**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO**

**ĐỒ ÁN CHUYÊN NGÀNH**

**Tên đề tài:** Tìm hiểu Laravel Framework – Xây dựng web bán hoa

**GVDH :** Bùi Nhật Bằng

**Họ tên SV :** Trần Ngọc Gia Hân D14-TH02 DH51401681

Thái Huỳnh Đức D14-TH01 DH51400250

**TpHCM 2015**

**Mục lục**

Chương 1: Tổng quan về laravel

1. Laravel là gì?
2. Điểm mạnh
3. Chức năng các thư mục và file trong laravel
4. Các cài đặt laravel

Chương 2 : Mô hình hoạt động của laravel

1. Mô hình MVC
2. Mô hình ORM
3. Route
4. Controller
5. View
6. Blade template

Chương 3 : Một vài chức năng thường dùng

1. Một số thư viện thường dùng trong laravel
2. Validation
3. Form request
4. Migrations
5. Seeding
6. Query Builder
7. Model
8. Làm việc với middleware
9. Các hàm trong Auth
10. Session
11. Pagination

Chương 4 : Ứng dụng minh họa

Chương 5: Bảng phân công công việc

***Chương 1:Tổng quan về Laravel***

**1.Laravel là gì?**

Laravel là một PHP framework  mã nguồn mở miễn phí, được phát triển bởi Taylor Otwell, mục tiêu hỗ trợ phát triển web theo mô hình MVC.

Mặc dù Laravel là một framework PHP tương đối mới (nó được phát hành vào năm 2011), theo khảo sát trực tuyến gần đây của Sitepoint thì nó là framework phổ biến nhất mà các nhà phát triển hay sử dụng. Laravel có một hệ sinh thái rất lớn với một nền tảng giúp triển khai ứng dụng của bạn một cách nhanh chóng, và trang web chính thức của nó cung cấp nhiều hướng dẫn bằng video gọi là Laracasts.

Laravel có nhiều tính năng giúp cho việc phát triển ứng dụng nhanh chóng nhất có thể. Laravel sở hữu một light-weight templating engine có tên là "Blade", cú pháp gọn nhẹ hỗ trợ cho các tác vụ thường xuyên cần phải làm trong ứng dụng của bạn, chẳng hạn như authentication, sessions, queueing, caching và RESTful routing. Laravel cũng bao gồm một môi trường phát triển cục bộ gọi là Homestead - đây thực chất là một Vagrant box được đóng gói.

**2. Điểm mạnh:**

–Là một open source( mở và hoàn toàn free)

\_ Được thừa hưởng những ưu điểm và thế mạnh của các framework khác như CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails…

– Có số lượng người sử dụng nhiều nhất (chiếm 22.7% trong tổng số 15 PHP Framework phổ biến nhất năm 2015)

– Tiếp cận dễ dàng, document rõ dàng, các lệnh tương tác với CSDL ngắn gọn, thân thiện, nó có thể giải quyết tốt những web project lớn.

– Autoload theo namespace.

– Sử dụng mô hình ORM rất đơn giản khi thao tác với DB

– Việc quản lý layout đơn giản với Balade Templating, giúp bạn có thể dễ dàng chia nhỏ code HTML ra nhiều phần, dễ dàng quản lý các assets

– Dễ dàng tích hợp các thư viện với các phiên bản phù hợp vào dự án, và được quản lý bởi Composer

– Phần route rất mạnh. Laravel Route có rất nhiều loại khác nhau như Route dành cho phương thức POST, Route dành cho phương thức GET ở cùng một URL.

- Mô hình MVC rõ ràng, dễ bảo trì.

- Hầu hết các chức năng của Laravel hoạt động liên tục với cấu hình rất nhỏ, dựa vào các quy tắc chuẩn để giảm bớt code-bloat

-Nhà lập trình, người tạo ra framework luôn cập nhật tài liệu trước khi cho ra một phiên bản mới, đảm bảo những người học lập trình luôn luôn có những tài liệu mới nhất.

**3.Chức năng các thư mục và file trong Laravel:**

|  |  |
| --- | --- |
| /app | Thư mục này rất quan trọng và bạn thường xuyên phải thao tác trong thư mục này: Nó chứa các Models, Controller, Providers ... |
| /bootstrap | Chứa các file khởi tạo và một số file cấu hình nạp tự động |
| /public | Thường dùng để chứa các file index.php,.htaccess, js, css của giao diện.Khi khởi chạy ứng dụng, file .htaccess sẽ chuyển hướng mọi yêu cầu (request) tới file index.php, index.php sẽ gọi đến các thành phần tương ứng của laravel (model, view, controller, ...) để thực thi và trả về kết quả (response). |
| /resources | Chứa trang views (V của mô hình MVC), chứa các file khác như (SASS, LESS, CoffeeScript) và các file hỗ trợ cho đa ngôn ngữ.. |
| /routes | Chứa file web.php và api.php có vai trò chỉ đường cho yêu cầu (request) đi đến đâu |
| /storage | Chứa các file đã biên dịch từ các file view xài Blade template, chứa file sessions, caches và các file được sinh ra tự động của framework |
| /tests | Chứa các file test của Framework |
| /vendor | Chứa các thư viện mà Composer cài đặt cho Framework. Bộ mã nguồn của laravel và các thành phần đi kèm laravel, cũng như các gói (packages) sau này sẽ thêm vào laravel |
| /app/config/ | Chứa các file cấu hình của Laravel Framework: cấu hình database, app, mail, session ... |
| /app/config/app.php | Cấu hình mức cài đặt cảu ứng dụng ở khía cạnh khác như timezone, locale, mode debug và khóa mã hóa duy nhất. |
| /app/config/auth.php | Drive xác thực |
| /app/config/cache.php | Nếu ứng dụng của bạn có lưu cache thì thời gian hồi đáp ứng dụng nhanh hơn |
| /app/database/migrations/ | Thư mục migration chứa những lớp PHP mà cho phép Laravel cập nhật Schema cơ sở dữ liệu của bạn mà vẫn giữ được toàn bộ versions cơ sở dữ liệu trong khi đồng bộ. Những file migration được tạo bởi tool Artisan. |
| /app/database/seeds/ | Chứa những file PHP cho phép Artisan đưa vào bảng csdl với những dữ liệu được đề xuất |
| /app/models/ | Chứa những file models của ứng dụng |
| /app/views/ | Thư mục chứa những file HTML của bạn được sử dụng bởi controller hay route ( Chú ý rằng bạn chỉ nên đặt những file mẫu view tại vị trí này vì những file css, javascript đã được đặt trong thư mục public.) |
| /app/lang/ | Mặc định thư mục này chứa những dòng ngôn ngữ cho việc phân trang và chứng thực form người dùng với ngôn ngữ là tiếng anh |
| /app/start/ | Chứa những thiết lập tùy chỉnh liên quan đến tool Artisan cũng như context local và global. |
| /app/storage | Thư mục storage được sử dụng để lưu trữ file tạm thời cho những dịch vụ Laravel khác nhau như session, cache, biên dịch template views. Thư mục này có thể ghi lại bởi web server. Thư mục này được duy trì bởi Laravel mà mình không cần can thiệp vào. |
| /app/routes.php | Đây là file route trong ứng dụng của bạn, nó lưu trứ toàn bộ những route để thông báo với Laravel cách như thế nào để kết nối khi có yêu cầu, theo hàm khai sẵn, controller và action. Đây là file cũng chứa cách khai báo cho một vài sự kiện bao gồm trang lỗi, và nó được sử dụng để định nghĩa views composers. |
| /app/filter.php | File này dùng để giới hạn một số khu vực không được truy cập của trang web. |
| Composer.json | File này chứa danh sách các dependencies (thư viện, framework) sử dụng trong app. Khi bạn chạy câu lệnh composer install hoặc composer update thì file này sẽ là cơ sở để xác định phiên bản nào cần tải về của các thư viện, framework được liệt kê trong file này. |
| Artisan | File này được dùng để thực thi các lệnh CLI (command line interface) để hỗ trợ phát triển ứng dụng. |
| .gitignore | Git (phần mềm để quản lý sự thay đổi của source code) sử dụng 2 file này để định nghĩa thuộc tính và danh sách các file không theo dõi. |

**4. Cách cài đặt Laravel.**

**4.1 Yêu cầu cài đặt**

Để sử dụng Laravel, yêu cầu máy chủ phải đáp ứng các yêu cầu:

* - PHP >= 7.0.0
* OpenSSL PHP Extension
* PDO PHP Extension
* Mbstring PHP Extension
* Tokenizer PHP Extension
* XML PHP Extension

Laravel sử dụng Composer để quản lí các thư viện trong PHP:

-Tạo mới một project laravel.

-Cập nhật các thư viện trong laravel, đúng với phiên bản của project bạn đang thực hiện.

-Quản lý package : Việc cập nhật rất đơn giản, chỉ cần gõ: composer up-date   
Composer sẽ tự động cập nhật các package.   
Nếu muốn cập nhật lên các phiên bản mới hơn hoặc các bản release, chỉ cần chỉnh sửa file composer.json)

**4.2 Composer**

**4.2.1 Định nghĩa**

Composer là một Dependency Management, chương trình hỗ trợ quản lý các gói thư viện trong PHP.Khi bạn làm việc với một Project, bạn chỉ cần khai báo Composer, nó sẽ tự động tải code của các thư viện, tạo file cần thiết, đúng phiên bản người dùng vào project của bạn và update các thư viện nếu cần

Composer về bản chất là một cách để đưa tất cả các thư viện, class, file... của 1 bên thứ 3 như CSS frameworks, jQuery plugins và những thứ khác vào project của bạn. Composer được gọi là Dependency management, nghĩa là nó giúp bạn quản lý các thư viện, nhưng không giống như Yum hoặc Apt trong linux, nó không cho các project của bạn dùng chung một đoạn code của thư viện, mà ở mỗi project, nó sẽ tải code của thư viện về và inject vào thư mục gốc trong project của bạn, và bạn hoàn toàn được auto update các thư viện và sử dụng cho project.

**Composer** là một mã nguồn mở (OpenSource) nên được cộng đồng hỗ trợ rất nhiều, bạn có thể tham gia phát triển từ trang Github chính thức của Composer.

**4.2.2 Lợi ích**

Trước đây khi chưa có composer, khi nào muốn sử dụng thư viện nào, bạn sẽ phải download code của thư viện đó về và đặt vào thư mục của project và sử dụng. Điều này sinh ra những bất lợi sau:

* Dung lượng project lớn vì phải cộng thêm code của các thư viện.
* Nếu bạn quản lý source code bằng git, SVN ( gọi chung là Version control system ) thì khi update các thư viện, bạn phải tạo các commit, mà thực sự các commit đó không phải của bạn.
* Bạn gặp những khó khăn trong việc update version của các thư viện. Nếu thư viện A, có sử dụng thư viện B, thư viện B sử dụng thư viện C, thì nếu một trong các thư viện này có update, bạn sẽ phải tự mình lần mò về nguồn cội của nó để update, rất phức tạp.

Composer giúp lập trình viên giảm bớt suy nghĩ về các package và chỉ tập trung vào ứng dụng chính của mình. Hiện tại, hầu hết các Framework đều hỗ trợ Composer, có thể kể đến như : CodeIgniter, Symfony2, Laravel, FuelPHP… Tất cả các components của các website lớn và danh tiếng như jQuery, Foundation, Bootstrap, thậm chí WordPress đều có sẵn một composer package. Thêm vào đó, các đoạn code nhỏ hơn nhưng không kém phần hữu ích có thể được khôi phục thông qua composer. Các Logging packages như monolog, PHP mailers, string manipulation classes, PHP Unit và các tools khác đều có mặt trong kho package của Composer.

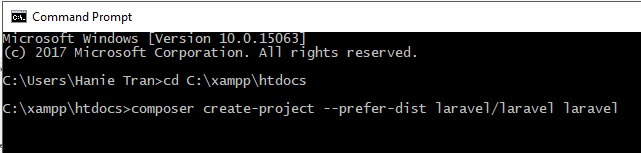
**4.2.3 Các câu lệnh của Composer**

* init : Tạo ra file composer.json nhằm khai báo các thông tin cho package
* install : Đọc thông tin từ file composer.json tại thư mục đang đứng, tổng hợp các package cần cài đặt, và cài đặt chúng vào một thư mục nào đó bên trong project.
* update : Update những dependencies đã được cài đặt lên version mới nhất, đồng thời update nội dung vào file composer.lock
* require : Add hoặc thay đổi nội dung một requirement vào file composer.json. Sau đó package được add vào hoặc thay đổi sẽ được cài đặt hoặc update.
* global : Là command cho phép ta thực hiện các command khác như install, update một cách global. Tuy nhiên câu lệnh global yêu cầu ta phải chạy từ thư mục COMPOSER\_HOME
* search : Cho phép bạn tìm kiếm một package từ trong project của mình, hoặc từ packagist.org
* show : Hiện ra những package khả dụng. Ngoài ra nếu đưa thêm tham số là tên package ở cuối thì sẽ hiện ra thông tin về package đó.
* depends : Thông tin về những package nào phụ thuộc vào package nào
* validate : Kiểm tra xem nội dung file composer.json có hợp hệ hay không
* status : Check xem có gì được thay đổi ở bên trong dependencies (do mình thực hiện) hay không.
* self-update : Update bản thân composer
* config : Chỉnh sửa các setting cơ bản
* create-project : Tạo ra một project từ package đã có sẵn. (chẳng hạn như bạn muốn deploy một package là một application, hay muốn develop tiếp từ một package đã có …)
* dump-autoload : Update autoloader khi có khi có class mới tong classmap package.
* licenses : Hiện thị tên và version của tưng license cho từng package được install.
* run-script : Dùng để chạy script một cách thủ công
* diagnose : Check xem có vấn đề gì với composer hay không
* archive : Tạo ra một file nén (zip hoặc tar) cho một package ở một version chỉ định
* help : Hiện thị thông tin về một câu lệnh nào đó

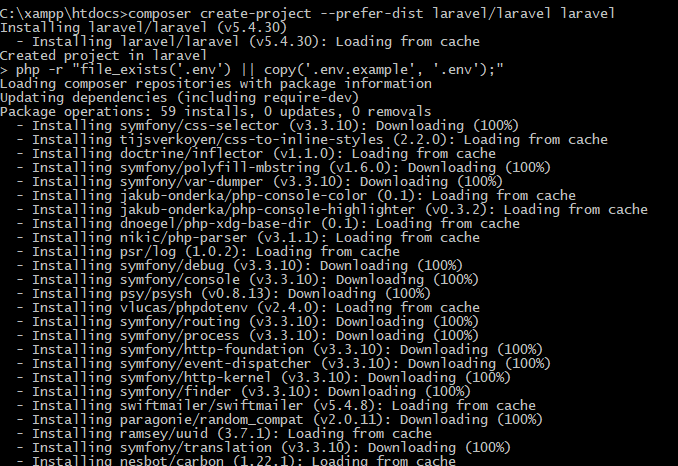
**4.2.4 Cài đặt**  
 - Yêu cầu hệ thống :  
 + PHP >=5.3.2  
 - Cài đặt  
   + Truy cập https://getcomposer.org/Composer-Setup.exe . Tải về và cài đặt như bình thường.

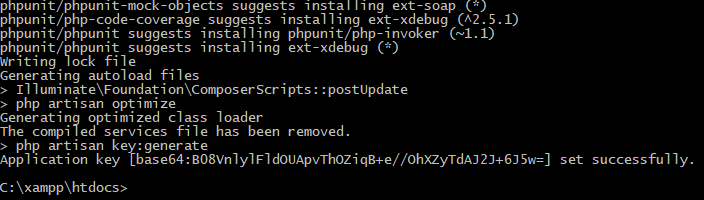
Sau khi cài đặt xong, chạy cmd và gõ lệnh : composer –v  
Nếu hiển thị version của composer nghĩa là thành công.

**4.3 Cài đặt laravel**  
Có nhiều nguồn để cài đặt laravel:   
- Cài đặt sử dụng Composer: Mở cmd, gõ dòng lệnh  
“composer create-project --prefer-dist laravel/laravel TenDuAn”  
với TenDuAn là tên dự án của bạn.

 VD: Sử dụng Composer để tạo một project mới có tên là Laravel

Composer sẽ tự động tải về các thư viện cần thiết cho project của bạn



Sau khi tải xong bạn sẽ nhận được kết quả như hình dưới

- Tải từ github:

Truy cập <https://github.com/laravel/laravel> sau đó tải file zip hoặc clone về. máy.

***Chương 2: Mô hình hoạt động Laravel***

**1.Mô hình MVC**

Vòng đời chuẩn là:

-       Yêu cầu HTTP từ Routed tới một Controller (routing nằm trong thư mục app/routes.php)

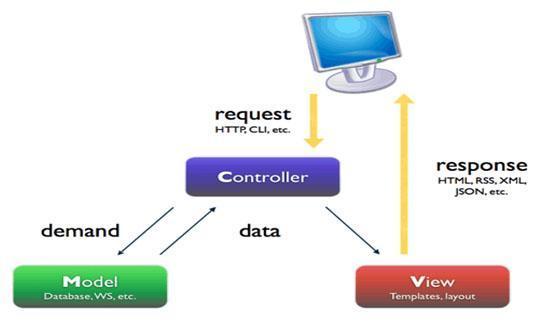
-       Controller sẽ thực hiện những action và gửi kết quả tới view (app/controllers)

-       View sẽ hiển thị những kiểu dữ liệu phù hợp và gửi lại HTTP Response (app/views)

**M là Model:** cấu trúc dữ liệu theo cách tin cậy và chuẩn bị dữ liệu theo lệnh của controller

**V là View:** Hiển thị dữ liệu cho người dùng theo cách dễ hiểu dựa trên hành động của người dùng.

**C là Controller:** Nhận lệnh từ người dùng, gửi lệnh đến cho Model để cập nhập dữ liệu, truyền lệnh đến View để cập nhập giao diện hiển thị.



Khi có một yêu cầu từ phía client gửi đến server, Controller có nhiệm vụ nhận yêu cầu, xử lý yêu cầu đó. Và nếu cần, nó sẽ gọi đến phần model, vốn là bộ phần làm việc với Database.

Sau khi xử lý xong, toàn bộ kết quả được đẩy về phần View. Tại View, sẽ gen ra mã Html tạo nên giao diện, và trả toàn bộ html về trình duyệt để hiển thị.

**Nhược điểm** của mô hình này: Đối với dự án nhỏ việc áp dụng mô hình MVC gây cồng kềnh, tốn thời gian trong quá trình phát triển. Tốn thời gian trung chuyển dữ liệu của các thành phần.

**Sự tương tác giữa các lớp trong mô hình MVC**

**Controller – View** sẽ lấy những hình ảnh, nút bấm…hoặc hiển thị dữ liệu được trả ra từ Controller để người dùng có thể quan sát và thao tác. Trong sự tương tác này cũng có thể không có dữ liệu được lấy từ Model và khi đó nó chỉ chịu trách nhiệm hiển thị đơn thuần như hình ảnh, nút bấm…

**Controller – Model** là luồng xử lý khi controller tiếp nhận yêu cầu và các tham số đầu vào từ người dùng, controller sẽ sử dụng các lớp/hàm trong Model cần thiết để lấy ra những dữ liệu chính xác.

**View – Model** có thể tương tác với nhau mà không qua Controller, nó chỉ đảm nhận hiển thị dữ liệu chứ không phải qua bất kỳ xử lý nghiệp vụ logics nào. Nó giống như các vùng dữ liệu hiển thị tĩnh trên các website như block slidebar…

**2. Mô hình ORM**

**ORM (Object Relational Mapping)** là một kỹ thuật lập trình dùng để chuyển đổi dữ liệu giữa một hệ thống không hướng đối tượng như cơ sở dữ liệu sang hệ thống hướng đối tượng như lập trình hướng đôi tượng trong PHP. Kỹ thuật này tạo ra các đối tượng CSDL ảo có thể được lập trình trong mã nguồn và có nhiều ưu điểm như mã nguồn trở lên rõ ràng và dễ bảo trì, dễ dàng thao tác với dữ liệu và thực hiện việc tối ưu hệ thống thông qua việc sử dụng bộ đệm… Các công việc khó hoặc không thể xử lý ở database layer sẽ được đưa lên lớp ứng dụng

**3. Route**

Route trong Laravel có vai trò định ra các tuyến đến các controller cụ thể từ một request của người dùng.

**3.1 Danh sách các loại Route**

* **Route::get** phương thức GET
* **Route::post** phương thức POST
* **Route::match** kết hợp hiều phương phức
* **Route::any** nhận tất cả các phương thức
* **Route::filter** tạo ra bộ lọc nhằm mục đích nào đó, ví dụ tạo bộ lọc kiểm tra đã đăng nhập hay chưa
* **Route::group**gom các route lại với nhau thành một nhóm
* **Route::controller**gọi đến controller tương ứng
* **Route::resource**sử dụng với resource controller

**3.2 Cách sử dụng**

Để sử dụng Route, mở thư mục chứa dự án Laravel, sau đó vào route/web.php

**3.2.1** **Route::get** : tiếp nhận các request với phương thức GET  
 Cú pháp : Route::get($url, $action);  
 $url là dường dẫn của Route  
 $action là các hành động thực hiện trong route.  
 Ví dụ:

Route::get('/', function () {

return view('welcome');

});

Route::get('index',[

'as'=>'index',

'uses'=>'PageController@getIndex'

]);

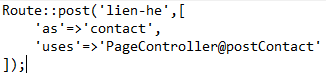
Trong ví dụ trên ta có index là phần đường dẫn sẽ hiển thị trên thanh

‘as’=>’index’ : index này sẽ được gọi ra trong các href

'uses'=>'PageController@getIndex' : hàm function getIndex sẽ được viết trong PageController

**3.2.2 Route::post** tiếp nhận các request với phương thức POST

Cú pháp : Route::post($url, $action);  
 $url là dường dẫn của Route  
 $action là các hành động thực hiện trong route.  
 Ví dụ:



Trong ví dụ trên ta có

‘lien-he’ là phần hiển thị trên thanh đường link

'as'=>'contact' : contact sẽ được gọi ra trong trang blade phần href, gọi đến route nào đó để được chuyển đến đúng trang

'uses'=>'PageController@postContact' : hàm postContact' sẽ được viết trong 'PageController

**3.2.3 Route::match**

Ngoài các method POST và GET phổ biến, trong laravel còn có thêm PUT, PATCH, DELETE. Để sử dụng các phương thức PUT,PATCH, DELETE thì phải sử dụng Route::match

Cú pháp : Route::match($methods, $uri, $action);  
 $methods có thể là 1 mảng các method như ['GET', 'POST','PUT'].  
 $url là dường dẫn của Route.  
           $action là các hành động thực hiện trong route.

Ví dụ:

**3.2.4 Route::any**

Nhận Request của tất cả các method

Cú pháp : Route::any($url, $action);  
 $url là dường dẫn của Route.  
           $action là các hành động thực hiện trong route.

Ví dụ:

**3.2.5 Route::controller**  
Nhận Request của tất cả các method

Cú pháp : Route::any($url,$controller,  $ names);  
$names là một mảng quy định tên cho action trong controller, có thể có hoặc không.  
$url là dường dẫn của Route.  
$controller là tên controller

Ví dụ:

**3.2.5 Route::resource**  
Một chức năng giúp chúng ta Xây dựng RESTful(websevice) một cách nhanh chóng

Cú pháp : Route::resource(‘url’, ‘controller’,’tuybien’);

url là dường dẫn của Route.  
controller’ là tên controller  
tuybien là các phương thức được sử dụng trong Route(có thể bỏ)

Ví dụ:

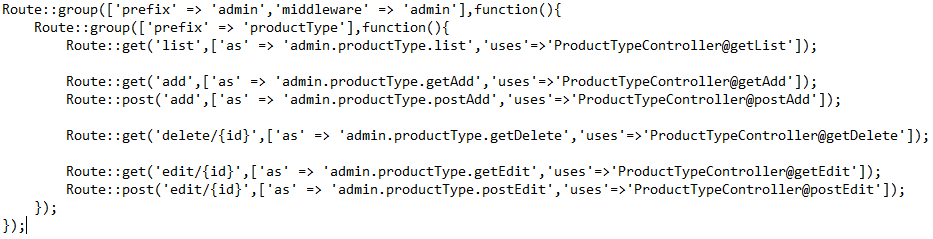
**3.2.6 Route::group**Nhóm các route với nhau

Cú pháp: Route::group($attr,$handle);

$attr: là các mảng thành phần điều kiện. VD: middleware,prefix,namespace,domain.

$handle: là các đoạn sử lý thường là hàm.

Ví dụ:



1 group route chung cho trang quản lý admin

**4. Controller**

Sau khi nhận Request thì sẽ chuyển hướng đến controller và controller sẽ xử lý. Controller sẽ gọi tới Model để tương tác với cơ sở dữ liệu và đưa dữ liệu tới View.

**Nguyên tắc tạo controller:**

* Controller phải được đặt trong đường dẫn App\Http\Controllers
* Tên của controller phải giống với tên class trong file controller đó.
* Class controller  phải kế thừa từ Controller

**4.1 Tạo controller**

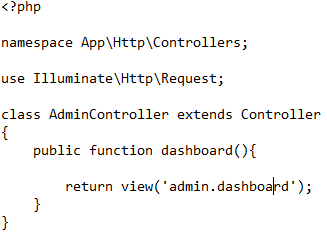
Có 2 cách tạo Controller.  
  
 Cách 1: Tạo bằng tay

Tạo một file controller trong thư mục App\Http\Controller và phải tuân thủ “Nguyên Tắc Tạo Controller”.  
 Ví dụ:AdminController

Mở cặp dấu <?php ?>

Tiếp theo ta gọi namespace App\Http\Controllers;

Viết 1 class AdminController ta muốn tạo ra và kế thừa từ Controller cha



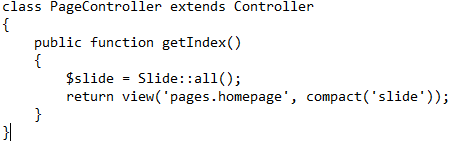
Cách 2: Tạo bằng lệnh laravel cung cấp sẵn  
 Mở cmd, trỏ vào thư mục chứa dự án Laravel và gõ theo cú pháp  
 **“php artisan make:controller controllername”**

trong đó controllername là tên của controller, khi đó laravel sẽ tự tạo ra một file controller có tên là controllername  được lưu trong App\Http\Controller

**4.2 Sử dụng Action trong controller**

Để tạo một action trong controllers thì chúng ta tạo một hàm trong controller và gọi ra ở trong Route.

Ví dụ:



Ta tạo hàm getIndex với biến $slide sẽ được gọi ra sau đó

Return view để trả về dữ liệu hiển thị ở trang homepage

**4.3 Truyền tham số trong action Controllers**

Ví dụ:

 Truyền vào một tham số $type

public function getProductType($idType)

{

$productsByIdType = Product::where('id\_type', $idType)->paginate(9);

return view('pages.productType', compact('productsByIdType'));

}

Ở đây tham số được truyền vào là $idType

**5.View**

View nhận dữ liệu dừ Controller và sau đó dựa vào layout của giao diện để xử lý dữ liệu theo yêu cầu.

**5.1 Tạo view**

Mặc định tất cả các file view sẽ được lưu ở thư mục resource/views.

Để tạo view, ta chỉ cần tạo ra file có đuôi là .php hoặc .blade.php.

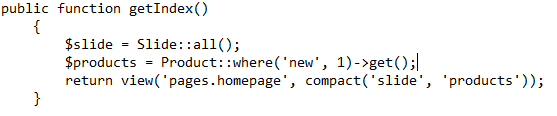
**5.2 Gọi view**

**Cú pháp: view(‘duongdantoiview’,** **dulieutruyenvaoview);**

Trong đó:  
 duongdantoiview  : Là đường dẫn tới file view lấy từ thư mục resources/views

dulieutruyenvaoview : là dữ liệu mà các bạn muốn truyền vào view(nếu không có thì bỏ qua).

Ví dụ:



‘page.homepage’ đường dẫn đến trang hompage

compact() các dữ liệu muốn truyền vào sẽ được viết trong ngoặc đơn bên trong

**5.3 Tạo view dùng chung** Khi cần chia sẽ một thông tin cho tất cả các view để hiển thị trên ứng dụng,ta có thể chia sẻ thông tin để dùng chung thay vì phải lặp lại việc lấy thông tin ra từ database.

Laravel cung cấp phương thức share chia sẻ dữ liệu cho toàn bộ Views.

Ví dụ:

**5.4 Các cách truyền dữ liệu cho view**

**5.4.1 Dùng compact()**

Cú pháp: compact(‘tencuabientruyenvao’);

Trong đó tencuabientruyenvao là tên của biến truyền vào.

Ví dụ:

**return** view(**'pages.homepage'**,*compact*(**'slide'**));

truyền biến slide từ controller sang view homepage.

**5.4.2 Dùng with()**

Cú pháp: view(‘tenview’)->with(‘key’,’value’);

Ví dụ:

**5.4.3 Dùng mảng**

Cú pháp: view(‘tenview’,[‘key’=>’value’]);

Ví dụ:

**return** view(**'pages.homepage'**,[**'slide'**=>$slide]);

**6. Blade template**

Blade là một templating engine đơn giản nhưng rất mạnh mẽ được tạo ra và đi cùng với Laravel

Blade không hạn chế sử dụng các mã PHP thuần túy trong view.

File Blade view có phần mở rộng là .blade.php và thông thường được lưu trong thư mục resources/views.

Hai lợi ích chính của việc sử dụng Blade là kế thừa template và sections.

**6.1 Các thành phần trong blade template**

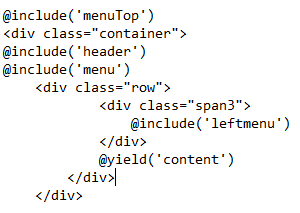
@extends : khai báo cho blade template biết file đang kế thừa.

@yield :có tác dụng như một biến, khai báo cho blade template biết vị trí để chèn nội dung của biến đó.

@section : chứa nội dung để hiển thị, nội dung của $section sẽ được hiển thị khi sử dùng @yield

@include : import một file vào file hiện tại.

VD:Ở trang master.blade.php chứa template của trang web ta có



@include('menuTop') : import giao diện menuTop vào template

Các phần @include sẽ được giữ nguyên và sẽ là template cứng, không thay đổi

@yield(‘content’) : phần nội dung sẽ được thay đổi ở các giao diện khác nhau

**6.2 Sử dụng blade template**

Muốn sử dụng blade template thì các tên file phải chứa “.blade” ở phía trước “.php”

Ví dụ: myView.blade.php

**6.2.1 Hiển thị dữ liệu trong blade template**

Khi sử dụng blade muốn in dữ liệu ra màn hình:

Cách 1: sử dụng “ {{ }} “

Ví dụ :

{{ $a = ‘ <b> Do An Chuyen Nganh </b>’  }}

=> Kết quả :  <b> Do An Chuyen Nganh </b>

Cách 2: Sử dụng { !! ‘biencanin’ !! }

Trong đó biencanin là dữ liệu cần xuất

Ví dụ:

{{ $a = ‘ <b> Do An Chuyen Nganh </b>’  }}

=> Kết quả :  Do An Chuyen Nganh

Sử dụng 3 cặp ngoặc {{{}}} để xuất giá trị dưới dạng HTML entities

**6.2.2 Dữ liệu mặc đinh trong blade template**

Để sử dụng dữ liệu mặc định trong blade template, ta dùng “ OR”

Ví dụ:

**6.2.3 Vòng lặp trong blade template**

Các loại vòng lặp trong blade template

@for ($i = 0; $i < 10; $i++)

    Giá trị hiện tại là {{ $i }}

@endfor

@foreach ($users as $user)

    <p>Đây là user có mã {{ $user->id }}</p>

@endforeach

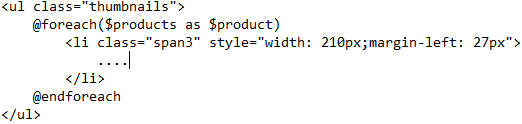
@while (true)

    <p> Vòng  lặp mãi mãi :(</p>

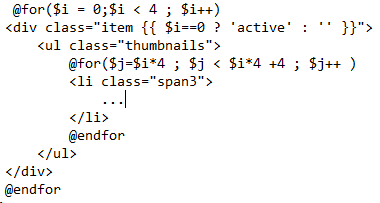
@endwhile

VD:

-Vòng lặp foreach



Vòng lặp for



**6.2.4 Câu lệnh if**

@if (count($records) === 1)

    có 1 bản ghi!

@elseif (count($records) > 1)

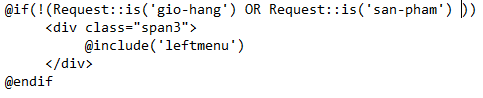
    có nhiều bản ghi!

@else

    không có bản ghi nào.

@endif

Ví dụ:



Nếu có tồn tại Request ‘gio-hang’ hoặc ‘san-pham’ thì ta sẽ thực hiện phần @inlude bên trong

Với câu lệnh if chỉ có một điều kiện, ta có thể sử dụng $unless

@unless ($count(record>1)

    Có giá trị

@endunless

**6.2.5 Sử dụng @extends ,@yield ,@section , @include**

Ví dụ:

Trang index chính master.blade.php

<div class="row">

@if(!Request::is('gio-hang') & !Request::is('san-pham'))

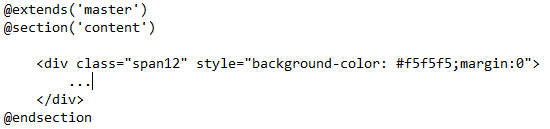
<div class="span3">@include('leftmenu')</div>

@endif

<div class="span9">@yield('content')</div>

</div>

Gọi ra để sử dụng cho các trang khác



Phần bên trong section(‘content’) sẽ là phần nội dung được thay đổi, phần này tương ứng với @yield(‘content’) ở trang template đã tạo

6.2.6 Hiển thị JSON

Cách 1: gọi thủ công json\_encode

<script> var app = <?php echo json\_encode($array); ?>; </script>

Cách 2: $json

<script> var app = @json($array); </script>

6.2.7: Phím xác thực @auth @guest

6.2.8 Biến loop

|  |  |
| --- | --- |
| $loop->index | Chỉ số lặp vòng lặp hiện tại (bắt đầu ở 0). |
| $loop->iteration | Vòng lặp vòng lặp hiện tại (bắt đầu từ 1). |
| $loop->remaining | Sự lặp lại còn lại trong vòng lặp. |
| $loop->count | Tổng số mục trong mảng đang được lặp lại. |
| $loop->first | Cho dù đây là vòng lặp đầu tiên thông qua vòng lặp. |
| $loop->last | Cho dù đây là lần lặp cuối cùng thông qua vòng lặp. |
| $loop->depth | Mức làm tổ của vòng lặp hiện tại. |
| $loop->parent | Khi trong một vòng lặp lồng nhau, biến vòng lặp của cha mẹ. |

***Chương 3:Một vài chức năng thường dùng***

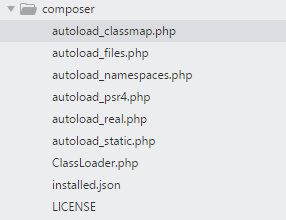
1. **Một số thư viện thường dùng trong Laravel**

Các thư viện được cài đặt tự động khi tạo mới một project laravel

Các thư viện này được lưu trong /vendor

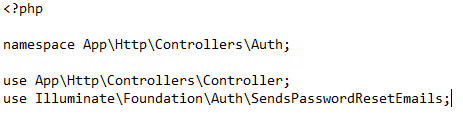
+ Debugbar : được dùng để debug các ứng dụng. Có thể dễ dàng xử lý các Request tới ứng dụng,các truy vấn với database, các views đang hiển thị…  
+ PHPMailer : được dùng để gửi Mail.(VD: web Hoasaigon.tk sử dụng thư viện swiftmailer)  
+ Carbon PHP : được dùng khi làm việc với ngày giờ, được phát triển dựa trên extension php DateTime kế thừa các class cơ sở của DateTime.  
+ Faker PHP : dùng để giả lập các mẫu dữ liệu khác nhau, thường được dùng để tạo dữ liệu mẫu khi debug.  
+ Omnipay PHP : dùng để phát triển các ứng dụng thanh toán trực tuyến.  
+ PHPGeo PHP : dùng để phát triển các ứng dụng bản đồ, cho phép ta tính khoảng cách giữa các toạ độ địa lý chính xác.  
+ Opauth PHP : dùng để giúp người dùng xác thực bản thân thông qua các tài khoản cá nhân trên các mạng xã hội như Facebook, Google+ , Twitter….  
+Composer

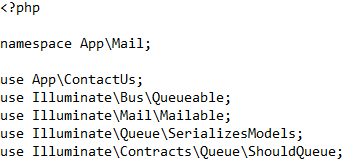
VD:



Ví dụ: Các thư viện khi muốn dùng thì cần phải khai báo

-Lấy lại mật khẩu khi quên, mật khẩu mới sẽ được gửi qua mail





**2. Validation**

Validation là tiến trình kiểm tra cái gì đó có tuân theo một qui tắc đã cho hay không. Và cụ thể đối với form nó là các ràng buộc về dữ liệu của form nhằm đảm bảo tính chặt chẽ của ứng dụng.

2.1 Form Validation

Trong Laravel hỗ trợ chúng ta mạnh về validation dữ liệu.

**Validation cơ bản :** hiển thị thông báo lỗi

-Đầu tiên là xác định route

+Get:Route hiển thị form cho người dùng tạo mới

+Post:Route lưu bài viết vào CSDL

-Tạo controller

-Code logic cho Validation

+Validate nhận Http request và đặt quy định cho Validation, nếu quy định thành công thì code thực thi bình thường, nếu thất bại thì một exception sẽ được tạo ra tích hợp lỗi response gửi đến cho người dùng

**Validation Trực tiếp trên Controller.**

-Để validation dữ liệu thì bắt buộc trên controller của chúng ta phải khai báo lớp http Request có namespace là use Illuminate\Http\Request;

-Tiếp đó chúng ta cần gọi hàm với cú pháp:

$this->validate($request,$pattern,$messenger,$customName);

Trong đó:

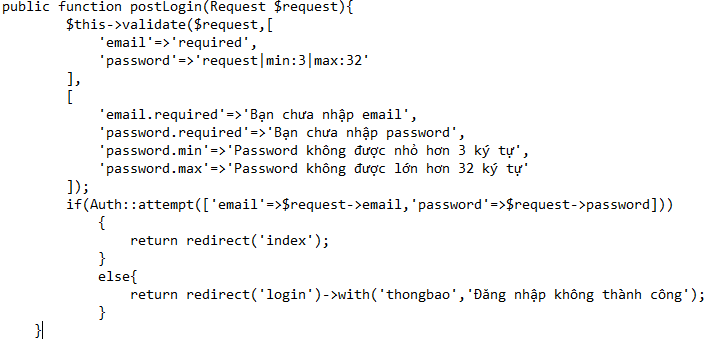
·         **$request:** Là biến tham chiếu đối tượng Request mà các bạn khai báo ở đầu hàm.

·         **$pattern:** Là mảng định nghĩa dữ liệu đầu vào của các trường trong form.

·         **$messenger:** Là mảng chứa nội dung báo lỗi (Nếu muốn thay đổi).

·         **$customName:** Là mảng chứa các tên cho các trường trong form.

VD:



**Validation với lớp validator.**

-Để sử dụng được lớp này trong controller thì chúng ta cần phải khai báo namespace của 2 lớp http Request và Validator.

use Validator;

use Illuminate\Http\Request;

Tiếp đó khởi tạo validator với cú pháp:

Validator::make($request,$pattern,$messenger,$customName);

-Nếu validator thành công thì sẽ trả về true và ngược lại

Trong đó:

·         **$request:** Là biến tham chiếu đối tượng Request mà các bạn khai báo ở đầu hàm

·         **$pattern:** Là mảng định nghĩa dữ liệu đầu vào của các trường trong form.

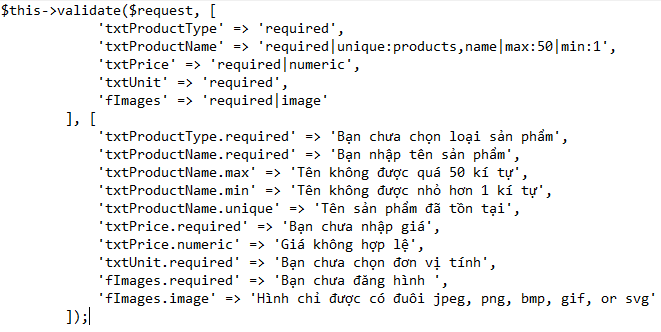
·         **$messenger:** Là mảng chứa nội dung báo lỗi (Nếu muốn thay đổi).

·         **$customName:** Là mảng chứa các tên cho các trường trong form.

Các Validation hay dùng

|  |  |
| --- | --- |
| Required | Không được để trống ô dữ liệu. |
| Boolean | Chỉ cho phép nhập vào các giá trị 0,1,true,false,'0','1' . |
| confirmed | Kiểm tra 2 input có trùng nhau không, input sau bắt buộc phải có tiền tố \_confirmation. VD: password,password\_confirmation. |
| dimensions | Giới hạn chiều rộng chiều cao của ảnh. VD: 'avatar' => 'dimensions:min\_width=100,min\_height=200' |
| email | Dữ liệu nhập vào phải là email. |
| file | Dữ liệu nhập vào phải là file. |
| image | Dữ liệu nhập vào phải là file ảnh(jpeg, png, bmp, gif, or svg). |
| in:foo,bar,.. | Dữ liệu nhập vào phải nằm trong list. |
| integer | Dữ liệu nhập vào phải là số nguyên. |
| json | Dữ liệu nhập vào phải là json |
| max:value | Độ dài lớn nhất của dữ liệu nhập vào. VD: max:255. |
| min:value | Độ dài nhỏ nhất của dữ liệu nhập vào. VD: min:5. |
| mimetypes:text/pain,... | Kiểu dữ liệu của file phải nằm trong list.  VD: 'video' => 'mimetypes:video/avi,video/mpeg,video/quicktime' |
| numeric | Dữ liệu nhập vào phải là số. |
| unique:table | Dữ liệu nhập vào phải là duy nhất trong bảng. |

Ví dụ



**3. Form Request**

Request trong Laravel hỗ trợ chúng ta rất mạnh mẽ về xử lí dữ liệu từ Form.

Request có thể được gọi ra trong Controller hoặc ở Route.

Để sử dụng Request, ta cần gọi class Request : “use Illuminate\Http\Request;”

3.1 Các phương thức của Request.

* $request->url() : trả về đường dẫn url đầy đủ của url không có query string
* $request->fullurl() : trả về đường dẫn url đầy đủ của url có query string
* $request->medthod() : trả về phương thức gửi.
* $request->isMethod() : kiểm tra phương thức gửi của form.

$method=$request->method();  
 if($request->isMethod('post')){  
 echo 'POST request';  
 } else {  
 echo 'GET request';  
 }

* $request->is(‘admin/\*’) : kiểm tra url có chứa chuỗi “admin” hay không.
* $request->all() : lấy hết dữ liệu của Form gửi lên.
* $request->input(‘tentheinput’) : lấy dữ liệu của thẻ input có tên là ‘tentheinput’
* $request->path() : trả về thông tin đường dẫn yêu cầu.
* $request->has(‘tentheinput’) : kiểm tra giá trị của thẻ input có tên là có trên request hay không.
* $request->filled(‘‘tentheinput’’) : kiểm tra giá trị của thẻ input có tên là có trên request và khác rỗng hay không.
* $request->cookie(‘‘tentheinput’’) : lấy giá trị cookie từ request.
* $request->file() : lấy tệp tin từ request.
* $request->hasFile() : kiểm tra tệp tin có trên request hay không.

Ví dụ:



**4. Migrations**

Migrations giống như một control database có tác dụng quản lý cũng như lưu trữ cấu trúc của database giúp cho việc sửa đổi database dễ dàng hơn.

Điều kiện sử dụng:

* Phải kết nối với database
* Migrations phải nằm trong thư mục : database\migrations

4.1 Tạo migrations

Mở cmd và trỏ vào thư mục chứa project laravel.  
Tuỳ theo mục đích, có các câu lệnh ta có thể sử dụng:

* php artisan make:migration TenMigrate : tạo migrate thông thường
* php artisan make:migration TenMigrate --create=TableName : tạo migrations cho bảng TableName
* php artisan make:migration TenMigrate --table=TableName: tạo migrations chỉnh sửa bảng cho bảng TableName

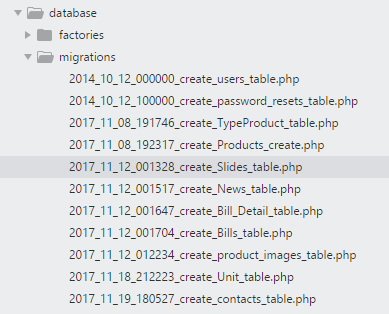
4.2 Các câu lệnh tạo bảng.

|  |  |
| --- | --- |
| $table->bigIncrements('id') | Tạo cột id khóa chính tự động tăng kiểu bigint |
| $table->bigInteger('votes') | Tạo cột votes với kiểu bigint |
| $table->binary('data'); | Tạo cột data với kiểu blob |
| $table->boolean('confirmed'); | Tạo cột confirmed với kiểu boolean |
| $table->char('name', 100); | Tạo cột name với kiểu char tối đa 4 kí tự |
| $table->date('created\_at'); | Tạo cột created\_at với kiểu date |
| $table->dateTime('created\_at'); | Tạo cột created\_at với kiểu dateTime |
| $table->double('amount', 8, 2); | Tạo cột amount với kiểu DOUBLE |
| $table->integer('votes'); | Tạo cột votes với kiểu DOUBLE |
| $table->timestamps(); | Tạo cột với kiểu timestamps |

Có thể xem thêm các câu lệnh khác tại : <https://laravel.com/docs/5.5/migrations#creating-columns>

Ví dụ:

Khi sử dụng câu lệnh migrate để tạo ta được các bảng như sau

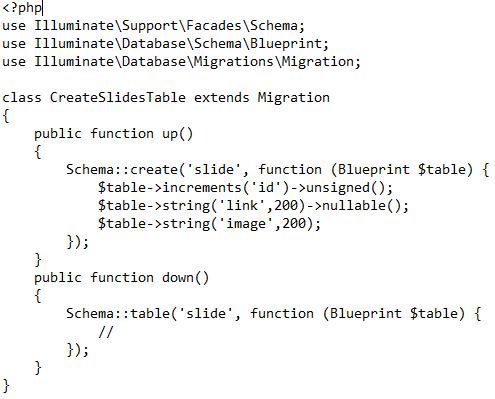


4.3 Thực thi Migrations

Sau khi thực hiện viết code, để có thể chạy được , ta có thể sử dụng các lệnh sau:

|  |  |
| --- | --- |
| Lệnh | Chức năng |
| php artisan migrate | chạy migration |
| php artisan migrate:resest | resest lại migration |
| php artisan migrate:refesh | chạy lại migration |
| php artisan migrate:status | xem trạng thái của migration |
| php artisan migrate:install | cài đặt migration |

Ví dụ:bảng Slide được tạo ra bằng lệnh migate, các hàm up, down sẽ được tạo sẵn



**5. Seeding**

-Seeding hay còn gọi là seeder,seed,.. thực ra nó là một class chứa code để tạo ra các dummy data(dữ liệu mẫu) cho database trong quá trình xây dựng ứng dụng.

**-Trong Seeder chúng ta có thể viết code của** [**Query builder**](http://toidicode.com/query-builder-trong-laravel-11.html)**.**

VD:

**-Tạo seed bằng lệnh**

php artisan make:seed “tên seed ban muốn tạo”

Thư mục sau khi tạo sẽ được lưu ở database/seeds, với code được tạo sẵn như sau

<?php

use Illuminate\Database\Seeder;

class Users extends Seeder

{

public function run(){}

}

**-Tiếp đó chúng ta sẽ có hai cách chạy:**

* Dùng lệnh: php artisan db:seed --class=Users
* Vào file DatabaseSeeder.php thêm đoạn $this->call(Users::class); vào hàm run (Users có thể thay đổi bằng tên file của bạn muốn thực hiện lệnh) và chúng ta lại dùng lệnh php artisan db:seed

Bạn cũng có thể sử dụng câu lệnh migrate:refresh để seed cơ sở dữ liệu của bạn, câu lệnh này cũng được dùng cho việc rollback hay chạy lại migration của bạn  
php artisan migrate:refresh --seed

Ví dụ:

**Gọi seeder bổ sung**Bện trong class DatabaseSeeder bạn có thể dùng hàm call để thực thi các seed class bổ sung.   
Sử dụng hàm call cho phép bạn phân tách cấu trúc seed vào database thành nhiều file,tránh làm các seed class trở nên quá lớn.  
Ta chỉ cần tên của seeder class mà muốn thực thi:  
public function run()  
{  
 $this->call(UsersTableSeeder::class);  
 $this->call(PostsTableSeeder::class);  
 $this->call(CommentsTableSeeder::class);  
}

**6.Query Builder**

Có tác dụng thay thế cho các câu lệnh truy vấn thông thường bằng các phương trong lớp DB.

Ví dụ : $users = DB::table('users')->get();

Sẽ lấy toàn bộ dữ liệu trong bảng users ra và lưu vào $users

Lệnh này sẽ tương đương với lệnh truy vấn thông thường : SELECT \* FROM users

**Một số lệnh truy vấn thường gặp**

- Chọn bảng trong CSDL: DB::table(‘users’)->get();

- Lấy dữ liệu: DB::table('users')->get();

- Lấy một dòng dữ liệu đầu tiên từ kết quả truy vấn:

DB::table('users')-> where('name', 'John')->first();

- Trả về dữ liệu cột đã khai báo:

DB::table('users')-> where('name', 'Joh')-> value('email');

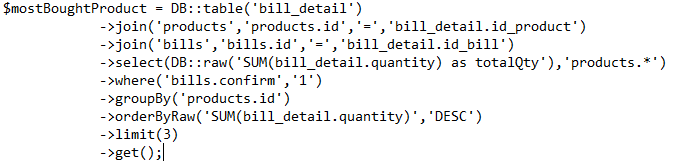
- Chọn tên cột cần truy vấn: DB::table('users')-> select(‘name’, ‘email’)->get();

- Điều kiện where: DB::table('users')-> where('votes', '=', 100)->get();

- Lệnh update: DB::table('users')->where('id', 1)-> update(['votes' => 1]);

- Lệnh insert: DB::table('users')->insert( ['email' => 'john@example.com', 'votes' => 0] );

Ví dụ



Lấy dữ liệu trong database từ bảng bill\_detail join với bảng product bằng khóa id\_product join tiếp với bảng bills bằng khóa id\_bill

Tính tổng quantity trong bill\_detail bằng lệnh SUM(bill\_detail.quantity) rồi tạo thành cột mới với tên totalQty

Gọi lệnh select để lấy dữ liệu từ cột mới tạo totalQty và tất cả dữ liệu trong bảng product

Lệnh where là điều kiện phải có với confirm trong bảng bills phải = 1

->limit(3) : lấy 3 sản phẩm

->get() : gọi lệnh ra để lấy dữ liệu

**7.Model**

Model là một lớp dữ liệu, có cấu trúc giống với bảng trong cơ sở dữ liệu, dùng để xử lý dữ liệu ra vào trong bảng.

**7.1Tạo model**

Các file model sẽ được lưu tại thư mục App/

php artisan make:model TenModel

Tạo một model và migrate tương ứng với nó :

Php artisan make:model TenModel -m

Kết nối Model tới bảng trong cơ sở dữ liệu

|  |  |
| --- | --- |
| Mã lệnh | Mô tả |
| protected $table = ‘tên bảng’; | Kết nối model với bảng trong cơ sở dữ liệu |
| public $timestamps = false; | Tắt/bật chế độ tự động quản lý ‘created\_at’ và ‘update\_at’ |

VD:

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model

{

protected $table = ‘user’;

public $timestamps = false;

}

**7.2.Các phương thức trong model**

|  |  |
| --- | --- |
| Mã lệnh | Mô tả |
| $user = new User();  echo $user->name; | Lấy giá trị thuộc tính của model |
| $user = User::all(); | Lấy toàn bộ dữ liệu trong bảng |
| $user = User::find(giá trị khóa chính); | Tìm user theo khóa chính |
| $user->toJson(); | Trả dữ liệu kiểu JSON |
| $user->save(); | Lưu dữ liệu từ model vào bảng |
| $user->delete(); | Xóa dữ liệu trong bảng |
| User::destroy(giá trị khóa chính); | Xóa dữ liệu bằng khóa chính trong bảng |

Kết hợp model với query builder

$user = User::where('active', 1)->orderBy('name', 'desc')->take(10)->get();

**7.3.Liên kết dữ liệu**

Model là đại diện cho các bảng trong cơ sở dữ liệu, chính vì thế mà nó cũng có các liên kết với nhau.

Khai báo các liên kết tới các model khác.

Ví dụ : Liên kết một nhiều. Ta khai báo hàm TenLienKet() trong class model.

|  |  |
| --- | --- |
| Khai báo | Sử dụng |
| public function TenLienKet()  {  return $this->hasMany( ‘TenModel’ , ‘KhoaPhu’, ‘KhoaChinh’ );  } | TenModel::TenLienKet() |

**7.4.Các liên kết**

- Một-Một, liên kết từ bảng cha tới bảng con: hasOne();

- Một-Một, liên kết từ bảng con tới bảng cha: belongsTo();

- Một-Nhiều: hasMany();

- Nhiều-Nhiều: belongsToMany();

- Liên kết qua bảng trung gian: hasManyThroungh();

**8. Làm việc với middleware**

**8.1 Cấu trúc của middleware**  
public function handle($request, Closure $next)  
 {  
 if ($request->input('age') <= 200) {  
 return redirect('home');  
 }  
 return $next($request);  
 }

**8.2 Tạo middleware mới cửa sổ cmd**

php artisan make:middleware TenMiddleware

**8.3 Đăng ký Middleware cho routes**

Mở app/Http/Kernel.php

protected $routeMiddleware = [

'auth' => \App\Http\Middleware\Authenticate::class,

'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

‘Định danh Middleware’ => \App\Http\Middleware\TenMiddleware::class,

];

**8.4 Gán middleware cho route**

|  |  |
| --- | --- |
| Cách 1 | Cách 2 |
| Route::get('user',[ 'middleware' => 'Định danh', function () { } ]); | Route::get('/', function () { })->middleware([''Định danh1', ''Định danh2']); |

**9.Các hàm trong Auth**

Các hàm trong auth được tạo ra bằng lệnh php artisan make:auth

Để sử dụng Auth bạn phải thêm thư viện

use Illuminate\Support\Facades\Auth;

Sử dụng Auth

- Auth::login($user);

- Đăng nhập với thông tin đăng nhập Auth::attempt(array())

- Đăng nhập với đối tượng : login(model)

- Đăng xuất Auth::logout();

- Lấy thông tin người đang đăng nhập :user()

- Kiểm tra đăng nhập chưa : check()

**10.Session**

**10.1.Cấu hình**

Cấu hình session trong config/session.php

|  |  |
| --- | --- |
| Tùy chỉnh trong config/session.php | Mô tả |
| 'lifetime' => 120, | Thời gian tồn tại của session tính theo phút |
| 'expire\_on\_close' => false, | true : mất session khi đóng trình duyệt, false : ngược lại |
| 'driver' => env('SESSION\_DRIVER', 'file'), C | Chọn nơi lưu trữ session |

Để sử dụng được session thì ta phải đặt routes trong middleware web Route::group(['middleware' => 'web'], function () {

Route::get('Route-Session', function(){

session()->put('KhoaHoc','Laravel');

echo session('KhoaHoc');

});

});

**10.2.Các hàm trong session**

Sử dụng Session

Session::PhuongThuc;

session()->PhuongThuc;

**Phương thức trong Session**

|  |  |
| --- | --- |
| Phương thức | Chức năng |
| put('Ten','GiaTri'); | Khai báo session |
| flash('flasha','Đây là session flash'); | Khai báo flash |
| session('Ten'); | Lấy dữ liệu từ session |
| has(‘Ten’) | Kiểm tra session tồn tại không |
| forget('Ten'); | Xóa bỏ 1 session |
| flush(); | Xóa hết các session |

**11.Pagination**

**11.1. Khai báo**

|  |  |
| --- | --- |
| ControllerPhanTrang | PhanTrang.blade.php |
| $sanpham = sanpham::paginate(5); return view('phantrang',['sanpham'=>$sanpham]); | {!! $sanpham->links() !!} |

Sử dụng Paginate

sanpham::paginate(5);

|  |  |
| --- | --- |
| Phương thức | Chức năng |
| paginate(Số tin trong 1 trang); | Phân trang |
| simplePaginate(Số tin trong 1 trang); | Phân trang không có số |
| paginate(5)->setPath('myurl/sanpham'); | Chỉnh đường dẫn phân trang |
| {!! $sanpham->links() !!} | Hiển thị phân trang trên view |
| {!! $sanpham->appends(['sort' => 'votes'])->links() !!} T | Thêm biến vào đường dẫn phân trang |
| {!! $sanpham->fragment('foo')->links() !!} | Thêm fragment vào sau đường dẫn |

**11.2. Cấu trúc thẻ phân trang**

Cấu trúc thẻ phân trang paginate();

<ul class = “pagination” >

<li class = “disabled”><span>«</span></li>

<li class = “active”><span>1</span></li>

<li><a href= “view?page=2”>2</a></li>

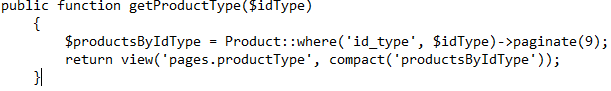
<li><a href= “view?page=3”>3</a></li>

<li><a href= “view?page=2” rel= “next”>»</a></li>

</ul>

Gọi pagitate trong controller với giá trị là số sản phẩm muốn hiển thị trên 1 trang

Ở ví dụ dưới, số sản phẩm muốn hiển thị trên 1 trang là 9



**Cấu trúc thẻ phân trang simplePaginate();**

<ul class= “pager”>

<li><a href = “<http://yourhost/view?page=4>” rel= “prev”>«</a></li>

<li><a href = “[http://yourhost/view?page=](http://yourhost/view?page=4)6” rel= “next”>»</a></li>

</ul>

Trang hiện tại được xác định theo giá trị của tham số ?page trên chuỗi truy vấn request HTTP

VD: gọi thanh phân trang ra trên 1 trang web

<div class="pagination row-fluid" align="center">{{$productType->links()}}</div>

***Chương 4: Ứng dụng minh họa***

**1.Website bán hoa áp dụng được các tính năng của laravel như**

- Sử dụng Auth

- Sử dụng lệnh để tạo database

php artisan make:model ten-database

- Sử dụng lệnh để tạo các controller

php artisan make:controller ten-controller

**2.Các kinh nghiệm rút ra được khi sử dụng laravel để xây dựng một trang web**

**3.Các mặt lợi, hại của laravel tự rút ra được**

**Chương 5: Bản phân công công việc**

