TRƯỜNG ĐẠI HỌC SƯ PHẠM
Thành phố Hồ Chí Minh
HCMC University of Education

CÔNG NGHỆ WEB

Đề tài: Tìm hiểu Laravel

Giảng viên: ThS. Trần Phước Tuấn

Nhóm thực hiện:

1. Trần Trung Hiếu – 44.01.104.087
2. Trương Văn Nam – 44.01.104.144
3. Trần Phước Lộc – 44.01.104.130

Mục lục

I. Giới thiệu

1. Laravel là gì?

Laravel là một PHP framework, có mã nguồn mở và miễn phí, được xây dựng nhằm hỗ trợ phát triển các phần mềm, ứng dụng, theo kiến trúc MVC. Hiện nay, Laravel đang là PHP framework phổ biến nhất và tốt nhất.

Những lý do khiến Laravel trở nên rộng rãi:

- Cú pháp dễ hiểu – rõ ràng
- Hệ thống đóng gói modular và quản lý gói phụ thuộc
- Nhiều cách khác nhau để truy cập vào các cơ sở dữ liệu quan hệ
- Nhiều tiện ích khác nhau hỗ trợ việc triển khai vào bảo trì ứng dụng.

Framework là các đoạn code đã được viết sẵn, cấu thành nên một bộ khung và các thư viện lập trình được đóng gói. Chúng cung cấp các tính năng có sẵn như mô hình, API và các yếu tố khác để tối giản cho việc phát triển các ứng dụng web phong phú, năng động. Các *framework* giống như là chúng ta có khung nhà được làm sẵn nền móng cơ bản, bạn chỉ cần vào xây dựng và nội thất theo ý mình.

2. Lịch sử phát triển của Laravel

Phiên bản đầu tiên của *Laravel* được Taylor Otwell tạo ra vào tháng 6 năm 2011 như một giải pháp thay thế cho *CodeIgniter*, cung cấp nhiều tính năng quan trọng hơn như xác thực và phân quyền. Với framework này, lập trình viên được hỗ trợ nhiều tính năng mới mẻ, hiệu quả và dễ thực hiện hơn.

Tháng 8/2014, *Laravel* đã trở thành project PHP phổ biến nhất và được theo dõi nhiều nhất trên Github. Và đến nay (10/2020) *Laravel* vẫn luôn đạt được vị trí quán quân cho PHP Framework phổ biến nhất.

3. Request Lifecycle

Khi sử dụng bất kỳ công cụ nào, chúng ta đều cảm thấy tự tin hơn nếu chúng ta có thể hiểu cách công cụ đó hoạt động. Việc phát triển ứng dụng cũng vậy. Khi bạn hiểu các công cụ phát triển hoạt động như thế nào, bạn cảm thấy thoải mái và tự tin hơn khi sử dụng chúng.

First Things

Giống như mọi loại framework khác, mọi request của một ứng dụng Laravel đều được bắt đầu từ file `public/index.php`. Tất cả các request chuyển đến file này

đều được chuyển đến từ Web Server của bạn (`Apache/Nginx`). Trong file `index.php` sẽ không bao gồm nhiều đoạn code. Thay vào đó nó là điểm khởi đầu để loading phần còn lại của framework.

Nhìn vào source code của file này chúng ta sẽ thấy nó sẽ load ra 3 phần:

- `bootstrap/autoload.php` ==> Khởi tạo Auto Loader để load ra các package được install từ `composer`
- `bootstrap/app.php` ==> Khợi tạo bộ khung làm việc của framwork bao gồm
    - `Http Kernel` => xử lý các request từ brower lên server
    - `Console Kernel` => xử lý các request từ các lệnh comand (Vd: `php artisan`, `php artisan migrate`, .v.v....v)
    - `ExceptionHandler` => xử lý lỗi
- Khởi tạo ra `Kernel` để làm tiếp tục đi tiếp

HTTP/Console Kernels

Tiếp theo, các request sẽ được gửi đến `Http Kernel` hoặc `Console Kernel` tùy thuộc vào loại yêu cầu đang cập nhật ứng dụng (Dùng browser hay command). 2 `Kernel` này sẽ như là trung tâm điều khiển mà tất cả các yêu cầu sẽ phải đi qua tuy nhiên trong bài viết này chúng ta hãy chỉ tập trung vào tìm hiểu `Http Kernel` thôi nhé. Vị trí của nó nằm ở `app/Http/Kernel.php`.

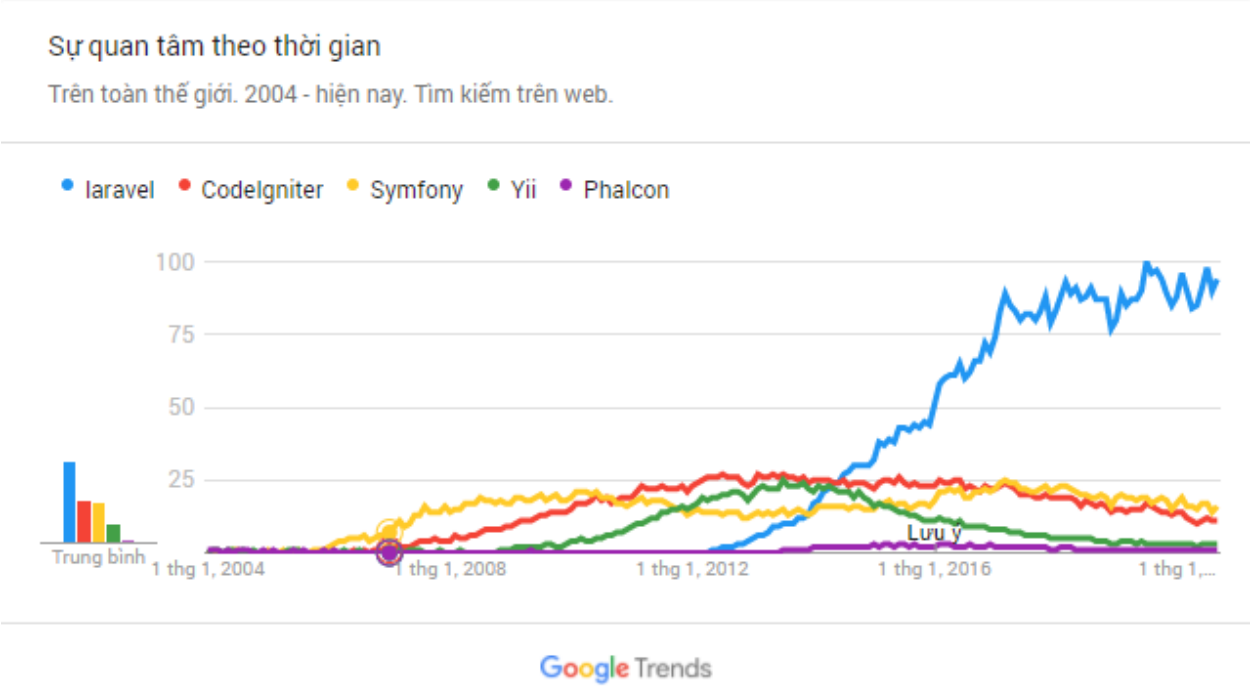
- `Http Kernel` được extends từ class `Illuminate\Foundation\Http\Kernel`. Nó định nghĩa một mảng `bootstrapper` sẽ được chạy trước khi request được thực thi. Những bootstrappers cấu hình việc xử lý lỗi, cấu hình `logs`, phát hiện môi trường ứng dụng (`develop`, `staging`, 'production`) và thực hiện các tác vụ khác cần được thực hiện trước khi request thực sự được xử lý.
- `Http Kernel` cũng định nghĩa một danh sách HTTP middleware mà tất cả các `request` phải đi qua trước khi được ứng dụng xử lý. Các HTTP middleware này xử lý việc đọc và ghi HTTP session, xác định xem ứng dụng có đang ở chế độ bảo trì hay không, xác minh token CSRF và hơn thế nữa.

- Hãy suy nghĩ về `Http Kernel` như là một trung tâm điều hành đại diện cho toàn bộ ứng dụng của bạn. Cung cấp cho nó các request HTTP và nó sẽ trả về các response HTTP.

Service Providers

- Một trong những hành động khởi động hạt nhân quan trọng nhất là tải các Service Providers cho ứng dụng của bạn.
- Tất cả các nhà cung cấp dịch vụ cho ứng dụng được cấu hình trong `config/app.php` trong mảng `providers`. Đầu tiên, phương thức `register` sẽ được gọi trên tất cả các `providers` sau đó, khi tất cả các `providers` đã được đăng ký, phương thức `boot` sẽ được gọi để nạp tất cả các `providers`
- Các `Service Providers` chịu trách nhiệm khởi động tất cả các thành phần khác nhau của Framework như: `database`, `queue`, `validation` và các thành phần định tuyến (`routing`). Vì chúng khởi động và cấu hình mọi tính năng được cung cấp bởi framework nên Service Providers là thành phần quan trọng nhất của toàn bộ quá trình khởi động Laravel.

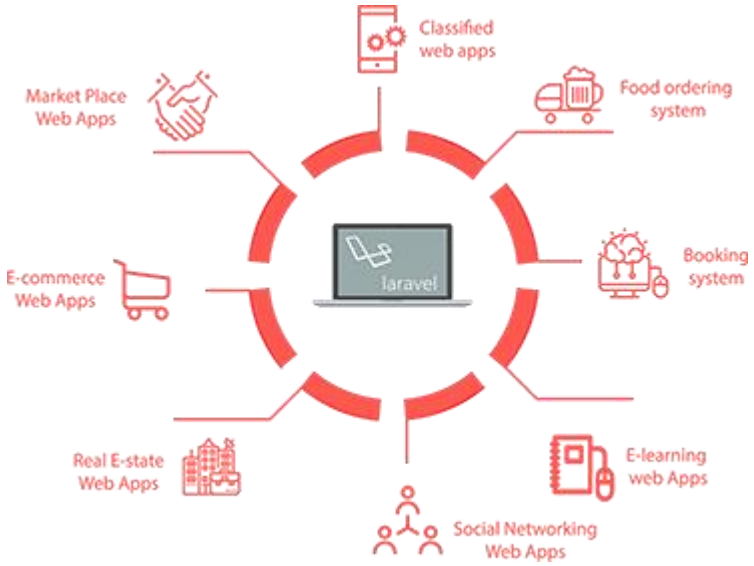
Dispatch Request



Khi ứng dụng đã được khởi động xong và tất cả các `Service Providers` đã được khởi tạo thành công. Các `Request` sẽ được đưa đến các bộ định tuyến (`router`). Ở đây, các yêu cầu sẽ được được kiểm tra xem có tồn tại hay không, có cần chạy qua HTTP middleware đặc biệt nào hay không và cuối cùng sẽ được điều chuyển đến các `Controller` hoặc `router` (Các function tự định nghĩ trong router).Sự phổ biến của *Laravel* so với các nền tảng khác.

II. Ưu điểm và nhược điểm của Laravel

1. Ưu điểm



Dễ sử dụng

Một trong những lý do khiến *Laravel* nhanh chóng được cộng đồng người dùng đón nhận và sử dụng nhiều là do nó rất dễ để có thể sử dụng. Ngay cả khi bạn chỉ mới chỉ có những kiến thức cơ bản nhất về lập trình web với PHP, thì chỉ mất vài giờ là bạn có thể bắt tay vào việc làm một project nhỏ với *Laravel*.

Mã nguồn mở

Laravel framework được xây dựng với mã nguồn mỡ và hoàn toàn miễn phí. Do đó, bạn không cần phải quan tâm đến việc trả thêm phí khi mở rộng ứng dụng hay trang web của mình. Mỗi lần nền tảng này được cập nhật, bạn lại có cơ hội khám phá thêm nhiều tính năng độc đáo và ứng dụng vào các sản phẩm công nghệ mà mình đang triển khai.

Được xây dựng theo đúng chuẩn WVC

WVC là tiêu chuẩn thiết kế web, bất kì website nào được đánh giá theo đúng chuẩn này. thì đều sẽ hoạt động tốt và ít nhiều mang lại những hiệu quả thực sự dành cho công ty, đơn vị sở hữu nó.

Có tích hợp sẵn nhiều tính năng

Bản thân *Laravel* đã cung cấp cho người dùng rất nhiều các nhóm tính năng giúp quá trình phát triển trở nên nhanh chóng hơn rất nhiều lần.

Module đa dạng

Laravel được xây dựng dựa trên hơn 20 thư viện khác nhau. Hiểu được cách thiết kế framework khiến các lập trình hoàn toàn có thể đóng góp cho framework cũng như mở rộng chúng một cách dễ dàng.

Tính bảo mật cao

Theo ý kiến của nhiều chuyên gia, *Laravel Framework* được đánh giá là có độ bảo mật cao hơn nền tảng *WordPress*. Để giúp lập trình viên có thể tối đa thời gian tập trung vào việc phát triển các tính năng mình cần, *Laravel* đã cung cấp sẵn cho người dùng các tính năng bảo mật cơ bản như:

- ORM của *Laravel* sử dụng PDO thay vì mysqli để chống lại tấn công SQL Injection.
- *Laravel* sử dụng một field token ẩn để chống lại tấn công kiểu CSRF.
- Các biến được đưa ra view mặc định đều được *Laravel* escape để tránh tấn công XSS

Do đó, khi thiết kế web với *Laravel*, bạn không cần quá lo lắng về khả năng bảo mật hay mất nhiều thời gian để cài đặt hay tối ưu thêm cho tính năng này. Tất cả đã có sẵn với *Laravel*.

Cộng đồng người dùng rộng lớn

Như nhiều nền tảng mã nguồn mở khác, *Laravel Framwork* cũng có cộng đồng người dùng rộng rãi và sẵn sàng hỗ trợ bạn trong quá trình thiết lập và vận hành dự án. Đặc biệt việc fix bug hay tìm lỗi trở nên nhanh chóng, dễ dàng và tiết kiệm thời gian hơn rất nhiều.

2. Nhược điểm

Không hỗ trợ tính năng thanh toán

Sẽ không đáng lo ngại nếu bạn không tự mình quản lý các khoản thanh toán, bởi vì bạn sẽ phải tuân thủ các quy tắc tuân thủ PCI. Trì hoãn các dịch vụ như Stripe và Paypal sẽ giải quyết vấn đề đó. Bạn cũng có thể thử bất kỳ trang web thương mại trực tuyến nào và xây dựng ứng dụng của mình trong kho template có sẵn, hoặc sử dụng các thư viện của Framework cho phép bạn tích hợp các phương thức thanh toán. Tuy nhiên, hầu hết các nhà giao dịch điện tử nên được nhúng bộ xử lý thanh toán của bên thứ ba nhằm thuận tiện hơn.

Thiếu sự liên tục giữa các phiên bản

Không có chuyển đổi liền mạch giữa các phiên bản Laravel. Nếu cố cập nhật code, bạn có thể sẽ phá vỡ ứng dụng.

Chất lượng

Một số thành phần trong framework không được thiết kế tốt. Ví dụ, dependency injection đôi khi trở nên phức tạp không cần thiết. Các tài liệu khá nặng. Bạn phải học hỏi nhiều trước khi bắt đầu xây dựng các ứng dụng.

Do đó, đây không phải là một lựa chọn tốt cho các nhà phát triển nghiệp dư. Tuy nhiên, framework vẫn đang được cải thiện rất nhiều. Phiên bản 5 đã tốt hơn nhiều với số lượng sai sót cũng ít hơn.

Một số nâng cấp có thể có vấn đề

Đây không chỉ là vấn đề của Laravel mà là của các PHP framework. Vì vậy, các nhà phát triển nên có biện pháp phòng ngừa trước khi nâng cấp mobile application/website.

Thường không cung cấp sự phong phú cho ứng dụng di động

Việc tải lại toàn trang có thể hơi nặng trong các mobile app khi so sánh với các website. Trong những trường hợp như vậy, các nhà phát triển web có xu hướng chỉ sử dụng framework như backend JSON API.

III. Cài đặt

Có rất nhiều cách để cài đặt *Laravel framework*, nhưng nhóm chọn cách 2 dưới đây vô cùng đơn giản, nhanh chóng cho những người mới bắt đầu.

Trước tiên bạn cần phải cài đặt:

- Composer
- Xampp (các bạn có thể chọn phần mềm khác)
- Một phần mềm để chạy command line (thường là Command Prompt của Windows hay Terminal của Mac...)

***Lưu ý**: PHP nên cài đặt ở phiên bản 7+ sẽ giúp ứng dụng Laravel chạy nhanh hơn.*

Những thứ trên bạn chỉ cần download và install khá đơn giản.

Bây giờ chúng ta sẽ đi qua từng cách cài đặt Laravel.

1. Với Laravel Installer (Via Laravel Installer)

   Đầu tiên, ta phải download *Laravel Installer* thông qua *Composer* với lệnh:

   ```
   composer global laravel/installer
   ```

   Tiếp theo, gõ tiếp lệnh bên dưới:

   ```
   laravel new blog
   ```

   Sau khi cửa sổ lệnh báo hoàn tất thì chúng ta đã khởi tạo thành công một project với tên *"blog"*.

2. Với Composer Create-Project (Via Composer Create-Project)

   Dùng command line `cd` tới thư mục muốn tạo project. Sau đó gõ dòng lệnh bên dưới và đợi ít thời gian.

   ```
   composer create-project --prefer-dist laravel/laravel blog
   ```

   Sau khi lệnh hoàn tất, chúng ta cũng thu được một kết quả tương tự như cách ở trên.

   Qua một trong hai cách trên, chúng ta đã có thể khởi tạo một ứng dụng *Laravel framework* rồi, tiếp theo chúng ta sẽ tìm hiểu cách để khởi động ứng dụng.

IV. Nạp máy chủ dành cho phát triển (Load local development server)

Nếu máy tính bạn đã cài đặt PHP thì có thể chạy dòng lệnh sau để khởi động server:

```
php artisan serve
```

Sau khi chạy dòng lệnh, mở trình duyệt và truy cập địa chỉ http://localhost:8000, chúng ta sẽ thu được kết quả như hình bên dưới:

Mặc định, Laravel khi khởi động chạy ở port 8000, nếu muốn thay đổi thiết lập này có thể thêm tham số `port` vào lệnh:

```
php artisan serve --port=8080
```

V. Một số tính năng nổi bật của Laravel

1. Migration trong Laravel

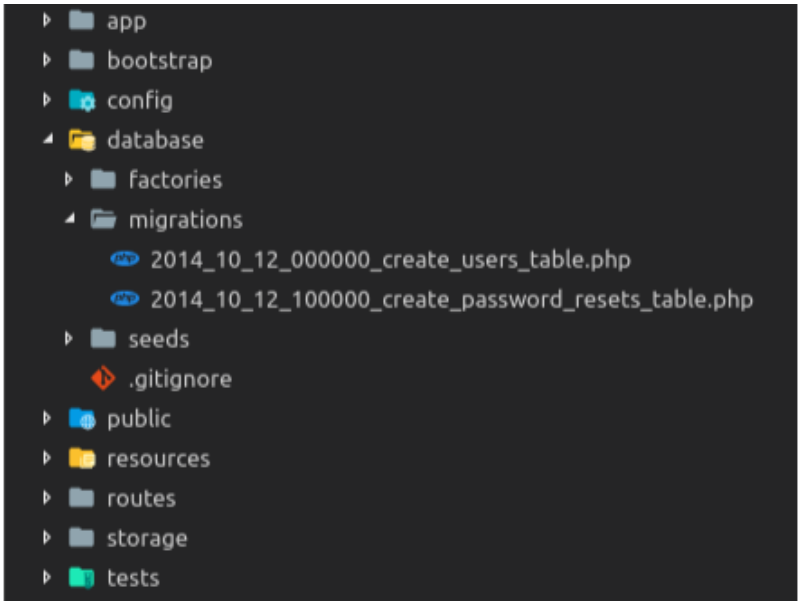
Migration giống như một hệ thống quản lý phiên bản giống như Git nhưng dành cho cơ sở dữ liệu của bạn. Migration cho phép bạn định nghĩa các bảng trong CSDL, định nghĩa nội dung các bảng cũng như cập nhật thay đổi các bảng đó hoàn toàn bằng PHP. Đồng thời các thao tác với CSDL này còn có thể sử dụng trên các loại CSDL khác nhau như MySQL, SQL Server, Postgres,... mà không cần phải chỉnh sửa lại code theo CSDL sử dụng.
Điều kiện tiên quyết để chạy migration một cách thành công:

- Phải có kết nối với database
- Migrations muốn sử dụng được thì phải nằm trong thư mục App\database\migrations

1.1 Tạo Migration

Để tạo một migration chúng ta có 2 cách để tạo, một là chúng ta sẽ vào *database/migrations* tạo một file mới trong đó. Nhưng cách này thường mọi người không dùng nhiều lắm, mọi người hay dùng cách thao tác với command line.

Để tạo một migration ta sẽ sử dụng câu lệnh `make:migration` File migration mới sẽ được đặt trong thư mục *database/migrations*



Mỗi file migration được đặt tên bao gồm timestamp để xác định thứ tự migartion với nhau. Ví dụ để tạo mới một bảng posts mới trong CSDl ta sẽ sử dụng câu lệnh sau:

```
php artisan make:migration create_table_posts_table --create=posts
```

Khi ta nhấn enter thì ngay lập tức trong file sẽ được tạo trong folder database/migration và nội dung của file sẽ như sau:

```
    <?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
```

```
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('posts');
    }
}
```

Để thêm các trường của bảng posts trong CSDL ta sẽ thêm vào `function up()` trong file migration.

```
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('slide_url');
            $table->string('title');
            $table->string('content');
            $table->timestamps();
        });
    }
```

Sau đó để tạo bảng trong CSDL thì chúng ta sẽ dùng câu lênh: `php artisan migrate`

Bây giờ một vấn đề phát sinh la khi chúng ta nhanh tay migrate mà sực nhớ ra là chúng ta thiếu trường **status** của bảng posts. Đừng lo, chúng ta chỉ cần tạo một file migration mới với tham số trong command là --table=posts. Lệnh sẽ là:

```
php artisan make:migration add_attribute_status_into_posts_table
--table=posts
```

Và sau đó chúng ta migrate như bình thường.

1.2 Rollback/Migrate trong một câu lệnh

Câu lệnh `php artisan migrate:refresh` sẽ đầu tiên rollback lại toàn bộ migration của chương trình, và thực hiện câu lệnh migrate. Tức là thực hiện *function down()* xong rồi thực hiện *function up()* trong file migration.
Câu lệnh sẽ thực hiện tái cấu trúc toàn bộ database:
`php artisan migrate:refresh`
`php artisan migrate:refresh –seed`
Tham số `--seed` là để chạy tất cả các Seeder chúng ta sẽ tìm hiểu ở phía dưới.
Nếu muốn thực hiện *function down()* với số lượng batch muốn rollback thì ta dùng câu lệnh: `php artisan migrate:rollback -step=n`

1.3 Schema

Trong file migration để dùng `Schema` thì chúng ta sẽ `use Illuminate\Support\Facades\Schema`.
Nếu muốn tạo một bảng mới trong DB của mình thì chúng ta có thể sử dụng

```
Schema::create('users', function (Blueprint $table) {
    $table->increments('id');
});
```

Nếu các bạn muốn kiểm tra xem `table` hoặc `column` có tồn tại hay không thì ta dùng

```
    if (Schema::hasTable('users')) {
        //
    }

    if (Schema::hasColumn('users', 'email')) {
        //
    }
```

Nếu muốn đổi tên bảng từ `post` sang `posts` thì ta dùng

```
    Schema::rename('post', 'posts')
```

Khi chúng ta muốn xóa bảng thì có thể sử dụng `Schema::drop()`

```
Schema::drop('users');


Schema::dropIfExists('users');
```

Các kiểu column

Tìm hiểu thêm tại: https://laravel.com/docs/8.x/migrations

Column modifier

Nhiều khi chúng ta có thể muốn cột trong table có giá trị `null` hay `not null`, vì thế Laravel hỗ trợ các modifier
Ví dụ như ta muốn để cột `address` trong table `users` được phép `null`

```
Schema::table('users', function ($table) {
    $table->string('address')->nullable();
});
```

Và còn rất nhiều modifier các bạn có thể tham khảo thêm tại:
https://laravel.com/docs/8.x/migrations

Modifying columns

Các bạn có thể mở terminal lên và cài thư viện `doctrine` để có thể sử dụng các hàm hữu ích trong nó.`composer require doctrine/dbal`
Chúng ta thường hay mắc phải sau khi migrate bảng rồi lại sực nhớ ra mình lại không muốn đặt tên cột như thế nữa. Để giải quyết vấn đề đó , thư viện `doctrine` có hàm xử lý được.

```
Schema::table('users', function ($table) {
    $table->renameColumn('from', 'to');
});
```

Hay một vấn đề nữa đó chính là mình muốn giới hạn giá trị kiểu dữ liệu của một cột

```
Schema::table('users', function ($table) {
    $table->string('name', 50)->nullable()->change();
```

```
});
```

Foreign Key Constraints

Đôi khi chúng ta muốn tạo các rằng buộc cho các bảng, chúng ta có thể sử dụng cú pháp sau để rằng buộc cho 2 bảng:

```
Schema::table('posts', function ($table) {
    $table->integer('user_id')->unsigned();

    $table->foreign('user_id')->references('id')->on('users');
});
```

Chú ý nếu không migrate mà không chạy được thì các bạn có thể tách ra làm 2 file migration để chạy.
Để drop một foreign ta dùng : `$table->dropForeign('posts_user_id_foreign');`
Chúng ta nên để ý quy tắc đặt tên foreign `<tên_table>_<tên_khóa_ngoại>_foreign`
Bạn có thể kích hoạt hay bỏ kích hoạt việc sử dụng foreign key constraint trong migration sử dụng hai hàm sau:

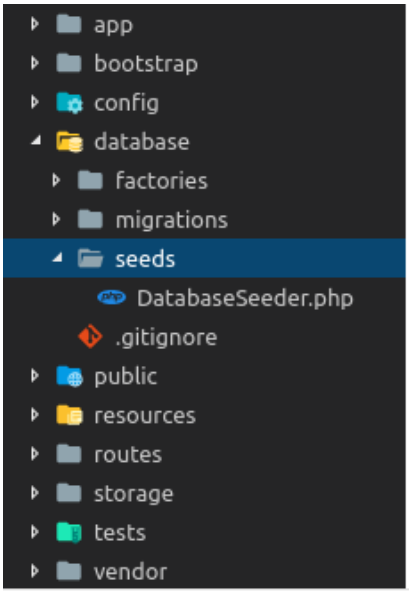
```
Schema::enableForeignKeyConstraints();

Schema::disableForeignKeyConstraints();
```

2. Seeder trong Laravel

Thay vì cách nhập tay thủ công thông thường vừa tốn thời gian vừa tốn công thì Laravel hỗ trợ Seeder để nhanh chóng tạo dữ liệu, để phát triển tính năng của chúng ta.

2.1 Cấu trúc thư mục

Tất cả các seed class được đặt trong thư mục `database/seeds`. trong thư mục này đã có sẵn file `DataTableSeeder.php`



Mặc định trong file này sẽ có class DatabaseSeeder có function `run()` call tới các seed class khác, cho phép bạn có thể control được thứ tự thêm dữ liệu theo thứ tự bạn sắp xếp.

2.2 Các cách insert dữ liệu bằng Seeder

Có rất nhiều cách để thêm dữ liệu bằng seeder, sau đây nhóm xin giới thiệu về một trong những cách đó.

Viết thẳng trong file DataTableSeeder.php

Chúng ta có thể viết trực tiếp trong function `run()`

```php
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
```

```
        DB::table('categories)->insert([
            ['name' => 'Điện thoại'],
            ['name' => 'Laptop'],
            ['name' => 'Tivi'],
        ]);
    }
}
```

hoặc chúng ta cũng có thể tách riêng ra 1 class và sử dụng hàm hàm `call()` trong function `run()`.

Lúc này xong, chúng ta sẽ chạy câu lệnh để thực thi chèn dữ liệu vào bảng `catgories`: `php artisan db:seed`

Sử dụng Model Factories

Để sinh ra một lượng dữ liệu lớn như 100 records chúng ta sẽ sử dụng Model Factories.

Ví dụ đặt ra là chúng ta cần tạo **1000 records** trong bảng users. Trong folder database ta sẽ thấy còn có một folder nữa là factories, bên trong folder này có 1 file tên là UserFactory.php dùng để định nghĩa cấu trúc của dữ liệu mẫu mà ta

muốn thêm vào CSDL theo từng bảng. Trong folder đó có chứa sẵn UserFactory.php

Bây giờ chúng ta tạo 1 file giống ý hệt file này nhưng đổi tên thành `SupplierFactory.php` hoặc có thể sử dụng terminal để tạo `php artisan make:factory SupplierFactory`

```
<?php

use Faker\Generator as Faker;

$factory->define(App\Models\Supplier::class, function (Faker
$faker) {
    return [
        'name' => $faker->name,
        'address' => $faker->address,
        'phone' => $faker->phoneNumber,
        'email' => $faker->unique()->safeEmail,
        'created_at' => new DateTime,
        'updated_at' => new DateTime,
    ];
});
```

Thư viện `Faker` là một thư viện fake data rất hữu ích, chúng ta không cần phải nghĩ ra những địa chỉ, những số điện thoại hay email để thêm vào table. Điều đó đã được thư viện `Faker` hỗ trợ hết.
Tiếp theo chúng ta sẽ sẽ tạo file seeder : `php artisan make:seed SupplierTableSeeder`, sau khi ấn enter một file SuplierTableSeeder sẽ được sinh ra trong folder database/seeds.

Trong function `run()` chúng ta sẽ sử dụng hàm của helper trong Laravel là hàm `factory()`. Đối số thứ nhất truyền vào class Model, đối số thứ hai truyền vào số lượng bản ghi muốn sinh ra.
Giờ chúng ta vào file `DatableSeeder.php` trong hàm `run()`:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
```

```
     * @return void
     */
    public function run()
    {
        $this->call(SuppliersTableSeeder::class);
    }
}
```

Và cuối cùng chạy lệnh `php artisan db:seed`
Hoặc chúng ta cũng có thể chạy file `SupplierTableSeeder.php` bằng câu lệnh: `php artisan db:seed --class=SupplierTableSeeder`

Chú ý!

Để chạy nhiều Seeder ta sẽ tạo nhiều file Seeder rồi gọi chúng trong function`run()` ở file `DataTableSeeder.php`:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UsersTableSeeder::class);
        $this->call(RolesTableSeeder::class);
        $this->call(CategoriesTableSeeder::class);
        $this->call(SuppliersTableSeeder::class);
        $this->call(ProductsTableSeeder::class);
        $this->call(ImagesTableSeeder::class);
    }
}
```

Các bạn hãy chú ý table nào cần thêm dữ liệu trước để sắp xếp thứ tự trong function `run()` nhé.

3. Eloquent Model

ORM(Object Relational Mapping) đây là tên gọi chỉ việc ánh xạ các record dữ liệu trong hệ quản trị cơ sở dữ liệu sang dạng đối tượng mà mã nguồn đang định

dạng trong class. **Eloquent ORM:** Larvel đã sử dụng kỹ thuật ORM giúp lập trình viên thao tác dễ dàng hơn với DB. Trong phần này chúng ta sẽ nói nhiều đến phần kiến thức `Eloquent Model` (Model) - là một phần trong mô hình MVC ở trên. Các Model này sẽ thao tác trực tiếp với DB, xử lý logic nghiệp vụ và trả về dữ liệu cho controller.

3.1 Định nghĩa model và các thiết lập cơ bản

Chúng ta sẽ định nghĩa ra model bằng câu lệnh `make:model` trong command Artisan.

```
php artisan make:model Post
```

Sau khi nhấn lệnh thì trong app/Post.php sẽ sinh ra đoạn code sau:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    //
}
```

Tên bảng

Mặc định nếu như bạn đặt tên model là `Post` thì laravel sẽ mapping với bảng tên `posts`. Những nếu bạn muốn đặt tên bảng khác đi nhưng vẫn phải theo đúng quy chuẩn convention đặt tên trong Laravel nhé(snake case). Ví dụ mình không đặt tên `posts` nữa mà mình đặt tên là `my_posts` chẳng hạn thì biến `$table` trong `Illuminate\Database\Eloquent\Model` sẽ mapping đúng model với tên bảng chúng ta khai báo.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    /**
     * The table associated with the model.
```

```
     *
     * @var string
     */
    protected $table = 'my_posts';
}
```

Khóa chính của bảng

Theo mặc định trong Laravel thì khóa chính của mỗi bảng sẽ là `id`. Nhưng đôi lúc chúng ta muốn thay đổi trường khóa chính này với tên khác như là `id_post` thì chúng ta có thể khai báo qua biến `$primaryKey`.

```
protected $primaryKey = 'id_post';
```

Kết nối DB

Tất cả các Eloquent models sẽ sử dụng DB mặc định khai báo trong file .env, nhưng nếu bạn muốn model này kết nối tới một bảng trong CSDL nào đó thì chúng ta sẽ set như sau trong model:

```
protected $connection = 'connection-name';
```

3.2 Lấy dữ liệu từ database

all

Khi chúng ta đã thiết lập những tham số như trên xong, thì chúng ta bắt tay vào lấy dữ liệu từ DB về. Để lấy tất cả các bản ghi trong 1 table chúng ta dùng `all()`:

```
<?php

use App\Post;

$flights = App\Post::all();

foreach ($posts as $post) {
    echo $post->title. "<br>";
}
```

Thêm rằng buộc cho câu truy vấn

Có những trường hợp mà chúng ta không cần thiết lấy tất cả các record của table ra, nó làm hiệu năng chương trình của chúng ta kém. Nên thêm những rằng buộc cho câu truy vấn là rất quan trọng để giảm bớt query không cần thiết và tăng hiệu năng chương trình.

```php
use App\Post;

$post = Post::where('published', true)
             ->orderBy('title', DESC)
             ->take(5)
             ->get();
```

Collections

Vì các hàm của Eloquent như `all` và `get` đều trả về nhiều kết quả, hay đó là một instance từ `Illuminate\Database\Eloquent\Collection` sẽ được trả về. Class `collection` cũng cấp các hàm hữu ích để làm việc với các kết quả Eloquent trả về. Các bạn có thể tham khảo các collection ở đây. Mình sẽ lấy một ví dụ về cách sử dụng collections nhé.

```php
use App\Post;
$posts = Post::all();
$posts = $posts->reject(function ($post) {
    return $post->publish;
});
```

Chunk

Nếu bạn muốn xử lý hàng ngán kết quả từ Eloquent, sử dụng `chunk` sẽ tiết kiệm được memory khi thao tác với tập dữ liệu kết quả lớn. hàm này sẽ lấy từng `chunk` của Eloquent models, cung cấp chúng thông qua `Closure` để xử lý.

```php
Post::chunk(200, function ($posts) {
    foreach ($posts as $post) {
        echo $post->title. "<br>";
    }
    echo "Hết một chunk";
});
```

Cursors

Hàm `cursor` cho phép bạn duyetj qua records bằng cách sử dụng một cursor(con trỏ), nó chỉ thực thi cho một truy vấn. Khi dữ liệu lớn, hàm `cursor` có thể được sử dụng để giảm memory sử dụng.

```php
foreach(Post::where('publish', true)->cursor() as $post) {
    //code
};
```

3.3 Insert và Update Models

Insert

Để tạo một bản ghi mới trong table, thì chúng ta sẽ tạo một model instance, sau đó chúng ta set giá trị của thuộc tính, sau đó dùng phương thức `save()`.

```
<?php

namespace App\Http\Controllers;

use App\Post;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Session;

class PostController extends Controller
{
    public function store(Request $request)
    {

        $post = new Post;
        $post->title = $request->title;
        $post->content = $request->content;
        $post->save();
        Session::flash('success', 'Bạn tạo bài post thành công');
        return redirect()->route('posts')
    }
}
```

Trong ví dụ trên chúng ta không tạo 2 trường `created_at` và `updated_at` mà khi `save()` sẽ tự động fill dữ liệu hai trường đó trong CSDL.

Updates

Hàm `save()` cũng được dùng để cập nhật model đã tồn tại trong database, dầu tiên bạn cần lấy model instance ra trước, bạn thay đổi các thuốc tính trong model instance, rồi gọi hàm `save()`. Giá trị hàm `updated_at` sẽ tự động được cập nhật, bạn không cần thay đổi thủ công giá trị này.

```
<?php

namespace App\Http\Controllers;

use App\Post;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
```

```
use Session;

class PostController extends Controller
{
    public function store(Request $request)
    {
        $post = Post::find(1);
        $post->title = "Title bài viết thứ 2";
        $post->save();
        Session::flash('success', 'Bạn thay đổi post thành công');
        return redirect()->route('posts')
    }
}
```

Mass Updates

Nhiều khi chúng ta cần update nhiều bản ghi một lúc thì chúng ta sẽ làm sau đây. Ví dụ như tất cả các các bài post chưa pulish thì sẽ được pulish hết.

```
      use App\Post;


      Post::where('pulish', false)
          ->update(['publish' => true]);
```

Hàm `update` sẽ nhận một mảng `key => value`, với key chính là tên trường cần update và value chính là giá trị update.

Mass Assignment

`Mass Assignment` là tính năng cho phép lập trình một cách tự động gán các tham số của một HTTP request vào các biến hoặc đối tượng trong lập trình. Ví dụ chúng ta có một form đang ký sản phẩm của người dùng như sau:

```
{!! Form::open(['method' => 'POST', 'route' =>
'products.store']) !!}
    {!! Form::label('name') !!} : {!! Form::text('name') !!}
    {!! Form::label('price') !!} : {!! Form::text('price') !!}
    {!! Form::submit('Create') !!}
{!! Form::close() !!}
```

Sau kh i submit form dữ liệu lên chúng ta có thể ghi dữ liệu này vào CSDL bằng đoạn code như sau(ta bỏ qua vấn đề validate dữ liệu nhập vào):

```
use App\Product;
```

```php
public function store(Request $request)
{
    $data = $request->all();
    $product = Product::create($data);
    if ($product) {
        echo "Tạo mới sản phẩm thành công";
    } else {
        echo "Tạo mới sản phẩm không thành công";
    }
}
```

Tuy nhiên có một lỗ hổng bảo mật xảy ra nếu một người truy cập cố tính gửi dữ liệu `role = 'admin'` trong bảng users chẳng hạn để người đó có quyền là admin. Để xử lý lỗ hổng trong Mass Assignment, Laravel đưa ra thêm hai thuộc tính cho Model là `$fillable` và `$guarded`.

Deleting Model

Để xóa bản ghi dữ liệu đơn gainr bằng cách chúng ta gọi đến phương thức `delete()`

```php
$post = Post::find(1);
$post->delete();
//hoặc
$post = Post::where('id', 1)->delete();
```

Laravel có cơ chế hỗ trợ người dùng khi lỡ tay xóa một bản ghi có thể lấy lại được đó chính là soft delete (xóa mềm). Tức là tưởng xóa những thực chất không xóa. Thực chất là chúng ta chỉ thêm một trường `deleted_at` để đánh dấu bản ghi này đã được xóa. Để cho phép một Model có thể thực hiện được đánh dấu bản ghi đã xóa, chúng ta sử dụng trait `Illuminate\Database\Eloquent\SoftDeletes` và thêm `deleted_at` vào thược tính `$dates` của nó.

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
```

```
class Post extends Model
{
    use SoftDeletes;
    protected $fillable = [
        'title',
        'content',
        'publish'
    ];
    protected $dates = ['deleted_at'];
}
```

Trong file migration tạo bảng `posts` bạn nhớ thêm `$table->softDeletes()`. Khi đó, nếu bạn thực hiện phương thức `delete()` thay vì nó sẽ xóa triệt để trong CSDL thì nó sẽ cập nhật thời gian hiện tại vào trường `deleted_at`, và như vậy bản ghi này coi như là đã xóa khỏi table. Để xem xem bản ghi này đã được đánh dấu là xóa tạm thời hay chưa các bạn sử dụng phương thức `trashed()`.

```
if ($post->trashed()) {
    echo "bản ghi này đã dánh dấu là xóa mềm";
}
```

Để truy vấn những bản ghi đã xóa thì chúng ta sử dụng `withTrashed()`

```
$post = Post::withTrashed()
            ->where('id', 1)
            ->get();
// Trả về kết quả bài Post người dùng có id = 1 đã bị xóa mềm.
```

Ngược lại nếu bạn muốn truy vấn kết quả từ những bản ghi mà đã xóa mềm thì bạn sử dụng phương thức `onlyTrashed()`.

Đôi lúc bạn sẽ muốn tất cả các bạn ghi đã xóa mềm rồi quay lại như trước. sử dụng phương thức `restore()`.

```
Post::withTrashed()
    ->where('id', 1)
    ->restore();
```

```
// Bài post có id = 1 sẽ được restore
```

Chú ý, để xóa vĩnh viễn bản ghi dùng phương thức `forceDelete()`.

VI. Tài liệu tham khảo

http://webfaver.com/php-coding/laravel-5/tut-laravel-5-0-laravel-la-gi-gioi-thieu-laravel-framework.html

https://viblo.asia/p/tap-1-cai-dat-laravel-installation-laravel-Eb85owV252G

https://viblo.asia/p/tim-hieu-eloquent-trong-laravel-phan-1-eloquent-model-database-QpmleBAo5rd

https://viblo.asia/p/tim-hieu-ve-migration-trong-laravel-bWrZn1MpKxw

https://viblo.asia/p/tim-hieu-ve-seeder-trong-laravel-bWrZn1MmKxw

https://laravel.com/docs/8.x/migrations

https://laravel.com/docs/8.x/lifecycle