VIEW

1.1. Giới thiệu

Nhiệm vụ của View là nhận dữ liệu từ Controller và sau đó dựa vào layout của giao diện nó sẽ xử lý dữ liệu theo yêu cầu. Mặc định các view sẽ được đặt trong thư mục `resources/views`. Nếu muốn thay đổi thư mục có thể vào `config/view.php` thay đổi lại giá trị của `paths`.

Mặc định Laravel hỗ trợ view file có phần mở rộng là `.html`, `.css`, `.php`, `.blade.php`. Nếu trong thư mục có cùng các view trùng tên thì Laravel sẽ ưu tiên theo thứ tự như sau: `.blade.php` , `.php`, `.css`, `.html`.

```
<!-- 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`, có thể xem bằng cách sử dụng `view helper` như sau:

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

1.2. Creating & Rendering Views

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 rằng tập tin có chứa `Blade template`. Các mẫu Blade chứa HTML cũng như các chỉ thị Blade cho phép dễ dàng lặp lại các giá trị, tạo câu lệnh rẽ nhánh, câu lệnh lặp và hơn thế nữa.

Tạo view có tên là `demo-view.blade.php` với nội dung như sau:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Demo View</title>
</head>
```

```
<body>
    <h1>View trong Laravel!</h1>
</body>
</html>
```

Để render view trong Laravel có thể sử dụng hàm `view()` hoặc `View::make()` với cú pháp sau:

```
view($view, $data);
//Hoặc
use \Illuminate\Support\Facades\View;
View::make($view, $data);
```

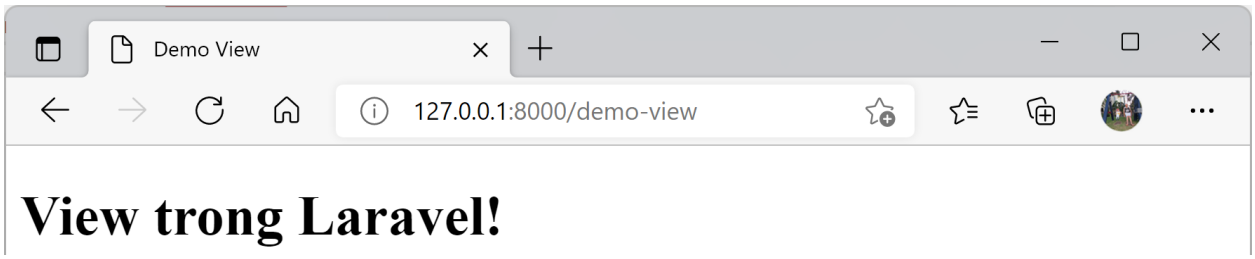
Trong đó:

- `$view` là path đến view (tính từ thư mục `views`).
- `$data` là mảng data bạn muốn truyền vào trong view. Tham số này có thể bỏ trống.

Để render view `demo-view.blade.php` ở trên:

```
//Rendering Views
Route::get('demo-view', function() {
    return view('demo-view');
})->name('demo-view');
```

Kết quả:



1.3. Nested View Directories

Các view cũng có thể được lồng trong các thư mục con của thư mục `resources/views` . Ký hiệu dấu chấm (`.`) được sử dụng để tham chiếu các view lồng nhau.

Ví dụ: Nếu view `demo-view.blade.php` bên trên nằm trong thư mục con `demo` của `views` (`views/demo`).

File `routes/web.php`:

```
//Nested View Directories
Route::get('demo-view', function() {
    return view('demo.demo-view');
})->name('demo-view');
```

1.4. Creating The First Available View

Trong một số trường hợp, nếu muốn render view tồn tại đầu tiên trong danh sách view thì có thể sử dụng phương thức `first` trong `View` object với cú pháp như sau:

```
//use Illuminate\Support\Facades\View;
View::first($views, $data);
```

Trong đó:

- `$views` là mảng path đến view (tính từ thư mục views).
- `$data` là mảng data bạn muốn truyền vào trong view. Tham số này có thể bỏ trống.

File `routes/web.php`:

```
//Creating The First Available View
Route::get('demo-view', function() {
    return View::first(['no_exist', 'demo-view']);
})->name('demo-view');
```

Ở đây global helper function view sẽ trả về một object nên có thể tham chiếu tiếp đến method `first`. Laravel sẽ lần lượt kiểm tra các view này từ trái qua phải, nếu view nào không tồn tại thì nó sẽ bỏ qua. View `no_exist` không tồn tại nên bỏ qua, với đường dẫn `http://localhost:8000/demo-view` thì chúng ta nhận được view `demo-view`.

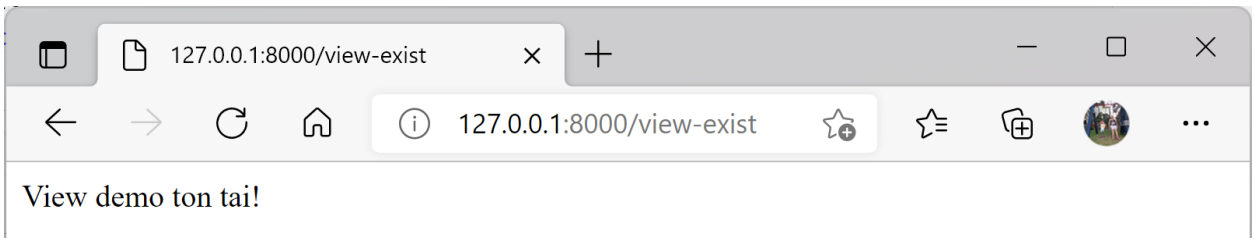
1.5. Determining If A View Exists

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

File `routes/web.php`:

```
//Determining If A View Exists
Route::get('view-exist', function () {
    if (View::exists('demo-view')) {
        return 'View demo ton tai!';
    } else {
        return 'View demo khong ton tai!';
    }
 })->name('view-exist');
```

Kết quả:



1.6. Passing Data To Views

Như trong các ví dụ trước, có thể chuyển một mảng dữ liệu đến các view:

```
//demo-view.blade.php
<body>
    <h1>Truye du lieu vao view trong Laravel!</h1>
    <h2><?=$name?></h2>          //<?=$name?> hoặc
    <h3>{{ $city }}</h3>         //{{ $city }}
</body>
```

File `routes/web.php`:

```
//Passing Data To Views
 Route::get('demo-view', function() {
    return view('demo-view', ['name'=>'NVTeo', 'city'=>'Ho Chi Minh']);
 })->name('demo-view');
```

Kết quả:

1.6.1. Truyền data cho tất cả view

Để truyền dữ liệu cho tất cả các view có trong source code, có thể sử dụng phương thức `share` trong `View` facade. Phương thức này sẽ được khi báo ở `AppServiceProvider.php` tại phương thức `boot` trong thư mục `App\Providers`:

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
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');
    }
}
```

Bây giờ tất cả các file view đều có thể gọi `$key` với giá trị là `value`

Ví dụ:

```
    public function boot() {
        View::share('name', 'Laravel');
    }
```

File `routes/web.php`:

```
//Sharing Data With All Views
Route::get('demo-view', function() {
    return view('demo-view', ['city'=>'Ho Chi Minh']);
})->name('demo-view');
```

Kết quả:

1.6.2. View Composers

View composer là một `class` hoặc một `callback` nó sẽ được gọi khi `render view`. Nếu muốn truyền data vào view khi render thì View composer cũng là một giải pháp.

Trong một số trường hợp nhiều view cần dùng chung một đoạn logic (nhưng không phải tất cả) và truyền vào view thì nên sử dụng `view composer` giúp code đỡ bị lặp lại.

Cần đăng ký `view composer` vào trong `provider`.Có hai cách để khai báo `view composer` đó là dùng `closure` hoặc là `class`.

Đối với `view composer` dùng `closure` thì khai báo theo cú pháp sau:

```
View::composer($views, function ($view) {
    // Code
});
```

Trong đó:

- `$view` là view muốn tác động.
- Nếu muốn tác động đến nhiều view, thì có thể truyền vào một mảng và nếu muốn tác động hết thì chỉ cần khai báo `$view = '*'`.

Ví dụ: Thêm `$title` vào trong view `demo-view`. Sử dụng view composer:

File `demo-view.blade.php`:

```
<body>
     <h1>View composer trong Laravel!</h1>
     <h2>{{ $name }}</h2>
     <h3>{{ $city }}</h3>
     <h4>{{ $title }}</h4>
</body>
```

File `App\Providers\AppServiceProvider`:

```
    public function boot()
    {
        View::share('name', 'Laravel');

        View::composer('demo-view', function ($view) {
            $view->with('title', 'View composer');
        });
    }
```

File routes/web.php:

```
//View Composers
Route::get('demo-view', function() {
    return view('demo-view', ['city'=>'Ho Chi Minh']);
})->name('demo-view');
```

Kết quả:

Trong view demo-view chỉ cần gọi tên biến $title đã được truyền vào từ view composer.

Đối với view composer dùng class thì bạn chỉ việc khai báo theo cú pháp sau:

```
View::composer($views, $className);
```

Trong đó:

- $view thì tương tự như đối với view composer closure.
- $className là tên của class chưa logic composer. Class composer này bắt buộc phải có method compose có visibility là public và nhận tham số truyền vào là một View object.

Đăng ký view composer

Đầu tiên khởi tạo provider ViewComposerProvider với lệnh Artisan sau:

```
php artisan make:provider ViewComposerProvider
```

Liệt kê ViewComposerProvider vừa khởi tạo trong mảng providers ở config/app.php.

```
'providers' => [
    //
    App\Providers\ViewComposerProvider::class,
],
```

Tiếp theo mở file app/Providers/ViewComposerProvider.php và thay đổi nội dung thành:

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

//Thêm 2 dòng
use Illuminate\Support\Facades\View;
use App\Http\View\Composers\ProfileComposer;

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

    /**
     * Bootstrap services.
     *
     * @return void
     */
    public function boot()
    {
        // Registering composer with Class
        View::composer('profile', ProfileComposer::class );
    }
}
```

Mặc định thì Laravel không tạo thư mục chứa các file composer. Có thể tạo cấp thư mục như sau để chứa các composer:

```
app/
├── Http/
|   ├── View/
|   |   ├── Composers/
|   |   |   |   ProfileComposer.php
```

File app/Http/View/Composers/ProfileComposer.php:

```php
<?php
namespace App\Http\View\Composers;
use Illuminate\View\View;
```

```
class ProfileComposer {
    public function compose(View $view) {
        $view->with('title', 'View composer');
    }
```

Bắt buộc trong file composer phải định nghĩa method `compose` và thêm class `Illuminate\View\View`, từ đó mới có thể dùng phương thức `with` để truyền dữ liệu cho `view` đã đăng ký ở ViewComposerProvider.

Cách hoạt động của hình thức `đăng ký composer với class` đó là khi một view tương ứng chuẩn bị render thì sẽ gọi phương thức `compose` của class đã đăng ký chung với view đó. Trong trường hợp trên thì khi `view profile` sắp render thì `ProfileComposer@compose` sẽ được thực thi để truyền các dữ liệu.

File `resources/view/profile.blade.php`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>View Composer</title>
</head>
<body>
    <h1>View Composer trong Laravel!</h1>
    <h2>{{ $title }}</h2>
</body>
</html>
```

Tiếp đó là đăng ký một route để render view `profile`, file `routes/web.php`:

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

Kết quả: