CẤU HÌNH LARAVEL VÀ CẤU TRÚC THƯ MỤC

1. CẤU HÌNH

Tất cả các files cấu hình của Laravel framework nó sẽ được đặt trong thư mục `config`. Với mỗi file trong thư mục đó, có thể chỉnh sửa cấu hình theo ý muốn.

1.1. Cấu hình môi trường

Thông thường, khá là hữu ích khi ứng dụng có giá trị cấu hình khác nhau trên các môi trường khác nhau. Giả sử, cấu hình giá trị cache trên local và trên production khác nhau.

Để làm việc đó, Laravel tận dụng thư viện PHP DotEnv được phát triển bởi Vance Lucas. Khi một ứng dụng mới được cài đặt, tại thư mục gốc sẽ có file `.env.example` file. Nếu cài bằng Laravel Composer, file đấy sẽ tự động đổi tên thành `.env`. Nếu không thì đổi tên file.

1.2. Truy xuất cấu hình môi trường

Tất cả các biến cấu hình sẽ được nạp bởi biến PHP toàn cục `$_ENV` khi ứng dụng nhận request. Tuy nhiên, có thể sử dụng hàm `env` để nhận giá trị cấu hình. Thực tế, nếu xem lại các file cấu hình, sẽ thấy một vài biến đã được sử dụng nó rồi:

```
'debug' => env('APP_DEBUG', false),
```

Giá trị thứ 2 truyền vào hàm `env` là "giá trị mặc định". Giá trị truyền vào sẽ được sử dụng nếu không có biến môi trường nào ứng với key đó.

1.3. Xác định môi trường hiện tại

File `.env` không nên đẩy lên source code ứng dụng, mỗi một developer/server sử dụng ứng dụng có thể có các cấu hình khác nhau.

Nếu ứng dụng phát triển bởi 1 nhóm, có thể tiếp tục lưu lại file `.env.example` trong ứng dụng. Bằng cách đặt các giá trị vào file này, các thành viên khác của nhóm có thể nhìn thấy được các biến môi trường cần thiết để chạy ứng dụng.

Môi trường hiện tại được xác định thông qua biến `APP_ENV` trong file `.env`. Có thể lấy biến này thông qua hàm `environment` trong facade `App.`

```
use Illuminate\Support\Facades\App;
```

```
$environment = App::environment();
```

Ngoài ra có thể truyền tham số vào hàm `environment` để kiểm tra xem môi trường có giống hay không. Hàm này sẽ trả về `true` nếu tham số truyền vào giống với biến môi trường:

```
if (App::environment('local')) {
    // The environment is local
}

if (App::environment(['local', 'staging'])) {
    // The environment is either local OR staging...
}
```

1.4. Truy cập các giá trị cấu hình

Có thể dễ dàng lấy được giá trị cấu hình trong ứng dụng bằng cách sử dụng hàm toàn cục `config`. Để lấy giá trị cấu hình sử dụng "dấu chấm", bắt đầu bằng tên của file và giá trị muốn lấy. Giá trị mặc định có thể được trả vè nếu thông số về biến cấu hình đó không tồn tại:

```
$value = config('app.timezone');

// Retrieve a default value if the configuration value does not exist...
$value = config('app.timezone', 'Asia/Seoul');
```

Để thiết lập giá trị cấu hình lúc thực thi, truyền vào một mảng `config` như sau:

```
config(['app.timezone' => 'America/Chicago']);
```

1.5. Cấu hình Caching

Để tăng hiệu năng cho ứng dụng, nên cache tất cả các file cấu hình bằng cách sử dụng lệnh `config:cache` Artisan command. Lệnh này sẽ gộp tất cả các file cấu hình trong ứng dụng thành một file duy nhất để tăng tốc độ tải của framework.

Nên chạy lệnh `php artisan config:cache` trên production của ứng dụng. Không nên chạy nó ở môi trường phát triển nó vì các thông số cấu hình liên tục thay đổi khi phát triển ứng dụng.

1.6. Chế độ bảo trì

Khi ứng dụng ở chế độ bảo trì, một giao diện sẽ được hiển thị cho tất cả request vào ứng dụng. Để "vô hiệu hóa" ứng dụng trong khi đang cập nhật hoặc bảo trì. Việc bảo trì nằm trong Stack Middleware cho ứng dụng. Nếu ứng dụng ở chế độ bảo trì, `MaintenanceModeException` sẽ bắn ra một status code là 503.

Để bật chế độ bảo trì, bạn chỉ cần thực thi lệnh `down`:

```
php artisan down
```

Nếu bạn muốn `Refresh` HTTP header được gửi cùng với tất cả các phản hồi của chế độ bảo trì, có thể cung cấp tùy chọn `refresh` khi gọi lệnh down. The `Refresh` header sẽ hướng dẫn trình duyệt tự động làm mới trang sau số giây được chỉ định:

```
php artisan down --refresh=15
```

Có thể cung cấp tùy chọn retry cho lệnh down , lệnh này sẽ được đặt làm giá trị của Retry-After HTTP header, mặc dù các trình duyệt thường bỏ qua tiêu đề này:

```
php artisan down --retry=60
```

1.7. Tắt chế độ bảo trì

Để vô hiệu hóa chế độ bảo trì, sử dụng lệnh `up`:

```
php artisan up
```

1.8. Bỏ qua chế độ bảo trì

Trong các bản phát hành trước đây của Laravel, tính năng chế độ bảo trì php artian down có thể bị bỏ qua bằng cách sử dụng "danh sách cho phép" các địa chỉ IP được phép truy cập ứng dụng. Tính năng này đã bị loại bỏ để chuyển sang giải pháp "`secret`" / "mã thông báo" đơn giản hơn. Khi ở chế độ bảo trì, có thể sử dụng tùy chọn secret để chỉ định mã thông báo bỏ qua chế độ bảo trì:

```
php artisan down --secret="1630542a-246b-4b66-afa1-dd72a4c43515"
```

Sau khi đặt ứng dụng ở chế độ bảo trì, có thể điều hướng đến URL ứng dụng khớp với mã

thông báo này và Laravel sẽ đưa ra cookie bỏ qua chế độ bảo trì cho trình duyệt:

```
https://example.com/1630542a-246b-4b66-afa1-dd72a4c43515
```

Khi truy cập vào tuyến đường ẩn này, sau đó sẽ được chuyển hướng đến / tuyến đường của ứng dụng. Khi cookie đã được cấp cho trình duyệt, sẽ có thể duyệt ứng dụng bình thường như thể nó không ở chế độ bảo trì.

1.9. Trang Pre-Rendering trong chế độ bảo trì

Trong các phiên bản Laravel 7 trở về trước, nếu chạy `php artisan down` để đưa ứng dụng vào chế độ bảo trì trong deploy và sau đó chạy Composer như một phần của quá trình deploy này, ứng dụng có thể sẽ gặp lỗi trong khi các dependency được thay đổi và file autoload được ghi, nghĩa là người dùng cuối sẽ nhìn thấy một trang lỗi, thay vì trang chế độ bảo trì như mong đợi. Vấn đề này đã được giải quyết trong Laravel 8, có thể truyền tên của một view cho flag "`render`" như một phần của lệnh `artisan down`. Ví dụ:

```
php artisan down --render="errors::503"
```

Laravel sẽ pre-render view này (trong trường hợp này là `error/503.blade.php`) và sau đó đưa ứng dụng vào chế độ bảo trì. Sau đó, nếu bất kỳ ai cố gắng truy cập vào trang web, họ sẽ thấy view này. Laravel thậm chí sẽ không cố tải file autoload của Composer, nghĩa là lỗi sẽ không được ném ra.

1.10. Redirecting Maintenance Mode Requests

Trong khi ở chế độ bảo trì, Laravel sẽ hiển thị chế độ xem chế độ bảo trì cho tất cả các URL ứng dụng mà người dùng cố gắng truy cập. Nếu muốn, có thể hướng dẫn Laravel chuyển hướng tất cả các yêu cầu đến một URL cụ thể. Điều này có thể được thực hiện bằng cách sử dụng tùy chọn redirect. Ví dụ: có thể muốn chuyển hướng tất cả các yêu cầu đến / URI:

```
php artisan down --redirect=/
```

2. CẤU TRÚC ỨNG DỤNG LARAVEL

2.1. Giới thiệu

Cấu trúc ứng dụng Laravel mặc định nhằm cung cấp một điểm khởi đầu tuyệt vời cho cả các ứng dụng lớn và nhỏ. Nhưng có thể tự do sắp xếp ứng dụng của mình theo cách mong muốn.

Laravel hầu như không áp đặt bất kỳ hạn chế nào đối với vị trí của bất kỳ lớp nhất định nào, miễn sao các Class tuân thủ nguyên tắc autoload của Composer.

2.2. Cấu trúc ứng dụng

Thư mục App

Thư mục app chứa tất cả các Class của project.

Thư mục app/Console

Thư mục chứa các tập tin định nghĩa các câu lệnh trên artisan.

Thư mục app/Exceptions

Thư mục chứa các tập tin quản lý, điều hướng lỗi.

Thư mục app/Http/Controllers

Thư mục chứa các controller của project.

Thư mục app/Http/Middleware

Thư mục chứa các tập tin lọc và ngăn chặn các requests.

Thư mục app/Providers

Thư mục chứa các file thực hiện việc khai báo service và bind vào trong Service Container.

Thư mục app/Models

Thư mục chứa các model của project (Laravel 8 sẽ có sẵn thư mục Models).

Thư mục bootstrap

Thư mục `bootstrap` chứa những file khởi động của Frameword và những file cấu hình autoloading. Ngoài ra nó còn có thư mục `cache` chứa những file mà framework sinh ra để tăng hiệu năng như route và services cache files.

Thư mục config

Thư mục chứa tất cả những file cấu hình.

Thư mục database

Thư mục chứa 2 thư mục migration (tạo và thao tác database) và seeds (tạo dữ liệu mẫu). Nếu muốn, có thể sử dụng nó để tổ chức một cơ sở dữ liệu SQLite.

Thư mục database/factories

Thư mục chứa các file định nghĩa các cột bảng dữ liệu để tạo ra các dữ liệu mẫu.

Thư mục database/migrations

Thư mục chứa các file tạo và chỉnh sửa dữ liệu.

Thư mục database/seeds

Thư mục chứa các file tạo dữ liệu thêm vào CSDL.

Thư mục public

Thư mục chứa file index.php giống như cổng cho tất cả các request vào project. Ngoài ra nó còn chứa một số tài nguyên như ảnh, JavaScript, và CSS.

Thư mục resources

Thư mục chứa những file view và raw, các file biên soạn như LESS, SASS, hoặc JavaScript. Ngoài ra còn chứa tất cả các file ngôn ngữ trong project.

Thư mục resources/views

Thư mục chứa các file view xuất giao diện người dùng.

Thư mục routes

Thư mục chứa tất cả các điều khiển route (đường dẫn) trong project.

Chứa các file route sẵn có: web.php, channels.php, api.php, và console.php

Thư mục routes/api.php

Cấu hình các route liên quan đến API.

Thư mục routes/web.php

Cấu hình các route liên quan đến web (Có giao diện người dùng).

Thư mục storage

Thư mục chứa các file biên soạn blade templates của bạn, file based sessions, file caches, và những file sinh ra từ project.

- Thư mục app, dùng để chứa những file sinh ra từ project.
- Thư mục framework, chứa những file sinh ra từ framework và caches.
- Thư mục logs, chứa những file logs.

- Thư mục /storage/app/public, lưu những file người dùng tạo ra như hình ảnh.

Thư mục tests

Thư mục chứa những file tests.

Thư mục vendor

Thư mục chứa các thư mục, file thư viện của Composer.

File .env

File chứa các config chính của Laravel.

File artisan

File thực hiện lệnh của Laravel.

File .gitattributes, .gitignore

File dùng để xử lý git.

File composer.json, composer.lock, composer-setup.php

File sinh ra của composer.

File package.json

File chứa các package cần dùng cho projects.

File phpunit.xml

File phpunit.xml, xml của phpunit dùng để testing project.

File webpack.mix.js

File dùng để build các webpack.