LARAVEL CƠ BẢN

# Giới thiệu

* Trang chủ: [https://laravel.com](https://laravel.com/)
* Document: <https://laravel.com/docs/7.x>
* Là 1 framework PHP mã nguồn mở, hoạt động theo mô hình MVC
* Là framwork được sử dụng phổ biến nhất tại thời điểm hiện tại
* Version mới nhất tại thời điểm dạy là 5.8
* Được tạo bởi Taylor Otwell
* Cần biết về OOP trước khi học laravel

# Cài đặt

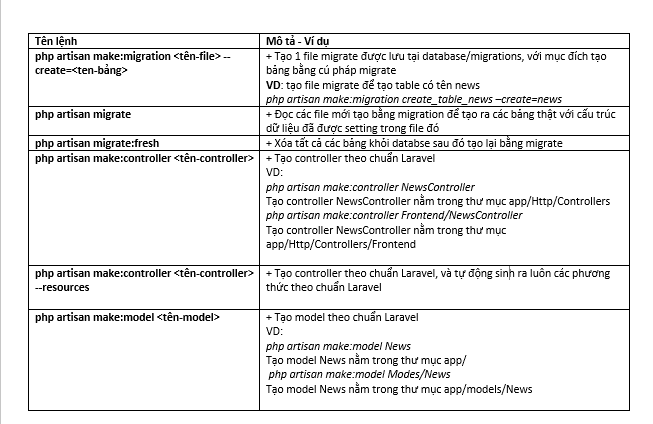
* Laravel ngoài code của bản thân nó thì còn sử dụng rất nhiều thư viện từ bên ngoài, do đó sẽ được cài đặt thông qua Composer

## Composer là gì

* Composer là một công cụ quản lý sự phụ thuộc trong PHP. Nó cho phép bạn khai
  + báo các thư viện mà dự án của bạn phụ thuộc vào và nó sẽ quản lý (cài đặt / cập
  + nhật) các thư viện cho bạn.
* Hãy tưởng tượng project của bạn có sử dụng thêm các library bên ngoài (kiểu như tích hợp các module từ 1 bên thứ 3 chẳng hạn), trong 1 ngày nào đó library ý có sự thay đổi (VD như bên thứ 3 cập nhật tính năng mới chẳng hạn), nếu bạn không sử dụng Composer, bạn sẽ phải copy thủ công các thay đổi đó, cập nhật vào module đó trên project của mình. Trong khi composer sẽ làm điều này thay bạn, chỉ bằng 1 vài dòng lệnh đơn giản
* Cài đặt: Vào link <https://getcomposer.org/>  để tại về file .exe để chạy trên Window, giữ nguyên các cấu hình mặc định
* Các libraries được update trong file composer.json, ngang hàng với file index.php gốc của ứng dụng

# Command line Artisan

* Là công cụ trên giao diện command line của Laravel, bao gồm các tập lệnh cho phép xây dựng ứng dụng web nhanh hơn
* Một số lệnh thông dụng



# Các phương thức cài đặt

* Phụ thuộc version PHP trên máy bạn là gì thì composer sẽ download version laravel tương ứng
* Do bản thân Laravel phụ thuộc các dependencies, nên nó sẽ sử dụng composer để cài đặt
* Cách 1: thông qua Laravel Installer, cần chạy 2 câu lệnh:

composer global require laravel/installer

laravel new <tên-thư-mục-project>

* Cách 2: thông qua lệnh create-project của Composer, cá nhân mình thường sử dụng cách này cho đơn giản

composer create-project --prefer-dist laravel/laravel <tên-thư-mục-project>

* OK – DONE ! Rất đơn giản, bạn đã có 1 project đồ sộ chỉ với 1 vài câu lệnh đơn giản như trên, đây cũng là 1 cách học framework, là học những cái có sẵn của nó

# Cấu trúc thư mục

# https://lh4.googleusercontent.com/XbB1CxoaQNVVIJpB6uNr1YoP0SSntEvbVFjdaRJZnT0lfG2RYkIdG7qAGOu2JcRtADOCxVRu79Rooxepb_7L9Rkzt63mzVcSambQkTi2NJvccAjIfcOrs1gCc_m3Ev35odJ7ntXZ

## Thư mục App

* Là thư muc ứng dụng, chứa toàn bộ source code của project, nó chứa events, excpertion và các khai báo midleware (giống như tươngf lửa)
* Chỉ cần chú ý tới thư mục Http, là thư mục chứa các class Controller của bạn. Thư mục Http này còn chứa các thư mục con sau
* Controllers: chứa các controllers của bạn
* Middleware: có thể hiểu như bộ lọc (tường lửa) từ các request gửi tới controller, cũng như các response trả về từ controller
* Requests: thư mục này mặc định chưa có, sẽ được tạo ra bằng lệnh artisan, có tác dụng check validate cho controller

## Thư mục Bootstrap

* Không phải bootstrap đâu nhé, đây là thư mục liên quan đến cache của ứng dụng, thường bạn sẽ ko đụng gì vào thư mục này

## Thư mục config

* Chứa các cấu hình của ứng dụng, được tách ra làm nhiều file với các chức năng riêng biệt, thường bạn cũng sẽ ít khi động vào thư mục này

## Thư mục database

* Không nên nhầm lẫn với cấu hình database (các khai báo username, password, db\_name), mà đây là thư mục chứa các thao tác liên quan đến csdl. Cụ thể các thư mục con như sau:
* Seeds: thêm dữ liệu thủ công vào các bảng, để chạy sử dụng lệnh ***php artisan db:seed***
* Migrations: tạo và xóa bảng bằng lệnh *artisan migrate*, khai báo các field của bảng trong phương thức up(), và khi rollback - hành động chạy ngược lại với phương thức up() - là phương thức down(). Ví dụ về tạo 1 migrate để tạo ra 1 bảng posts trong CSDL:
  + ***php artisan make:migration create\_table\_posts\_table --create=posts.***
* Factories: Cơ chế giống seed nhưng thêm khối lượng lớn dữ liệu

## Thư mục public

* Thư mục khởi tạo ứng dụng, chứa file index.php gốc của ứng dụng – hay đây chính là file index.php mà các bạn đã từng học trong mô hình MVC, đây là file đón nhận mọi request gửi tới
* Có các thư mục trong thư mục assets trước mình đã học là css, js, images, uploads, ckeditor, ckfinder

## Thư mục resources

* Thư mục quan trọng nhất là thư mục views, chính là lớp view của ứng dụng MVC mà các bạn đã học,
* Ngoài ra view của laravel sử dụng template là .blade.php, là 1 engine template để code nhìn sáng sủa hơn, thay vì trộn lẫn các mã html và php như trc, nên khi code trong view của Laravel các bạn sẽ gặp các ký tự @ thay cho <?php … như thông thường

## Thư mục routes

* Có thể hiểu đơn gian đây chính là file .htaccess dùng để rewrite lại url mà các bạn đã học trong mô hình MVC
* Là nơi bạn sẽ định nghĩa ra các url của hệ thống mà đã được rewrite, thay vì trước phải tự rewrite trong file .htaccess, thì Laravel cung cấp các khai báo có sẵn trong file web.php

## Thư mục storage

* Chứa các log liên quan khi ứng dụng của bạn đc chạy, thường bạn sẽ không đụng gì vào thư mục này

## Thư mục vendor

* Chứa các thư viện của bên thứ 3, do Laravel hoạt động dựa trên Composer, nên thư mục này sẽ tự động được sinh ra khi bạn chạy lệnh *composer install,* bạn sẽ không động gì vào thư mục này

# Luồng xử lý trong Laravel

# https://lh3.googleusercontent.com/3DdnSlZn1YYSq_ptpJlCYfk0uLYI6q6kiPGsKCBAf8NDb-f3LTRy54hzI0k88UvQykM2i5Kk_3b4D7LK_zLRpGNoXoqlvejrw6At6WnijW35NhNc71QD415NBXk0AstUAJIVhEtB

* Laravel hoạt động dựa trên mô hình MVC
* Mô tả hình trên

|  |  |
| --- | --- |
| **Folder / File** | **Mô tả** |
| User | User gửi yêu cầu. |
| Routing | Yêu cầu từ User sẽ được Routing điều hướng:   * Tới Controller để xử lý yêu cầu. * Tới Middleware để kiểm tra điều hướng cần thiết. * Tới thẳng View nếu không cần xử lý. |
| Middleware | Middleware có thể xem như bộ lọc HTTP từ request, VD như dùng để kiểm tra, xác thực người dùng đăng nhập vào hệ thống. Sau khi kiểm tra, xác thực xong sẽ trả về Controller để tiếp tục xử lý. |
| Controller | Controller được xem như trung tâm điều khiển của hệ thống, tất cả thao tác xử lý nên được thực hiện ở đây.   * Controller kết nối với Database thông qua điều khiển từ Model. Hoặc có thể thao tác trực tiếp tới Database thông qua các Query Builder * Kết quả xử lý sẽ được trả về view. |
| Model | Khi có yêu cầu từ Controller, Model sẽ tương tác với Database và trả kết quả về Controller, một số trường hợp cần thiết thì Model cũng có thể trả thẳng kết quả về View. |
| Database | Lưu trữ và chứa dữ liệu. |
| Migration & Seeding | * Migration, được dùng để tạo Database. * Seeding, được dùng để tạo dữ liệu ảo cho Database. |
| Query Builder | Câu truy vấn Database. |
| View | View nhận dữ liệu xử lý từ Controller (hoặc Model, Routing), hiển thị kết quả cho người dùng. |

# File cài đặt môi trường .env

* Thay vì phải vào thư mục config, hay vào các file để cấu hình, Laravel đưa ra file .env như là 1 nơi thống nhất để cài đặt cho môi trường, các biến trong file .env đã được ánh xạ (mapping) qua các constant (hằng số ) cấu hình tương ứng của thư mục config
* Lưu ý, do môi trường giữa các máy là khác nhau, nên file .env này sẽ được thêm vào file .gitignore và sẽ ko đc đẩy lên Git
* File .env.example là file backup cho file.env của bạn
* Biến $\_ENV là biến toàn cục mà Laravel tạo sẵn ra, chứa toàn bộ các thông tin trong file .env

# Chế độ maintainance (bảo trì) trong Laravel

* Chế độ bảo trì trong website, sử dụng artisan để thao tác

***php artisan down***

* Để thoát chế độ này sử dụng lệnh

***php artisan up***

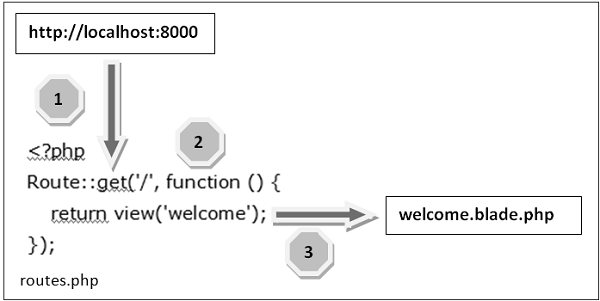
# Routing

* Trong Laravel, mọi request tới controller đều phải thông qua việc định nghĩa lại thông qua các route
* Nôm na là Laravel không cho phép chạy file PHP bất kỳ nào một cách tùy tiện bằng việc gõ tên file lên thanh địa chỉ trình duyệt, mà tất cả phải được điều khiển thông qua Routing.
* Laravel đã xử lý việc có hay ko dấu / ở trước tên route cho bạn
* Có thể hiểu cơ chế Routing trong Laravel có phần nào đó giống file .htaccess mà bạn đã thao tác
* Laravel quy định rất chặt chẽ việc sử dụng phương thức POST/GEt trong các trường hợp cụ thể. Ví dụ để hiển thị thì sẽ dùng GET, để thay đổi dữ liệu sẽ sử dụng POST, tuy nhiên tùy mục đích thì bạn hoàn toàn có thể sử dụng POST/GET sao cho phù hợp, tất nhiên điều này có thể làm sai lệch mục đích mà Laravel mong muốn
* File thao tác với routes nằm tại: **routes/web.php**
* Nếu ko muốn quy định phương thức, sử dụng Routes::any
* Giải thích về 1 khai báo route:

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

**return** view(**'welcome'**);

});



* Bước 1: Chạy url gốc của ứng dụng, là ‘/’
* Bước 2: Kiểm tra phương thức của ứng dụng là GET, sau đó chạy function callback - là phương thức sẽ chạy sau khi lệnh trước đó chạy xong - bên trong
* Bước 3: Trong function callback này, thực thi việc gọi ra 1 view có tên là welcome, đây chính là file welcome.blade.php nằm trong thư mục resoureces/views.
* VD1:

Route::*get*(**'/welcome/myname'**, **function** (){

**return** view(**'welcome/myname'**);

});

* VD2:

Route::*group*([**'prefix'** => **'/admin'**], **function** () {

   Route::*get*(**'/comments'**, **'NewsController@comment'**);

});

//đây là route dạng group, để tránh việc phải viết lại nhiều route mà có url bắt đầu bằng /admin. Ví dụ /admin/create, /admin/edit ...

## Route có tham số

* Ví dụ bài toán lấy chi tiết sản phẩm, ta cần phải biết id của sản phẩm đó, và như url trong MVC các bạn đã biết là nó có dạng index.php?controller=abc&action=def&id=2, thì route trong Laravel cũng có thể định nghĩa được như vậy với cú pháp như sau

Route::get('ID/{id}',function($id) {

   echo 'ID: '.$id;

});

## Route name

* Là chuẩn convention khi gọi tên route

Route::*get*(**'/user'**, **'Backend\UserController@register'**)->name(**"register"**);

Route::*post*(**'/user/save'**, **'Backend\UserController@save'**)->name(**"save"**);

* Nếu đường dẫn dài và khó nhớ, thì nên đặt route name thay thế

# Middleware

* Được hiểu như 1 firewall, đứng giữa request gửi tới và response trả về,  có tác dụng lọc . VD: bình thường luồng request/response của bạn đang chạy ok, khi có middleware vào nó đứng giữa và làm nhiệm vụ lọc xem có cho phép request gửi tới hay không, dựa vào 1 điều kiện gì đó mà bạn đã set sẵn
* Laravel cung cấp sẵn 1 middleware, kiểm tra xem user đã đăng nhập hay chưa, nếu chưa sẽ chuyển hướng về trang login
* Tạo middleware: php artisan make:middleware AgeMiddleware
* Sau khi tạo middleware, cần đăng ký nó để có thể sử dụng. Trong laravel có 2 phạm vi đăng ký là ở mức toàn cục (,mọi request-response đều phải qua) và mức route (chỉ các route nào mới bị ảnh hướng)
* Đăng ký trong app/Http/Kernel.php
* Tạo middleware bằng artisan:

*php artisan make:middleware <tên-middleware>*

VD: *php artisan make:middleware MyMiddleware*

* Ví dụ:
  + Thêm routes:

<?php

Route::get('/usercontroller/path',[

   'middleware' => 'First',

   'uses' => 'UserController@showPath'

]);

* Tạo middleware

php artisan make:middleware FirstMiddleware

* Đăng ký middleware trong Kernel.php

**protected $routeMiddleware** = [

**'First'** => FirstMiddleware::**class**

];

* Middleware có thể được sử dụng trong route hoặc trong hàm khởi tạo của controller
* Trong route:

Route::*get*(**'testing'**, [

**'middleware'** => **'auth'**,

**'uses'** => **'TestController@testing'**

]);

* Trong hàm khởi tạo

**<?php**

**namespace** App\Http\Controllers;

**use** Illuminate\Http\Request;

**class** TestController **extends** Controller

{

**public function** \_\_construct()

 {

   $this->middleware(**'auth'**);

 }

*//*

**public function** index() {

**echo "Đây là test controller"**;

 }

**public function** test() {

**echo "testing"**;

 }

**public function** login() {

**echo "login"**;

 }

}

# Namespace

* Được sử dụng như 1 định danh duy nhất cho tên class, vì có thể xảy ra trường hợp có nhiều class trùng tên, nên namespace sẽ phân biệt được các trường hợp này
* Sử dụng từ khóa **use (**tương tự như chức năng import file mà bạn đã học**)**

**use** Illuminate\Http\Request;

# Controller

* Chữ C trong mô hình MVC
* Nên tạo controller bằng công cụ artisan của Laravel, để có đc đúng dạng controller chuẩn
* Controller sau khi được tạo sẽ nằm tại app/Http/Controllers
* Controller sẽ được gọi thông qua route tại routes/web.php
* Tạo với artisan php artisan ***make:controller <tên-controller>***

## Restful Resource Controller

* CRUD là chức năng căn bản của 1 ứng dụng web, Laravel có cú pháp artisan rất thuận tiện cho việc sinh ra các routes cho toàn bộ các phương thức này của controller, hay còn gọi là resource controller
* Ví dụ:
* Tạo controller

*php artisan make:controller MyController –resource*

* File app/Http/Controller/MyController.php

Sau khi chạy xong, sẽ sinh ra 1 controller với tên MyController đã có sẵn các phương thức, đây chính là controller dạng resource của Laravel

**class MyController extends Controller**

**{**

***//***

**public function index() {**

**echo 'index';**

**}**

**public function create() {**

**echo 'create';**

**}**

**public function store(Request $request) {**

**echo 'store';**

**}**

**public function show($id) {**

**echo 'show';**

**}**

**public function edit($id) {**

**echo 'show';**

**}**

**public function update(Request $request, $id) {**

**echo 'update';**

**}**

**public function destroy($id) {**

**echo 'destroy';**

**}**

**}**

* Thêm route dạng resource tại routes/web.php:

Route::*resource*(**'my'**, **'MyController'**);

Khi đó các url để truy cập liên quan đến MyController sẽ như sau

|  |  |  |  |
| --- | --- | --- | --- |
| **Verb** | **Path** | **Action** | **Route Name** |
| GET | /my | index | my.index |
| GET | /my/create | create | my.create |
| POST | /my | store | my.store |
| GET | /my/{my} | show | my.show |
| GET | /my/{my}/edit | edit | my.edit |
| PUT/PATCH | /my/{my} | update | my.update |
| DELETE | /my/{my} | destroy | my.destroy |

# Request

* Là tất cả các thông tin gửi gửi đến (request) ứng dụng của bạn, Laravel sinh ra class Request để bắt được toàn bộ các thông tin này
* Bạn có thể truyền class này như 1 tham số vào mọi phương thức của bạn, vì nó đã đc Laravel xây dựng sẵn
* Trong Laravel, sử dụng class Request này để lấy toàn bộ các thông tin chung từ POST, GET, FILES

## Lấy thông tin về URL

* Lấy mọi thông tin liên quan đến url
* Ví dụ
* Tạo class UrlController

*php artisan make:controller UrlController*

* Code cho UrlController:

**class** UrlController **extends** Controller

{

**public function** index(Request $request) {

*//Cach su dụng phuong thuc path*

$path = $request->path();

**echo "<br />Path: "** . $path;

*//Cach su dung phuong thuc is*

$pattern = $request->is(**'foo/\*'**);

**echo "<br /> is Method:** $pattern**"**;

*//cach su dung phuong thuc url*

$url = $request->url();

**echo "<br />URL method:** $url**"**;

 }

}

## Lấy thông tin về input form

* Không cần quan tâm đến phương thức POST hay GET, Laravel sử dụng class Request này để xử lý chung cho cả 2 phương thức, sử dụng phương thức input()
* Có thể lấy các thông tin này thông qua đối tượng của lớp Request hoặc phương thức request()
* VD:
* Tạo routes tại routes/web.php:

Route::*get*(**'/signup'**, **'UserRegistrationController@signup'**);  
Route::*post*(**'/user/register'**, **'UserRegistrationController@postRegister'**);

* Tạo view resources/views/register.blade.php:

<**html**>

<**head**>

   <**title**>Form Example</**title**>

</**head**>

<**body**>

<**form action = "user/register" method = "post"**>

   <**input type = "hidden" name = "\_token" value = "<?php echo** csrf\_token() **?>"**>

   <**table**>

       <**tr**>

           <**td**>Name</**td**>

           <**td**><**input type = "text" name = "name"** /></**td**>

       </**tr**>

       <**tr**>

           <**td**>Username</**td**>

           <**td**><**input type = "text" name = "username"** /></**td**>

       </**tr**>

       <**tr**>

           <**td**>Password</**td**>

           <**td**><**input type = "text" name = "password"** /></**td**>

       </**tr**>

       <**tr**>

           <**td colspan = "2" align = "center"**>

               <**input type = "submit" value = "Register"** />

           </**td**>

       </**tr**>

   </**table**>

</**form**>

</**body**>

</**html**>

* Tạo UserRegistrationController:

*php artisan make:controller UserRegistration*

* Code cho class UserRegistrationController.php:

**<?php**

**namespace** App\Http\Controllers;

**use** Illuminate\Http\Request;

**class** UserRegistrationController **extends** Controller

{

*//*

**public function** signup() {

**return** view(**'register'**);

 }

**public function** postRegister(Request $request) {

   $name = $request->input(**'name'**);

**echo 'Name: '** . $name;

**echo '<br />'**;

   $username = $request->**username**;

**echo 'Username: '** . $username;

**echo '<br />'**;

   $password = $request->**password**;

**echo 'Password: '** . $password;

**echo '<br />'**;

 }

}

* Chạy đường dẫn /signup, submit form và xem kết quả

# Session

# Laravel sử dụng session thông qua các phương thức đc cung cấp sẵn

# Các thao tác với session:

* Set session:
  + session()->put(**key**, **value**);
  + VD: session()->put(**"manh123"**, **"12121"**)
* Get session:
  + session()->get(**key**);
  + VD: session()->get(**"manh123”**);
* Get all session:
  + session()->all();
* Xóa session:

session()->forget(**key**);

VD: session()->forget(**'manh123'**);

Xóa all: session()->flush();

# Cookie

* Laravel sử dụng cookie thông qua các phương thức đc cung cấp sẵn, ngoài ra vẫn có thể truy cập biến $\_COOKIE của PHP như bình thường
* Các thao tác với cookie:
* Set cookie

$cookie = cookie(**'name'**, **'123'**, **'30'**);

Cookie::**queue**($cookie);

* Get cookie:

Cookie::*get*(**'abc'**);

# Views

* Chữ V trong mô hình MVC
* Được xác định tại resources/views
* Views trong Laravel sử dụng template blade, nên đuôi file sẽ có dạng blade.php
* Ví dụ:
* Tạo view resources/views/test.blade.php

<**html**>

<**body**>

<**h1**>Hello, World</**h1**>

</**body**>

</**html**>

* Tạo route

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

**return** view(**'test'**);

});

## Truyền tham số qua view

* Route:

Route::*get*(**'/test-param'**, **function** (){

**return** view(**'test-param'**, [

**'name'** => **'Manh'**

]);

});

* Views resources/views/test-param.blade.php

<**html**>

<**body**>

<**h1**>Hello **<?php echo** $name**?>**, World</**h1**>

<**h1**>Hello {{ $name }}, World</**h1**>

</**body**>

# </html>

## Blade template

* Là 1 engine template được Laravel áp dụng
* Nhìn code rõ ràng sạch sẽ hơn
* Blade template sinh ra để hạn chế sử dụng các thẻ php ở trong file, ví dụ các thẻ <?php, <?php if(): ?>... mà sẽsử dụng cú pháp thay thế của nó, thông thường là ký tự @ ở đầu
* Hỗ trợ cơ chế cache, ko load lại view cho đến khi có sự thay đổi
* Đuôi file có dạng .blade.php
* So sánh một vài cấu trúc tương đồng giữa PHP và Blade

|  |  |
| --- | --- |
| **PHP** | **Blade** |
| <?php echo $name ?> | {{ $name }} |
| <?php include 'footer.php' ?> | {{ @include('footer.php') }} |
| if(điều\_kiện):      câu\_lệnh  elseif(điều\_kiện):      câu\_lệnh;  else:      câu\_lệnh;  endif; | @if(điều\_kiện)      câu\_lệnh;  @elseif(điều\_kiện)      câu\_lệnh;  @else      câu\_lệnh;  @endif; |
| for ($i = 0; $i < 10; $i++) {      câu\_lệnh;  } | @for ($i = 0; $i < 10; $i++)      câu\_lệnh;  @endfor |
| while (true) {      câu\_lệnh;  } | @while (true)      câu\_lệnh;  @endwhile |
| foreach ($users as $user) {      câu\_lệnh;  } | @foreach ($users as $user)      câu\_lệnh;  @endforeach |

## Layout

* Khi truy cập vào hầu hết các website, nếu các bạn để ý một chút thì chúng ta sẽ thấy khi ta duy chuyển hoặc truy cập nhiều phần hoặc nhiều trang khác nhau, thì sẽ có một số thành phần nó sẽ giống nhau hoàn toàn và gần như không thay đổi ở tất cả các trang khác nhau như Header, Footer, Menu… Có bao giờ các bạn tự hỏi liệu người ta có phải định nghĩa và khai báo các thành phần này ở tất cả các trang? Chắc chắn là không rồi các bạn ạ, không ai đi thiết kế web mà đi làm như vậy cả, mà người ta sẽ khai báo 1 file layout để chứa các thành phần dùng chung và ở mỗi trang view chúng ta chỉ cần gọi lại layout đó mà không cần phải viết lại hay định nghĩa lại. Để hiểu rõ hơn thì chúng ta sẽ đi làm 1 ví dụ nhé.
* Layout trong Laravel sử dụng 1 số biến kiểu dạng tham số, để khi các view con gọi layout thì sẽ phải set giá trị thật cho các tham số này. Một số tham số trong layout có thể gặp:
  + @yield có thể coi như lệnh đặt chỗ trước, sẽ lấy những phần ở nơi khác gắn vào đây.
  + @extends cho biết sẽ kết nối tới file nào
  + Tại các view con, nội dung bên trong @section sẽ được gắn vào @yield tương ứng
* Ví dụ
* Tạo thư mục layouts (có thể là bất cứ tên gì) trong views/resources
* Trong thư mục layouts này, tạo file với tên là master.blade.php, đây sẽ là file layout chính của ứng dụng, nội dung như sau

<**html**>

<**head**>

   <**title**>Laravel - **@yield('title')**</**title**>

</**head**>

<**body**>

<**div class="header"**>

   This is my header

</**div**>

<**div class="main-content"**>

**@yield('content')**

</**div**>

<**div class="footer"**>

   This is my header

</**div**>

</**body**>

</**html**>

* Tạo file layout-test.blade.php trong views/resources, trong file view này sẽ extend layout mới xây dựng ở trên, sử dụng từ khóa @extends như sau

**@extends('layouts.master')**

**@section('title'**, **'This is title')**

**@section('content')**

Content của tôi

**@endsection**

# Redirect

* Phương thức chuyển hướng, tương đương với hàm header(‘Location: ’) trong PHP
* Có thể chuyển hướng theo route name hoặc url
* VD1:

redirect(**'home'**);

redirect()->route(**'route.destination'**);

* VD2: chuyển hướng theo route name
* web.php

Route::*get*(**'destination'**, **'TestController@destination'**)->name(**'route.destination'**);

* Controller:

redirect()->route(**'route.destination'**);

# Làm viêc với Database

* Trong Laravel, các thông tin kết nối tới CSDL sẽ được setting tại file .env của ứng dụng
* Nhập thông tin kết nối tại file .env, với các thông tin kết nối phụ thuộc vào hệ thống của bạn, giả sử như sau

**DB\_CONNECTION**=**mysql  
DB\_HOST**=**127.0.0.1  
DB\_PORT**=**3306  
DB\_DATABASE**=**laravel\_demo  
DB\_USERNAME**=**root  
DB\_PASSWORD**=

* Trong Laravel có 2 cơ chế truy xuất CSDL là QueryBuilder và Eloquent ORM
* QueryBuilder: có thể hiểu 1 cách đơn giản như 1 câu query thuần mà các bạn đã học, sử dụng class DB có sẵn của LAravel làm tiền để để thao tác
* Eloquent ORM: truy cập CSDL dựa theo model, với cách thức này thì mỗi bảng trong CSDL sẽ tương đương với 1 model của mô hình MVC
* Tùy mục đích mà có thể sử dụng QueryBuilder hoặc Eloquent để thao tác với CSDL, về mặt cơ bản Eloquent thì có tính bảo mật hơn QueryBuilder, Eloquent thì dễ sử dụng hơn do cách viết đơn giản, ngắn gọn và mang tính gợi nhớ hơn vì sử dụng model để tương tác, QueryBuilder tốc độ truy cập sẽ nhanh hơn Eloquent vì nó thực hiện truy cập thẳng vào CSDL mà ko cần thông qua 1 class model như Eloquent
* Bảng sau so sánh 2 cách viết của QueryBuilder và Eloquent:

|  |  |
| --- | --- |
| **QueryBuilder** | **Eloquent ORM** |
| Lấy tất cả bản ghi | |
| $news = DB::table(‘news')->get(); | $news = News::all() |
| Lấy 1 bản ghi theo id | |
| $new = DB::table('news')->where('id', 1)->first(); | $new = News::find(1); |
| Lấy 1 trường của 1 bản ghi | |
| $new = DB::table('news')->where("id", 1)->select('title')->first(); | News::find(1)->select('title')->first() |
| Lấy 1 trường của tất cả bản ghi | |
| $new = DB::table('news')->select("title")->get(); | $news = News::all('title'); |
| Insert bản ghi | |
| $isInsert = DB::table('news')->insert([  'title' => 'new'  ]); | $newModel = new News();  $newModel->title = "Title insert";  $newModel->save(); |
| Update bản ghi | |
| $isUpdate = DB::table('news')->where('id', 1)->update([  'title' => 'title new'  ]); | $news = News::find(1);  $news->title = "Title update";  $news->save(); |
| Xóa bản ghi | |
| $isDelete = DB::table('news')->where('id', 1)->delete(); | $news = News::find(1);  $news->delete(); |
| Hàm count và max | |
| $count = DB::table('news')->count();  $max = DB::table('news')->max('id'); | $count = News::count();  $max = News::max('id'); |

# Validation

* Laravel cung cấp phương thức validate để check dữ liệu đầu vào, dựa vào các rules mà nó quy định
* Danh sách các rule có sẵn trong Laravel có thể tham khảo tại: <https://laravel.com/docs/7.x/validation#available-validation-rules>

**public function** validateForm(Request $request) {

   $this->validate($request, [

**'username'** => **'required|max:8'**,

**'password'** => **'required|'**

]);

 }

}

# Laravel cũng tự động sinh ra biến toàn cục $errors, chứa toàn bộ thông tin về error validate trong Laravel, việc cần làm của bạn là chỉ cần hiển thị biến này ra view để show lỗi validate cho người dùng

# Migrate

* Migration là một công cụ rất hay của Laravel, cho phép tạo các bảng, các cột cơ sở dữ liệu (Database) bằng các dòng lệnh PHP, ngoài ra có thể dễ dàng cập nhật nội dung các bảng, các cột đã tồn tại.
* Migration giúp cho việc đồng bộ Database giữa các môi trường khá thuận lợi, thay vì muốn thêm một bảng hay một cột trong các môi trường, thì ta phải làm thủ công từng môi trường, thì với Migration bạn chỉ cần thực hiện bằng dòng lệnh đơn giản.
* Điểm tốt của Migration là làm việc được với nhiều cơ sở dữ liệu như: MySQL, SQL Server, PostgresSQL, ...
* Migration được đặt bên trong thư mục */database/migrations/*.
* Để Migration hoạt động, chúng ta cần kết nối với cơ sở dữ liệu
* Cú pháp tạo migrate:

php artisan make:migration create\_news\_table

* php artisan - Công cụ hỗ trợ viết command line tích hợp sẵn trong Laravel, sẽ còn gặp lại nhiều.
* make:migration - Lệnh tạo Migration.
* create\_news\_table - Tên Migration do mình tự đặt.

* Sau khi chạy, sẽ sinh ra 1 file tương ứng, trong file này cần quan tâm là 2 function up() và down():
* public function up() dùng để thêm, bớt, thay đổi, ... nội dung bảng cơ sở dữ liệu - để thực thi ta cần sử dụng lệnh php artisan migrate
* public function down() dùng phục hồi hay xóa bảng, ... - để thực thi ta cần sử dụng lệnh php artisan migrate:rollback