile', $data);
```

Kiểm tra sự tồn tại view

- Sử dụng phương thức exists() của View facade

```php
use Illuminate\Support\Facades\View;

if (View::exists('emails.customer')) {
    //
}
```

Truyền dữ liệu vào View

- Ví dụ: Truyền một mảng vào view:

```
return view('greetings', ['name' => 'Victoria']);
```

- $data nên là một cặp key/value.
- Sử dụng dữ liệu trong View:

<?php echo $key; ?>

Truyền dữ liệu vào view

- Có thể sử dụng hàm with() để truyền dữ liệu sang View

```php
return view('greeting')->with('name', 'Victoria');
```

Chia sẻ dữ liệu cho tất cả các View

- Sử dụng phương thức share của View facade
- Gọi phương thức View::share() trong phương thức boot của một service provider, chẳng hạn như lớp AppServiceProvider

Chia sẻ dữ liệu cho tất cả các View

```php
<?php

namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return  void
     */
    public function boot()
    {
        View::share('key', 'value');
    }

    /**
     * Register the service provider.
     *
     * @return  void
     */
    public function register()
    {
        //
    }
}
```

View Composer

- View Composer là cơ chế cho phép View gọi đến các phương thức mỗi khi View đó được render

```php
<?php

namespace App\Providers;

use Illuminate\Support\Facades\View;
use Illuminate\Support\ServiceProvider;

class ComposerServiceProvider extends ServiceProvider
{
    public function boot()
    {
       View::composer(
            'profile', 'App\Http\ViewComposers\ProfileComposer'
        );
        View::composer('dashboard', function ($view) {
            //
        });
    }
    public function register()
    {}
}
```

Đính kèm Composer vào nhiều View

- Có thể gán một View Composer vào nhiều View

```
View::composer(
    ['profile', 'dashboard'],
    'App\Http\ViewComposers\MyViewComposer'
);
```

```
View::composer('*', function ($view) {
    //
});
```

Blade Templates

Giới thiệu, Template Inheritance
Hiển thị dữ liệu, Control Structures
Including Sub-Views , Stacks
Service Injection, Extending Blade

Giới thiệu

- Blade là templating engine đơn giản nhưng rất tuyệt vợi cung cấp bởi Laravel.
- Không như những templating engine của PHP, Blade không cấm bạn sử dụng code PHP thuần ở trong view.
- Thực tế, tất cả các Blade view được compiled từ code PHP và được cache cho đến khi chúng được chỉnh sửa, nghĩa là Blade không làm tăng thêm chi phí cho ứng dụng của bạn.
- Tất cả các Blade view sử dụng đuôi .blade.php và được lưu trong thư mục resources/views directory.

Template Inheritance

- Định nghĩa một layout
- Kế thừa một layout

Định nghĩa một layout

- Có 2 lợi ích của việc sử dụng Blade là template inheritance và sections.
- Để bắt đầu, Chúng ta hãy xem ví dụ sau. Đầu tiên, chúng ta cùng xem một trang layout "master".
- Vì hầu hết các ứng dụng wb đều có một mẫu layout chung giữa các trang với nhau, nó sẽ rất tiện nếu tạo ra layout này thành một Blade view riêng:

Định nghĩa một layout

```
<!-- Stored in resources/views/layouts/app.blade.php -->

<html>
    <head>
        <title>App Name - @yield('title')</title>
    </head>
    <body>
        @section('sidebar')
            This is the master sidebar.
        @endsection

        <div class="container">
            @yield('content')
        </div>
    </body>
</html>
```

Định nghĩa một layout

- Như bạn có thể thấy, file này có chứa mã HTML mark-up.
- Tuy nhiên, chú ý ở @section và @yield directives.
- The @section directive, đúng như tên của nó, định nghĩa một nội dungtrong khi @yield directive sử dụng để hiển thị dữ liệu ở một vị trí đặt trước.
- Bây giờ chúng ta đã tạo xong một layout cho ứng dúng, hãy cùng tạo ra các trang con kế thừa từ layout này

Kế thừa một layout

- Khi bạn tạo một trang con, sử dụng Blade @extends directive để chỉ ra layout của trang con này "inherit" từ đâu.
- Views kế thừa một Blade layout có thể inject nội dung vào trong sections using @section directives của layout.
- Nhớ rằng, như ví dụ trên, nội dung của những section này được hiển thị khi sử dụng @yield :

Kế thừa một layout

```
<!-- Stored in resources/views/child.blade.php -->

@extends('layouts.app')

@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
```

Kế thừa một layout

- Trong ví dụ trên, phần sidebar để thực hiện @parent directive thêm nội dung vài sidebar (thay vì ghi đè toàn bộ).
- @parent directive sẽ được thay thế bởi nội dung của layout khi view được render.
- Blade views có thể được trả về từ routes bằng cách sử dụng hàm global view :

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

Hiển thị dữ liệu

- Blade & JavaScript Frameworks

Hiển thị dữ liệu

- Bạn có thể truyền dữ liệu vào Blade views bằng cách đặt biết trong cặp ngowacj nhọn. Ví dụ, với route dưới:

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

- Bạn có thể hiển thị nội dung của biến name variable:

```
Hello, {{ $name }}.
```

Hiển thị dữ liệu

- Tất nhiên, bạn không hề bị giới hạn trong việc hiển thị nội dung của biến vào trong view.
- Bạn cũng có thể sử dụng hàm echo của PHP để hiển thị biến.
- Thực tế, you bạn có thể đặt code PHP bạn muốn vào Blade:

```
The current UNIX timestamp is {{ time() }}.
```

Cặp `{{ }}` của Blade được tự động gửi tới hàm `htmlentities` của PHP để ngăn chặn các hành vi tấn công XSS.

Hiển thị dữ liệu nếu tồn tại

- Thỉnh thoảng bạn muốn hiện giá trị một biến, nhưng bạn không chắc nếu biết đó có giá trị.
- Chúng ta có thể thể hiện theo kiểu code PHP :

```
{{ isset($name) ? $name : 'Default' }}
```

- Tuy nhiên, thay vì viết kiểu ternary, Blade provides cung cấp cho bạn một cách ngắn gọn hơn:

```
{{ $name or 'Default' }}
```

- Trong ví dụ trên, nếu biến $name tồn tại, giá trị sẽ được hiện thị. Tuy nhiên, nếu nó không tồn tại, Từ Default sẽ được hiển thị.

Hiện dữ liệu chưa Unescaped

- Mặc định, cặp {{ }} được tự động gửi qua hàm htmlentities của PHP để ngăn chặn tấn công XSS.
- Nếu bạn không muốn dự liệu bị escaped,bạn có thể sử dụng cú pháp:

```
Hello, {!! $name !!}.
```

Phải cẩn thận khi hiện nội dung được người dùng cung cấp. Luôn luôn sử dung cặp ngoặc nhọn để ngăn chặn tấn công XSS attacks khi hiển thị dữ liệu được cung cấp.

Blade & JavaScript Frameworks

- Vì nhiều JavaScript frameworks cũng sử dụng cặp "ngoặc nhọn" để cho biết một biểu thức cần được hiển thị lên trình duyệt, bạn có thể sử dụng biểu tượng @ để nói cho Blade biết được biểu thức này cần được giữ lại. Ví dụ:

```
<h1>Laravel</h1>

Hello, @{{ name }}.
```

- Trong ví dụ này, biểu tượng @ sẽ bị xóa bởi Blade ; tuy nhiên, {{ name }} được dữ lại cho phép nó được render tiếp bởi Javascript framkwork của bạn.

The @verbatim @verbatim Directive

- Nếu bạn muốn hiển thị biến JavaScript trong phần lớn template của bạn, bạn có thể bọc chúng trong directive khi đó bạn sẽ không cần tiền tố @ trước biểu thức điều kiện:

```
@verbatim
    <div class="container">
        Hello, {{ name }}.
    </div>
```

Control Structures

- Cấu trúc điều kiện
- Vòng lặp
- Biến vòng lặp
- Comments

Control Structures

- Ngoài template inheritance và hiển thị dữ liệu, Blade còn cung cấp một số short-cuts PHP control structures, như biểu thức điều kiện và vòng lắm.
- Các short-cuts provide rất rõ ràng, là cách ngắn gọn khi làm việc với PHP control structures, và giống cấu trúc của PHP counterparts.

Cấu trúc điều kiện

- Bạn có xây dựng cấu trúc if sbằng cách sử dụng @if , @elseif , @else , và @endif directives.
- Những directives tương ứng giống các từ khóa của PHP:

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)

    I have multiple records!
@else
    I don't have any records!
@endif
```

Cấu trúc điều kiện

- For convenience, Blade also provides an @unless directive:

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

Vòng lặp

- Ngoài cấu trúc điều kiện, Blade provides cũng cung cấp phương thức hỗ trợ cho việc xử lý vòng lặp.
- Một lần nữa, mỗi directives tương ứng giống các từ khóa 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 (true)
    <p>I'm looping forever.</p>
@endwhile
```

Vòng lặp

Trong vòng lặp, bạn có thể sử dụng biến vòng lặp để lấy được thông tin giá trị của vòng lặp, chẳng hạn như bạn muốn lấy giá trị đầu tiên hoặc cuối cùng của vòng lặp.

- Khi sử dụng vòng lặp bạn cũng có thể kết thúc hoặc bỏ qua vòng lặp hiện tại:

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

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

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

Vòng lặp

- Bạn cũng có thể thêm điều kiện directive biểu diễn trong một dòng:

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

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

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

Biến vòng lặp

- Trong vòng lặp, một biến $loop sẽ tồn tại bên trong vòng lặp. Biến này cho phép ta truy cập một số thông tin hữu ích của vòng lặp như index của vòng lặp hiện tại và vòng lặp đầu hoặc vòng lặp cuối của nó:

```
@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
```

Biến vòng lặp

- Nếu bạn có vòng lặp lồng nhau, bạn có thể truy cập biên $loop của vòng lặp tra qua thuộc tính parent :

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

Biến vòng lặp

- Biến $loop còn chứa một số thông tin hữu ích:

Thuộc tính	Miêu tả
`$loop->index`	Chỉ số index hiện tại của vòng lặp (starts at 0).
`$loop->iteration`	Các vòng lặp hiện tại (starts at 1).
`$loop->remaining`	Số vòng lặp còn lại.
`$loop->count`	Tổng số vòng lặp.
`$loop->first`	Vòng lặp đầu tiên.
`$loop->last`	Vòng lặp cuối cùng.
`$loop->depth`	Độ sâu của vòng lặp hiện tại.
`$loop->parent`	Biến parent loop của vòng lặp trong 1 vòng lặp lồng.

Comments

- Blade còn cho phép bạn comment trong view.
- Tuy nhiên, không như comment của HTML , comment của Blade không đi kèm nội dung HTML được trả về:

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

Including Sub-Views

- Rendering Views cho Collections

Including Sub-Views

- Blade's @include directive cho phép bạn chèn một Blade view từ một view khác. Tất cả các biến tồn tại trong view cha đều có thể sử dụng ở viree chèn thêm:

```
<div>
    @include('shared.errors')

    <form>
        <!-- Form Contents -->
    </form>
</div>
```

Including Sub-Views

- Mặc dù các view được chèn thêm kế thừa tất cả dữ liệu từ view cha, bạn cũng có thể truyền một mảng dữ liệu bổ sung:

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

Bạn nên tránh sử dụng `__DIR__` và `__FILE__` ở trong Blade views, vì chúng sẽ tham chiếu tới vị trí file bị cache.

Rendering Views cho Collections

- Bạn có thể kết hợp vòng lặp và view chèn thêm trong một dòng với @each directive:

```
@each('view.name', $jobs, 'job')
```

- Tham sô thứ nhất là tên của view partial để render các element trong mảng hay collection.
- Tham số thứ hai là một mảng hoặc collection mà bạn muốn lặp, tham số thứ ba là tên của biến được gán vào trong vòng lặp bên view.

Rendering Views cho Collections

- Vì vậy, ví dụ, nếu bạn muốn lặp qua một mảng tên jobs , bạn phải truy xuất vào mỗi biến job trong view partial.
- Key của vòng lặp hiện tại sẽ tồn tại như là key bên trong view partial.
- Bạn cũng có thể truyền tham số thứ tư vào @each directive. tham số này sẽ chỉ định view sẽ dduwcoj render nếu như mảng bị rỗng.

```
@each('view.name', $jobs, 'job', 'view.empty')
```

Stacks

- Blade cho phép bạn đẩy tên stack để cho việc render ở một vị trí nào trong view hoặc layout khac.
- Việc này rất hữu ích cho việc xác định thư viện JavaScript libraries cần cho view con:

```
@push('scripts')
    <script src="/example.js"></script>
@endpush
```

Stacks

- Bạn có thể đẩy một hoặc nhiều vào stack.
- Để render thành công một nội dung stack, truyền vào tên của stack trong @stack directive:

```
<head>
    <!-- Head Contents -->

    @stack('scripts')
</head>
```

Service Injection

- Để @inject directive có thể được sử dụng để lấy lại một service từ Laravel service container.
- Tham số thứ nhất @inject là tên biến của service sẽ được đặt vào, tham số thứ hai là class hoặc tên interface của service bạn muốn resolve:

```
@inject('metrics', 'App\Services\MetricsService')

<div>
    Monthly Revenue: {{ $metrics->monthlyRevenue() }}.
</div>
```

Mở rộng Blade

- Blade còn cho phép bạn tùy biên directives bằng phương thức directive .
- Khi trình viên dich của Blade gặp directive, nó sẽ gọi tới callback được cung cấp với tham số tương ứng.
- Ví dụ dưới đây tạo một @datetime($var) directive để thực hiện format một biết $var , nó sẽ là một thể hiện của DateTime :

Mở rộng Blade

```php
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Blade;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Perform post-registration booting of services.
     *
     * @return  void
     */
    public function boot()
    {
        Blade::directive('datetime', function($expression) {
            return "<?php echo $expression->format('m/d/Y H:i'); ?>";
        });
    }
```

```php
    /**
     * Register bindings in the container.
     *
     * @return  void
     */
    public function register()
    {
        //
    }
}
```

Mở rộng Blade

- Như bạn thấy, chúng ta sẽ móc lỗi phương thức format trong bất cứ biểu thức nào được gửi qua directive.
- Vì vậy, Trong ví dụ trên, Mã PHP được tạo ra bởi directivesẽ là:

```php
<?php echo $var->format('m/d/Y H:i'); ?>
```

> Sau khi cập nhật logiccủa mộtBlade directive, bạn cần xóa hết tất cả các Blade view bị cache. cache Blade views có thể xóa bằng lệnh `view:clear` Artisan.

- Nguồn: https://laravel.com/docs/5.3/blade

Tóm tắt bài học

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: ***ORM and Eloquent***