Course - Laravel Framework

# View

*Sẽ là không thực tế nếu trả về toàn bộ chuỗi tài liệu HTML trực tiếp từ các route và controller của bạn. Rất may, view cung cấp một cách thuận tiện để đặt tất cả HTML của chúng ta trong các tập tin riêng biệt.*

Tags: laravel view, laravel framework

### Giới thiệu

Tất nhiên, không thực tế nếu trả về toàn bộ chuỗi tài liệu HTML trực tiếp từ các route và controller của bạn. Rất may, các chế độ xem cung cấp một cách thuận tiện để đặt tất cả HTML của chúng ta trong các tập tin riêng biệt. View sẽ tách controller / logic ứng dụng của bạn khỏi logic trình bày và được lưu trữ trong thư mục *resources/views*. Một view đơn giản có thể trông giống như sau:

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

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

Vì view này được lưu trữ tại *resources/views/greeting.blade.php*, nên chúng ta có thể trả lại nó bằng cách sử dụng hàm `view` toàn cục như sau:

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

> Tìm kiếm thêm thông tin về cách viết các template Blade? Hãy xem toàn bộ tài liệu về Blade để bắt đầu.

## Tạo và trình bày view

Bạn có thể tạo view bằng cách đặt tập tin có phần mở rộng *.blade.php* vào thư mục resources/views của ứng dụng. Phần mở rộng *.blade.php* thông báo cho framework biết rằng tập tin có chứa Blade template. Các Blade template chứa HTML cũng như các chỉ thị Blade cho phép bạn dễ dàng dùng lại các giá trị, tạo câu lệnh "if", lặp qua dữ liệu và nhiều thứ khác nữa.

Khi bạn đã tạo một view, bạn có thể trả lại nó từ một trong các route hoặc controller của ứng dụng bằng cách sử dụng hàm toàn cục `view` chung:

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

Các chế độ xem cũng có thể được trả lại bằng cách sử dụng facade của `View`:

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

return View::make('greeting', ['name' => 'James']);
```

Như bạn có thể thấy, đối số đầu tiên được truyền cho hàm `view` tương ứng với tên của tập tin `view` trong thư mục *resources/views*. Đối số thứ hai là một mảng dữ liệu sẽ được cung cấp cho view template. Trong trường hợp này, chúng ta đang truyến biến `name`, và nó sẽ được hiển thị trong view bằng cú pháp Blade.

### Lồng các thư mục view

View cũng có thể được lồng ghép trong các thư mục con của thư mục *resources/views*. Ký hiệu "dấu chấm" có thể được sử dụng để tham chiếu các view template lồng nhau. Ví dụ: nếu view của bạn được lưu trữ tại *resources/views/admin/profile.blade.php*, bạn có thể trả lại nó từ một trong các route/controller của ứng dụng như sau:

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

> **Chú ý:** Đừng đặt "dấu chấm" khi đặt tên cho thư mục.

### Tạo view đầu tiên

Sử dụng phương thức `first` của facade `View`, bạn có thể tạo view đầu tiên cho một mảng view nào đó. Điều này có thể hữu ích nếu ứng dụng hoặc package của bạn cho phép các view template được tùy chỉnh hoặc ghi đè:

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

return View::first(['custom.admin', 'admin'], $data);
```

### Kiểm tra sự tồn tại của view

Nếu bạn cần xác định xem một view nào đó có tồn tại hay không, bạn có thể sử dụng `View` facade. Phương thức `exists` sẽ trả về `true` nếu view template tồn tại:

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

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

## Truyền dữ liệu vào view

Như bạn đã thấy trong các ví dụ trước, bạn có thể truyền một mảng dữ liệu đến các view template để cung cấp dữ liệu đó cho view template:

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

Khi truyền thông tin theo cách này, dữ liệu phải là một mảng với các cặp khóa/giá trị. Sau khi cung cấp dữ liệu cho một view template, bạn có thể truy cập từng giá trị trong view template của mình bằng cách sử dụng các khóa của dữ liệu như một biến bình thường, ví dụ như `<?php echo $name ?>`.

Để thay thế cho việc truyền một mảng dữ liệu hoàn chỉnh đến hàm `view`, bạn có thể sử dụng phương thức `with` để thêm các phần dữ liệu riêng lẻ vào view template. Phương thức `with` trả về đối tượng `View` để bạn có thể tiếp tục gọi các phương thức khác của `View` trước khi trả lại view template:

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

### Chia sẽ dữ liệu với tất cả view

Đôi khi, bạn có thể cần chia sẻ dữ liệu với tất cả các view được hiển thị bởi ứng dụng của bạn. Bạn có thể làm như vậy bằng cách sử dụng phương thức `share` của facade `View`. Thông thường, bạn nên gọi phương thức `share` trong phương thức `boot` của service provider. Bạn có thể tự do thêm chúng vào class `App\Providers\AppServiceProvider` hoặc tạo một service provider riêng để chứa chúng:

```php
<?php
namespace App\Providers;
use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    //
  }

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

```
    }
}
```

## View composer

*View composers* là các callback hoặc các phương thức class được gọi khi một view được hiển thị. Nếu bạn có dữ liệu mà bạn muốn liên kết với một view template mỗi khi view template đó được hiển thị, thì một view composer có thể giúp bạn bố trí logic đó vào một vị trí đơn lẻ nào đó. View composer có thể tỏ ra đặc biệt hữu ích nếu cùng một view template được trả về bởi nhiều route hoặc controller khác nhau trong ứng dụng của bạn và luôn cần một phần dữ liệu cụ thể.

Thông thường, các view composer sẽ được đăng ký với một trong các service provider của ứng dụng của bạn. Trong ví dụ này, chúng ta sẽ giả định rằng chúng ta đã tạo một `App\Providers\ViewServiceProvider` mới để chứa logic này.

Chúng ta sẽ sử dụng phương thức `composer` của facade `View` để đăng ký view composer. Laravel không có một thư mục mặc định nào cho các view composer dựa trên class, vì vậy bạn có thể tự do sắp xếp chúng theo cách bạn muốn. Ví dụ: bạn có thể tạo thư mục *app/View/Composers* để chứa tất cả các view composer trong ứng dụng của bạn:

```
<?php

namespace App\Providers;

use App\View\Composers\ProfileComposer;
use Illuminate\Support\Facades\View;
use Illuminate\Support\ServiceProvider;

class ViewServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
```

```
    //
  }

  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
    // Using class based composers...
    View::composer('profile', ProfileComposer::class);

    // Using closure based composers...
    View::composer('dashboard', function ($view) {
      //
    });
  }
}
```

**Chú ý:** Hãy nhớ rằng, nếu bạn tạo một service provider mới để chứa các bản đăng ký view composer của mình, thì bạn sẽ cần thêm service provider vào mảng `providers` trong tệp cấu hình *config/app.php*.

Bây giờ giả định chúng ta đã đăng ký composer, phương thức `compose` của class `App\View\Composers\ProfileComposer` sẽ được thực thi mỗi khi hiển thị hồ sơ. Hãy xem một ví dụ về một class composer:

```
<?php

namespace App\View\Composers;

use App\Repositories\UserRepository;
use Illuminate\View\View;

class ProfileComposer
{
  /**
   * The user repository implementation.
```

```
     *
     * @var \App\Repositories\UserRepository
     */
    protected $users;

    /**
     * Create a new profile composer.
     *
     * @param  \App\Repositories\UserRepository  $users
     * @return void
     */
    public function __construct(UserRepository $users)
    {
        // Dependencies are automatically resolved by the service container...
        $this->users = $users;
    }

    /**
     * Bind data to the view.
     *
     * @param  \Illuminate\View\View  $view
     * @return void
     */
    public function compose(View $view)
    {
        $view->with('count', $this->users->count());
    }
}
```

Như bạn có thể thấy, tất cả các view composer đều được cung cấp thông qua service container, vì vậy bạn có thể khai kiểu cho bất kỳ thư viện nào mà bạn cần trong constructor của class composer.

### Đính kèm một composer vào nhiều view

Bạn có thể đính kèm một view composer vào nhiều view cùng một lúc bằng cách truyền một mảng view template làm đối số đầu tiên cho phương thức `composer`:

```php
use App\Views\Composers\MultiComposer;

View::composer( ['profile', 'dashboard'], MultiComposer::class );
```

Phương thức `composer` cũng chấp nhận ký tự * làm ký tự đại diện, cho phép bạn đính kèm composer vào tất cả các view template:

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

### View creator

View "creator" rất giống với view composer; tuy nhiên, chúng được thực thi ngay sau khi view tempate được khởi tạo thay vì đợi cho đến khi view template sắp render. Để đăng ký một view creator, hãy sử dụng phương thức `creator`:

```php
use App\View\Creators\ProfileCreator;
use Illuminate\Support\Facades\View;

View::creator('profile', ProfileCreator::class);
```

## Tối ưu hóa view

Mặc định, thì các view template của Blade được biên dịch theo yêu cầu. Khi một request được thực hiện nó sẽ hiển thị một view template, Laravel sẽ xác định xem có tồn tại một phiên bản đã biên dịch của view template hay không. Nếu tập tin tồn tại, Laravel sau đó sẽ xác định xem view template chưa biên dịch có được sửa đổi gần đây hơn view template đã biên dịch hay không. Nếu view template đã biên dịch không tồn tại hoặc view template chưa được biên dịch đã được sửa đổi, Laravel sẽ biên dịch lại view template.

Việc biên dịch các view template trong quá trình request có thể có tác động tiêu cực nhỏ đến hiệu suất, vì vậy Laravel cung cấp lệnh Artisan `view:cache` để biên dịch trước tất cả các view template được ứng dụng của bạn sử dụng. Để tăng hiệu suất, bạn có thể muốn chạy lệnh này trong quá trình phân phối ứng dụng của mình:

```
php artisan view:cache
```

Bạn có thể sử dụng lệnh `view:clear` để xóa bộ nhớ cache của view template:

```
php artisan view:clear
```