

Contents

Bài 1: Cài đặt Laravel , cấu trúc thư mục của Laravel.....	3
1.Cài đặt Laravel	3
2.Giới thiệu cấu trúc thư mục trong Laravel	4
Bài 2: Tìm hiểu Route, Controller	4
1. Cấu trúc của Route	4
2.Controller	6
Bài 3: Gửi nhận dữ liệu với Request và Responses	7
1.Request, responses là gì?.....	7
2.Làm việc với URL sử dụng Request	7
3.Gửi nhận tham số trên Request.....	8
4. Sử dụng Cookie với request và response	9
5.Upload file.....	9
6. Trả về dữ liệu dạng JSON	11
Bài 4: Views.....	11
1.Views là gì?.....	11
2. Cách gọi Views	12
3. Cách truyền tham số sang View	12
Bài 5: Blade Template	13
1.Blade Template là gì?.....	13
2. Tạo template cho trang web	14
3. Các câu lệnh sử dụng trong Blade template	17
Bài 6: Làm việc với Database	18
1.Thao tác chuẩn bị cơ sở dữ liệu.....	18
2.Tạo, sửa, xóa bảng bằng Schema	19
3.Quản lý dữ liệu với Migrations	21
4.Tạo dữ liệu mẫu với seed	24
5. Query Builder.....	25
6. Eloquent – Model.....	28
Bài 7: Kiểm soát Route với Middleware	33

1. Kiểm soát Route với middleware	33
2. Tạo và sử dụng Middleware	34
Bài 8: Authentication trong Laravel	36
1. Auth là gì	36
2. Sử dụng Auth để thực hiện đăng nhập, đăng xuất.....	37
3. Custom Auth mặc định của Laravel	39
Bài 9: Validation trong Laravel	44
1. Validation là gì?	44
2. Validation trực tiếp trên controller	44
3. Tạo Validation với FormRequest	46
Bài 10: Làm việc với Session	48
1. Cấu hình Session	48
2. Thiết lập và truy xuất Session	48
Bài 11: Phân trang (Pagination)	49
Bài 12: Shopping cart.....	50
1. Shopping Cart trong Laravel	50
2. Các bước cài đặt package vào project	51
3. cách sử dụng package shopping cart	52
Bài 13: RESTful Controller	54
1. Khái niệm ban đầu về API và Web API.....	54
2. Web API.....	55
3. RestFul Controller	55

Bài 1: Cài đặt Laravel , cấu trúc thư mục của Laravel

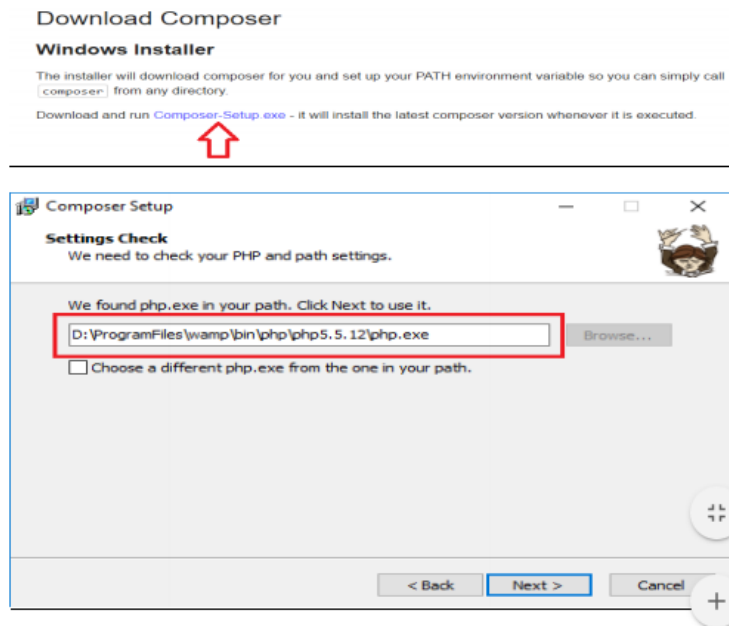
1.Cài đặt Laravel

Để cài Laravel, các bạn sẽ đi qua 2 bước :

Bước 1: Cài đặt Composer

bạn vào trang <http://getcomposer.org/> để dowload composer.

Trong quá trình cài đặt, bạn sẽ được composer báo chọn đến file php.exe trong xamp (hoặc wamp)



Sau khi cài đặt xong, composer có yêu cầu bạn làm các bước sau:

1. Mở cmd.exe
2. Đóng tất cả các cửa sổ windows lại (bao gồm cả cmd.exe)
3. Mở lại cmd.exe
4. Đóng lại rồi logout ra khỏi windows , sau đó login lại.
5. Cuối cùng ta bật cmd.exe lên là xong.

→ xong các bước trên tức là bạn đã cài xong composer rồi đó, cái composer là để làm môi trường để bạn download Laravel phiên bản mới nhất về.

Bước 2: Cài đặt Laravel

truy cập vào thư mục muốn cài đặt Laravel
(thường là C:/xamp/htdocts/)

Nhấn Shift+click chuột phải → chọn open command window để mở cửa sổ cmd lên.

gõ vào dòng code sau:

```
composer create-project --prefer-dist laravel/laravel Tên_project →enter
```

chờ vài phút để hệ thống download Laravel về máy của bạn.

2. Giới thiệu cấu trúc thư mục trong Laravel

- + App\Http\Controller : Chứa các bộ điều khiển controller
 - + Routes/web.php: là chứa các điều hướng route của web
 - + Config : Chứa các file cấu hình cho hệ thống.
 - + Database : Nơi chúng ta cấu hình các bộ dữ liệu mẫu : migrate, seed.
 - + Public : Nơi lưu trữ các thư viện CSS, JavaScript, các hình ảnh.
 - + Resources\Views : Lưu trữ các file giao diện mã html views.
 - + File .env: Cài đặt liên kết tới database cho hệ thống.
-

Bài 2: Tìm hiểu Route, Controller

1. Cấu trúc của Route

A) Route là gì?

Đúng như tên gọi của nó. Route thực hiện chức năng định tuyến, dẫn đường cho các HTTP request được gửi đến đúng nơi ta muốn nó đến. Với các ứng dụng web ngày nay, việc làm cho ứng dụng có chức năng tốt – giao diện đẹp là một chuyện, nhưng để có một trang web thực sự tốt thì “đường dẫn thân thiện” là không thể thiếu.

Ví dụ bạn có 1 trang web thể hiện danh sách xe hơi của hãng Honda:

`http://showrom.com/category.php?vendor=honda&type=car`

Thay vì để một đường dẫn ngớ ngẩn như vậy, chúng ta cần tạo ra 1 trang web có thể chạy dưới đường dẫn “đẹp” kiểu như:

`http://showrom.com/car/honda`

Rõ ràng đường dẫn đẹp thì có lợi cho người dùng, rất hiệu quả về mặt marketing cũng như SEO. Nhưng để tạo được đường dẫn đẹp như vậy thường là nỗi đau khổ khi viết bằng cú pháp trong .htaccess. Nhằm bắt được điều này, Laravel đã sớm đưa ra tính năng routing để tạo ra các đường dẫn đẹp và đơn giản, mà hiệu quả như một phép màu.

tham khảo: <http://webfaver.com/php-coding/laravel-5/co-ban-ve-route.html>

B) Cấu trúc một Route

Bạn sẽ định nghĩa hầu hết các route của mình trong file routes/web.php (laravel 5.4)

Phương thức	Đường dẫn	Tham số
Route::get	'home/laravel'	function(){} ;

Trong đó các phương thức bao gồm get(), post(), put(), delete() chính là các HTTP METHOD dùng cho route. Tham số thứ nhất – URL – là phần phía sau domain trang web. Tham số thứ

2 là hàm nặc danh, thực hiện xử lý cho từng route. Thử truy cập vào địa chỉ http://localhost/Ten_Project/public bạn sẽ thấy trang chủ Laravel hiện lên

Một số cách viết Route cơ bản

1.Route mac dinh

```
//URL: http://localhost:81/MyLaravel/public/
Route::get('/', function () {
    return view('welcome');
});
```

2.Route tự tạo

```
//URL: ../Hello
Route::get('Hello', function() {
    return "Xin chao cac ban!";
});

//URL: ../Hello/Index
Route::get('Hello/index', function() {
    echo "<h1>Hello index</h1>";
});
```

3.Router có truyền tham số

```
//URL: ../GioiThieu/Nguyen Van A
Route::get('GioiThieu/{ten}', function($ten) {
    echo "Ten của bạn là:". $ten;
});

//tham số có điều kiện where
Route::get('DiaChi/{dchi}', function($dchi) {
    echo "Địa chỉ của bạn là:". $dchi;
})->where(['dchi'=>'[a-zA-Z]+']); //ĐK là địa chỉ phải ở dạng ký tự (quy định bởi biểu thức Regular Expression)
```

4.Đặt tên định danh(hay gọi là tên của route)

```
//URL: ../GoiTen
Route::get('Route1', function() {
    echo "Đây là Route1";
})->name('TenDinhDanh');

Hoặc
Route::get('Route1', ['as'=>'TenDinhDanh', function() { return "Đã đổi tên"; }]);

//gọi Route thông qua tên của nó
Route::get('GoiTen', function() {
    return redirect()->route('TenDinhDanh');
});
```

5.Route Group

```
//URL: ../MyGroup/User2
Route::group(['prefix'=>'MyGroup'], function() {
    Route::get('User1', function() {
        echo 'User1';
    });
    Route::get('User2', function() {
        echo 'User2';
    });
    Route::get('User3', function() {
        echo 'User3';
    });
});
```

Link tham khảo: <https://khoapham.vn/download/laravel/bai2.pdf>

2.Controller

A) controller là gì

Thay vì định nghĩa tất cả các xử lý trong routes/web.php, bạn có thể tổ chức logic hợp lý, gọn gàng hơn bằng cách sử dụng Controller trong Laravel 5. Mục tiêu của Controller là gom nhóm các xử lý từ request HTTP vào chung trong một class. Các Controller được lưu trữ tại thư mục app/Http/Controllers

Cách tạo một Controller mới trong Laravel

vào thư mục của laravel nhấn Shift+ click chuột phải --> Open command window gõ vào:
php artisan make:controller Tên_controller --> Enter để hoàn tất.

B) sử dụng Controller

trong controller ta thêm dòng: `use app\Http\Controller\Auth;` ở đầu file để link tới file Mycontroller của ta (có thể ko cần)

Gọi Controller từ Route

Route	Controller
<code>Route::get('GoiController','MyController@XinChao');</code> Tức là nó sẽ gọi phương thức XinChao trong controller của ta là MyController	<code>public function XinChao(){ echo "chào các bạn"; }</code> xuất ra dòng chữ: chào các bạn

Gọi và truyền tham số từ Route sang Controller

Route	Controller
-------	------------

```
Route::get('KhoaHoc/{tenKH}', 'MyController@KhoaHoc');
```

Tức là nó sẽ gọi phương thức KhoaHoc trong MyController

URL... KhoaHoc/Laravel

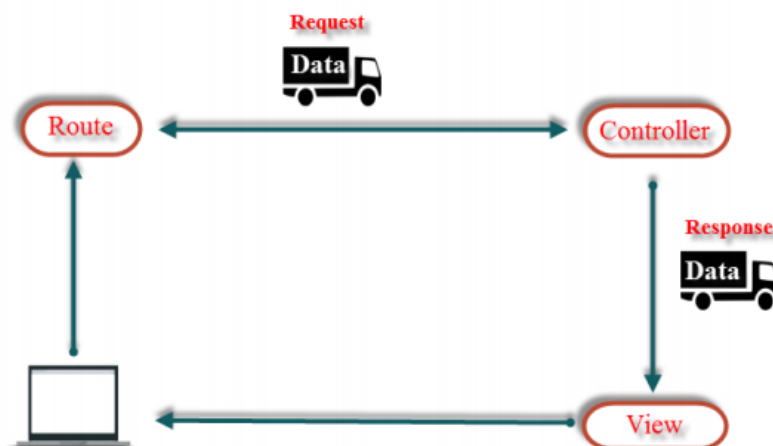
```
public function KhoaHoc($tenKH){  
    echo "Tên khoá học là:". $tenKH;  
}
```

Tên khoá học là:laravel

Bài 3: Gửi nhận dữ liệu với Request và Responses

1.Request, responses là gì?

Hiểu đơn giản thì request là gửi yêu cầu từ client lên server và response là server trả kết quả về cho client.



2.Làm việc với URL sử dụng Request

Để sử dụng Request ta khai báo đầu file controller của bạn dòng code sau:

```
use Illuminate\Http\Request;
```

Route

Controller

```
Route::get('goi-Controller','MyController@GetURL');
```

URL: ../goi-Controller

```
//học REQUEST
public function GetURL(Request $request){
    //return $request->path();//lấy đường dẫn cục bộ
    //return $request->url(); //lấy toàn bộ đường dẫn

    //kiểm tra URL có từ 'goi' hay không
    /*if($request->is('goi*'))
        echo 'có xuất hiện từ goi trong URL';
    else
        echo 'không tồn tại từ goi trong URL';*/

    //kiểm tra phương thức gửi qua là gì
    if($request->isMethod('post')){
        echo 'đây là phương thức POST';
    }
    else
        echo 'no method';
}
```

3. Gửi nhận tham số trên Request

```
//gọi giao diện view
//URL: ../getForm --> gọi từ View/postForm.blade.php
Route::get('getForm',function(){
    return view('postForm');//tự hiểu là postForm.blade.php
});
/*sau khi nhập form xong nhấn submit nó sẽ gọi Route postForm,
lúc này Route postForm sẽ gọi phương thức postForm trong MyController và in ra họ tên*/
Route::post('postForm',['as'=>'postForm','uses'=>'MyController@postForm']);
```

Route đầu tiên thực thi → gọi view (là trang `view/postForm.blade.php`)

```
<input type="hidden" name="_token" value="{{ csrf_token() }}"><!--dòng 1-->

<form action="{{route('postForm')}}" method="post">

    <input type="hidden" name="_token" value="{{ csrf_token() }}"><!--dòng 2-->

    <input type="text" name="hoten">
    <input type="submit">
</form>
```

Đối với Laravel 5.x khi xử lý form thì bắt buộc phải có `csrf_token` (tránh tấn công xss) nếu không thì sẽ không thực hiện submit form được. Để tạo `csrf_token` cho form chúng ta thêm 2 dòng code được đánh dấu <dong 1> và <dong 2>

ở đây action của form là `{{route('postForm')}}` điều này nghĩa là sau khi submit form dữ liệu sẽ được gửi về Route có name là `postForm`.

Tại Route `postForm` ta sẽ gọi controller

File: MyController


```
public function postForm(Request $request){
    if($request->has('hoten')) //kiểm tra bạn đã nhập vào biến hoten chưa
        echo $request->hoten;
    else
        echo "bạn chưa nhập họ tên";
}
```

Một số phương thức mở rộng

Phương thức	Chức năng
<code>\$request->has('name');</code>	Kiểm tra tham số name có tồn tại không.
<code>\$request->input('id');</code>	Nhận dữ liệu từ thẻ <code><input name='id' ></code>
<code>\$request->input('products.0.name');</code>	Nhận dữ liệu từ mảng
<code>\$request->input('user.name');</code>	Nhận dữ liệu từ JSON dạng mảng
<code>\$request->all();</code>	Nhận hết dữ liệu, lưu thành dạng mảng.
<code>\$request->only('age');</code>	Chỉ nhận tham số age
<code>\$request->except('age');</code>	Nhận tất cả tham số ngoại trừ tham số age

4. Sử dụng Cookie với request và response

Để sử dụng được Response ta phải khai báo thư viện cho nó trong file MyController của ta
 use Illuminate\Http\Response;

file Route/web.php

```
//-----COOKIE-----
//1. sử dụng Cookie với Request và response
Route::get('setCookie', 'MyController@setCookie');//thiết lập cookie
Route::get('getCookie', 'MyController@getCookie');//lấy Cookie
```

File: Mycontroller.php

```
//sử dụng Cookie với Request và response
public function setCookie(){
    $response=new Response();
    $response->withCookie('KhoaHoc','Laravel co ban',1);//thời gian của cookie là 1 phút
    echo 'bạn đã thiết lập cookie thành công, hãy thay đổi đường dẫn thành getCookie để thấy kết quả';
    return $response;
}
public function getCookie(Request $request){
    echo 'cookie của bạn là: ';
    return $request->cookie('KhoaHoc');
}
```

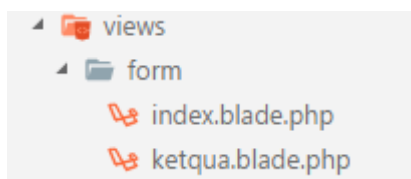
ở đây `$response->withCookie("tên-cookie","giá-trị", thời-gian-sống-cho-cookie);`
 thời gian sống của Cookie có đơn vị là phút

`$request->cookie("name");` → lấy giá trị của cookie có tên đó.

5.Upload file

```
Route::get('Form/index', function(){
    return view('form.index');
});
Route::post('Form/confirm', 'MyController@getform')->name('confirm');
```

(1) Route Form/index sẽ gọi view **index.blade.php**



```
<form action="{{route('confirm')}}" method="post" enctype="multipart/form-data">
    {{ csrf_field() }}
    <p><input type="text" name="hoten"></p>
    <p><input type="file" name="hinh"></p>
    <input type="submit" value="Submit">
</form>
```

View này sẽ hiển thị form nhập liệu. Sau nhập liệu xong nhấn submit sẽ gọi route Form/confirm

(2) Route Form/confirm sẽ gọi phương thức getform trong controller

đây là Mycontroller

```
use Illuminate\Http\Request;
```

```
class MyController extends Controller
```

```
{
    public function getform(Request $request){
        if($request->hasFile('hinh')){
            //upload file
            $file=$request->file('hinh');
            $file->move('images',$file->getClientOriginalName('hinh'));

            return view('form.ketqua',['result'=>$request]);
        }
    }
}
```

Trong phương thức getform ta đã lấy file upload rồi di chuyển vào thư mục public/**images** (thư mục images do ta tự tạo để lưu trữ hình ảnh).

move(Vị trí lưu hình, tên hình)

Sau đó ta gọi view **ketqua.blade.php**

```
<div></div>
<div>{{$result->hoten}}</div>
```

← → ↻ localhost:81/testLaravel/public/Form/confirm



ABC

Hàm **asset()** → sẽ trở đến thư mục public

getClientOriginalName() → lấy tên file

Mở rộng

Phương thức	Chức năng
<code>getClientSize('myFile')</code>	Trả về dung lượng của file , tính theo bytes
<code>getClientMimeType('myFile')</code>	Trả về kiểu của file : image/png
<code>getClientOriginalName('myFile')</code>	Trả về tên của file
<code>getClientOriginalExtension('myFile')</code>	Trả về đuôi của file : png
<code>isValid('myFile')</code>	Kiểm tra upload file có thành công hay không

6. Trả về dữ liệu dạng JSON

File: Route/web.php

```
Route::get('getJson','MyController@getJson');
```

File: MyController.php

```
public function getJson(){  
    $arr=['laravel','php','ASP.NET'];  
    return response()->json($arr);  
}
```

→ kết quả in ra màn hình mảng arr trên

Bài 4: Views

1.Views là gì?

Là các file có đuôi **.php**, chứa mã nguồn **html**, hiển thị dữ liệu cho người dùng xem và được lưu tại thư mục **resources/views** trong Laravel.



2. Cách gọi Views

+ gọi view từ Route:

Cài đặt Route	Resources/views/myView.php
<pre>Route::get('myView', function(){ return view('myView'); });</pre>	<pre><h1>Here is my view</h1></pre>
Kết quả : Here is my view	

+ Gọi Views từ Controller

File:Route/web.php

```
Route::get('view1','MyController@view1');
```

File:MyController.php

```
public function view1(){
    return View('trang1');//tự hiểu là trang1.php
}
```

Trong thư mục views ta tạo file trang1.php với nội dung sau:

```
<h1>THIS IS PAGE ONE</h1>
```

➔ kết quả in ra màn hình THIS IS PAGE ONE.

Giả sử trang1.php có đường dẫn như sau: views/subfolder/trang1.php

thì trong Mycontroller ta thay đổi lại thành

```
public function view1(){
    return View('subfolder.trang1');//trang1 nằm trong thư mục subfolder
}
```

3. Cách truyền tham số sang View

File: Route/web.php

//truyền tham số sang view: URL:../view3/abc

```
Route::get('view3/{param}','MyController@view3');
```

File: Mycontroller.php

```
public function view3($param){  
    return view('trang3',['t'=>$param]); // tham số t sẽ được truyền sang trang3.php  
}
```

File: views/trang3.php

```
<h1>THIS IS PAGE THREE</h1>  
<h2><?php echo $t; ?></h2>
```

→ kết quả in ra biến \$t là **abc**

Ngoài ra còn có thể sử dụng phương thức compact để truyền biến sang view

```
public function getGioHang(){  
    $content=Cart::content();  
    $total=Cart::subtotal();  
    return view('pages.cart',compact('content','total'));  
}
```

*** Share dữ liệu dùng chung cho các views**

File: Route/web.php

View::share('KhoaHoc','Laravel'); // các view sử dụng biến \$KhoaHoc một cách bình thường

Khi đó các view sử dụng biến \$KhoaHoc một cách bình thường

File: views/trang1.php

```
<h1>THIS IS PAGE ONE</h1>  
<h1><?php echo $KhoaHoc ?></h1>
```

File: views/trang2.php

```
<h1>THIS IS PAGE TWO</h1>  
<h1><?php echo $KhoaHoc ?></h1>
```

 **Tìm hiểu Mô hình MVC trong Laravel:**
<https://khoapham.vn/download/laravel/bai6.pdf>

Bài 5: Blade Template

1.Blade Template là gì?

Template là gì?

Template đó chính là 1 mẫu bố cục chung cho tất cả các trang có sử dụng lại những thành phần giống nhau mà không phải viết lại toàn bộ, từ đó trên mỗi trang, chỉ cần thay đổi nội dung ở một số vị trí trên trang so với trang chủ.

Template engine có tác dụng giúp tách đi những đoạn code PHP nằm trong View nên tách biệt hoàn toàn giữa người cắt CSS và người code PHP.

Hiện nay có khá nhiều template engine như Smarty, XiTemplate, TinyButStrong, Blade...

Blade template là gì?

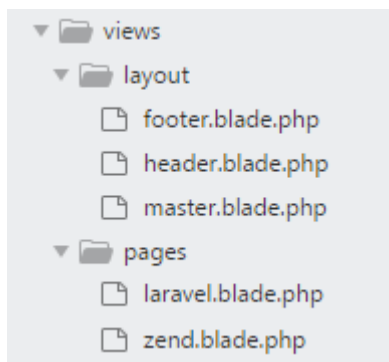
Blade rất đơn giản, nhưng lại là một templating engine đầy mạnh mẽ! Không giống những PHP templating engine phổ biến khác, Blade không giới hạn chúng ta sử dụng code PHP trong views. Tất cả các file Blade sẽ được dịch thành file code PHP và cache cho đến khi file Blade bị thay đổi, điều đó cũng có nghĩa là Blade tự làm tất cả những việc cần thiết để có thể chạy views cho ứng dụng của bạn.

Các file view dùng cho Blade có phần tên đuôi file là **.blade.php** và được lưu trong thư mục mặc định **resources/views** (với laravel 5.x).

2. Tạo template cho trang web

File: Route/web.php

```
Route::get('blade',function(){
    return view('pages.laravel');//gọi đến trang laravel có template là trang master
});
```



File laravel.blade.php

```
@extends('layout.master')<!--sử dụng layout-->

@section('NoiDung')
<h4>Đây là trang laravel.blade.php</h4> <!--Nội dung trang web đặt trong này-->
@endsection
```

File master.blade.php

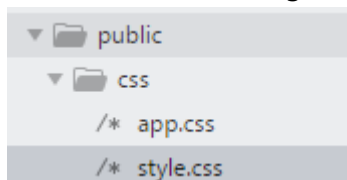
```

<!DOCTYPE html>
<html>
<head>
  <title></title>
  <link rel="stylesheet" type="text/css" href="{{asset('css/style.css')}}">
</head>
<body>
  @include('layout.header')<!--gọi trang header-->
  <div id='content'>
    <h1>master page</h1>
    @yield('NoiDung') <!--gọi trang laravel-->
  </div>
  @include('layout.footer') <!--gọi trang footer-->
</body>
</html>

```

Trong đó `href="{{asset('css/style.css')}}"` là nhúng file CSS để định dạng cho website này

Hàm `asset` có tác dụng trỏ đến thư mục `public`



Nội dung file `style.css` như sau:

```

#content{
    width: 800px;
    height: 500px;
    border: 1px solid black;
    background-color: yellow;
    text-align: center;
    margin: auto;
}

#header{
    width: 800px;
    height: 100px;
    border: 1px solid black;
    background-color: blue;
    text-align: center;
    margin: auto;
}

#footer{

```

```
width: 800px;
height: 100px;
border: 1px solid black;
background-color: tan;
text-align: center;
margin: auto;
}
```

File: header.blade.php

```
<div id='header'>Đây là tiêu đề website</div>
```

File: footer.blade.php

```
<div id='footer'>Đây là footer</div>
```

→ như vậy ở đây ta hiểu trang laravel.blade.php sử dụng khung giao diện của trang master.blade.php, chỉ khác là phần nội dung của trang master được thay bằng nội dung của trang laravel này.

Demo: nhập URL

<http://localhost:81/MyLaravel/public/blade>

Xuất hiện giao diện sau:



Giả sử bây giờ tôi muốn truyền vào phía sau URL trên
../blade/laravel → xuất hiện nội dung là trang laravel

../blade/zend → xuất hiện nội dung là trang zend

Khi đó ta phải truyền tham số cho Route như sau

Route::get('blade2/{str}', 'MyController@blade');

File: MyController.php

```
//blade template
//thay đổi URL
public function blade($str){
    if($str=='laravel') //nếu URL là ../blade/laravel thì gọi view laravel
        return view('pages.laravel');
    else if($str=='zend') //nếu URL là ../blade/laravel thì gọi view zend
        return view('pages.zend', ['kh'=>'zend framework']); //có truyền tham số từ controller->view
}
```

Trong câu lệnh else

Bạn thấy tôi đã truyền tham số \$kh với giá trị là "zend framwork" sang cho view zend

Trong view zend.blade.php có 3 cách để in ra biến \$kh nhận được

@extends('layout.master')

@section('NoiDung')

<h4>Đây là trang zend.plade.php</h4>

<?php echo \$kh ?>

<h4>{{ \$kh }}</h4>

<h4>{!! \$kh !!}</h4>

@endsection

Cách 1: sử dụng php thuần

Cách 2 và cách 3 sử dụng cú pháp trong Laravel, điểm khác biệt của 2 cách này là:

{{ }}-> in ra biến (không nhận biết được mã HTML trong biến đó)

{!! !!}> in ra biến (nhận biết và định dạng HTML nếu biến đó là 1 đối tượng HTML)

3. Các câu lệnh sử dụng trong Blade template

+ Câu lệnh if...else

```
@if($tmp!="")
    {{$tmp}}
@else
    {"đây là biến rỗng "}
@endif
```

+ Câu lệnh for

```
@for($i=1;$i<=10;$i++)
    {{$i." "}}
@endfor
```

+ Câu lệnh foreach

```
<?php $khoahoc =array('php','IOS','ASP.NET'); ?>
@foreach($khoahoc as $value)
    {{$value." "}}
@endforeach
```

+ While

```
@while( $dieukien )
    các câu lệnh;
@endwhile
```

+break, continue

đối với 2 lệnh này có 2 dạng là ko có điều kiện và có điều kiện.

```
@break
@continue

@break( dieukien ) //thực hiện khi có điều kiện
@continue( dieukien ) //thực hiện khi có điều kiện
```

Vd:

```
@foreach($khoahoc as $value)
    @break($value=='IOS') <!--nếu chạy đến IOS thì thoát không lặp nữa-->
    {{$value." "}}
@endforeach
```

Lưu ý: các câu lệnh trong blade template sẽ không có dấu chấm phẩy (;) ở cuối câu lệnh.

Bài 6: Làm việc với Database

1. Thao tác chuẩn bị cơ sở dữ liệu

B1: Kết nối với cơ sở dữ liệu trong laravel

Mở file **.env** lên và tìm đến đoạn code sau:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1          //địa chỉ localhost
DB_PORT=3306
DB_DATABASE=homestead //tên database
DB_USERNAME=homestead//tên db: thường là root
DB_PASSWORD=secret       //pass các bạn nên để trống
```

➔ hãy chỉnh sửa lại như sau:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=mylaravel
DB_USERNAME=root
DB_PASSWORD=
```

B2: tạo database

vào phpmyadmin tạo một database có tên là **mylaravel**

2. Tạo, sửa, xóa bảng bằng Schema

Muốn làm việc với Schema thì phải khai báo

Use `Illuminate\Support\Facades\Schema`

Tuy nhiên khi sử dụng trong route thì không cần

A) Tạo bảng

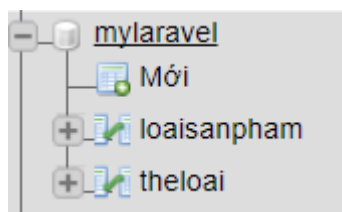
Cấu trúc tạo bảng

```
Schema::create("ten_bảng", function($table){
    $table->KDL("tên_cột_1",kích_thức);
    $table->KDL("tên_cột_2",kích_thức);
    ...
})
```

```
Route::get('database',function(){
    Schema::create('loaisanpham',function($table){
        $table->increments('id'); //tạo cột id tự động tăng
        $table->string('ten',200); //tạo cột ten cố độ dài là 200
    });
    echo "đã thực hiện lệnh tạo bảng thành công";

    Schema::create('theloai',function($table){
        $table->increments('id');
        $table->string('ten',200)->nullable(); //cho phépnull
        $table->string('nsx')->default('nhà sản xuất');//gán giá trị mặc định cho cột
    });
    echo "đã thực hiện lệnh tạo bảng thành công";
});
```

ở trên tôi đã tạo 2 bảng loaisanpham (id, ten) và bảng theloai(id, ten, nsx)



Bạn hãy vào phpmyadmin để kiểm tra, đây là cấu trúc bảng theloai

#	Tên	Kiểu	Bảng mã đối chiếu	Thuộc tính	Null	Mặc định	Ghi chú	Thêm
<input type="checkbox"/> 1	id	int(10)		UNSIGNED	Không	Không		AUTO_INCREMENT
<input type="checkbox"/> 2	ten	varchar(200)	utf8mb4_unicode_ci		Có	NULL		
<input type="checkbox"/> 3	nsx	varchar(255)	utf8mb4_unicode_ci		Không	nhà sản xuất		

Như vậy với kiểu string thì nếu bạn không quy định kích thước cho nó thì mặc định kích thước tối đa là 255 ký tự.

Một số KDL khi tạo cột:

<code>\$table->increments(ten_cot);</code>	kiểu số nguyên tự động tăng khi qua dòng mới
<code>\$table->string(ten_cot, kích_thước);</code>	kiểu chuỗi
<code>\$table->integer(ten_cot);</code>	kiểu số nguyên
<code>\$table->float(ten_cot);</code>	kiểu số thực
<code>\$table->date(ten_cot);</code>	kiểu ngày
<code>\$table->time(ten_cot);</code>	kiểu giờ
<code>\$table->datetime(ten_cot);</code>	kiểu ngày giờ
<code>\$table->boolean(ten_cot);</code>	kiểu logic

Một số ràng buộc cho cột

Câu lệnh	Mô tả
<code>->nullable();</code>	Cho phép giá trị null
<code>->default(\$value);</code>	Gán giá trị mặc định cho cột
<code>->unsigned();</code>	Đặt unsigned cho integer

b) Liên kết giữa các bảng

Câu lệnh	Mô tả
<code>\$table->primary('TenKhoaChinh');</code>	Tạo khóa chính
<code>\$table->foreign('KhoaPhu')->references('KhoaChinh')->on('Bang');</code>	Tạo khóa phụ
<code>\$table->unique('TênCột');</code>	Ràng buộc unique

Vd: tôi có bảng

loaisanpham (**id**, **ten**) và tôi sẽ tạo tiếp bảng **sanpham**(**id**, **ten**, **gia**, **soluong**, **id_loaisanpham**)

trong đó id_loaisanpham sẽ tham chiếu tới khoá chính id trong bảng loaisanpham

```
$table->foreign('id_loaisanpham')->references('id')->on('loaisanpham');
```

```
//2.liên kết bảng
Route::get('lienketbang',function(){
    Schema::create('sanpham',function($table){
        $table->increments('id');
        $table->string('ten');
        $table->float('gia');
        $table->integer('soluong')->default(0);
        $table->integer('id_loaisanpham')->unsigned();//tạo thuộc tính cho giống khoá chính
        //tạo khoá ngoại và cho tham chiếu tới khoá chính
        $table->foreign('id_loaisanpham')->references('id')->on('loaisanpham');
    });
});
```

C) chỉnh sửa bảng

Thêm cột mới vào bảng

```
Schema::table('ten_bảng', function($table){
    $table->string('ten_cột_mới');
});
```

Xoá 1 cột trong bảng

```
Schema::table('ten_bảng', function($table){
    $table->dropColumn('ten_cột');
});
```

Đổi tên bảng

```
Schema::rename('ten_cũ', 'ten_mới');
```

Xoá bảng

```
Schema::drop('ten_bảng');
```

//xoá bảng nếu bảng đó tồn tại

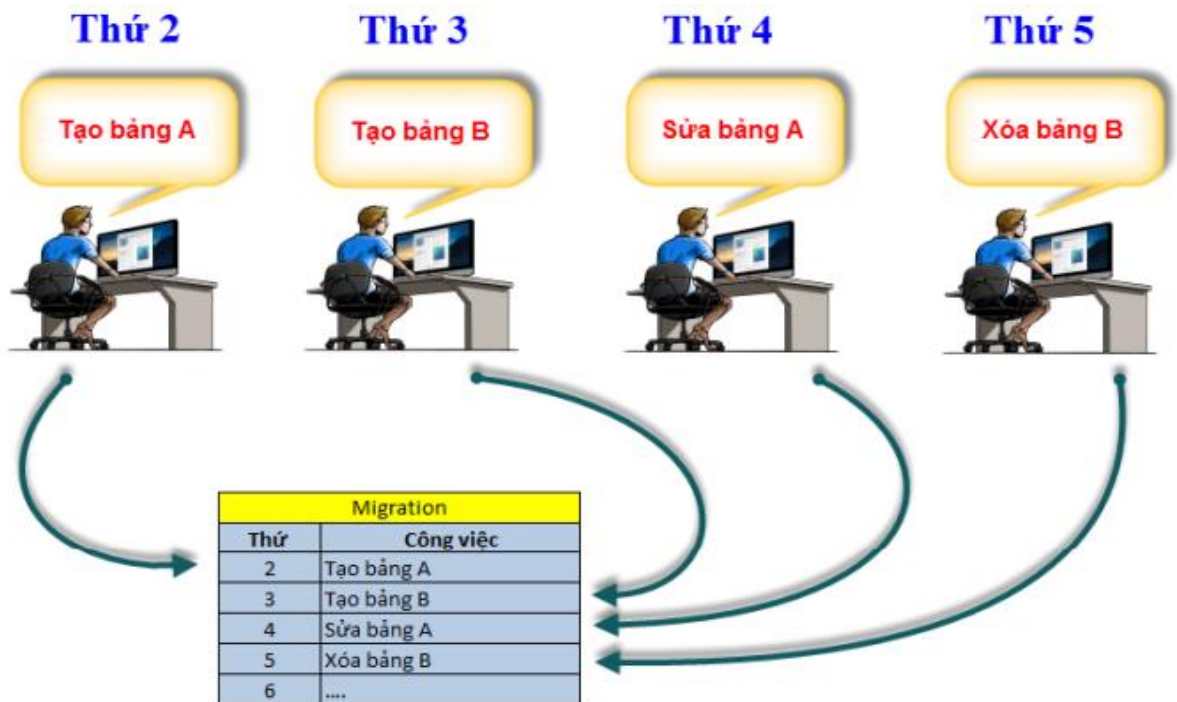
```
Schema::dropIfExists('ten_bảng');
```

3.Quản lý dữ liệu với Migrations

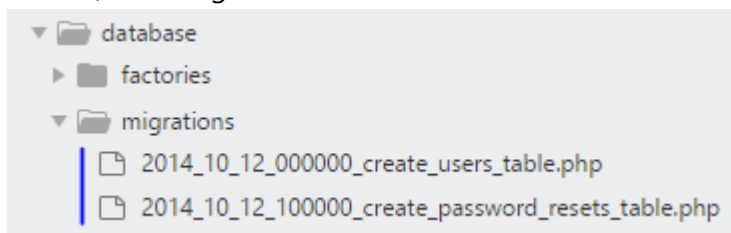
A) giới thiệu về migrations

Migration trong Laravel giống như một control database có tác dụng quản lý cũng như lưu trữ lại cấu trúc của database giúp cho việc sửa đổi database trở lên dễ dàng hơn.

Hay nói cách khác Migrations giúp ta lưu lại các thao tác thay đổi trên database từ đó dễ dàng khôi phục lại dữ liệu .



Thư mục lưu Migrations



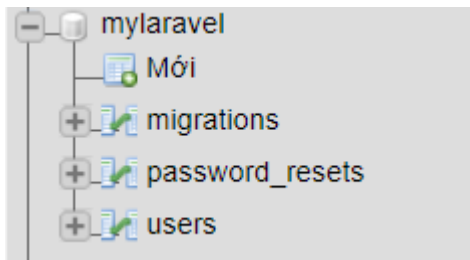
Trong migrations sẽ có 2 file, mỗi file sẽ có 2 hàm là **up** (được gọi khi thực thi migrate đó) và **down**(được gọi khi muốn Rollback)

Để chạy migration ta vào thư mục project của ta, Shift+nhấn chuột phải → open command window.. → gõ vào dòng lệnh **php artisan migrate** để thực thi các file trong migrations

```
C:\xampp\htdocs\MyLaravel>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
```

Nếu bị lỗi thì là bạn vào 2 file trong thư mục migrations , chỗ nào KDL là string thì thêm vào giới hạn số ký tự cho nó là 191 nhé. chẳng hạn `$table->string('name')` thành `$table->string('name',191);`

Sau khi thực thi xong bạn vào phpmyadmin→ vào CSDL của mình sẽ thấy có thêm 3 table mới được tạo ra như sau:



Trong đó bảng migrations sẽ lưu lại các quá trình thao tác với database

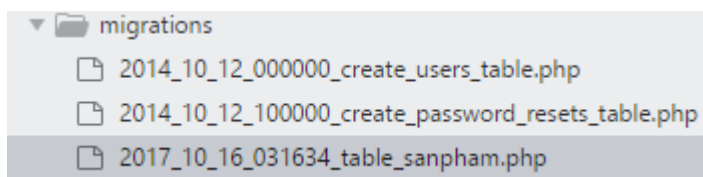
id	migration	batch
1	2014_10_12_000000_create_users_table	1
2	2014_10_12_100000_create_password_resets_table	1

B) Các thao tác bằng migrations

Tạo file migrate mới

Vào thư mục của project nhấn Shift+nhấn chuột phải → open command window.. → gõ vào dòng lệnh `php artisan make:migration TenFile` để tạo file migrate mới trong migrations.

Giả sử ở đây tôi tạo file có tên là table_sanpham



Trong file này tôi code như sau:

```
public function up()
{
    //tạo bảng sanpham
    Schema::create('sanpham',function($table){
        $table->increments('id');
        $table->string('ten');
        $table->integer('soluong');
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    //xóa bảng
    Schema::drop('sanpham');
}
```

Như vậy 2 thao tác tạo và xóa bảng phải luôn đi kèm với nhau trong mỗi file migration

Để thực thi file này bạn vào cửa sổ cmd và gõ `php artisan migrate`. vào phpmyadmin bạn sẽ thấy có 1 table mới tạo ra có tên là **sanpham(id, ten, soluong)**. Và trong bảng **migrations** sẽ xuất hiện thêm dòng mới cho biết lần thao tác gần nhất bạn làm gì.

Đưa CSDL về trạng thái trước khi thực hiện migrate

Trong cửa sổ cmd bạn thực hiện các câu lệnh sau:

`php artisan migrate:rollback` → Hủy bỏ các thao tác đã thực thi của migrate lần trước trên database.

`php artisan migrate:reset` → Hủy bỏ hết các thao tác đã thực thi của migrate, đưa database về trạng thái ban đầu.

nếu như sau khi thực thi các câu lệnh trên, bạn muốn thực thi lại migrate thì đơn giản là bạn cứ nhập `php artisan migrate` là đc, lúc này các table sẽ được tạo lại.

Lưu ý: việc đưa CSDL về trạng thái trước đó ntn là tùy thuộc vào `down()` trong mỗi migrate bạn cài đặt gì trong đó (có thể là xoá table, hoặc là chỉ cập nhật lại table đó).

Tóm tắt:

Sử dụng migrate với cửa sổ cmd

<code>php artisan make:migration TenMigrate</code>	Tạo file migrate với artisan
<code>php artisan migrate</code>	Thực thi file migrate
<code>php artisan migrate:rollback</code>	Hủy bỏ việc thực thi của migrate trước
<code>php artisan migrate:reset</code>	Hủy bỏ hết công việc của migrate

4. Tạo dữ liệu mẫu với seed

Seed là bộ dữ liệu mẫu, nó giúp chúng ta quản lý dữ liệu trong bảng một cách thuận tiện, dễ dàng khôi phục lại khi cần thiết.

Các file seed được lưu tại thư mục `database/seeds/DatabaseSeeder.php`

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        //nơi thực thi các câu lệnh khi seed được chạy
    }
}
```

Tạo dữ liệu mẫu với seed:

+cách 1: viết trực tiếp trong class DatabaseSeeder


```

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

        DB::table('users')->insert([
            'name'=>'Phu',
            'email'=>'phu123@gmail.com',
            'password'=>bcrypt('123456') //mã hoá mật khẩu
        ]);
    }
}

```

Để thực thi bạn vào cửa sổ cmd gõ: **php artisan db:seed**

+ Cách 2 Tạo class riêng sau đó gọi vào trong class DatabaseSeeder

```

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

class userSeeder extends Seeder
{
    public function run()
    {
        DB::table('users')->insert([
            //thực hiện insert nhiều dòng dữ liệu cùng lúc
            ['name'=>'PhuA', 'email'=>'phu003@gmail.com', 'password'=>bcrypt('aaaa') ],
            ['name'=>'PhuB', 'email'=>'phu456@gmail.com', 'password'=>bcrypt('bbbb') ],
            ['name'=>'PhuC', 'email'=>'phu789@gmail.com', 'password'=>bcrypt('cccc') ]
        ]);
    }
}

```

Thực thi trong Cmd: **php artisan db:seed**

5. Query Builder

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

Để sử dụng được các câu truy vấn trên bắt buộc các bạn phải:

- Nếu truy vấn trong controllers thì các bạn cần phải khai báo use **Illuminate\Support\Facades\DB** còn trong Route thì không cần.

Ví dụ: để truy vấn dữ liệu trong bảng users trong SQL bạn viết như sau
 SELECT * FROM users

Tuy nhiên đối với laravel ta có cú pháp truy vấn sau:

`$users = DB::table('users')->get();` câu lệnh này sẽ lấy toàn bộ dữ liệu trong bảng **users** ra và lưu vào biến **\$users**

Danh sách các lệnh truy vấn cơ bản trong SQL

Query

- Lấy tất cả dữ liệu trong bảng

```
$data=DB::table('sanpham')->get();

foreach($data as $value){
    echo $value->tensp;
}
```

- lấy một số thuộc tính

```
$data=DB::table('sanpham')->select('id', 'tensp', 'gia')->get();
```

-Lấy dòng đầu tiên trong bảng

```
$sanpham = DB::table('sanpham')->where('id', 1)->first();
Echo $sanpham->tensp;
```

- truy vấn có điều kiện

Điều kiện bằng

```
//select * from sanpham where tensp='Dell'
$data=DB::table('sanpham')->where('tensp','=', 'Dell')->get();
Hoặc
$data=DB::table('sanpham')->where('tensp','Dell')->get();
```

So sánh chuỗi

```
DB::table('sanpham')->where('tensp','like','dienthoai')->get();
```

Điều kiện >, <, <>

```
DB::table('tablename')->where('column','>','value')->get();
```

```
DB::table('tablename')->where('column','<','value')->get();
```

```
DB::table('tablename')->where('column','<>','value')->get();
```

Điều kiện and, or

```
//select * from sanpham where gia=1000 and id_loaisp=2
$data=DB::table('sanpham')->where([
    ['gia',1000],
    ['id_loaisp',2]
])->get();
```

```
//select * from sanpham where gia=1000 or tensp='SAMSUNG A5'
$data=DB::table('sanpham')->where('gia',1000)->orWhere('tensp','Dell')->get();
```

Select kết hợp với where

```
$data=DB::table('sanpham')->select('id', 'tensp', 'gia')
->where('id_loaisp',1)->get();
```

Join các bảng

```
$users = DB::table('sanpham')->select('tensp', 'tenloai')
->join('loaisp', 'sanpham.id_loaisp', '=', 'loaisp.id')->get();
```

Left join:

```
$users = DB::table('sanpham')
->leftjoin('loaisp', 'sanpham.id_loaisp', '=', 'loaisp.id')->get();
```

Order by

```
$data=DB::table('sanpham')->select('id', 'tensp', 'gia')
->where('id_loaisp',1)->orderBy('gia','desc')->get();
```

Group by and having

Trong một số trường hợp câu lệnh groupBy trong laravel bị lỗi khi đó fix như sau:
bạn vào config/database.php và sửa lại:

```
'mysql' => [
    'driver' => 'mysql',
    'host' => env('DB_HOST', 'localhost'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'charset' => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix' => '',
    'strict' => false,
    'engine' => null,
],
```

```
$data=DB::table('sanpham')->groupBy('soluong')
->having('soluong', '>=', 2)->get();
```

Insert

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

Insert nhiều dòng cùng lúc

```
DB::table('users')->insert([
    //thực hiện insert nhiều dòng dữ liệu cùng lúc
    ['name'=>'PhuA','email'=>'phu003@gmail.com','password'=>bcrypt('aaaa') ],
    ['name'=>'PhuB','email'=>'phu456@gmail.com','password'=>bcrypt('bbbb') ],
    ['name'=>'PhuC','email'=>'phu789@gmail.com','password'=>bcrypt('cccc') ]
]);
```

Update, delete

Mô tả	Trong Laravel
-------	---------------

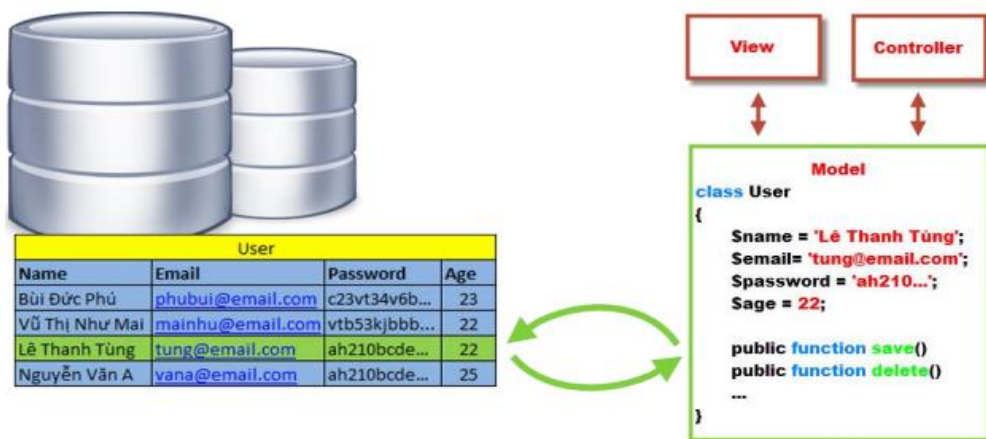
Update cột name, email dựa vào id	DB::table('users')->where('id',1) ->update(['name'=>'ninja' , 'email'=>'ninja@gmail.com']);
Delete dòng theo id	DB::table('users')->where('id',1)->delete();
Xoá tất cả dữ liệu trong bảng	DB::table('users')->delete();

6. Eloquent – Model

Như các bạn biết trong Laravel Framework hỗ trợ cho chúng ta 2 cách phổ biến để có thể làm việc với cơ sở dữ liệu. Trong bài trước mình đã giới thiệu cho các bạn về **Query Builder**. Hôm nay mình sẽ giới thiệu cho các bạn về **Eloquent ORM**.

a) Eloquent ORM là gì?

Eloquent ORM đi kèm với Laravel cung cấp ActiveRecord đầy đủ, đẹp đẽ và đơn giản để làm việc với database. Mỗi bảng của database sẽ được ánh xạ qua 'Model', và model này được sử dụng để tương tác với bảng.



Tạo Model:

Các file Model sẽ được lưu ở thư mục **App\...**

Ban đầu sẽ có một file model được tạo sẵn là User.php, nếu bạn muốn tạo thêm một model mới thì vào cửa sổ cmd của project gõ: **php artisan make:model TenModel**

b) Kết nối model với bảng trong CSDL

Giả sử tôi tạo thêm một Model mới tên là **sanpham**

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class sanpham extends Model
{
    protected $table="sanpham";//Kết nối model với bảng trong cơ sở dữ liệu
```

```

        protected $fillable = ['id','tensp','soluong', 'gia', 'mota','id_loaisp'];
        public $timestamps = false;
    }

```

Trong đó:

- + Thuộc tính **\$table** sẽ khai báo bảng dữ liệu mà ta sẽ thao tác.
- + Thuộc tính **\$timestamps** cho biết có cho phép tự động tạo thêm 2 cột updated_at và created_at trong bảng hay không (nếu không muốn tạo thì cho giá trị là false)
- + thuộc tính **\$fillable** là liệt kê các tên cột cần sử dụng trong của table trong CSDL(điều này giúp hạn chế các truy vấn trên các cột không cần thiết)

c) Các thao tác truy vấn CSDL sử dụng Eloquent ORM

Về cơ bản, các câu lệnh truy vấn của Eloquent ORM có 1 chút thay đổi so với Query Builder, khiến câu lệnh trông ngắn gọn và đẹp dễ hơn:

để thực hiện truy vấn bạn phải khai báo sử dụng file Model trên trong Controller/Route

```
use App\sanpham;
```

SELECT

- Lấy tất cả dữ liệu trong bảng sanpham

```

$sanpham = sanpham::all();
foreach ($sanpham as $row)
{
    echo $row->tensp;
}

```

- Lấy 1 dòng dữ liệu thông qua khóa chính:

```

$sanpham =sanpham::find(1);
echo $sanpham ->tensp;

```

Nếu không lấy được dữ liệu nào trả về thì kết quả sẽ là NULL

- Bạn cũng có thể sử dụng các hàm trong query builder

```

$data=sanpham::where('id_loaisp', 1)->get();
foreach($data as $value){
    echo $value->tensp;
}

```

INSERT

- Cách 1: Để tạo một record mới vào bảng, đơn giản bạn tạo một thực thể của model và gọi phương thức save.

```

$sp = new sanpham;
$sp->tensp="Lenovo";
$sp->soluong=10;
$sp->gia=2400;
$sp->mota="Lenovo in USA";
$sp->save();

```

- Cách 2: Sử dụng phương thức Create

```
$sanpham = sanpham::create([
    'tensp' => 'Nokia',
    'soluong'=>1,
    'gia'=>4200,
    'mota'=>'Dien thoai nhut ban',
]);
```

UPDATE

- Cập nhật một model, bạn có thể truy vấn nó, thay đổi thuộc tính và lưu nó lại thông qua khóa chính.

```
$sanpham = sanpham::find(1);
$sanpham->tensp = 'Test';
$sanpham->save();
```

- Bạn có thể update theo điều kiện

```
$result = sanpham::where('gia', '>=', 2400)->update(['tensp'=>'test2']);
echo $result; // số dòng update thành công
```

DELETE

Để xóa một record, đơn giản bạn gọi phương thức delete dựa vào khóa chính

```
$sanpham = sanpham::find(1);
$sanpham->delete();
```

Xóa bằng khóa

```
sanpham::destroy(1);
```

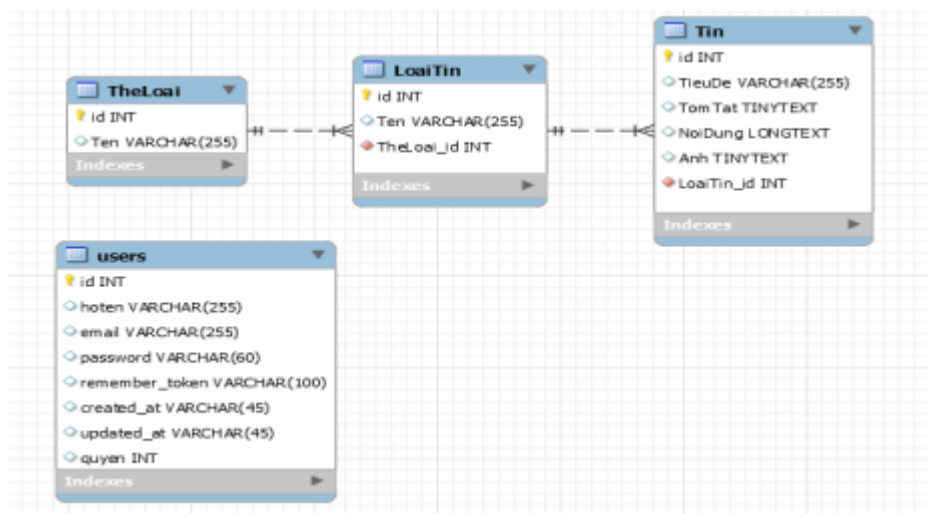
Xóa dựa vào điều kiện

```
$result = sanpham::where('gia', '=', 3000)->delete();
```

➡ So sánh giữa Query Builder và Eloquent ORM.

<https://techtalk.vn/so-sanh-eloquent-orm-va-query-builder-trong-laravel.html>

d) Tạo liên kết bảng thông qua các model (tìm hiểu thêm)



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

Cú pháp tạo liên kết bảng trong model

public function tenphuongthuc()

```
{  
    return $this->Kiểu_kết_nối('App\TenModel', 'khoa_ngoai', 'khoa_chinh');  
}
```

Ví dụ

bảng: tacgia

id	hoten	diachi
1	nguyen van a	TPHCM
2	duyle	Binh Dinh

Bảng: baiviet

id	tieude	noidung	id_tacgia
1	bai dau tien	day la noi dung bai dau tien	2
2	bai thu 2	day la noi dung bai thu 2	1
3	bai viet 3	day la noi dung bai viet 3	1

Mối quan hệ: một tác giả có nhiều bài viết (1-n), một bài viết chỉ thuộc về một tác giả(1-1)

xác định khoá: khoá chính-id (trong bảng tacgia), khoá ngoại-id_tacgia (trong bảng baiviet)

Model tacgia

```
namespace App;  
use Illuminate\Database\Eloquent\Model;  
  
class tacgia extends Model  
{  
    protected $table="tacgia";  
    protected $filltable=['id','hoten','diachi'];  
    public $timestamps = false;  
  
    public function baiviet(){  
        return $this->hasMany('App\baiviet','id_tacgia','id');//1-n  
    }  
}
```

Model baiviet

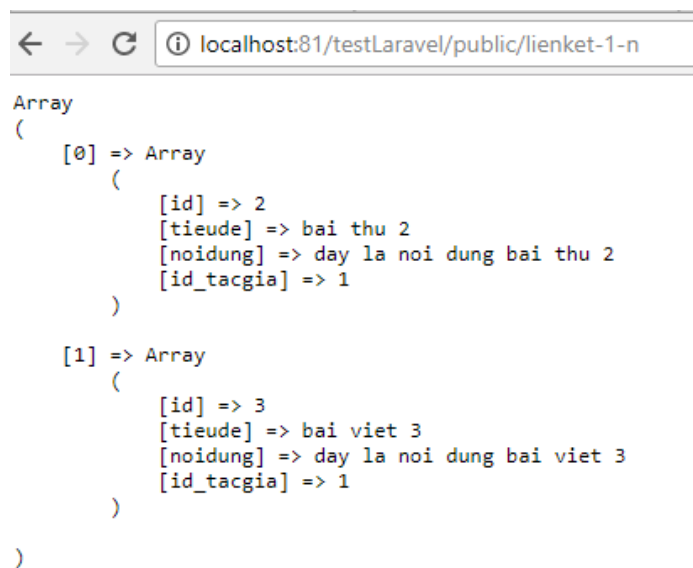
```
namespace App;  
use Illuminate\Database\Eloquent\Model;  
  
class baiviet extends Model  
{  
    protected $table="baiviet";
```

```
protected $fillable=['id','tieude','noidung','id_tacgia'];
public $timestamps = false;

public function tacgia(){
    return $this->belongsTo('App\tacgia', 'id_tacgia','id');//1-1
}
```

Test: lấy thông tin bài các viết của tác giả có id=1

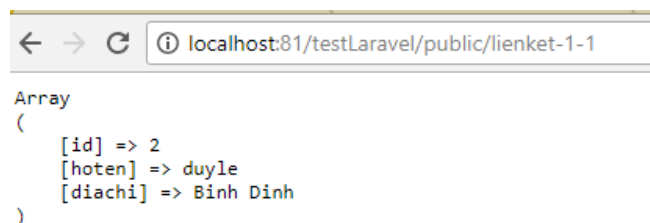
```
Route::get('lienket-1-n', function(){
    $data=App\tacgia::find(1)->baiviet->toArray();
    echo "<pre>";
    print_r($data);
    echo "</pre>";
});
```



```
Array
(
    [0] => Array
        (
            [id] => 2
            [tieude] => bai thu 2
            [noidung] => day la noi dung bai thu 2
            [id_tacgia] => 1
        )
    [1] => Array
        (
            [id] => 3
            [tieude] => bai viet 3
            [noidung] => day la noi dung bai viet 3
            [id_tacgia] => 1
        )
)
```

Test: lấy thông tin tác giả của bài viết có id=1

```
Route::get('lienket-1-1', function(){
    $data=App\baiviet::find(1)->tacgia->toArray();
    echo "<pre>";
    print_r($data);
    echo "</pre>";
});
```



```
Array
(
    [id] => 2
    [hoten] => duyle
    [diachi] => Bình Định
)
```

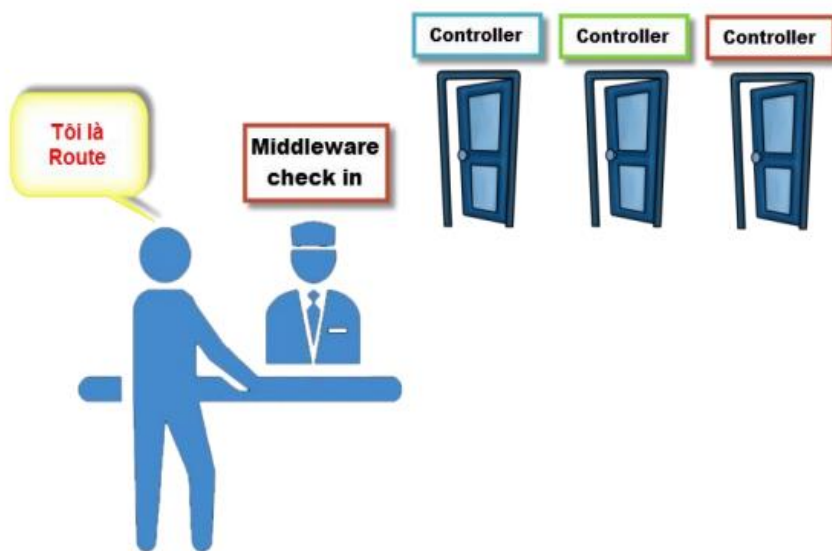

Bài 7: Kiểm soát Route với Middleware

1. Kiểm soát Route với middleware

middleware là một cơ chế trung gian trong việc quản lý các route, nghĩa là nó sẽ cho phép route nào sẽ được thực thi route nào không được.

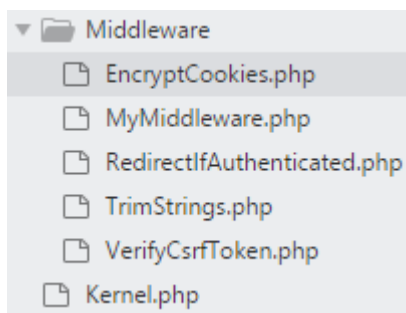
Middleware cung cấp một giải pháp khá tiện ích cho các HTTP requests trong ứng dụng của bạn. Ví dụ, Laravel có chứa một middleware xác thực user đăng nhập vào ứng dụng của bạn. Nếu user không chính xác, middleware sẽ chuyển hướng (redirect) tới màn hình đăng nhập. Tuy nhiên, nếu user hợp lệ, middleware sẽ cho phép request sau được thực hiện tiếp.

Hình dưới minh họa cho cơ chế middleware này



Cấu trúc thư mục của middleware

App/Http/Middleware



Trong project mặc định đã có các file Middleware, ngoài ra bạn có thể tạo thêm một file Middleware mới bằng câu lệnh CMD sau:

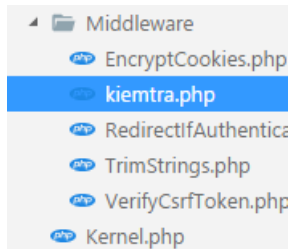
```
php artisan make:middleware TenMiddleware
```

File **Kernel.php** chứa các Middleware đã được đăng ký hoạt động, nếu bạn tạo một Middleware mới thì để nó hoạt động được thì bạn phải đăng ký trong file này.

2. Tạo và sử dụng Middleware

B1: tạo file Middleware

Php artisan make: middleware kiểmtra



Trong Middleware thì mọi xử lý đều nằm trong phương thức handle

```
public function handle($request, Closure $next)
{
    if($request->has('txtEmail') && $request->txtEmail == "duyle@gmail.com"){
        return $next($request);
    }
    else
        return redirect()->route("loi-email");
}
```

Ở đây tôi tạo Middleware để xác thực email người dùng nhập vào đúng không. Nếu đúng thì cho phép đi đến các request khác, ngược lại trả về request thông báo lỗi.

B2: đăng ký Middleware vừa tạo với hệ thống

Có 3 kiểu đăng ký thường dùng là Global middleware, Route middleware và Nhóm middleware.

Global middleware:

là một middleware mà bất cứ HTTP request nào muốn thực hiện được cũng bắt buộc phải qua nó.

Để đăng ký global middleware bạn sẽ phải vào: Kernel.php tìm đến đoạn sau và đăng ký:

```
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    \App\Http\Middleware\kiemtra::class
];
```

Route middleware

Là middleware chỉ sử dụng được khi bạn gọi nó ở trong Route, Middleware này thường dùng để chặn một số HTTP Request nhất định.

Cách đăng ký

Cú pháp:

'Tên middleware'=> \App\Http\Middleware\TenMiddleware::class

```
protected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'check-email' => \App\Http\Middleware\kiemtra::class
];
```

Route sử dụng như sau:

```
Route::post('kiemtra-email', function(){
    echo "Email success";
})->middleware('check-email')->name('kiemtra-email');
```

→ nghĩa là trước khi thực hiện route kiemtrea-email nó sẽ vào middleware có tên là check-email kiểm tra trước, nếu thoả mãn middleware thì quay lại thực hiện route này.

Nhóm middleware.

Với loại middleware này chúng ta có thể gộp các middleware thành các nhóm để gọi cho nhanh.

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        // \Illuminate\Session\Middleware\AuthenticateSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:60,1',
        'bindings',
    ],
];
```

Mặc định của Laravel đã có 2 nhóm middleware là **web** và **api**.

Các bạn muốn đăng ký thêm middleware thì đăng ký theo cú pháp tương tự như mặc định của Laravel.

Và gọi trong Route như sau:

```
Route::get('/', function () {
    //
})->middleware('web');
```

Ví dụ sử dụng Route Middleware

- Route

```
Route::get('nhap-email',function(){
    return view('email');
});

Route::post('kiemtra-email',function(){
    echo "Email success";
})->middleware('check-email')->name('kiemtra-email');

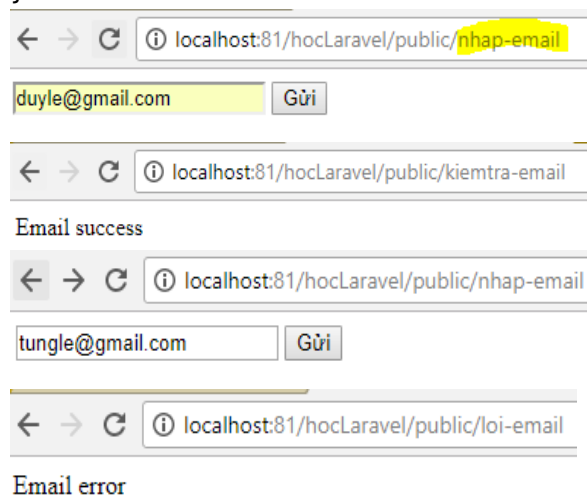
Route::get('loi-email', function(){
    echo "Email error";
})->name('loi-email');
```

- View nhập email

```
<form action="{{route('kiemtra-email')}}" method="post">
    <input type="hidden" name="_token" value="{{csrf_token()}}">
    <input type="text" name="txtEmail" >
    <input type="submit">
</form>
```

- Middleware kiểm tra email có hợp lệ không

```
public function handle($request, Closure $next)
{
    if($request->has('txtEmail') && $request->txtEmail == "duyle@gmail.com"){
        return $next($request);
    }
    else
        return redirect()->route("loi-email");
}
```



Bài 8: Authentication trong Laravel

1. Auth là gì

Auth dùng để quản lý việc đăng nhập/đăng xuất trong laravel

Với Auth bạn chỉ cần cung cấp thông tin người dùng sau đó hệ thống sẽ trả về kết quả cho ta và cho đăng nhập vào hệ thống nếu thông tin bạn cung cấp chính xác.

Đầu tiên mọi người hãy mở tệp tin `config/auth.php` ra và xem:

Đoạn code trong file này khá dài đúng không các bạn?(toàn comment) Chức năng của đoạn code đó là xác định phương thức xác thực cũng như lưu trữ mặc định, và tùy biến.

-Vì nó khá dài nên ở series này mình sẽ không trình bày nhiều mà chỉ cần các bạn quan tâm đoạn code này là được.

```
'providers' => [
    'users' => [
        'driver' => 'eloquent',
        'model' => App\User::class,
    ],
],
```

Đây là đoạn xác lập phương thức, và nguồn dữ liệu được lấy ra để xác thực.

Chú ý đến model : Đây là thiết lập nguồn dữ liệu lấy ra từ đâu. Ở đây mặc định Laravel chọn là model User (App\User.php). Thông số này bạn có thể chỉnh thành tên model mà bạn chọn để làm Auth.

2. Sử dụng Auth để thực hiện đăng nhập, đăng xuất

Lưu ý: khi sử dụng Auth trong controller thì controller đó phải khai báo

use Illuminate\Support\Facades\Auth;

còn nếu dùng trong Route thì không cần khai báo gì cả.

ở đây tôi sẽ hướng dẫn bạn thông qua ví dụ minh họa sau:

Giả sử bạn có bảng users như sau:

id	name	email	password	remember_token
4	duyle	duyle@gmail.com	\$2y\$10\$b5dR9XS2k1tmmb4l9DdW5e632e2BLb90Suo8DVZNZa2...	21nkeVsTB5B8qltvkYjeJh08

Chúng ta sẽ thực hiện đăng nhập thông qua 2 thuộc tính name và password:

-Route

```
Route::get('dangnhap',function(){
    return view('dangnhap.dangnhap');
});

Route::post('kiemtra',function(Request $request){
    $name=$request->txtName;
    $password=$request->txtPassword;
    if(Auth::attempt(['name'=>$name, 'password'=>$password])){
        return view('dangnhap.thanhcong',['user'=>Auth::user()]);
    }
    else{
        return view('dangnhap.dangnhap',['message'=>'name hoặc password không chính xác']);
    }
}
```

```

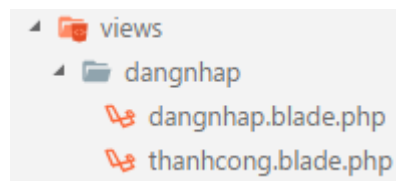
})->name('kiemtra');

Route::get('dangxuat',function(){
    Auth::logout();
    return view('dangnhap.dangnhap');
});

```

(1) Route dangnhap

sẽ gọi view dangnhap.blade.php, view này là form để nhập name và password



View dangnhap như sau:

```

<form action="{{route('kiemtra')}}" method="post">
    {{ csrf_field() }}
    Name: <input type="text" name="txtName"><br/>
    Password: <input type="text" name="txtPassword"><br/>
    <input type="submit"/>
</form>
@if(isset($message))
    {{$message}}
@endif

```

Sau khi người dùng nhập username và password xong nhấn submit → gọi route kiemtra để kiểm tra xem thông tin đăng nhập đúng ko.

(2) Route kiemtra

Auth::attempt(): Đây là phương thức check input có hợp lệ không, nếu hợp lệ sẽ trả về true và ngược lại trả về false. Hoàn toàn có thể thêm điều kiện nếu muốn, chẳng hạn:

```

$data=[
    'username'=>$request->username,
    'password'=>$request->password,
    'level'=>2
];
if(Auth::attempt($data)){
    //true
}else{
    //false
}

```

Nếu kiểm tra đúng thì gọi view thanhcong.blade.php và truyền tham số user qua bên đó.

Auth::user() → trả về một đối tượng user đang đăng nhập (đối tượng trả về sẽ ở dạng JSON)

do đó bạn có thể lấy các thông tin trong nó như **Auth::user()->name**

Auth::id(); → lấy ID của User đang đăng nhập

Nếu đăng nhập thành công thì sẽ gọi view thanhcong.blade.php


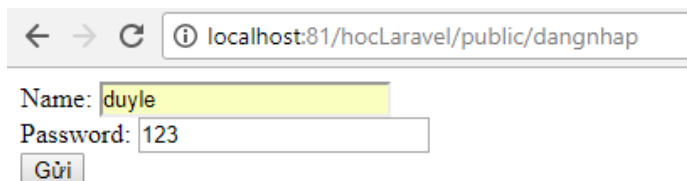
```
<h2 style="color:blue">ĐĂNG NHẬP THÀNH CÔNG</h2>
<p>Name: {{ $user->name }}</p>
<p>Password: {{ $user->password }}</p>
<a href="{{url('dangxuat')}}">Đăng xuất</a>
```

(3) Route dangxuat

Để đăng xuất bạn chỉ cần gọi phương thức `Auth::logout()`;

* Mở rộng: kiểm tra đăng nhập với `check()`

```
if (Auth::check()) {
    // Đã đăng nhập.
}
else{
    //chưa đăng nhập.
}
```



ĐĂNG NHẬP THÀNH CÔNG

Name: duyle

Password: \$2y\$10\$b5dR9XS2k1tmmb4l9DdW5e632e2BLb90Suo8DVZNZa2VYxGCjKaBa

[Đăng xuất](#)

3.Custom Auth mặc định của Laravel

a) login và Register mặc định trong Laravel

- Để tạo Auth trong Laravel thì cũng hết sức đơn giản. Các bạn chỉ cần dùng lệnh:

```
php artisan make:auth
```

Sau khi chạy lệnh này lên thì Laravel sẽ thêm cho chúng ta một homeController, route channels và rất nhiều view mới.

- Ngay lúc này bạn run project lên và sẽ thấy một số thay đổi về giao diện welcome mặc định của Laravel.

Laravel

[DOCUMENTATION](#) [LARACASTS](#) [NEWS](#) [FORGE](#) [GITHUB](#)

-Trên màn hình welcome của laravel đã xuất hiện thêm login và register, Tiếp tục click vào từng action ta thấy:

Toidicode.com [Login](#) [Register](#)

Login

E-Mail Address

Password

☐ Remember Me

[Login](#) [Forgot Your Password?](#)

Toidicode.com

Toidicode.com

Login Register

Register


Name

E-Mail Address

Password

Confirm Password

Register



- Sau đó bạn cứ thử đăng ký một users, đăng nhập vào hệ thống rồi vào database xem có gì ở bảng users nhé!

b) Custom lại Register trong Laravel

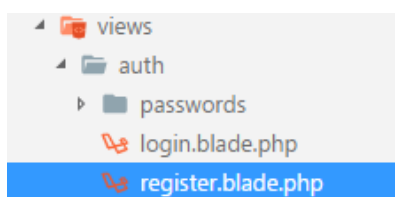
Chuẩn bị: Nếu như mặc định khi đang ký tài khoản User thì sẽ không có trường username cho người dùng nhập vào. Bây giờ tôi muốn thêm trường này vào khi đăng ký User mới thì làm thế nào? Để làm được như vậy trước hết bạn phải thêm thuộc tính username và table users trong database. Có 2 cách thêm

cách 1: Các bạn truy cập vào database/migrations mở tệp

2014_10_12_000000_create_users_table.php và thêm vào `$table->string('username')` trong phương thức up() sau đó chạy lại file migrate này

Cách 2: Vào trực tiếp trong database thêm thuộc tính này vào bảng users

Bước 1: Chỉnh sửa form Register



Thêm đoạn code sau vào form để hiển thị trường nhập username

```
<div class="form-group{{ $errors->has('username') ? ' has-error' : '' }}">
  <label for="username" class="col-md-4 control-label">Username</label>

  <div class="col-md-6">
    <input id="username" type="text" class="form-control" name="username" value="{{ old('username') }}" required>

    @if ($errors->has('username'))
      <span class="help-block">
        <strong>{{ $errors->first('username') }}</strong>
      </span>
    @endif
  </div>
</div>
```

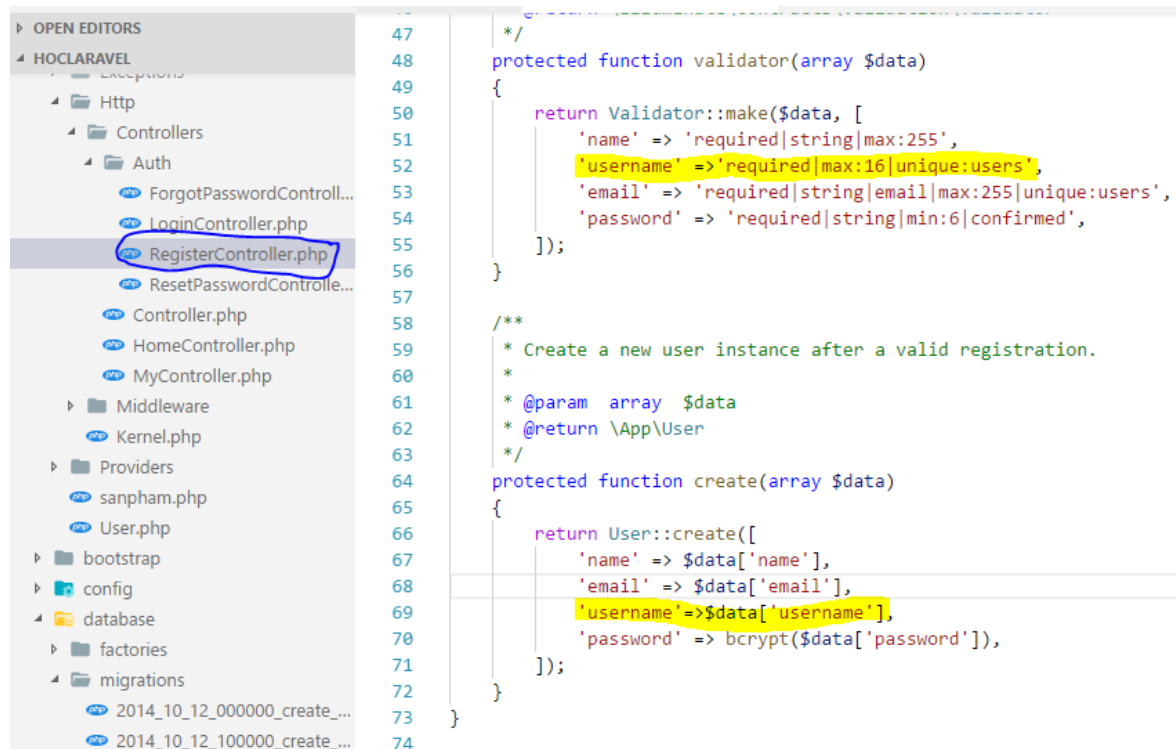
Giờ các bạn run project lên và vào register (như bài trước) và thấy giao diện đã được thêm input username rồi đúng không? Nhưng lúc này các bạn chưa đăng ký được vì các bạn mới chỉ khai báo giao diện mà chưa báo cho Laravel biết là phải thêm input username vào.

Bước 2: chỉnh sửa hệ thống

- Trong Model user bạn thêm trường username vào

```
protected $fillable = [  
    'name', 'username', 'email', 'password',  
];
```

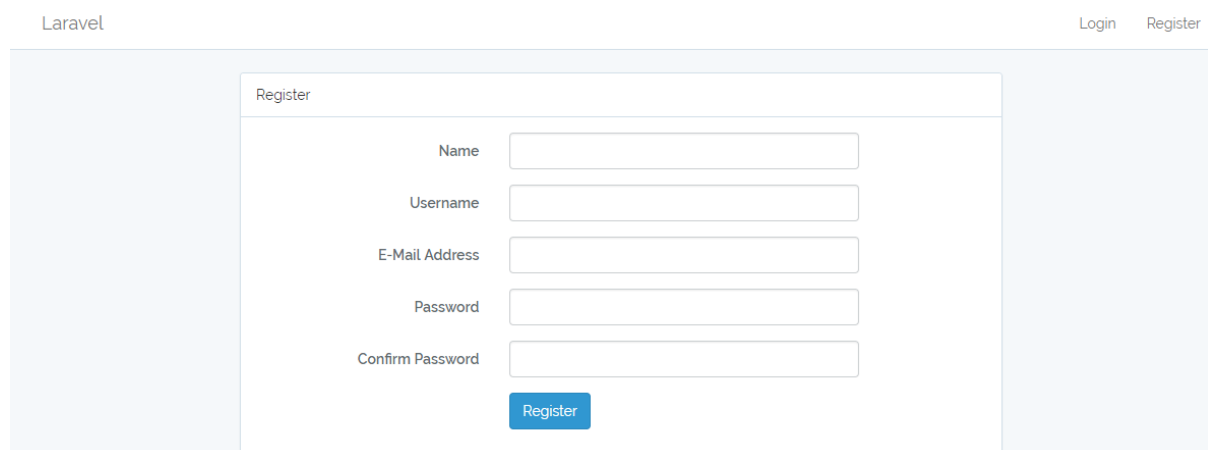
- trong RegisterController bạn thêm 2 dòng sau:



```
47  */  
48  protected function validator(array $data)  
49  {  
50      return Validator::make($data, [  
51          'name' => 'required|string|max:255',  
52          'username' => 'required|max:16|unique:users',  
53          'email' => 'required|string|email|max:255|unique:users',  
54          'password' => 'required|string|min:6|confirmed',  
55      ]);  
56  }  
57  
58  /**  
59   * Create a new user instance after a valid registration.  
60   *  
61   * @param array $data  
62   * @return \App\User  
63   */  
64  protected function create(array $data)  
65  {  
66      return User::create([  
67          'name' => $data['name'],  
68          'email' => $data['email'],  
69          'username' => $data['username'],  
70          'password' => bcrypt($data['password']),  
71      ]);  
72  }  
73  }  
74
```

dòng đầu dùng cho mục đích Validate và dòng còn lại mục đích lưu dữ liệu username từ form Register vào database.

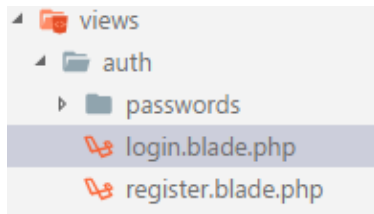
➔OK. Giờ đã xong mọi người có thể sử dụng được chức năng Đăng ký (register) của Laravel



c) Custom lại form Login

Mặc định chức năng Login của Laravel gồm có 2 thông tin là email-address và Password. Tuy nhiên giờ ta lại muốn đăng nhập bằng username và password thì sao? Để làm được như vậy ta phải custom lại chức năng Login của nó.

Bước 1: chỉnh sửa Form login



Sửa Email-address thành username:

```
<div class="form-group{{ $errors->has('username') ? ' has-error' : '' }}">
  <label for="username" class="col-md-4 control-label">username</label>

  <div class="col-md-6">
    <input id="username" type="text" class="form-control" name="username" value="{{ old('username') }}" required />

    @if ($errors->has('username'))
      <span class="help-block">
        <strong>{{ $errors->first('username') }}</strong>
      </span>
    @endif
  </div>
</div>
```

Bước 2: chỉnh sửa hệ thống

-Tiếp đó ta cần khai báo cho Laravel biết rằng là dữ liệu bạn lấy ra từ trường nào trong bảng và để làm điều đó mọi người cần phải vào LoginController (đường dẫn: app/Http/Controllers/auth/loginController.php) và thêm hàm username() vào trong class với cú pháp:

```
public function username()
{
    return 'username';//tên cột cần lấy
}
```

-Mình quay lại login.blade.php của Laravel nhưng sẽ sửa action của form thành:

```
<form class="form-horizontal" method="POST" action="{{ route('logintest') }}">
```

-Tiếp đó các bạn vào Route/web.php thêm một route để điều hướng dữ liệu như sau:

```
Route::post('logintest', 'TestController@check')->name('logintest');
```

- Tạo thêm TestController để xử lý

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class TestController extends Controller
{
    public function check(Request $request)
    {
        $data=[
            'username'=>$request->username,
            'password'=>$request->password,
        ];
        if(Auth::attempt($data)){
            //true
        }else{
            //false
        }
    }
}
```

→ như vậy chúng ta đã chỉnh sửa lại form login theo ý thích của mình rồi đấy

Bài 9: Validation trong Laravel

1. Validation là gì?

Validation là mục đích kiểm tra xem dữ liệu nhập vào form có hợp lệ hay không.

Trong Laravel hỗ trợ chúng ta mạnh về validation dữ liệu, có 3 cách Validation là:

- + Validation trực tiếp trên controller
- + Validation với lớp validator
- + Validation với FormRequest

Trong nội dung bài này mình sẽ trình bày cách đầu và cách thứ 3, còn cách sử dụng lớp validator bạn tự tìm hiểu qua: <https://toidicode.com/validation-trong-laravel-44.html>

2. Validation trực tiếp trên controller

- Để validation dữ liệu thì bắt buộc trên controller của chúng ta phải khai báo lớp http Request

```
use Illuminate\Http\Request;
```

- Route

```
Route::get('nhapthongtin', function(){
    return view('thongtin');
});
```

```
Route::post('validation', 'MyController@validation')->name('validation');
```

(1) route nhapthongtin để gọi view thongtin.blade.php, view này sẽ hiển thị form cho phép nhập name và age

```
<form action="{{route('validation')}}" method="post">
    {{ csrf_field() }}
    <p> Name: <input type="text" name="name" value="{{ old('name') }}" />
    </p>
    <p> Age: <input type="text" name="age" value="{{ old('age') }}" />
    </p>
    <p><input type="submit" name="submit"/></p>
</form>
```

→form này sau khi submit sẽ gọi route validation

(2) route validation sẽ gọi phương thức validation trong Mycontroller

```
public function validation(Request $request){
    $this->validate($request,
        [
            'name' =>'required|min:5|max:25',
            'age' =>'required|integer|max:3',
        ],
        [
            'required'=>':attribute Không được để trống',
            'min'=>':attribute Không được nhỏ hơn :min ký tự',
            'max'=>':attribute Không được lớn hơn :max ký tự',
            'integer' => ':attribute Chỉ được nhập số',
        ],
        [
            'name'=>'Tên',
            'age'=>'Tuổi',
        ]
    );
}
```

Phương thức có tác dụng kiểm tra tính hợp lệ các trường input trong form
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 các ràng buộc dữ liệu cho các trường input.

- \$messenger: Là mảng chứa nội dung báo lỗi (Nếu muốn thay đổi).
- \$customName: Là mảng chứa các tên cho các trường trong form.

Đoạn trên mình đã validation cho trường name không được để trống và dữ liệu nhập vào phải có độ dài lớn hơn 5 và nhỏ hơn 25, trường age không được để trống và dữ liệu nhập vào phải là số.

- Hiện thị lỗi trong view

```
<form action="{{route('validation')}}" method="post">
    {{ csrf_field() }}
    <p> Name: <input type="text" name="name" value="{{ old('name') }}" />
        @if ($errors->has('name'))
            <div style="color:red">{{ $errors->first('name') }}</div>
        @endif
    </p>
    <p> Age: <input type="text" name="age" value="{{ old('age') }}" />
        @if ($errors->has('age'))
            <div style="color:red">{{ $errors->first('age') }}</div>
        @endif
    </p>
    <p><input type="submit" name="submit"/></p>
```

Các cách hiển thị lỗi trong form

`$errors->has('inputName');` Để kiểm tra có tồn tại lỗi khi nhập vào input hay không.

`$errors->first('inputName');` hiển thị ra lỗi đầu tiên tìm thấy trong input đó

`$errors->all();` hiển thị ra lấy ra tất cả các lỗi trong input đó

3. Tạo Validation với FormRequest

Phần trước mình đã giới thiệu cách validation form trực tiếp trên Controller rồi, nhưng với cách đó được cho là không hay vì bạn phải lặp lại việc validate ở rất nhiều nơi, và điều đó thì đi ngược lại với nguyên tắc lập trình DRY(Don't Repeat Yourself). Và một framework được cho là siêu mạnh như Laravel thì nó đều có cách khắc phục cả.

Bước 1: Tạo file Request cho form

Để tạo một FormRequest trong Laravel thì cũng có 2 cách là tạo bằng tay và bằng lệnh mà cái gì nó hỗ trợ rồi thì tội gì mà phải tạo bằng tay:

```
php artisan make:request TênRequest
```

VD: Mình tạo một LoginRequest.

php artisan make:request LoginRequest

Ngay sau đó bạn truy cập vào [app/Http/Requests](#) sẽ thấy có một file [LoginRequest.php](#).

Bước 2: Cấu hình file Request này

Bật tắt Validate.

- Để bật tắt chức năng validation Request thì các bạn chỉ cần cấu hình giá trị trả về của authorize function:

+TRUE: bật tính năng validation.

+FALSE: tắt tính năng validation.

```
public function authorize()
{
    return true;
}
```

- Cấu hình chuỗi pattern.

Trong FormRequest để cấu hình chuỗi \$pattern các bạn phải viết trong hàm rules

```
public function rules()
{
    return [
        'name' => 'required|min:5|max:25',
        'age' => 'required|integer|max:3'
    ];
}
```

- Thay đổi nội dung báo lỗi.

Để thay đổi nội dung lỗi hiển thị thì mọi người cần phải tạo một hàm có tên messages()

```
public function messages()
{
    return [
        'required' => ':attribute Không được để trống',
        'min' => ':attribute Không được nhỏ hơn :min ký tự',
        'max' => ':attribute Không được lớn hơn :max ký tự',
        'integer' => ':attribute Chỉ được nhập số',
    ];
}
```

- Thay đổi tên input.

Và để thay đổi tên cho input thì mọi người cũng cần phải tạo thêm một hàm có tên attributes()

```
public function attributes()
{
    return [
        'name' => 'Tên',
    ];
}
```

```

        'age' => 'Tuổi',
    ];
}

```

→ nhận xét: các thiết lập trên có cú pháp giống như khi Validation trực tiếp trong Controller

Bước 3: Sử dụng lớp FormRequest trong Controllers.

- Để sử dụng lớp FormRequest trong Controllers thì trước hết các bạn phải gọi namespace của Request đó với cú pháp:

use App\Http\Requests\TênRequest;

- Tiếp đó hàm nào các bạn muốn sử dụng validation thì chỉ việc ánh xạ nó vào hàm với cú pháp:

```

public function Myfunction(LoginRequest $request)
{
    return $request->all();
}

```

→ như vậy là bạn đã thực hiện xong validation cho form

Bài 10: Làm việc với Session

1. Cấu hình Session

Để xem thông tin cấu hình của Session bạn vào file config/session.php

ở file này bạn tìm đến các dòng sau:

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'),	Chọn nơi lưu trữ session

bạn có thể chỉnh sửa lại giá trị của các thuộc tính trên cho phù hợp với website của bạn

2. Thiết lập và truy xuất Session

Để sử dụng được session thì ta phải đặt routes trong middleware web


```
//URL: ../Session
Route::group(['middleware'=>['web']],function(){
    Route::get('Session',function(){
        Session::put('KhoaHoc','Laravel');
        echo "Đã đặt Session";
        echo "<br/>";

        if(Session::has('KhoaHoc')) //kiểm tra session có tồn tại không
            echo Session::get('KhoaHoc');
        else
            echo "Không có Session KhoaHoc";

        //Session::forget('KhoaHoc');//xóa Session
    });
});
```

Thiết lập Session: Session::put("Ten_Session", "Giá_trị");

Lấy Session: echo Session::get('Ten_Session');

Kiểm tra Session có tồn tại ko: Session::has("Ten_Session") → true/false

Xóa Session: Session::forget('Ten_Session');

Bài 11: Phân trang (Pagination)

B1: Chuẩn bị CSDL

ở bài này chúng ta sẽ dùng csdl khoaphamtraining, do đó chúng ta phải kết nối với CSDL này trong file .env

chúng ta sẽ sử dụng bảng tin để thực hiện phân trang.

B2: Tạo model kết nối đến table tin

Cmd: php artisan make:model Tin

Cho model này kết nối với bản tin trong csdl

```
class Tin extends Model
{
    protected $table="tin";
}
```

B3: Tạo TinController

```

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Tin;
class TinController extends Controller
{
    public function index(){
        $tin=Tin::paginate(5);//5 tin trên một trang
        return view('tin',['tin'=>$tin]);
    }
}

```

B4: tạo view tin.blade.php

```

@foreach ($tin as $value)

    {{ $value->TieuDe}}

<br/>

```

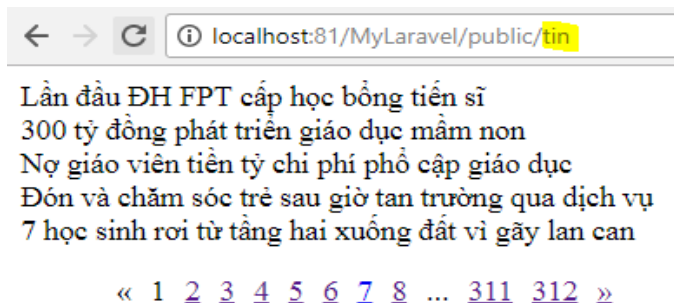
```
@endforeach
```

```
{!! $tin->links() !!}
```

B5: tạo Route

```
Route::get('tin','TinController@index');
```

→ thực thi và xem kết quả:



Bài 12: Shopping cart

1. Shopping Cart trong Laravel

Ở mỗi trang web thương mại điện tử thì chức năng đặt hàng luôn là chức năng quan trọng nhất. Ngoài cách tự viết các hàm sử dụng cho Cart thì một cách khác là bạn sử dụng các package có sẵn.

Đối với framework Laravel thì package phổ biến nhất là Crinsane - LaravelShoppingcart

Nó được hướng dẫn cụ thể tại: <https://github.com/Crinsane/LaravelShoppingcart>

2. Các bước cài đặt package vào project

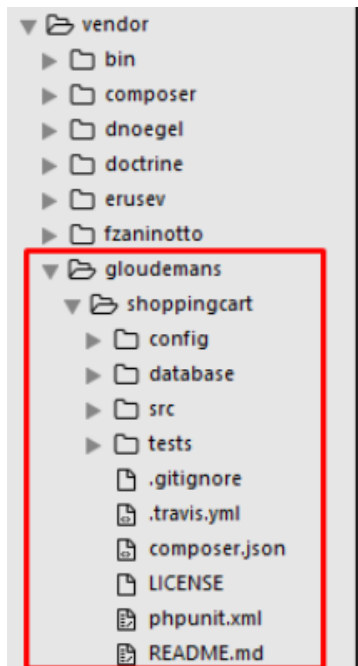
Đối với phiên bản 5.6 trở về trước

B1: Cài đặt package thông qua composer

Mở cửa sổ cmd tại thư mục project của bạn và gõ:

```
composer require gloudemans/shoppingcart
```

Chờ cho quá trình download package về.



Nếu bạn ở phiên bản 5.4 trở về trước thì mới làm tiếp b2

B2: Bạn mở config/app.php

- thêm dòng sau vào mảng providers .

```
Gloudemans\Shoppingcart\ShoppingcartServiceProvider::class
```

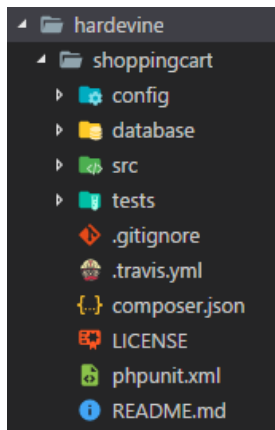
- Thêm dòng sau vào mảng aliases

```
'Cart' => Gloudemans\Shoppingcart\Facades\Cart::class
```

→Vây là xong phần cài đặt, Bây giờ chúng ta đã sẵn sàng để bắt đầu sử dụng giỏ hàng trong ứng dụng của mình.

Đối với phiên bản 5.7

```
composer require hardevine/shoppingcart
```



3. cách sử dụng package shopping cart

Shoppingcart cung cấp sẵn cho ta 1 số phương thức sau để sử dụng.

Để sử dụng các phương thức của Shoppingcart thì bạn phải khai báo trong Controller

use Cart

- Thêm sản phẩm vào giỏ hàng

```
Cart::add(id_san_pham, ten_san_pham, so_luong, gia);
```

Hoặc cấu trúc nâng cao sau:

```
Cart::add(id_san_pham, ten_san_pham, so_luong, gia, ['thongtin' => 'noi dung  
thong tin']);
```

Như vậy tham số thứ 5 là một mảng 1 chiều chứa các thông tin mà bạn muốn bổ sung vào cart ngoài các thông tin bắt buộc như id sản phẩm, tên sản phẩm, số lượng, giá.

Vd:

```
public function getMuaHang($id){  
    $sanpham=Product::find($id);  
    Cart::add($id, $sanpham->name, 1, $sanpham->unit_price,['img' =>  
$sanpham->image]);  
    return redirect()->back();  
}
```

- Cập nhật giỏ hàng

Để cập nhật một mặt hàng trong giỏ hàng, đầu tiên bạn cần rowId của mặt hàng đó (rowId là một giá trị id đại diện cho mặt hàng trong Cart do package này quy định)

+ Nếu bạn chỉ muốn update số lượng, bạn chỉ cần gọi phương thức update rowId và số lượng mới:

```
$rowId = 'da39a3ee5e6b4b0d3255bfef95601890afd80709';
```

```
Cart::update($rowId, 2);
```

+ Nếu bạn muốn cập nhật thêm các thuộc tính của sản phẩm, bạn có thể truyền vào phương thức update một mảng. Bằng cách này bạn có thể cập nhật tất cả thông tin của sản phẩm với rowId.

```
Cart::update($rowId, ['name' => 'Product 1']); // Will update the name
```

- Lấy thông tin của tất cả sản phẩm trong giỏ hàng

```
$products = Cart::content();
```

Nếu bạn xuất ra `$products` thì sẽ có các thông tin sau:

```
Illuminate\Support\Collection Object
(
    [items:protected] => Array
        (
            [104a7fa4f09bb496c05b7a460aa982de] => Gloudemans\Shoppingcart\CartItem Object
                (
                    [rowId] => 104a7fa4f09bb496c05b7a460aa982de
                    [id] => 7
                    [qty] => 1
                    [name] => Bánh Crepe Táo
                    [price] => 160000
                    [options] => Gloudemans\Shoppingcart\CartItemOptions Object
                        (
                            [items:protected] => Array
                                (
                                    [img] => crepe-tao.jpg
                                )
                        )
                    [associatedModel:Gloudemans\Shoppingcart\CartItem:private] =>
                    [taxRate:Gloudemans\Shoppingcart\CartItem:private] => 21
                )
        )
    )
)
```

Do đó để xuất thông tin của từng sản phẩm trong giỏ hàng thì bạn phải gọi đúng theo tên thuộc tính mà package này quy định (khác với tên thuộc tính của bảng sản phẩm trong CSDL)

```
@foreach($content as $row)
<tr class="cart_item">
    <td class="product-name">
        {{ $row->name }}
    </td>

    <td style="font-size:14px">
        {{ $row->price }}
    </td>

    <td class="product-quantity">
        {{ $row->qty }}
    </td>
</tr>
```

- Xóa một sản phẩm trong giỏ hàng

```
$rowId = 'da39a3ee5e6b4b0d3255bfef95601890afd80709';
```

```
Cart::remove($rowId);
```

- Huỷ giỏ hàng

```
Cart::destroy();
```

- Tính tổng giá của tất cả sản phẩm trong giỏ hàng

```
$total=Cart::subtotal();
```

Ngoài ra còn nhiều phương thức khác bạn có thể tham khảo ở đây

<https://viblo.asia/p/laravel-5x-shopping-cart-p1-bJzKmWXwI9N>

Hoặc trực tiếp trên trang tài liệu github của backage này (link để ở đầu bài)

Bài 13: RESTful Controller

1. Khái niệm ban đầu về API và Web API

API là gì

API là cụm viết tắt của *Application Programming Interface* (giao diện lập trình ứng dụng). Đây là một giao tiếp phần mềm được dùng bởi các ứng dụng khác nhau. Cũng giống như bàn phím là thiết bị giao tiếp giữa người dùng và máy tính, thì API là giao tiếp giữa các phần mềm với nhau, ví dụ như giữa chương trình và hệ điều hành (OS).

Phân loại API

1. Hệ thống API trên nền tảng web, hay gọi là web API

Loại API này hiện đang rất phổ biến, các website lớn đều cung cấp hệ thống API cho phép bạn kết nối, lấy dữ liệu hoặc cập nhật dữ liệu vào hệ thống.

Ví dụ, nếu bạn sử dụng dịch vụ bán hàng trực tuyến của Lazada, bạn sẽ cần phải thực hiện một số các tác vụ như tạo sản phẩm mới, cập nhật sản phẩm mới. Giả sử bạn có 1000 mặt hàng, việc cập nhật số lượng tồn kho bằng tay rất vất vả chưa kể nhầm lẫn do chủ quan người nhập. Lazada cung cấp hệ thống API, từ đây bạn có thể xây dựng một kết nối từ hệ thống của bạn sang Lazada và mọi thứ (trong đó có kho hàng, số tồn...) sẽ được đồng bộ với nhau.

2. Hệ thống API trên Hệ điều hành

Khái niệm này có trước cả web API, Microsoft cung cấp các hệ điều hành Windows cùng các tài liệu API là đặc tả các hàm, phương thức, lời gọi hàm cũng như các giao thức kết nối cho lập trình viên, giúp lập trình viên có thể tạo ra các phần mềm ứng dụng có thể tương tác trực tiếp với hệ điều hành.

3. Các API của thư viện phần mềm hoặc framework

API mô tả và quy định các hành vi mong muốn mà các thư viện cung cấp, một API có thể có nhiều các triển khai khác nhau và nó cũng giúp cho một chương trình viết bằng ngôn ngữ này có thể sử dụng thư viện được viết bằng ngôn ngữ khác. API cũng có thể liên quan đến các framework khi framework được xây dựng trên nhiều các thư viện và thực thi nhiều các API khác nhau. Tuy nhiên việc sử dụng API trên framework không giống với thông thường, truy cập đến các API được xây dựng trong framework sẽ mở rộng nội dung của nó và các class mới được "cắm" vào (plug) khung tự nó. Kiểm soát tổng thể luồng ứng dụng có thể

nằm ngoài tầm kiểm soát của thành phần gọi bằng cách đảo ngược kiểm soát – Inversion of Control.

Hiểu hơn về API qua ví dụ về Facebook

Bạn có thể sử dụng tài khoản Facebook của mình để đăng nhập vào rất nhiều trang web không do Facebook kiểm soát. Để người dùng của mình có thể sử dụng thông tin cá nhân Facebook trên các trang này, điều duy nhất mạng xã hội này cần làm là tạo ra một API đăng nhập tài khoản Facebook. Mỗi lần bạn click vào nút "Đăng nhập với Facebook" trên Instagram, WhatsApp hay Quora thì các trang web/ứng dụng này sẽ "gọi" tới API của Facebook. Công việc xác thực danh tính sẽ được Facebook thực hiện, các trang web và các ứng dụng không cần phải nhúng tay vào. Sau khi xác thực xong, Facebook sẽ "ném" lại cho các trang web và ứng dụng gọi tới API của mình trên một gói tin có nội dung đại loại như "Đây là anh Lê Hoàng, tài khoản Facebook là abcxyz" chẳng hạn. Nhờ có API mà Facebook có thể thực hiện tính năng xác thực hộ các dịch vụ khác.

2. Web API

Web API hỗ trợ RESTfull đầy đủ phương thức: GET/ POST/ PUT/ DELETE dữ liệu.

REST (Representational State Transfer) là kiến trúc được sử dụng trong việc giao tiếp giữa các máy khách và máy chủ, trong việc quản lý các tài nguyên trên internet. REST được sử dụng rất nhiều trong việc phát triển các ứng dụng **Web Services** sử dụng giao thức HTTP trong giao tiếp thông qua mạng internet. Các ứng dụng sử dụng kiến trúc REST này được gọi là ứng dụng phát triển theo kiểu RESTful

3. RestFul Controller

1. Tạo một Restful controller

- Tạo một restful controller trong laravel

php artisan make:controller TenController --resource

- Một RESTful controller sẽ có dạng như sau:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyController extends Controller
{
    public function index() {
        //
    }

    public function create() {
        //
    }

    public function store(Request $request) {
        //
    }
}
```

```

    }

    public function show($id) {

        //

    }

    public function edit($id) {

        //

    }

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

        //

    }


    public function destroy($id) {

        //

    }

}

```

- index(): Hiển thị một danh sách.
- create(): Thêm mới.
- store(): Lưu mới.
- show(\$id): Hiển thị một dữ liệu theo tham số truyền vào.
- edit(\$id): Sửa một dữ liệu theo tham số truyền vào.
- update(\$id): Cập nhật một dữ liệu theo tham số truyền vào.
- destroy(\$id): Xóa một dữ liệu theo tham số truyền vào.

- Để thực thi Restful controller này ta phải tạo một route
Route::resource('url', 'TenController');

- Tùy biến cho route

Route::resource('url', 'tencontroller', 'tuybien');

Qua phần trên có bạn nào thắc mắc là : "chẳng lẽ RESTful Controller nào cũng phải đáp ứng đủ các action như trên Nếu không đủ thì không dùng được à" --> Vấn đề này các bạn khỏi lo nhé, vì Laravel cung cấp chức năng lọc các phương thức được sử dụng trong RESTful

+ chỉ cho sử dụng các action được khai báo còn lại sẽ không được sử dụng:

Route::resource('url', 'TenController', ['only' => ['index', 'create', 'show', 'edit']]);

+ Cấm không cho sử dụng action index còn lại được sử dụng hết:

Route::resource('url', 'TenController', ['except' => ['index']]);

2. Ví dụ demo về các viết cũng như cách gọi các action trong RESTful controller

B1: Tạo route

```
Route::resource('sanpham', 'ResController');
```

B2: Tạo RESTful controller

B3: Tạo app giả lập cho thiết bị bên ngoài để thao tác với các action trong RESTful controller của ta.

Ở đây tôi tạo 1 file app.php nằm trong thư mục chứ project

Các thực thi các action

Controller SanPham	Action	/SanPham	/SanPham	/SanPham/1	/SanPham/1	/SanPham/1
	Method	GET	POST	GET	POST	POST
	Parameters				PUT	DELETE
	Function	index	store	show	update	destroy
	Mô tả	Lấy tất cả danh sách sản phẩm	Thêm sản phẩm mới	Lấy chi tiết sản phẩm có id=1	Cập nhật sản phẩm có id=1	Xóa sản phẩm có id=1

1.Action Index (lấy danh sách sản phẩm)

- app.php

```
<form action="http://localhost:81/HocLaravel/Shop/public/sanpham"
method="GET">
    <input type="submit" value="Lấy tất cả sp">
</form>
```

- Controller

```
public function index()
{
    $sanpham = product::all();
    return $sanpham; //trả về json
}
```

→ thực thi: chạy file app.php -> hiển thị form -> nhấn submit

← → ↻ ⓘ localhost:81/HocLaravel/Shop/public/sanpham?

```
[
  - {
    id: 1,
    name: "Bánh Crepe nhân Sầu riêng",
    id_type: 5,
    description: "Bánh crepe sầu riêng nhà làm",
    unit_price: 150000,
    promotion_price: 120000,
    image: "1430967449-pancake-sau-rieng-6.jpg",
    unit: "hộp",
    new: 1,
    created_at: "2016-10-26 10:00:16",
    updated_at: "2019-01-25 03:01:59"
  },
  - {
    id: 2,
    name: "Bánh Crepe Chocolate",
    id_type: 6,
    description: "",
    unit_price: 180000,
    promotion_price: 160000,
    image: "crepe-chocolate.jpg",
    unit: "hộp",
    new: 1,
    created_at: "2016-10-26 10:00:16",
    updated_at: "2016-10-25 05:11:00"
  },
  ,
]
```

2. Action store (thêm mới 1 sản phẩm)

- App.php

```
<form action="http://localhost:81/HocLaravel/Shop/public/sanpham"
method="POST">
  <input type="text" name="tensp">
  <input type="file" name="hinh">
  <input type="submit" value="Thêm sp mới">
</form>
```

-Controller

```
public function store(Request $request)
{
    $sanpham = new product;
    $sanpham->name = $request->tensp;
    $sanpham->id_type = 5;
    $sanpham->description = "";
    $sanpham->unit_price = 160000;
    $sanpham->promotion_price= 0;
    $sanpham->save();

    return response($sanpham,201);//trả về json
}
```

➔ thực thi

- Chạy file app.php -> hiển thị form

sản phẩm tạo từ api

Chọn tệp

Không có tệp nào được chọn

Thêm sp mới

- submit

← → ↻ ⓘ localhost:81/HocLaravel/Shop/public/sanpham

```
{
  name: "sản phẩm tạo từ api",
  id_type: 5,
  description: "",
  unit_price: 160000,
  promotion_price: 0,
  updated_at: "2019-01-26 13:22:06",
  created_at: "2019-01-26 13:22:06",
  id: 63
}
```

* Nếu có lỗi do csrf thì vào app/Http/Middleware/VerifyCsrfToken.php

```
protected $except = [
    '*' //cho phép tất cả các controller của chúng ta được truy cập từ bên
ngoài
];
```

Tương tự các action còn lại tham khảo trong project Laravel_Shop

TÀI LIỆU THAM KHẢO:

[1] <https://toidicode.com/hoc-laravel>

[2] <https://khoapham.vn/download/laravel/bai1.pdf> ... Bai12.pdf

Laravel 5.x cơ bản version-1/2018