# **Bài 1: Laravel là gì? cách cài đặt laravel**

## 1, Laravel là gì?

-Laravel là một php framework mới, ra đời vào tháng 04/2011.Ngay khi vừa mới ra mắt thì nó đã được cộng đồng chú ý đến bởi nhiều đặc điểm và tính năng mới như Eloquent ORM, RESTful Controllers, ...

- Từ lúc ra đời đến nay thì laravel đã trải qua rất nhiều bản cập nhật, đến tại thời điểm mình viết bài thì nó đang là laravel version 5.3 và cũng đương nhiên là mình sẽ hướng dẫn các bạn tiếp cận với laravel version 5.3.

## 2, Cấu hình yêu cầu khi cài đặt laravel.

* PHP >= 5.6.4
* OpenSSL PHP Extension
* PDO PHP Extension
* Mbstring PHP Extension
* Tokenizer PHP Extension
* XML PHP Extension

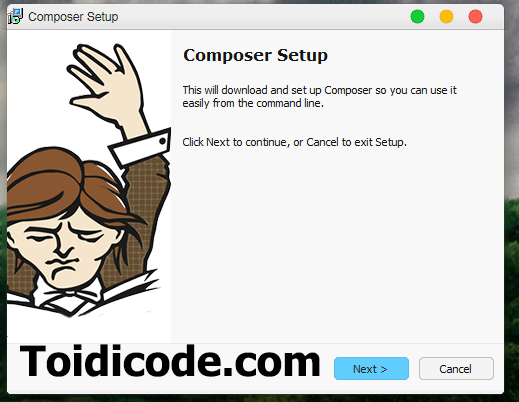
## 3, Download và cài đặt laravel.

-Đầu tiên các bạn cần phải cài đặt biến môi trường cho PHP ( xem thêm).

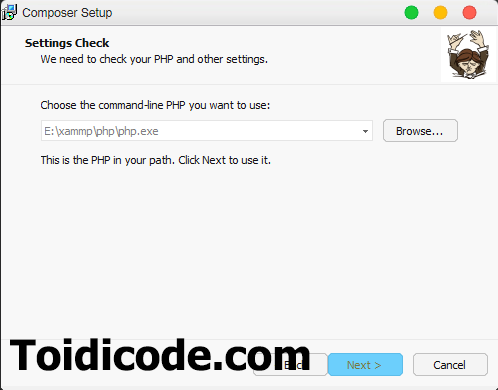
- Sau đó các bạn sẽ tiến hành cài Laravel bằng **2 cách** sau đây.

### Cách 1: cài laravel bằng comporser.exe

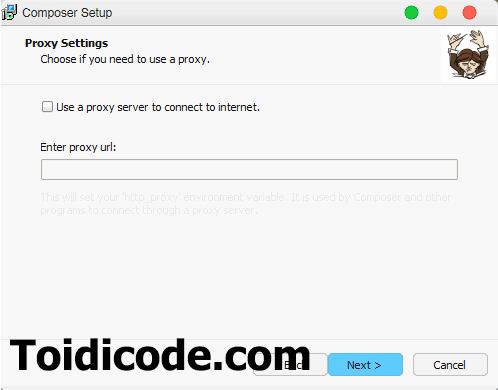
* Đầu tiên các bạn lên trang chủ  getcomposer.org và download file [**composer\_setup.exe**](https://getcomposer.org/Composer-Setup.exe)về**.**
* Sau đó mở file lên và cài đặt:

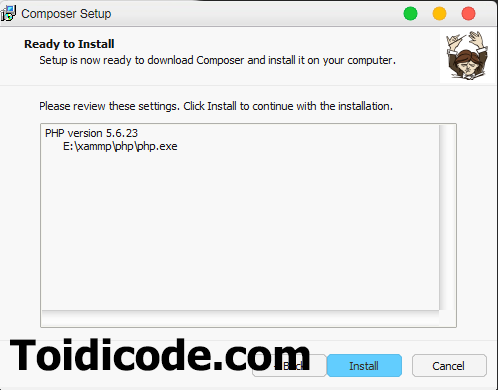


Nhấn next để tiếp tục



Chọn thư mục chứa php (mặc định thì composer tự tìm cho chúng ta) sau đó chọn next

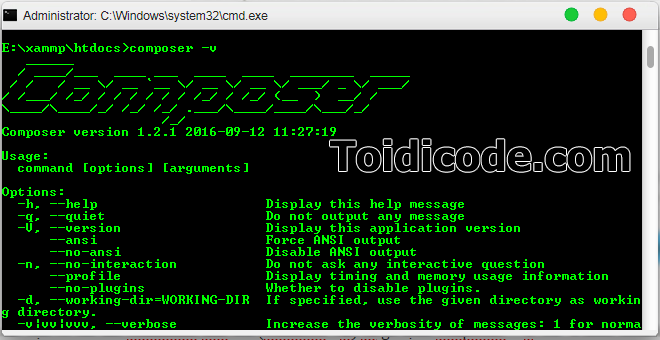


Và lại tiếp tục next

Cuối cùng là install.

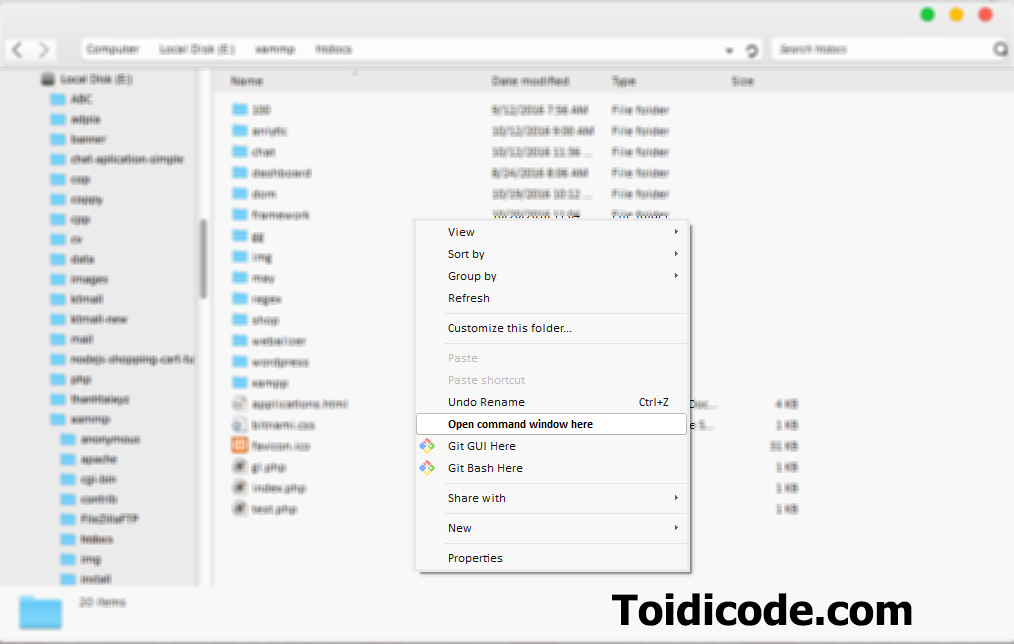
-Sau khi đã cài đặt xong các bạn có thể mở command line lên (windows +R) và gõ lệnh composer -v.

Nếu màn hình hiện nên dạng như này là bạn đã cài đặt thành công rồi.



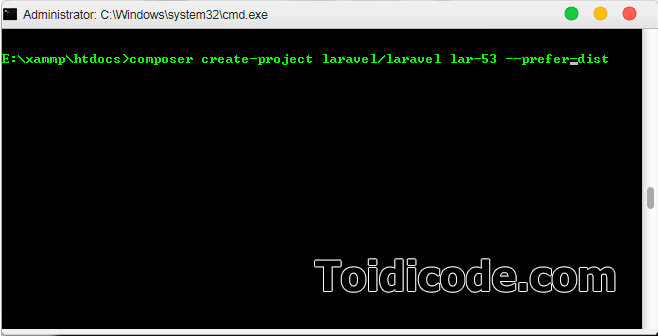
Ok! giờ chúng ta bắt đầu tiến hành tạo 1 project laraver đầu tiên nhé!

Các bạn sử dụng command line để di chuyển đến thư mục cần tạo project laravel (của mình cài xammp lên mình sẽ cho trỏ đến **htdocs**) hoặc các bạn có thể vào thư mục muốn tạo project rồi ấn **shift + chuột phải --> open command windows here**



-Sau đó các bạn gõ lệnh composer create-project laravel/laravel lar-53 --prefer-dist.

* Ở đây lar-53 là thư mục chứa laravel các bạn có thể tùy chỉnh thành thư mục của bạn.



Ấn enter và đợi nó download thôi



### Cách 2: Cài bằng composer.phar.

-Để cài bằng cách này thì các bạn cần phải tải srouce code của laravel về trước : Tải tại [**đây**](https://github.com/laravel/laravel)**.**

-Sau khi đã tải xong các bạn dùng cmd trỏ tới thư mục chứa project và gõ lệnh composer.phar install . Và đợi nó download thư viện vendor. tiếp theo các bạn phải sử dụng tiếp lệnh php artisan key:generate để kích hoạt **application key** của laravel (Từ phiên bản 5.2 trở lên mới có).

## 4,Khởi chạy Laravel.

-Sau khi đã hoàn thành các bước cài đặt ở trên thì các bạn: tiếp tục mở command line lên mà gõ lệnh artisan serve để khởi động server cho project laravel. hoặc Nếu bạn dùng **xammp** hay các phầm khác thì có thể dẫn đến thư mục public của laravel. vd: localhost/lar-53/public . 

## 5, Lời kết.

-Bài này mình đã giới thiệu qua về laravel và các cách cài đặt cũng như chạy nó trên hệ điều hành windows còn bạn nào muốn mình hướng dẫn cài laravel trên linux thì để lại thông tin ở phía dưới nhé!

# **Bài 2: Cấu trúc của laravel (5.3)**

## 1,Cấu trúc của Laravel.

-Ở bài trước mình đã giới thiệu qua về Laravel và các cách [cài đặt](https://toidicode.com/laravel-la-gi-cach-cai-dat-laravel-1.html) nó trên windows rồi. Sau khi đã cài đặt được nó rồi thì chúng ta bắt đầu xem qua về cấu trúc của nó nhé.

lar-53/

|--- app/

│ |--- Console

│ |--- Exceptions

│ |--- Http

│ │ |--- Controllers/

│ |--Auth

│ |--Controller.php

│ │ |--- Middleware

│ │ └── Kernel.php

│ |--- Providers

| └--- User.php

|--- bootstrap/

|--- config/

|--- database/

|--- public/

| |--- ...

| |--- index.php

|--- resources/

| |--- assets

| |--- lang

| |--- views

|--- routes/

| |--- api.php

| |--- console.php

| |--- web.php

|--- storage/

|--- tests/

|--- vendor/

|--- .env

|--- .env.example

|--- .gitattributes

|--- .gitignore

|--- artisan

|--- composer.json

|--- composer.lock

|--- gulpfile.js

|--- package.json

|--- phpunit.xml

|--- readme.md

|--- server.php

- **Chú thích**:

* **app:** Chứa các thư mục, các tập tin php, thư viện, models.
  + **Console:** Chứa các tập tin định nghĩa các câu lệnh trên artisan.
  + **Exception:**  Chứa các tập tin quản lý, điều hướng lỗi.
  + **Http**
    - **Controllers** : Chứa các controllers của project.
    - **Middleware:**Chứa các tập tin lọc và ngăn chặn các requests.
    - **Kernel.php:**Cấu hình, định nghĩa Middleware.
  + **Providers:**Chứ các providers thực hiện việc **binding vào service container**(ở phần nâng cao mình sẽ nói sau).
  + **User.php:**Là model User mà Laravel tự địn sẵn cho chúng ta.
* **bootstrap:**Chứa tập tin điều hướng hệ thống.
* **config:** Chứa mọi tập tin cấu hình của Laravel.
* **database:** Chứa các thư mục tập tin vềdatabase.
  + **migrations:** Chứa các tập tin định nghĩa,khởi tạo và sửa bảng.
  + **seeds:** Chứa các tập tin định nghĩa dữ liệu insert(thêm) vào trong database.
  + **factories:** Chứa các tập tin định nghĩa các cột bảng dữ liệu để tạo ra các dữ liệu ảo.
* **public:** Chứa các tập tin css, js, image.
  + **index.php:**Đây là tệp tin root của Laravel.
* **resources:** Chứa các views, ngôn ngữ(language) của project.
* **routes:** Chứa các tập tin định nghĩa các router, xử lý điều hướng router bao gồm : **web, api và console**(Mình sẽ nói thêm ở phần sau).
* **storage:** Chứa các tập tin hệ thống cache, session, ...
* **tests: ...**
* **vendor:** Chứa các package mà laravel yêu cầu.
* **.env:** Là tập tin cấu hình chính của laravel như key app,database.
* **.env.example:**Tệp tin cấu hình mẫu của laraver.
* **composer.json:** tập tin của composer.
* **composer.lock:** tập tin của composer.
* **package.js:** Tập tin cấu hình của [nodejs](https://toidicode.com/nodejs-la-gi-26.html) (chứa các package cần dùng cho projects).
* **gulpfile.js:** Là tập tin gulp builder.
* **phpunit.xml:** Là tập tin xml của phpunit dùng để testing project.
* **server.php:** Là tập tin để artisan trỏ đến tạo server khi gõ lệnh php artisan serve .
* **artisan:** Tập tin thực thi lệnh của Laravel.

-Ở trên là một số các thư mục trong laravel mà chúng ta lên biết thôi vẫn còn rất nhiều các thư mục khác cấu thành lên laravel nữa các bạn tự tìm hiểu thêm nhé.

## 2, Lời kết.

-Qua phần trên mình mong rằng các bạn đã hiểu thêm về chức năng của các thư mục trong Laravel. Vào các phần sau mình sẽ bắt đầu đi vào tìm hiều từng thành một trong laravel nhé.

# **Bài 3: Route trong Laravel (phần 1)**

Bài trước chúng ta đã cùng tìm hiểu về cấu trúc thư mục của Laravel rồi. Ở bài này chúng ta sẽ tìm hiểu tiếp về route trong Laraver.

## 1, Route trong laravel là gì?

-Route trong Larvel cũng như các framework khác  đều có chức năng là định ra các dạng resquest(hiểu nôm na là url) định sẵn và cũng có các action( hành động) do chúng ta định nghĩa cho nó . Hơi khó hiểu đúng không các bạn? Mình sẽ trình bày ví dụ cụ thể ở dưới cho mọi người dễ hiểu.

## 2,Các Loại Route trong Larvel.

-Laravel là một framework hỗ trợ chúng ta rất đa dạng về route, và cũng rất dễ sử dụng. Nó gồm có các loại như sau:

* **Route::get**nhận resquest với phương thức GET.
* **Route::post** nhận resquest với phương thức POST.
* **Route::put** nhận resquest với phương thức PUT.
* **Route::delete** nhận resquest với phương thức DELETE.
* **Route::match** kết hợp nhiều phương phức như POST,GET,PUT,..
* **Route::any** nhận tất cả các phương thức.
* **Route::group**tạo ra các nhóm route.
* **Route::controller**gọi đến controller tương ứng mà chúng ta tự định.
* **Route::resource**sử dụng với resource controller.

## 3,Sử dụng route trong Laravel.

-Để sử dụng được route thì các bạn mở thư mục laravel vừa tạo ở [bài 1](https://toidicode.com/laravel-la-gi-cach-cai-dat-laravel-1.html) sau đó chúng ta truy cập routes/web.php để bắt đầu viết code route.

### a) Route::get

-Đây là route sẽ tiếp nhận các request với phương thức GET.

**-Cú Pháp:**

Route::get($url, $action);

Trong đó: $url là đường dẫn của route, $action là các hành động thực hiện trong router đó.

**-Ví Dụ:**

* Ví dụ mình muốn khi người dùng vào http://localhost/welcome (ở đây server ảo của mình là localhost) thì sẽ cho in ra chữ "Chào mừng các bạn đã đến với toidicode.com" thì mình sẽ định route như sau.

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

return 'Chào mừng các bạn đã đến với toidicode.com';

});

hoặc có thể sử dụng 2 cách sau nữa nhưng mình sẽ giải thích thêm ở phần controller sau.

Route::get('welcome', 'WelcomeController@index');

hoặc

Route::get('welcome', ['as' => 'name', 'uses' => 'WelcomeController@index']);

* Ngoài ra vẫn còn 1 kiểu nữa nhưng mình sẽ nói ở phần view trong laravel nhé.

### b) Route::post

-Đối với loại route này thì nó không khác gì với Route::get() ngoài : Route::get() nhận phương phương thức **GET** còn Route::post() nhận phương thức **POST.** **-Cú Pháp:**

Route::post($url, $action);

### c)Route::match

-Nếu muốn sử dụng hai phương thức GET,POST mà không muốn dùng phương thức PUT,PATCH,DELE,... thì làm sao nhỉ? .Nhận thấy điều đó nên Laravel đã cho chúng ta thêm một route nữa là match route.

**-Cú Pháp:**

Route::match('phuong thuc', 'url', 'action');

* Phương thức: có thể là một mảng POST,GET,...  vd: ['POST','GET']
* url: thì không khác gì như Route::get.
* action: cũng không khác gì Route::get.

### d)Route::any

-Sử dụng một số phương thức trong route thì dùng R**oute::any()** rồi, còn nếu như muốn dùng tất cả các phương thức thì chẳng lẽ chúng ta cứ phải viết đi viết lại 1 url với từng phương thức sao? Cũng vì điều này mà Laravel đã cung cấp cho chúng ta Route::any().

**-Cú pháp:**

Route::any($url, $action);

-Còn cách sử dụng thì không khác gì Route::get() cho mấy.

## 3,Lời Kết.

-Ở bài này chúng ta đã tìm hiểu một số route trong Laravel rồi, còn những route put,delete thì cũng tương tự như **Route::get** nhé. Bài sau mình sẽ cùng tìm hiểu về **Route::group()** và **Route::controller()**.

# **Bài 4: Route trong Laravel (phần 2)**

Ở [bài trước](https://toidicode.com/route-trong-laravel-phan-1-3.html) chúng ta đã tìm hiểu về Route::get(),post(),any(),match(),.. rồi, ở phần này chúng ta sẽ tìm hiểu tiếp về Route::resource() và Route::group().

## 1,Route::resource().

-Route::resource() là một chức năng giúp chúng ta Xây dựng RESTful(websevice) một cách nhanh chóng. Đối với những người mới tiếp cận với PHP,Laravel thì chưa cảm thấy được ý nghĩa tuyệt vời của RESTful controller.

**-Cú Pháp:**

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

Trong đó:

* Url: thì cũng như các route khác là đường dẫn của route.
* tencontroller: là tên của controller (không đi kèm đuôi .php).
* tuybien: là các tùy biến phương thức được sử dụng trong Route, Tham số này có thể bỏ qua nếu không cần thiết.

**-Chú ý:**Để sử dụng được Route::resource() thì các bạn cần phải tạo ra một RESTful Controller. -Để tạo Một RESTful Controller chúng ta có hai các tạo:

* Cách 1: Là tạo bằng tay và viết code cũng bằng tay.
* Cách 2: Dùng lệnh mà laravel cung cấp sẵn cho chúng ta. Các bạn mở commanline lên và trỏ đến thư mục chứa project Laravel (VD của tôi là lar-53) và gõ lệnh.

php artisan make:controller TenController --resource

Trong đó tencontroller là tên của các bạn muốn tạo.

-Một RESTful controller sẽ có dạng như sau(trong code demo của tôi tên controller là HomeController):

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

//

}

/\*\*

\* Show the form for creating a new resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function create()

{

//

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

//

}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

//

}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit($id)

{

//

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

//

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

**-Chú thích các function:**

* 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.

-Chú Ý: Điểm mạnh của RESTful Controller là sinh ra cho chúng ta các phương thức luôn có trong một ứng dụng **VD:**Tôi có một route.

Route::resource('photo', 'PhotoController');

-Thì phương thức cũng như kiểu request của nó sẽ như sau.

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

-**Trong đó:**{photo} là các tham số id của photo (xem truyền tham số trong route).

#### -Tùy biến:

-Qua phần trên có bạn nào thắc mắc là : "chẵng 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  đang đứng top 1 về số lượng developer sử dụng nó, nên trong Laravel đã cung cấp cho chúng ta lọc action trong Route::resource(). +Lọc các phương thức được sử dụng trong RESTful :

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

Chú thích: Ở trên là đoạn code chỉ cho sử dụng các action được khai báo còn lại sẽ không được sử dụng +Lọc các phương thức không được sử dụng trong RESTful:

Route::resource('duongdan', 'HomeController', ['except' => ['index']]);

Chú thích: Cấm không cho sử dụng action index còn lại được sử dụng hết.

+Truyền nhiều tham số trong action.

Route::resource('anh.author', 'HomeController');

Khi đó bên action show của controller sẽ có dạng như sau:

public function show($id, $author)

{

// code

}

Để sử dụng các bạn chạy đường dẫn:

anh/{id}/author/{author}

// vd anh/1/author/tai

## 2,Route::group()

-Chúng ta muốn xây dựng trang quản trị cho website mà đường dẫn bắt đầu lúc nào cũng phải là admin và phải đăng nhập mới được sử dụng. VD:admin/product/addproduct,admin/category/addcategory,... thì cứ phải gõ từng route rồi check authencation cho từng route sao. -Đối với vấn đề trên Laraver đã cung cấp cho chúng ta một loại Route::group() để giải quyết.

**-Cú pháp:**

Route::group($attr, $handle);

Trong đó:

* $attr: là các mảng thành phần điều kiện. VD: middleware,prefix,namespace,domain.
* $handle: là các đoạn sử lý thường là hàm.

Ví dụ: -Check đăng nhập mới được sử dụng Route bằng middeware( những phần sau mình sẽ nói về middeware).

Route::group(['middleware' => 'auth'], function () {

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

// Uses Auth Middleware

});

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

// Uses Auth Middleware

});

});

-Thêm tiền tố trước các thành phàn trong group.

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

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

// route nay se co duong dan là admin/users

});

});

-Các thành phần điều kiện khác các bạn có thể xem tại :  https://laravel.com/docs/5.3/routing#route-groups

## 3,Lời kết.

-Qua bài này mình đã trình bày cho các bạn về 2 lại route resource và group . Bài sau mình sẽ nói thêm về cách truyền tham số trong route.

# **Bài 5: Route trong Laravel (Phần 3)**

Ở phần trước mình đã giới thiệu về các route trong Laravel rồi. Bài này mình sẽ giới thiệu tiếp về truyền tham số trong route.

## 1,Truyền biến trong route.

-Route ở trong Laravel hay cũng như bất kỳ các route của các framework khác, thì đều có hỗ trợ chúng ta truyền tham số (biến).

-Ở trong Laravel để khai báo tham số trong route thì cần phải đặt trong dấu {}. VD:

Route::get('hoten/{ten}', function ($ten) {

return 'hello ' . $ten;

});

Tương tự chúng ta có thể truyền nhiều tham số trong route được. VD:

Route::get('thongtin/{tuoi}/{ten}', function ($tuoi, $ten) {

return "Hello $ten , $tuoi tuổi";

});

## 2,Điều kiện cho tham số truyền vào.

-Như ở 2 ví dụ trên các bạn cũng đã thấy, nếu như ở biến {ten} ta truyền vào là số và ngược lại biến {tuoi} ta truyền vô là chuỗi thì có phải là kỳ quá không. Thì ở đây laravel cũng đã hỗ trợ chúng ta regex(Regular Expression) để ràng buộc các tham số truyền vào trong route.

-Như ở ví dụ số 2 trên thì chúng ta có thể ràn buộc như sau.

Route::get('thongtin/{tuoi}/{ten}', function ($tuoi, $ten) {

return "hello $ten , $tuoi tuổi";

})->where(['tuoi' => '[0-9]+', 'ten' => '[a-z]+']);

## 3,Lời kết.

-Qua  phần trên chắc hẳn các bạn đã biết tryền tham số trong route rồi đúng không. Phần tiếp theo mình sẽ trình bày tiếp về view và blade template trong Laravel

# **Bài 6: View Trong Laravel**

Ở các [bài trước](https://toidicode.com/route-trong-laravel-phan-3-5.html) mọi người đã được tìm hiểu về Cấu trúc cũng như là route trong Laravel rồi đúng không? Thì tiếp đến phần này mình sẽ giới thiệu về view Một thành phần khá là quan trọng trong mô hình MVC framework.

## 1,Tạo view trong Laravel.

-View Trong Laravel thì bắt buộc phải được nằm trong thư mục resources/views và phải có đuôi là .php hoặc .blade.php (Nếu như bạn muốn sử dụng blade template).  Còn việc tạo 2 file này như thế nào thì chắc hẳn các bạn cũng viết rồi.

-Trong view thì các bạn có thể sử dụng tất cả các ngôn ngữ trong file PHP hỗ trợ như: HTML,CSS,JS,.. VD: Mình sẽ tạo một view có tên hello.php và một view toidicode.blade.php vào trong thư mục resources/views. File hello.php có nội dung:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>hello cac ban</title>

<link rel="stylesheet" href="">

</head>

<body>

hello cac ban!

</body>

</html>

Và trong toidicode.blade.php có nội dung:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>hello cac ban</title>

<link rel="stylesheet" href="">

</head>

<body>

{{ 'toidicode.com' }} hello cac ban!

</body>

</html>

-Như ở trên các bạn thấy trong file toidicode.blade.php có sử dụng đoạn {{ 'toidicode.com'}} đây là đoạn code xử lý của blade template. Và đó cũng là lý do mình khuyên các bạn nên sử dụng phần view mở rộng của laraver.

Chú Ý: Khi sử dụng kết hợp với angularjs thì không được dùng blade template.

#### -Tạo thư mục cho view:

+Để tạo các thư mục con cho view thì các bạn chỉ cần tạo một thư mục nữa trong resources/views thì larver cũng chấp nhận đó là view(Cứ nằm trong thư mục resources/views là được).

VD: Trong thư mục Resources/views mình sẽ tạo thêm một thư mục admin , và trong thư mục admin mình tạo thêm một view có tên là login.blade.php có nội dung như sau:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Login</title>

<link rel="stylesheet" href="">

</head>

<body>

Đây là trang login

</body>

</html>

-như thế là chúng ta đã tạo được view rồi đúng không.

## 2,Gọi view trong Laravel.

-Ở phần  1 chúng ta đã tạo được view ở trong laravel rồi đúng không. Giờ chúng ta sẽ tiến hành triệu tập view trong Laravel nhé. Cú pháp:

view('duongdantoiview', 'dulieucantruyenvaoview');

Chú thích:

* duongdantoiview: Là đường dẫn tới file view lấy từ thư mục resources/views . VD như mình muốn gọi file view hello ở vd trên: view('hello'). Gọi view login trong thư mục admin **view('admin.login')**
* dulieutruyenvaoview: là dữ liệu mà các bạn muốn truyền vào view(nếu không có thì bỏ qua).

#### -Gọi view trong route:

-Cú Pháp:

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

return view('tenview', 'dataneuco');

});

VD: mình sẽ tạo 1 route rồi gọi một view và truyền luôn tham số vào bên trong cho view.

* Route:

Route::get('chao/{user}', function ($user) {

return view('hello-user', ['user' => $user]);

});

* View: hello-user.php:

<?php

echo $user;

?>

Nếu view của bạn sử dụng blade template thì sẽ có tên: hello-user.blade.php Và có nội dung như sau

{{ $user }}

=>Mình sẽ nói chi tiết về blade template ở bài sau.

#### -Gọi view trong controller:

-Cú pháp:

public function index()

{

return view('view', 'dataneuco');

}

-Cũng Tương tự như gọi view trong route các bạn thực hành tương tự lại VD trên nhé!

## 3,Tạo View dùng chung.

-Đôi khi, bạn cần chia sẻ một thông tin cho tất cả các view sẽ hiển thị trên ứng dụng, ví dụ: hiển thị thông tin thành viên đã đăng nhập lên thanh trạng thái của các trang trong ứng dụng chẳng hạn.Chẳng lẽ cứ phải  lặp đi lặp lại việc insert data trong hàm view().

-Thay vào đó,Laravel có hỗ trợ cho chúng ta chia sẻ view cho toàn ứng dụng với  hàm share().Để sử dụng các bạn chỉ cần thêm nó vào trong hàm boot trong file app/Providers/AppServiceProvider.php.

VD:

<?php

namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider

{

/\*\*

\* Bootstrap any application services.

\*

\* @return void

\*/

public function boot()

{

View::share('key', 'value');

}

/\*\*

\* Register the service provider.

\*

\* @return void

\*/

public function register()

{

//

}

}

## 4,Các các truyền dữ liệu cho view.

### Dùng compact().

Cú Pháp:

compact('tencuabientruyenvao')

VD:

Route::get('chao/{user}', function ($user) {

return view('hello-user', compact('user'));

});

// view muon hien thi thi goi bien $user

### Dùng with().

Cú Pháp:

view('tenview')->with('key', 'value');

VD:

Route::get('chao/{user}', function ($user) {

return view('hello-user')->with('user', $user);

});

//de hien thi trong view goi bien $user

### Dùng Mảng.

Cú pháp:

view('tenview', ['key' => 'value']);

VD:

Route::get('chao/{user}', function ($user) {

return view('hello-user', ['user' => $user]);

});

## 5,Lời Kết.

-Qua bài này mình đã giới thiệu cho các bạn về view trong laravel rôi, còn một số phần như view::composer nếu các bạn cần thì mình sẽ viết thêm nhé!Bài sau mình sẽ tìm hiểu về blade template trong laravel.

# **Bài 7: Blade template engine trong Laravel**

-Hiện nay cũng có khá nhiều các PHP framework nổi tiếng như Codeigniter, zend, CAKE,... Nhưng chỉ một số framework có tích hợp template engine , vì đơn thuần khi sử dụng template engine thì nó sẽ làm giảm hiệu năng của trang web. Nhưng với Laravel thì lại khác, Laravel có tích hợp sẵn cho chúng ta blade template với tốc độ xử lý rất nhanh.

## 1,Tạo blade template trong Laravel.

-Để tạo một blade template trong Laravel thì trước hết các bạn phải hiểu **blade template**bản chất nó là một view trong Laravel nên các bạn phải đặt nó trong thư mục resources/views ([view trong laravel](https://toidicode.com/view-trong-laravel-6.html)). Và tên file của blade template phải kết thúc bằng **.blade.php**

VD: mình sẽ tạo 1 blade template toidicode.blade.php trong thư mục resources/views

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Blade template</title>

<link rel="stylesheet" href="">

</head>

<body>

<h1>Chào mừng các bạn đã đến toidicode.com</h1>

</body>

</html>

## 2,Sử dụng blade template.

### Hiển thị dữ liệu trong blade template.

-Trong [PHP](https://toidicode.com/series/php-co-ban) khi muốn in dữ liệu ra màn hình các bạn có thể dụng echo,printf,... Nhưng ở đây khi các bạn sử dụng **blade template** thì sẽ có 2 cách để in dữ liệu ra màn hình khác.

#### -Sử dụng {{}}.

{{ 'bien' }}

-Thẻ này có tác dụng in ra dữ liệu dưới dạng thô . VD:

{{ '<b>toidicode.com</b>' }}

KQ:

|  |
| --- |
| <b>toidicode.com</b> |

#### -Sử dụng

{!! 'bien' !!}

-Thẻ này cho in ra dự liệu có appen HTML. VD:

{!! '<b>toidicode.com</b>' !!}

KQ:

|  |
| --- |
| **toidicode.com** |

### Dữ liệu mặc định trong blade template.

-Để sử dụng dữ liệu mặc định trong blade template  chúng ta có thể dùng **OR**.

VD: in ra bến **$a** nếu không có thì là **5**.

{{ $a or 5 }}

### Vòng lặp trong blade template.

#### -For:

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

In ra giá trị {{ $i }} <br/>

@endfor

#### -Foreach:

@foreach($users as $user)

<p>This is user {{ $user->id }}</p>

@endforeach

#### -Forelse:

+ Lặp dữ liệu nếu trống thì...

@forelse($users as $user)

<li>{{ $user->name }}</li>

@empty

<p>No users</p>

@endforelse

#### -while:

@while (true)

<p>I'm looping forever.</p>

@endwhile

### Câu lệnh rẽ nhánh trong blade template engine.

#### -if-else:

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

I have one record!

@elseif (count($records) > 1)

I have multiple records!

@else

I don't have any records!

@endif

## 3 ,Lời kết.

-Qua phần này mình đã giới thiệu cơ bản  về **blade template engine** trong laravel, nhưng trong **blade template engine** nó còn hỗ trợ chúng ta một phần cực kỳ hay nữa mình sẽ giới thiệu đến mọi người ở bài sau nhé!

# **Bài 8: Blade template engine trong Laravel(phần 2)**

Ở [phần trước](https://toidicode.com/blade-template-engine-trong-laravel-7.html) mọi người đã được tìm hiểu về biến, câu lệnh rẽ nhánh và vòng lặp trong blade template rồi . Tiếp đến, ở phần này mình sẽ tiếp tục giới thiệu đến mọi người một chức năng rất là hay ở trong blade template nữa, đó là template inheritance (kế thừa giao diện).

## 1, Template inheritance (kế thừa giao diện).

-Mình sẽ làm ví dụ trước để các bạn dễ hiểu nhé.

**Ví Dụ 1:** .-Đầu tiên mình sẽ tạo ra 1 blade template có tên **master.blade.php** ở đường dẫn resoures\views\master.blade.php.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title> @yield('title')</title>

<link rel="stylesheet" href="">

</head>

<body>

<div class="container">

@section('content')

@show

</div>

</body>

</html>

-Và sau đó mình tạo tiếp một file home.blade.php ở đường dẫn resoures\views\home.blade.php

@extends('master')

@section('title','đây là trang chủ')

@section('content')

<p> Đây là phần content</p>

@endsection

-Giờ tiếp đến chúng ta tạo ra một Route để gọi view.

Route::get('call-view', function () {

return view('home');

});

-Tiếp đó các bạn khởi động server lên và chạy đường dẫn :  yourdomain/call-view lên và thử nhận xét.

**=>Chú thích:**

-Như ở trên mình có sử dụng các từ khóa lạ như @yield(),@section(),@extends().

-**@yield(), @section-@show**: có tác dụng như một biến nó được tạo ra nhằm báo cho blade template  biết vị trí ý sẽ dùng để chèn nội dung cho biến đó.

-**@extends()**: Có tác dụng khai báo cho blade template biết là file hiện tại đang kế thừa từ file nào

. **=>Thực tiễn:** Như ở trong ví dụ trên: Đầu tiên thì mình khởi tạo 2 biến @yeild('title'),@section('content') trong file **master.blade.php**, tiếp sau đó mình tạo tiếp file **home.blade.php** và dùng @extends('master') để kế thừa tất cả thuộc tính từ file **master.blade.php** tiếp đó mình lại khai báo nội dung cho 2 biến ở file **master.blade.php** bằng section('tenbien','noidungngan'),@section('tenbien') noidungdai @endsection

Ví dụ 2: - Mình sẽ thay đổi code trong file master ở vd1 một chút.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title> @yield('title')</title>

<link rel="stylesheet" href="">

</head>

<body>

<div class="container">

@section('content')

<p>Dòng này là của master.blade.php</p>

@show

</div>

</body>

</html>

Và file home.php

@extends('master')

@section('title', 'đây là trang chủ')

@section('content')

<p> Đây là phần content</p>

@endsection

Khi chạy lên như ở ví dụ 1 thì các bạn sẽ thấy source code của nó sẽ có dạng.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title> @yield('title')</title>

<link rel="stylesheet" href="">

</head>

<body>

<div class="container">

<p>Dòng này là của master.blade.php</p>

<p>Đây là phần content</p>

</div>

</body>

</html>

## 2, Lời kết.

-Qua ví dụ trên các bạn đã phát hiện ra chưa ạ?

+Ở phần trên mình có sử dụng thêm mới mục đích là muốn khai báo cho blade template biết là ở vẫn dữ nguyên lại phần @section('content') trong file master và thêm vào sau nó nội dung @section('content') ở trong file home.

->Chú ý: @Parent có thể đặt trước nội dung muốn thêm vào (nếu muốn phần được thêm sẽ ở sau phần khởi tạo) và ngược lại.

# **Bài 9: Controller trong Laravel**

-Trong mô hình MVC thì project sẽ được chia ra làm 3 phần(Model,[View](https://toidicode.com/view-trong-laravel-6.html),Controller) và trong Laravel framework cũng chia làm 3 phần như vậy. Ở phần trước mình đã giới thiêu với các bạn về view trong Laravel rồi. Và ở phần này mình sẽ giới thiệu với các bạn về **Controller trong Laravel.**

## 1,Tạo controller trong Laravel.

-Cũng giống như view, để tạo controller trong Laravel cũng phải có một số ràn buộc như sau:

* Controller phải được đặt trong đường dẫn App\Http\controllers
* Tên của controller phải giống với tên class trong file controller đó.
* Class controller (do bạn tạo) phải extends(kế thừa) từ Controller (chú ý chữ C phải viết hoa).

**=>**Cách tạo file: -Ở trong Laravel để tạo controller thì chúng ta có hai cách:

**Cách 1:**Tạo bằng tay.

-Chúng ta có thể tạo một file controller bằng tay hoặc bằng cmd thuần trong đường dẫn App\Http\Controllers. Và để sử dụng được controller đó thì bắt buộc các bạn phải tuân thủ 3 điều kiện phía trên nhé.

-Ví dụ: mình tạo một Homecontroller.php trong App\Http\Controllers thì trong file sẽ có nội dung như sau:

<?php

namespace App\Http\Controllers;

class Homecontroller extends Controller

{

//your code

}

**Cách 2:**Tạo bằng lệnh laravel cung cấp sẵn. + Để tạo một controller bằng lệnh trong Laravel thì các bạn chỉ việc mở commanline lên và trỏ vào thư mục chứa project Laravel của bạn (xem lại [bài 1](https://toidicode.com/laravel-la-gi-cach-cai-dat-laravel-1.html)) và gõ lệnh theo cú pháp.

php artisan make:controller Controllername

Hoặc

php artisan make:controller ControllerName --resource

để tạo resource controllers

* Chú thích: controllername là tên controller các bạn muốn tạo.

Khi đó Laravel sẽ tự sinh cho chúng ta 1 file C**ontrollerName**và có sẵn code như sau:

<?php

namespace App\Http\Controllers;

use App\User;

use App\Http\Controllers\Controller;

class ControllerName extends Controller

{

//

}

-Và với resource controllers:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class ControllerName extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

//

}

/\*\*

\* Show the form for creating a new resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function create()

{

//

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

//

}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id)

{

//

}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit($id)

{

//

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{

//

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

=>Các bạn thấy cách này khá đơn giản đúng không.

## 2,Sử dụng dụng các action trong Controllers.

-Để tạo một action trong controllers thì chúng ta chỉ việc tạo một **hàm** ở trong controllers và gọi nó ở trong Route là được.

-Ví dụ: Mình sẽ tạo một controllers home controller và một hàm là index.

<?php

namespace App\Http\Controllers;

class HomeController extends Controller

{

public function index(){

echo "Đây là index trong homecontroller";

}

}

Và chúng ta sẽ tạo một route để gọi đến action đó.

Route::get('call-controller', 'Homecontroller@index');

-Như thế là chúng ta đã gọi và sử dụng được action trong Controllers rồi.

## 3,Truyền tham số trong action Controllers.

-Để truyền tham số trong action Controllers thì nó hoàn toàn giống PHP thuần, nên mọi người đọc ví dụ sau của mình rồi tự suy ra kết luận nhé.

-Ví Dụ: Vẫn với homecontroller ở trên, nhưng mình sẽ cho truyền tham số vào controllers như sau:

<?php

namespace App\Http\Controllers;

class HomeController extends Controller

{

public function index($name, $age){

echo "Xin chào $name, $age tuổi";

}

}

Và mình sẽ khởi tạo một route có truyền tham số cho nó như sau:

Route::get('a/{name}/{age}', 'HomeController@index')->where(['name' => '[a-zA-Z]+', 'age' => '[0-9]+']);

(ai không hiểu xem lại [bài 5](https://toidicode.com/route-trong-laravel-phan-3-5.html) nhé) -Như thế là bạn đã có thể truyền tham số cho action trong controllers rồi.

## 4,Lời Kết.

-Qua phần chia sẻ cơ bản về controllers của mình ở trên thì mình rất hy vọng là mọi người có và nắm được cái nhìn tổng quan cũng như là cách sử dụng Controllers cơ bản trong Larvel. Phần sau mình sẽ giới thiệu với mọi người về cách cấu hình database và xử lý truy vấn với Query Buider trong Laravel.

# **Bài 10: Query Builder trong Laravel**

  -Trước đây khi làm việc với PHP thuần, để tương tác với database các bạn sẽ cần phải kết nối với database và viết những câu truy vấn hết sức là phức tạp để gọi được dữ liệu ra. Nhưng khi sử dụng các famework đặc biệt là Laravel thì các bạn sẽ không phải lo về điều đó nữa, bởi Laravel có cung cấp cho chúng ta một chức năng rất hay là **Query Builder.**

## ****1, Cấu hình database.****

-Trước tiên để sử dụng được Query builder các bạn cần phải cấu hình database kết nối đến cơ sở dữ liệu.

-Để cấu hình Database các bạn cần mở file **.env**lên và cấu hình như sau.

APP\_ENV=local

APP\_KEY=base64:w6dZccXCvdehAax2NYUt6f/igtsyw7QTQCx2eGEQBgI=

APP\_DEBUG=true

APP\_LOG\_LEVEL=debug

APP\_URL=http://localhost

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_USERNAME=root

DB\_PASSWORD=

BROADCAST\_DRIVER=log

CACHE\_DRIVER=file

SESSION\_DRIVER=file

QUEUE\_DRIVER=sync

REDIS\_HOST=127.0.0.1

REDIS\_PASSWORD=null

REDIS\_PORT=6379

MAIL\_DRIVER=smtp

MAIL\_HOST=mailtrap.io

MAIL\_PORT=2525

MAIL\_USERNAME=null

MAIL\_PASSWORD=null

MAIL\_ENCRYPTION=null

PUSHER\_APP\_ID=

PUSHER\_KEY=

PUSHER\_SECRET=

-Chú thích:

* DB\_HOST : Là server chứa mysql của bạn.
* DB\_PORT: Là cổng post của server Mysql của bạn (thường để mặc định là 3306).
* DB\_DATABASE: Là tên database của bạn.
* DB\_USERNAME: Là username để truy cập vào database của bạn.
* DB\_PASSWORD: Là password để truy cập vào database của bạn.

## 2, Sử dụng Query Builder.

-Sau khi đã cấu hình được database (ở phần trên). Thì giờ mọi người bắt đầu thực hành truy vấn Sql với Query Builder nhé.

### Lấy tất cả dữ liệu trong bảng.

-Cú Pháp:

DB::table('tablename')->get()

**VD:** Lấy tất cả dữ liệu trong bảng tbl\_product.

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

$data = DB::table('tb\_product')->get();

print\_r($data);

});

**Kết quả:**

Illuminate\Support\Collection Object

(

[items:protected] => Array

(

[0] => stdClass Object

(

[id] => 5

[title] => Váy đầm xòe Asose

)

[1] => stdClass Object

(

[id] => 6

[title] => Váy đầm xòe Asose

)

)

)

### Select cột trong bảng.

-Cú pháp:

DB::table('tablename')->select('columnfirst', 'columnsecond')->get();

### Select có điều kiện.

-Cú pháp:

+Điều kiện bằng:

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

+Điều kiện lớn hơn:

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

+Điều kiện nhỏ hơn:

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

+Điều kiện khác:

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

+Điều kiện lồng:

VD:

$users = DB::table('users')->where('votes', '>', 100)->orWhere('name', 'John')->get();

Truy vấn tìm kiếm.

DB::table('tablename')->where('column', 'like', 'filter')->get();

// vd : DB::table('product')->where('name', 'like', '%Sách%')->get();

### Join bảng.

$users = DB::table('users')->join('contacts', 'users.id', '=', 'contacts.user\_id')->get();

+Left join:

$users = DB::table('users')->leftjoin('contacts', 'users.id', '=', 'contacts.user\_id')->get();

### Unions.

$first = DB::table('users')->whereNull('first\_name');

$users = DB::table('users')

->whereNull('last\_name')

->union($first)

->get();

### OrderBy.

$users = DB::table('users')

->orderBy('name', 'desc')

->get();

### Random.

$randomUser = DB::table('users')

->inRandomOrder()

->first();

### GroupBy/having.

$users = DB::table('users')

->groupBy('account\_id')

->having('account\_id', '>', 100)

->get();

### Thêm (insert).

DB::table('users')->insert(

['email' => 'john@example.com', 'votes' => 0]

);

### Sửa (update).

DB::table('users')

->where('id', 1)

->update(['votes' => 1]);

### Xóa (delete).

DB::table('users')->delete();

//hoặc

DB::table('users')->where('votes', '>', 100)->delete();

## 3, Chú ý.

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

* Kết nối với cơ sở dữ liệu.
* Nếu truy vấn trong controllers thì các bạn cần phải khai báo use Illuminate\Support\Facades\DB; còn trong Route thì không cần.

## 4, Lời kết.

-Bài hôm nay khá là dài đúng không các bạn? Mình cũng thấy thế, nhưng ở phía trên mới là một số câu lệnh truy vấn cơ bản trong **Query Builder**còn một số câu truy vấn nâng cao nữa các bạn muốn tham khảo có thể xem tại [đây](https://laravel.com/docs/5.3/queries) nhé.

# **Bài 11: Model trong Laravel**

-Trong [bài trước](https://toidicode.com/query-builder-trong-laravel-11.html) mình đã hướng dẫn mọi người truy vấn CSDL với Query Builder trong Laravel. Bài này mình tiếp tục giới thiệu đến mọi người một phần khá là quan trọng trong mô hình MVC đó chính là **Model.**

## 1, Tạo Model trong Laravel.

-Cũng giống như trong Controllers, model của chúng ta cũng có hai các tạo là thủ công và bằng lệnh mà Laravel hỗ trợ.

### a) cách 1: Tạo thủ công.

-Để tạo một model có hiệu lực thì bắt buộc các bạn phải:

* Đặt nó ở trong thư mục **App/**
* Tên class bên trong file phải cùng với tên file. VD: file news.php thì phải có class là **news.**
* Class vừa tạo phải kế thừa **Model**gốc của Laravel. VD: **class news extends Model{...}**
* Để kế thừa thành công Model trong Laravel thì bắt buộc bạn phải khai báo sử dụng namespace Của Model gốc của Laravel và đồng thời khai báo namespace mới cho model vừa tạo.

VD: Tôi tạo một model News.php trong App/ của Laravel.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class News extends Model

{

}

### b) Cách 2: Tạo bằng lệnh Laravel cung cấp sẵn.

-Mình khuyến khích các bạn dùng cách này nhé.

-Tạo model bằng cách này thì khá là đơn giản. Các bạn chỉ cần mở cmd(commandline) lên và trỏ đến thư mục chứa project Laravel của các bạn và gõ lệnh.

php artisan make:model News

Hoặc

php arisan make:model News --migration

-Trong đó:

* News là tên model, các bạn có thể chỉnh thành tên model các bạn muốn tạo.
* Ở dòng thứ hai thì chúng ta cũng tạo được model nhưng đồng thời chúng ta sẽ được tạo luôn một file migration( nói ở bài sau).

-Sau đó enter và vào trong thư mục App/ bạn sẽ thấy có một file News.php trong đó và bên trong sẵn có đoạn code.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class News extends Model

{

//

}

-Như thế chúng ta thấy cách này khá là nhanh và không cần phải code mấy dòng code thủ công như cách 1 nữa đúng không?

## 2,Khai báo các thông số tùy chỉnh.

### Khai báo table cần sử dụng trong Model.

-Ở trong Laravel mỗi một model ứng với một bảng(table) dữ liệu trong CSDL và để khai báo model sử dụng bảng dữ liệu nào trong database thì mọi người khai báo dòng sau trong class model.

protected $table = 'tableName';

-Trong đó : tableName là tên table các bạn muốn sử dụng. Ví Dụ:

protected $table = 'tbl\_user';

### Lọc cột dữ liệu trong model.

-Trong laravel chẳng hạn như bảng user tôi chỉ muốn truy vấn cột username,useremail mà không muốn sử dụng cột passwork thì sau. Chính vì điều đó Laravel cũng đã cung cấp cho chúng ta một thông số là **fillable** để điều chỉnh các cột cần sử dụng.

protected $fillable = ['column1', 'column2', .., 'columnn'];

### Khai báo timestamps.

-Laravel cũng cung cấp cho chúng ta tùy biến có sử dụng time stamps hay không. Nếu để **true** là có và ngược lại **false** là không.

public $timestamps = true;

// hoặc

public $timestamps = false;

## 3, Lời kết.

-Qua phần giới thiệu cơ bản về Model trong Laravel của mình, mình hi vọng mọi người sẽ lắm rõ được model trong Laravel hoặc động như thế nào và cách cấu hình nó như thế nào. Phần sau mình sẽ hướng dẫn mọi người truy vấn dữ liệu với ORM trong Laravel.

# **Bài 12: Eloquent ORM trong Laravel**

-[Bài trước](https://toidicode.com/model-trong-laravel-12.html) mình đã giới thiệu với mọi người cách tạo Model cũng như cách cấu hình Model, nhưng chưa được ứng dụng vào thực tế ,Thì bài này mình sẽ hướng dẫn mọi người truy vấn cơ sở dữ liệu trong Model với **Eloquent ORM** của Laravel.

## ****1,Cách gọi Model.****

-Để gọi Model trong Controllers thì bắt buộc chúng ta phải gọi namespace của model đó trong Controllers.

* VD: mình muốn gọi Model **news** trong **homecontroller**.

<?php

namespace App\Http\Controllers;

use App\News;

class homecontroller extends Controller

{

News::all();

}

**Hoặc**

#### <?php

#### namespace App\Http\Controllers;

#### use App;

#### class homecontroller extends Controller

#### {

#### App\News::all();

#### }

-Còn nếu muốn gọi model trong Route thì các bạn không cần gọi namespace mà các bạn có thể dùng luôn với cú pháp.

App\TenModel::someThing;

VD: mình muốn gọi Model news trong route.

App\News::all();

## 2, Các câu truy vấn hay dùng trong Eloquent ORM.

-**Chú ý**: Tất cả các cú pháp dưới đây mình lấy **News**model Làm mẫu nhé

### Lấy ra dữ liệu trong bảng.

**-Lấy ra tất cả dữ liệu trong bảng.**

News::all();

**-Lấy ra một dòng dữ liệu thông qua khóa chính**.

News::find(1);

**Hoặc**

News::take(1)->get();

**-Truy vấn điều kiện.**

**+Bằng:**

News::where('id', 5)->get();

**+Lớn hơn, nhỏ hơn,..** (giống [Query Buider](https://toidicode.com/query-builder-trong-laravel-11.html#load-h3-2))

**-Chọn cột dữ liệu.**

News::select('id', 'title')->get();

**-Đếm dữ liệu.**

News::all()->count();

### Thêm dữ liệu.

#### $news = new News();

#### $news->title = "tin tức 1";

#### $news->categoryId = 1;

#### $news->save();

-Ở trên mình đã thêm vào cột title nội dung là 'tin tức 1' và cột categoryId là 1.

### Sửa dữ liệu.

-Ví Dụ mình muốn sửa tiêu đề của bảng news có id =1.

$news = News::find(1);

$news->title = 'toidicode.com';

$news->save();

**Hoặc**

News::where('id', 1)->update(['title' => 'toidicode.com']);

### Xóa

**-Cách 1:**

$news= News::find(1);

$news->delete();

**-Cách 2:**

News::destroy(1);

//or

News::destroy(1, 2);

//or

News::destroy([1, 2, 3]);

//or

News::destroy(array(1, 2, 3));

-Trong đó: 1, 2, 3 là các id(primary) của bảng cần truy vấn.

**-Cách 3:**

News::where('id', 1)->delete();

## 3, Lời kết.

-Phần trên mình đã trình bày cho các bạn về Eloquent ORM trong Laravel rồi, đây là một phần cũng rất là quan trọng trong project nên mình mong các bạn cố gắng đọc cho kĩ để hiểu hơn về nó.

# **Bài 13: Các mối quan hệ (Relationships) trong Eloquent**

-Trong Bài trước mình đã giới thiệu cho mọi người về Eloquent ORM trong laravel rồi, nhưng vấn đề là trong khi lập trình các ứng dụng thực tế thì sẽ tồn tại ra các mối quan hệ giữa các bảng với nhau chính vì điều đó Laravel đã cung cấp cho chúng ta các mối quan hệ có sẵn để giảm thiểu thời gian cũng như công sức của các Lập trình Viên.

## 1, Các mối quan hệ trong Eloquent.

-Trước đây Laravel version 4.0 thì chỉ cung cấp cho chúng ta 3 mối quan hệ chính, nhưng đến phiên bản hiện tại con số đó đã được nâng lên là **6**mối quan hệ.

-Gồm:

* **One to One** : Một Một.
* **One to Many**: Một nhiều.
* **Many to Many**: Nhiều nhiều.
* **Has Many Through**: Nhiều nhiều qua lại trung gian.
* **Polymorphic Relations**: Đa hình.
* **Many To Many Polymorphic Relations** : Nhiều nhiều đa hình.

-Hơi phức tạp đúng không mọi người? Để mình đi vào ví dụ cụ thể cho mọi người hiểu hơn nhé.

## 2, One To One.

-Đây là mối quan hệ đơn giản n nhất trong tất cả các mối quan hệ mà Laravel cung cấp.

 VD: Mình có một bảng **Posts** để đăng bài và một bảng **Featured\_images**để đặt ảnh đại diện cho bài post đó . Mình sẽ tạo ra mối quan hệ **1- 1** như sau: một bài viết của mình chỉ có thể có một ảnh đại diện và ngược lại một ảnh đại diện mình sẽ chỉ dùng cho một bài viết.

-Khi đó mình sẽ phải tạo 2 model là **Posts.**

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Posts extends Model

{

protected $table = 'Posts';

public $timestamp = false;

public function FeaturedImages()

{

return $this->hasOne('App\FeaturedImages');

}

}

và**FeaturedImages:**

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class FeaturedImages extends Model

{

protected $table = 'FeaturedImages';

public $timestamp = false;

}

- Khi này các bạn chỉ cần triệu gọi với lệnh **Eloquent** dạng như sau:

App\Posts::find(1)->FeaturedImages;

**-Chú ý:**Câu lệnh ở trên chỉ đúng nếu như khóa ngoại(forigen key) của bạn là FeaturedImages\_id. Nhưng không sao Laravel vẫn cung cấp cho chúng ta tùy biến cột khóa ngoại với cú pháp như sau:

public function FeaturedImages()

{

return $this->hasOne('App\FeaturedImages', 'foreign\_key');

}

-Nhưng nếu như ở bảng FeaturedImages của bạn cũng không phải liên kết với Post qua cột id thì bạn có thể khai báo thêm cho Larvel bằng cách sau:

public function FeaturedImages()

{

return $this->hasOne('App\FeaturedImages', 'foreign\_key', 'local\_key');

}

### -Định nghĩa các mối quan hệ đảo ngược.

-Như ở phía trên mình có nói là một FearturedImages cũng chỉ thuộc về một bài post thôi, nên chúng ta hoàn toàn có thể khai báo định nghĩa đảo ngược lại với model FearturedImages như sau:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class FeaturedImages extends Model

{

protected $table = 'FeaturedImages';

public $timestamp = false;

public function posts()

{

return $this->belongsto('App\Posts');

}

}

Cách gọi và truyền các khóa ngoại cho nó cũng tương tự như với model Posts.

public function posts()

{

return $this->belongsto('App\Posts', 'foreign\_key');

}

public function posts()

{

return $this->belongsto('App\Posts', 'forigen\_key', 'local\_key');

}

App\FearturedImages::find(1)->posts;

## ****3, One to Many.****

-Mối quan hệ này cũng tương tự như One to One nhưng nó chỉ khác nhau về câu lệnh thôi à.

-VD: Cũng với bảng Posts như ở VD trên nhưng lần này tôi sẽ tạo thêm một bảng Author , Nhưng ở đây nó sẽ là mối quan hệ một nhiều vì một Author thì có thể có nhiều bài posts và một bài post chỉ có một Author thôi.

-Mình sẽ tạo thêm model Author và phương thức **hasMany()** để tạo mối quan hệ cho nó.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Author extends Model

{

protected $table = 'Author';

public $timestamp = false;

public function posts()

{

return $this->hasMany('App\Posts');

}

}

Và Larvel cũng cho chúng ta truyền khóa ngoại, cột liên kết như mối Quan hệ **One to One.**

public function posts()

{

return $this->hasMany('App\Posts', 'author\_id');

}

public function posts()

{

return $this->hasMany('App\Posts', 'author\_id', 'local\_key');

}

Để gọi phương thức cũng không khác gì.

App\author::find(1)->posts;

### -Định nghĩa các mối quan hệ đảo ngược.

+Về phần này thì hoàn toàn giống phần **One to One** nên mình không viết lại nữa.

## 4, Many to Many.

-Mối quan hệ này nó hơi phức tạp hơn hai mối quan hệ kia một chút, nhưng không có gì là khó cả.

VD: Vẫn Là bảng posts như trên nhưng ta sẽ tạo thêm 2 bảng là posts\_category và category . Mối quan hệ được diễn tả như sau: Một bài posts thì sẽ thuộc một hoặc nhiều category và ngược lại một category cũng có một hoặc nhiều bài posts Và chúng được kết nối với nhau qua bảng posts\_category.

-Chúng ta sẽ thể hiện quan hệ giữa 2 bảng này bằng phương thức **belongstoMany()**như sau:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Posts extends Model

{

protected $table = 'Posts';

public $timestamp = false;

public function category()

{

return $this->belongstoMany('App\category');

}

}

-Gọi phương thức cũng tương tự như 2 quan hệ trên.

App\Posts::find()->category;

**->Chú ý:**Ở đây bảng trung gian của các bạn không phải là bảng posts\_category (**quy ước mặc định của Laravel**) mà là một bảng khác thì các bạn phải tùy biến như sau:

public function category()

{

return $this->belongstoMany('App\category', 'table\_medium');

}

// table\_medium la ten bang trung gian cua cac ban

**Và** nếu hai column liên kết của bạn khác với quy ước của Laravel thì bạn có thể tùy biến như sau:

public function category()

{

return $this->belongstoMany('App\category', 'table\_medium', 'posts\_id', 'category\_id');

}

### -Định nghĩa các mối quan hệ đảo ngược.

+Về phần này của quan hệ Many to Many thì nó lại giống hệt với phần chính của nó.

VD: chúng ta sẽ gọi phương thức posts trong category như sau.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class category extends Model

{

protected $table = 'category';

public $timestamp = false;

public function posts ()

{

return $this->belongstoMany('App\Posts');

}

}

### -Truy cập vào Table trung gian qua quan hệ Many to Many.

Như ở trên mình có nói là để sử dụng mối quan hệ Many to Many thì chúng ta cần phải giao tiếp qua một bảng trung gian. Và để truy cập vào bảng trung gian đó thì trong Laravel có cung cấp cho chúng ta một phương thức **pivot attribute.** VD: Tôi muốn xem mối quan hệ được xây dựng trên bảng trung gian posts\_category vào khi nào thì sẽ phải truy vấn như sau:

$post = App\Posts::find(1);

foreach ($post->category as $rows) {

echo $rows->pivot->created\_at;

}

## ****4, Has Many Through.****

- Ở mối quan hệ này chúng ta có thể cập vào các mối quan hệ xa thông qua một mối quan hệ trung gian.

Ví dụ, một Country model có thể có nhiều Post model thông qua một User model trung gian. Trong ví dụ này, bạn có thể dễ dàng lấy tất cả các blog post cho 1 country.

Hãy nhìn vào các bảng cần thiết để xác định mối quan hệ này:

countries

id - integer

name - string

users

id - integer

country\_id - integer

name - string

posts

id - integer

user\_id - integer

title - string

Mặc dù post không chứa cột country\_id, mối quan hệ hasManyThrough cung cấp quyền truy cập vào post của country thông qua $country->posts. Để thực hiện các truy vấn này, Eloquent kiểm tra các country\_id trên bảng user trung gian. Sau khi tìm ra id của user phù hợp, chúng được sử dụng để truy vấn bảng posts. Bây giờ chúng ta đã xem xét các cấu trúc bảng cho các mối quan hệ, hãy định nghĩa nó trên Country model.

Default

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Country extends Model

{

/\*\*

\* Get all of the posts for the country.

\*/

public function posts()

{

return $this->hasManyThrough('App\Post', 'App\User');

}

}

Đối số đầu tiên truyền cho phương thức hasManyThrough là tên của model cuối cùng chúng ta muốn truy cập, trong khi đối số thứ 2 là tên của model trung gian. Nếu bạn muốn tùy chỉnh các foreign key của relationship, bạn có thể truyền vào các đối số thứ 3 và thứ 4 của phương thức hasManyThrough. Đối số thứ 3 là foreign key của model trung gian, đối số thứ 4 là foreign key của model cuối cùng và đối số thứ 5 là local key.

class Country extends Model

{

public function posts()

{

return $this->hasManyThrough(

'App\Post', 'App\User',

'country\_id', 'user\_id', 'id'

);

}

}

## 5, Các mối quan hệ khác.

-Ở hai mối quan hệ còn lại rất ít khi sử dụng lên mình sẽ không nói đến ở bài này, các bạn nào quan tâm có thể xem tại [đây](https://laravel.com/docs/5.3/eloquent-relationships#many-to-many-polymorphic-relations).

## 6, Lời kết.

-Qua bài này mọi người có thấy Laravel hỗ trợ chúng ta ràng buộc các mối quan hệ khá là đầy đủ và chi tiết đúng không? Bài này cũng là một bài khá quan trọng nên mọi người chú ý đọc cho kỹ nhé!

# **Bài 14: Collections trong Laravel**

-Bài trước mình đã giới thiệu với mọi người về [Relationship trong Laravel](https://toidicode.com/cac-moi-quan-he-relationships-trong-eloquent-14.html)(Ràng buộc quan hệ) Và mình đã nhận xét là nó khá là tiết kiệm thời gian cho lập trình viên, và tiếp đến ở phần này mình lại tiếp tục giới thiệu đến mọi người một bột thư viện(Collections) trong Laravel hỗ trợ chúng ta Lọc,xử lý dữ liệu rất là nhanh.

## 1,Collection Trong Laravel là gì?

-Collection trong Laravel là một class có tích hợp sẵn các phương thức xử lý dữ liệu thường xuyên nhằm làm giảm thiểu tối đa thời gian cho các lập trình viên. Đặc biệt là làm API kết nối tới database vì dữ liệu trả về từ databasecó sẵn kiểu là Collection.

## 2, Các Khai báo, sử dụng Collections.

-Đầu tiên để khai báo cũng như sử dụng được Collections các bạn cần phải gọi namespace của nó.

use Illuminate\Support\Collection;

-Sau khi gọi namespace thành công. các bạn có thể khởi tạo collections bằng 2 cách sau đây:

// cách này dùng helper function nên không cần khai báo namespace

$collection = collect([]);

// hoac

$collection = Collection::make([]);

-Trong đó: [] là mảng giá trị các bạn truyền vào.

-Lúc này các bạn **prinf\_r($collection)**thì PHP sẽ trả về cho chúng ta một object có chứa giá trị tương ứng với mảng các bạn truyền vào.

## 3,Các Collection hay dùng.

### all

-Hàm này có tác dụng lấy ra tất cả các giá trị trong collections và trả về dưới dạng mảng.

$data = Collection::make([1, 2, 3, 4, 5, 6])->all();

//output [1,2,3,4,5,6];

### filter

-Hàm này có tác dụng lọc dữ liệu và trả về các dữ liệu đã được lọc.

$collection = collect(['toidicode.com', 'Vu Thanh Tai', 'PHP'])

->filter(function ($key, $value) {

return $value == 't';

});

Output

Array

(

[0] => toidicode.com

)

### expect

-Hàm này loại bỏ đi các giá trị không cần thiết.

$collection = collect(['product\_id' => 1, 'price' => 100, 'discount' => false]);

$filtered = $collection->except(['product\_id', 'discount']);

$filtered->all();

// output ['price'=>100]

### Count

-Hàm có tác dụng đếm các phần tử trong collections.

$collection = collect([1, 2, 3, 4]);

$collection->count();

//output 4

### avg

-Hàm tính giá trị trung bình của các phần tử trong mảng.

$data = Collection::make([1, 2, 3, 4, 5, 6])->avg();

//output 3.5

### sum.

-hàm tính tổng giá trị của các phần tử.

$collection = collect([1, 2, 3, 4, 5, 6, 7]);

$chunks = $collection->sum();

//output 28

### chumk.

-Hàm các tác dụng tách mảng ra thành các mảng con.

$collection = collect([1, 2, 3, 4, 5, 6, 7]);

$chunks = $collection->chunk(4);

$chunks->toArray();

//output [[1, 2, 3, 4], [5, 6, 7]]

### first.

-Hàm có tác dụng trả về giá trị đầu tiên.

collect([1, 2, 3, 4])->first(function ($value, $key) {

return $value > 2;

});

// 3

### get

-Hàm lấy giá trị của phần tử trong mảng.

$collection = collect(['website' => 'toidicode.com', 'author' => 'Vu Thanh Tai']);

$value = $collection->get('author');

// Vu Thanh Tai

### SortBy.

-Hàm sắp xếp lại theo thứ tự tăng dần.

$collection = collect([

['name' => 'Desk', 'price' => 200],

['name' => 'Chair', 'price' => 100],

['name' => 'Bookcase', 'price' => 150],

]);

$sorted = $collection->sortBy('price');

$sorted->values()->all();

/\*

[

['name' => 'Chair', 'price' => 100],

['name' => 'Bookcase', 'price' => 150],

['name' => 'Desk', 'price' => 200],

]

\*/

### SortByDesc.

-Sắp xếp lại giá trị theo thứ tự giảm dần.

$collection = collect([

['name' => 'Desk', 'price' => 200],

['name' => 'Chair', 'price' => 100],

['name' => 'Bookcase', 'price' => 150],

]);

$sorted = $collection->sortByDesc('price');

$sorted->values()->all();

/\*

[

['name' => 'Chair', 'price' => 100],

['name' => 'Bookcase', 'price' => 150],

['name' => 'Desk', 'price' => 200],

]

\*/

### Take.

-Hàm giới hạn số lượng trả về.

$collection = collect([0, 1, 2, 3, 4, 5]);

$chunk = $collection->take(3);

$chunk->all();

//output [0, 1, 2]

-Hoặc lấy từ cuối lên trên.

$collection = collect([0, 1, 2, 3, 4, 5]);

$chunk = $collection->take(-2);

$chunk->all();

//output [4, 5]

### Các hàm khác.

-Các hàm ít sử dụng khác các bạn có thể xem thêm tại [đây](https://laravel.com/docs/5.3/collections#available-methods).

## 3,Lời kết.

-Qua phần trên các bạn có thấy mình nói có đúng không? Collections giúp chúng ta tiết kiệm được rất nhiều thời gian làm việc . Ở phần này các bạn không cần phải nhớ nhiều chi cho mệt, các bạn chỉ cần nhớ những hàm mà các bạn cảm thấy là cần thiết là được rồi. Chúc các bạn học tốt!

# **Bài 15: Schema Buider trong Laravel**

phút đọc

Font size

Ở các bài trước mình đã giới thiệu với mọi người cách truy xuất dữ liệu trong Laravel Với [Query Buider](https://toidicode.com/query-builder-trong-laravel-11.html) và [Eloquent ORM](https://toidicode.com/eloquent-orm-trong-laravel-13.html) rồi. Hôm nay tiếp tục với series [Học Laravel](https://toidicode.com/series/hoc-laravel) của Toidicode.com mình xin được giới thiệu với mọi người một thuật ngữ mới nữa trong Laravel đó là **Schema Builder.**

## 1, ****Schema Builder**** trong Laravel là gì?

-Schema Builder Là một class trong bộ Facades của Laravel nó giúp chúng ta làm việc với tất cả các cơ sở dữ liệu mà Laravel hỗ trợ với các hàm được định nghĩa sẵn.

-Thường thì nó sẽ được kết hợp với Migrations để xây dự cấu trúc database.

-Điều kiện để sử dụng Schema Builder:

* Phải có kết nối với database .
* Muốn sử dụng được Schema Builder thì phải gọi namespace của nó **Illuminate\Support\Facades\Schema**

## 2, Các lệnh thường dùng trong Schema Builder.

### Tạo bảng.

Cú pháp:

Schema::table('tablename', function ($table) {

//code

});

Vd: Tạo bảng users.

Schema::table('users', function ($table) {

//code

});

### Đổi tên bảng.

Cú Pháp:

Schema::rename($from, $to);

VD: Đổi bảng users thành admin.

Schema::rename('users','admin');

### Xóa bảng.

Cú Pháp:

Schema::drop('tableName');

//hoac

Schema::dropIfExists('tableName');

VD: Xóa bảng users.

Schema::drop('users');

//hoặc

Schema::dropIfExists('users');

### Thêm cột.

Cú Pháp:

Schema::table('tableName', function ($table) {

$table->typeData('columnName');

});

VD: Thêm cột email với kiểu dữ liệu vachar.

Schema::table('users', function ($table) {

$table->string('email');

});

->Chú ý: Các kiểu dữ liệu mình sẽ nói rõ hơn ở [bài sau](https://toidicode.com/migrations-trong-laravel-17.html).

### Đổi tên cột.

Schema: table('tableName', function ($table) {

$table->renameColumn('from', 'to');

});

VD: Đổi cột email trong bảng users thành usermail.

Schema::table('users', function ($table) {

$table->renameColumn('email', 'usermail');

});

### Xóa cột.

Cú Pháp:

Schema::table('tableName', function ($table) {

$table->dropColumn('columnName');

//hoặc xóa nhiều cột cùng lúc.

$table->dropColumn(['columnName1','columnName2','columnName3']);

});

VD: xóa cột email trong bảng users.

Schema::table('users', function ($table) {

$table->dropColumn('email');

});

### Tạo index.

Cú Pháp:

$table->index('coumnName');

Các kiểu Index.

| Cú pháp | Mô tả |
| --- | --- |
| table->primary('id'); | Add a primary key. |
| table->primary(['first', 'last']); | Add composite keys. |
| $table->unique('email'); | Add a unique index. |
| $table->unique('state', 'my\_index\_name'); | Add a custom index name. |
| $table->unique(['first', 'last']); | Add a composite unique index. |
| $table->index('state'); | Add a basic index. |

-Mấy cái này dịch sang tiếng việt rất khó biểu diễn nên mình để tiếng anh cho dễ :D.

-VD: mình sẽ đánh unique index cho cột email.

$table->unique('email');

### Xóa Index.

Cú Pháp:

| Cú Pháp | Mô Tả |
| --- | --- |
| $table->dropPrimary('users\_id\_primary'); | Drop a primary key from the "users" table. |
| $table->dropUnique('users\_email\_unique'); | Drop a unique index from the "users" table. |
| $table->dropIndex('geo\_state\_index'); | Drop a basic index from the "geo" table. |

Thêm khóa ngoại.

Cú Pháp:

Schema::table('tableName', function ($table) {

$table->integer('conmName')->unsigned();

$table->foreign('Conmumname')->references('columnName')->on('tableName2');

});

Vd: Mình sẽ thêm cột users\_id của bảng posts làm khóa ngoại trên bảng users.

Schema::table('posts', function ($table) {

$table->integer('user\_id')->unsigned();

$table->foreign('user\_id')->references('id')->on('users');

});

Hoặc mình muốn khi xóa mội users thì đồng thời sẽ xóa hết các bài đăng của users đó đi thì .

$table->foreign('user\_id')

->references('id')->on('users')

->onDelete('cascade');

### Xóa khóa ngoại.

Cú Pháp

$table->dropForeign('columnName');

VD: Mình xóa khóa ngoại ở ví dụ trên đi.

$table->dropForeign('user\_id');

## 5,Lời kết.

-Sau khi đọc hết phần trên các bạn có thấy Laravel hỗ trợ chúng ta rất mạnh về cấu trúc của database đúng không?

# **Bài 16: Migrations trong Laravel**

Ở các bài trước mình đã giới thiệu với mọi người cách truy xuất dữ liệu trong Laravel Với [Query Buider](https://toidicode.com/query-builder-trong-laravel-11.html) và [Eloquent ORM](https://toidicode.com/eloquent-orm-trong-laravel-13.html) rồi. Hôm nay tiếp tục với series [Học Laravel](https://toidicode.com/hoc-laravel) của Toidicode.com mình xin được giới thiệu với mọi người một thuật ngữ mới nữa trong Laravel đó là **Migratios,**

## 1, Migrations trong Laravel là gì?

-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.

-Điều kiện để sử dụng Migations:

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

## 2, Tạo migrations.

-Để Tạo Migrations thì các bạn cũng có 2 cách tạo là dùng tay và dùng lệnh, nhưng mình khuyến khích mọi người dùng lệnh.

-Tạo Migrations bằng lệnh thì các bạn mở cmd lên và trỏ tới thư mục chứa project của các bạn(như mọi khi :-) ) và gõ 1 trong các lệnh sau tùy theo mục đích của bạn.

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

-Chú Thích: **TenMigrate,TableName** là các thông số các bạn có thể tùy chỉnh.

VD: Mình sẽ tạo Migrations create\_users\_table cho table users.

php artisan make:migration create\_users\_table --create=users

-Nếu tạo thành công nó sẽ báo dạng như sau: Created migration: xxxxxxxxxxxx: Lúc này bạn có thể kiểm tra lại bằng cách truy cập vào  **App\database\migrations**nếu thấy  có file tên trùng với phần xxxxxxxx ở trên thì là đã thành công.

-Tiếp đó các bạn mở file ra và sẽ thấy nội dung có dạng:

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('users', function (Blueprint $table) {

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

//

}

}

### Hàm up và hàm down.

-Hàm up trong Migrations có tác dụng thực thi migration

-Hàm down trong Migrations có tác dụng thực thi đoạn lệnh rollback(trở về trước đó).

## 3, Các cú pháp trong Migrations.

-Dưới đây là một số các câu lệnh tạo bảng hay dùng trong Migrations.

| Lệnh | Chức năng |
| --- | --- |
| $table->bigIncrements('id'); | Tạo cột id khóa chính tự động tăng kiểu bigint |
| $table->bigInteger('votes'); | Tạo cột votes với kiểu bigint |
| $table->binary('data'); | Tạo cột data với kiểu blob |
| $table->boolean('confirmed'); | Tạo cột confirmed với kiểu boolean |
| $table->char('name', 4); | Tạo cột name với kiểu char tối đa 4 kí tự |
| $table->date('created\_at'); | Tạo cột created\_atvới kiểu date |
| $table->dateTime('created\_at'); | Tạo cột created\_atvới kiểu dateTime |
| $table->dateTimeTz('created\_at'); | Tạo cột name với kiểu DATETIME (with timezone) |
| $table->decimal('amount', 5, 2); | Tạo cột name với kiểu DECIMAL |
| $table->double('column', 15, 8); | Tạo cột name với kiểu DOUBLE |
| $table->enum('choices', ['foo', 'bar']); | ENUM equivalent for the database. |
| $table->float('amount', 8, 2); | FLOAT equivalent for the database, 8 digits in total and 2 after the decimal point. |
| $table->increments('id'); | Incrementing ID (primary key) using a "UNSIGNED INTEGER" equivalent. |
| $table->integer('votes'); | INTEGER equivalent for the database. |
| $table->ipAddress('visitor'); | IP address equivalent for the database. |
| $table->json('options'); | JSON equivalent for the database. |
| $table->jsonb('options'); | JSONB equivalent for the database. |
| $table->longText('description'); | LONGTEXT equivalent for the database. |
| $table->macAddress('device'); | MAC address equivalent for the database. |
| $table->mediumIncrements('id'); | Incrementing ID (primary key) using a "UNSIGNED MEDIUM INTEGER" equivalent. |
| $table->mediumInteger('numbers'); | MEDIUMINT equivalent for the database. |
| $table->mediumText('description'); | MEDIUMTEXT equivalent for the database. |
| $table->morphs('taggable'); | Adds unsigned INTEGER taggable\_id and STRING taggable\_type. |
|  |  |
|  |  |
|  |  |
| $table->nullableTimestamps(); | Same as timestamps(). |
| $table->rememberToken(); | Adds remember\_token as VARCHAR(100) NULL. |
| $table->smallIncrements('id'); | Incrementing ID (primary key) using a "UNSIGNED SMALL INTEGER" equivalent. |
| $table->smallInteger('votes'); | SMALLINT equivalent for the database. |
| $table->softDeletes(); | Adds nullable deleted\_at column for soft deletes. |
| $table->string('email'); | VARCHAR equivalent column. |
| $table->string('name', 100); | VARCHAR equivalent with a length. |
| $table->text('description'); | TEXT equivalent for the database. |
| $table->time('sunrise'); | TIME equivalent for the database. |
| $table->timeTz('sunrise'); | TIME (with timezone) equivalent for the database. |
| $table->tinyInteger('numbers'); | TINYINT equivalent for the database. |
| $table->timestamp('added\_on'); | TIMESTAMP equivalent for the database. |
| $table->timestampTz('added\_on'); | TIMESTAMP (with timezone) equivalent for the database. |
| $table->timestamps(); | Adds nullable created\_at and updated\_at columns. |
| $table->timestampsTz(); | Adds nullable created\_at and updated\_at (with timezone) columns. |
| $table->unsignedBigInteger('votes'); | Unsigned BIGINT equivalent for the database. |
| $table->unsignedInteger('votes'); | Unsigned INT equivalent for the database. |
| $table->unsignedMediumInteger('votes'); | Unsigned MEDIUMINT equivalent for the database. |
| $table->unsignedSmallInteger('votes'); | Unsigned SMALLINT equivalent for the database. |
| $table->unsignedTinyInteger('votes'); | Unsigned TINYINT equivalent for the database. |
| $table->uuid('id'); | UUID equivalent for the database. |

-Nhiều quá các bạn chịu khó Google translate nhé : :-P

### -VD: Mình sẽ tạo bảng users ở trên với các cột:

* id; khóa chính tự động tăng.
* name: kiểu dữ liệu varchar.
* email:kiểu dữ liệu varchar và là duy nhất(unique).
* password: kiểu dữ liệu varchar.

Code:

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->bigIncrements('id');

$table->string('name');

$table->string('email')->unique();

$table->string('password');

});

}

-còn nếu các bnj muốn khi rollback thì sẽ xóa bảng users thì các bạn viết đoạn code sau:

public function down()

{

Schema::dropIfExists('users');

}

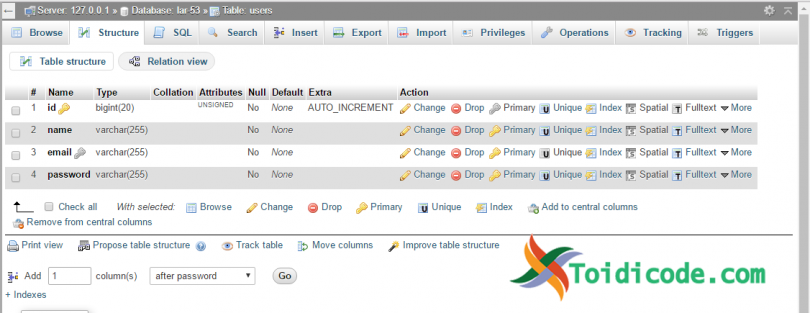
## 4, Thực thi Migrations.

-Sau khi đã tạo và viết code cho migrate thì tất nhiên là chúng ta cần phải chạy nó đúng không nhỉ?(không thì viết làm chi)

-Các lệnh thực thi migrations:

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

VD: Với bảng users ở trên mình sẽ tiến hành chạy lệnh php artisan migrate Kết quả:



## 5,Lời kết.

-Phần trên là các kiến thức cơ bản trong Migrations còn bạn này muốn tìm hiểu chi tiết hơn về nó có thể tham khảo tại [đây](https://laravel.com/docs/5.3/migrations) .

# **Bài 17: Seeding trong Laravel**

Ở Các phần trước mình đã giới thiệu với mọi người về [Migrations](https://toidicode.com/migrations-trong-laravel-17.html) với [Schema Builder](https://toidicode.com/migrations-trong-laravel-17.html) Nhưng vấn đề là nó mới chỉ tạo ra các cấu trúc của database còn dữ liệu thì thêm như thế nào, ở đây chắc chắn sẽ có bạn nói rằng sẽ thêm bằng phpMyadmin nhưng ở đây Laravel đã cung cấp cho chúng ta một thứ rất hay để thêm dữ liệu mẫu vào database.

## 1, Seeding là gì?

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

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

## 2,Sử dụng seeder.

### Sử dụng seeder gốc.

-Mặc định trong laravel có cung cấp cho chúng ta một **DatabaseSeeder.php** trong thư mục **database/seeds/**có code sẵn như sau:

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

// $this->call(UsersTableSeeder::class);

}

}

Và để sử dụng nó chúng ta chỉ cần viết code của Query Builder vào đó.

**VD:**Mình sẽ thêm dữ liệu cho bảng users (name,email,password) như sau:

<?php

use Illuminate\Database\Seeder;

use Illuminate\Database\Eloquent\Model;

class DatabaseSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

DB::table('users')->insert([

'name' => str\_random(10),

'email' => str\_random(10).'@gmail.com',

'password' => bcrypt('secret'),

]);

}

}

  Và để chạy nó chúng ta sẽ dùng lệnh.

php artisan db:seed

-Tuy nhiên thông thường khi làm dự án thực tế thì chúng ta sẽ không lên làm như thế, bởi vì như thế sẽ rất khó quản lý, nên chúng ta sẽ phải chia nhỏ ra từng file cho từng table để quản lý.

### Tạo seeder.

-Để tạo Seeder trong Laravel chúng ta dùng lệnh:

php artisan make:seed seedName

-Với seedName: là tên của seed các bạn muốn tạo (nên đặt tên có ý nghĩa để sau này dễ quản lý).

**VD:** Mình muốn tạo file Users.php để quản lý dữ liệu cho bảng users.

-Đầu tiên chúng ta sẽ dùng lệnh: **php artisan make:seed Users.php.**

-Sau đó chúng ta vào **database/seeds**sẽ thấy một file Users.php có nội dung như sau:

<?php

use Illuminate\Database\Seeder;

class Users extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

//

}

}

-Giờ chúng ta chỉ cần chèn code Query Builder vào hàm run như VD trên là được.

-Tiếp đó chúng ta sẽ có hai cách chạy:

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

## 3,Lời kết.

-Phần trên mình đã giới thiệu với các bạn về **Seeder** cơ bản, còn một số các kiến thức nâng cao về S**eeder** mình sẽ giới thiệu với các bạn vào một dịp khác nhé! Chúc các bạn học tốt!

# **Bài 18: Form Request trong Larvel**

-Sau khi đã giới thiệu với các bạn về [model](https://toidicode.com/model-trong-laravel-12.html), [controller](https://toidicode.com/controller-trong-laravel-10.html),.. thì tiếp đến ở bài này mình sẽ giới thiệu về Form Request một thứ không thể thiếu trong một framwwork.

-Và để bắt đầu chúng ta cần tạo cho nó một view.

## 1,Tạo View.

-Đầu tiên mình sẽ tạo ra một view FormRequest.blade.php có nội dung như sau:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Test form</title>

<link rel="stylesheet" href="">

</head>

<body>

<form action="{{ url('handle-form') }}" method="POST" role="form">

<legend>Test submit form</legend>

{{ csrf\_field()}}

<div class="form-group">

<label for="">label</label>

<input type="text" name="name" class="form-control" id="" placeholder="Input field">

</div>

<button type="submit" class="btn btn-primary">Submit</button>

</form>

</body>

</html>

-Ở đây mình có tạo ra một form có một trường name input, một button submit và một action {{ url('handle-form') }} (action này nó sẽ có chức năng gọi đến đường dẫn yourdomain.com/handle-form ).

**Chú ý:** đối với Laravel khi xử lý form thì bắt buộc phải có **csrf\_token (tránh tấn công xss)**nếu không thì sẽ không thực hiện được submit form được. Để tạo csrf\_token cho form chúng ta có 2 cách sau:

* Dùng: **{{ csrf\_field() }}**
* Dùng: **<input type="hidden" name="\_token" value="<?php echo csrf\_token(); ?>">** hoặc **<input type="hidden" name="\_token" value="{{ csrf\_token() }} ?>">**đối với blade template.

## 2,Tạo controller.

-Tiếp theo mình lại tạo thêm một handleController để xử lý Reuqest.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HandleController extends Controller

{

public function getForm(){

return view('FormRequest');

}

public function handleRequest(Request $request){

// nhan het du lieu co trong form

return $request->all();

}

}

-Ở trên mình có tạo ra 2 function:

* getForm: để hiển thị view form-request.blade.php ở trên.
* handleRequest: để xử lý nhận dữ liệu của form.

**=>Chú ý:  Để nhận dữ liệu từ form được thì chúng ta cần phải truyền vào hàm nhận một lớp Request.(Nói ở phần dưới).**

## 3,Tạo Route.

-Sau khi đã tạo xong View vào Controller giờ mình sẽ tạo tiếp 2 Route để hiện view và xử lý dữ liệu.

Route::get('get-form',['uses'=> 'HandleController@getForm']);

Route::post('handle-form',['uses'=> 'HandleController@handleRequest']);

//Hoặc ngắn gọn hơn như sau

Route::get('get-form', 'HandleController@getForm');

Route::post('handle-form', 'HandleController@handleRequest');

-Cái này thì mình không giải thích nữa nhé (ai không hiểu xem lại [route](https://toidicode.com/route-trong-laravel-phan-1-3.html) ).

## 4,Các hàm xử lý request.

-Trong Laravel hỗ trợ chúng ta rất mạnh về xử lý nhận dữ liệu từ form. Nên trong bài này mình sẽ không thể nêu hết các phương thức, thuộc tính mà Laravel hỗ trợ được, mình sẽ nêu ra các phương thức, thuộc tính hay dùng thôi.

### Lấy hết dữ liệu của form.

-Để hết dữ liệu của form gửi lên thì Laravel. thì mọi người sử dụng phương thức all().

VD: đối với view trên mọi người sẽ nhận như sau.

public function handleRequest(Request $request)

{

// nhan het du lieu co trong form

return $request->all();

}

### Lấy dữ liệu theo name định sẵn.

-Để lấy dữ liệu của name định sẵn các bạn sử dụng cú pháp:

$request->input('inputname');

Hoặc

$request->inputname;

Hoặc:

$request->get('inputname');

* Chú thích: **inputName** là name của input các bạn đặt ở form.

VD: Mình sẽ lấy giá trị của name='name' ở trong view trên như sau:

public function handleRequest(Request $request)

{

return $request->get('name');

}

### Lấy dữ liệu của nhiều name.

Cú Pháp:

public function handleRequest(Request $request)

{

return $request->get('name');

}

-Hàm này cho phép lấy giá trị của nhiều name và trả về dữ liệu mảng.

### Kiểm tra phương thức gửi của form.

-Laravel có cung cấp luôn cho chúng ta hàm kiểm kiểu submit của form là GET,POST,.. Cú pháp xử lý như sau:

public function handleRequest(Request $request)

{

return $request->isMethod('post');

}

-phương thức này trả về kiểu giá trị boolean (true,false);

### Lấy path của form.

-Cú Pháp:

public function handleRequest(Request $request)

{

return $request->path();

}

-Hàm này cho phép chúng ta lấy đường dẫn tới form.

VD: http://domain.com/form/submit/

### Giới hạn đường dẫn gửi form.

public function handleRequest(Request $request){

return $request->is('something');

}

-Hàm này có tác dụng giới hạn các url được chấp nhận gửi form.

VD: chỉ muốn nhận dữ liệu từ đường dẫn có path bắt đầu bằng admin

public function handleRequest(Request $request)

{

if ($request->is('admin/\*')) {

// code

}

}

### Sử dụng collections trong form.

-Với sự hỗ trợ mạnh mẽ của Laravel thì các bạn cũng có thể sử dụng collection trong Form Request.

VD:

$input = $request->only(['username', 'password']);

$input = $request->only('username', 'password');

$input = $request->except(['credit\_card']);

$input = $request->except('credit\_card');

## 3, Lời kết.

-Phần trên mình đã giới thiệu với mọi người về Form Request trong Laravel rồi, tuy nó chỉ là lý thuyết nhưng nó sẽ là nền tảng cho bạn khi làm việc với Laravel về sau.

# **Bài 19: Upload files trong Laravel**

-Ở phần trước mình đã giới thiệu với mọi người về [Form Request](https://toidicode.com/form-request-trong-larvel-42.html) rồi, nhưng đó mới chỉ là gửi nhận dữ liệu dưới dạng text/plain vậy còn gửi và nhận dữ liệu với file thì làm thế nào?

Hôm nay chúng ta sẽ cùng tìm hiểu nốt về upload file trong Laravel

## 1, Chuẩn Bị.

### Tạo View.

-Đầu tiên mình cần tạo ra một view form có một input file và một input submit để thực hiện upload. (name: d**emo\_upload.blade.php**)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>Demo Upload File - Toidicode.com</title>

<meta name="author" content="ThanhTai">

</head>

<body>

<form action="{{ url('file') }}" enctype="multipart/form-data" method="POST">

{{ csrf\_field() }}

<input type="file" name="filesTest" required="true">

<br/>

<input type="submit" value="upload">

</form>

</body>

</html>

### Tạo Controller.

Tiếp theo chúng ta cần tạo một controller có tên: **FileController.php**

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FileController extends Controller

{

public function index()

{

return view('demo\_upload');

}

public function doUpload(Request $request)

{

//xử lý upload ở đây

}

}

### Tạo Route.

- Cuối cùng là tạo route để điều hướng.

Route::get('file','FileController@index');

Route::post('file','Filecontroller@doUpload');

## 2, Các Hàm lấy thông tin của file.

-Để cho dễ hiểu mình sẽ làm luôn VD đối với form ở trên.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FileController extends Controller

{

public function index()

{

return view('file');

}

public function doUpload(Request $request)

{

//Kiểm tra file

if ($request->hasFile('fileTest')) {

$file = $request->filesTest;

//Lấy Tên files

echo 'Tên Files: ' . $file->getClientOriginalName();

echo '<br/>';

//Lấy Đuôi File

echo 'Đuôi file: ' . $file->getClientOriginalExtension();

echo '<br/>';

//Lấy đường dẫn tạm thời của file

echo 'Đường dẫn tạm: ' . $file->getRealPath();

echo '<br/>';

//Lấy kích cỡ của file đơn vị tính theo bytes

echo 'Kích cỡ file: ' . $file->getSize();

echo '<br/>';

//Lấy kiểu file

echo 'Kiểu files: ' . $file->getMimeType();

}

}

}

## 3, Upload File.

-Sau khi đã tạo được View,Controller,route phục vụ cho việc upload file rồi, giờ chúng ta sẽ tiến hành xử lý upload file.

-Để upload files với Request thì Laravel có cung cấp cho chúng ta hàm **move()**sử dụng với 2 thông số truyền vào:

move($Location, $filesName)

Trong đó:

* $Location: Là thư mục chứa file upload lên Sever.
* $filesName: Là tên mới của file.

VD: Mình sẽ upload form ở VD trên vào thư mục uploads:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FileController extends Controller

{

public function index()

{

return view('file');

}

public function doUpload(Request $request)

{

$file = $request->filesTest;

$file->move('upload', $file->getClientOriginalName());

// hàm sẽ trả về đường dẫn mới của file trên server nếu thành công

// còn nếu không nó sẽ raise ra exception.

}

}

## 4) Lời kết.

-Phần trên mình đã giới thiệu với mọi người về uploadfiles trong Laravel. Còn cách ràn buộc loại files, dung lượng files mình sẽ trình bày tiếp ở các phần sau.

Chúc các bạn học tốt!

# **Bài 20: Validation trong Laravel**

Phần trước mình đã giới thiệu với mọi người về [form request](https://toidicode.com/form-request-trong-larvel-42.html) trong Laravel rồi, Phần này mình tiếp tục giới thiệu với mọi người về validation để ràng buộc dữ liệu.

**1, Validation là gì?**

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

**2, Validation Form Trong Laravel.**

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

**Validation Trực tiếp trên Controller.**

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

use Illuminate\Http\Request;

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

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

Trong đó:

* **$request:** Là biến tham chiếu đối tượng Request mà các bạn khai báo ở đầu hàm.
* **$pattern:** Là mảng định nghĩa dữ liệu đầu vào của các trường trong form.
* **$messenger:** Là mảng chứa nội dung báo lỗi (Nếu muốn thay đổi).
* **$customName:** Là mảng chứa các tên cho các trường trong form.

-**Chú ý:**Nếu không cần thay đổi nội dung lỗi,tên input mặc định thì không cần truyền giá trị cho 2 tham số này.

**VD1:**Mình muốn validation dữ liệu cho một form có 2 input: name,age thì mình sẽ phải viết đoạn validation như sau:

public function handleForm(Request $request)

{

$this->validate($request,

[

'name' => 'required|min:5|max:255',

'age' => 'required|integer|max:3',

],

[

'required' => ':attribute Không được để trống',

'min' => ':attribute Không được nhỏ hơn :min',

'max' => ':attribute Không được lớn hơn :max',

'integer' => ':attribute Chỉ được nhập số',

],

[

'name' => 'Tiêu đề',

'age' => 'Tuổi',

]

);

}

-Đoạn trên mình đã validation cho trường name không được để trống và dữ liệu nhập vào phải có độ dài lơn hơn 5 và nhỏ hơn 255, trường age không được để trống và dữ liệu nhập vào phải là số.

**Validation với lớp validator.**

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

use Validator;

use Illuminate\Http\Request;

Tiếp đó khởi tạo validator với cú pháp:

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

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

Trong đó:

* **$request:** Là biến tham chiếu đối tượng Request mà các bạn khai báo ở đầu hàm (Nhưng ở đây phải là mảng).
* **$pattern:** Là mảng định nghĩa dữ liệu đầu vào của các trường trong form.
* **$messenger:** Là mảng chứa nội dung báo lỗi (Nếu muốn thay đổi).
* **$customName:** Là mảng chứa các tên cho các trường trong form.

-Chú ý: Cũng giống validate nếu không cần thay đổi nội dung lỗi,tên input mặc định thì không cần truyền giá trị cho 2 tham số này.

**VD2:**Giống VD1 giờ mình sẽ làm theo validator.

$validate = Validator::make(

$request->all(),

[

'title' => 'required|min:5|max:255',

'content' => 'required',

],

[

'required' => ':attribute Không được để trống',

'min' => ':attribute Không được nhỏ hơn :min',

'max' => ':attribute Không được lớn hơn :max',

],

[

'title' => 'Tiêu đề',

'content' => 'Nội dung',

]

);

if ($validate->fails()) {

return View('ValidationView')->withErrors($validate);

}

-Ở đây ValidationView là view ban đầu submit lên còn các thông số withErrors mình đã giới thiệu ở các phần trước rồi.

**3, Hiển Thị Thông báo lỗi Trong Laravel.**

-Khi mà form submit không thành công do không nhập đúng yêu cầu thì bạn cần phải thông báo lỗi ra trình duyệt. Và Laravel có hỗ trợ chúng ta một số phương thức để hiển thị lỗi.

**Kiểm tra có tồn tại lỗi không.**

-Để kiểm tra có tồn tại lỗi hay không thì Laravel có cung cấp cho chúng ta phương thức **has()**với cú pháp:

$errors->has('inputName');

// output true||false

-inputName: Ở đây là tên của input cần kiểm tra.

VD: Kiểm tra xem input title có tồn tại lỗi không.

if($errors->has('title')) {

//true

}

**Lấy hết lỗi.**

-Để lấy hết lỗi mọi người sử dụng phương thức **all()**với cú pháp

$errors->all();

**Lấy lỗi đầu tiên của input.**

-Để lấy lỗi đầu tiên của input Laravel có cung cấp cho chúng ta phương thức **first()** với cú pháp:

$errors->first('inputName')

VD: Lấy lỗi đầu tiên của input có name='content' nếu có:

if ($errors->has('content') {

echo $errors->first('content');

}

**4, Các validation Hay dùng.**

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

-Ngoài ra các bạn có thể xem thêm tại: [đây](https://laravel.com/docs/5.3/validation#available-validation-rules)

**3, Lời kết,**

-Phần trên mình đã hướng dẫn mọi người 2 cách validation trong Laravel, nhưng bên cạnh đó Laravel còn cung cấp cho chúng ta một cách nữa, và cách này là chuẩn với nguyên tắc DRY(Don't Repeat Yourself) mình sẽ giới thiệu ở phần tiếp theo.

# **Bài 21: Validation trong Laravel (Phần 2)**

Phần trước mình đã giới thiệu với mọi người 2 cách validation form trực tiếp trên Controller rồi, nhưng các cách đó đều được cho là không hay vì bạn sẽ phải lặp lại việc validate ở rất nhiều nơi, và điều đó thì đi ngược lại với nguyên tắc lập trình cụ thể ở đây là **DRY**(Don't Repeat Yourself). Và một framework được cho là siêu mạnh như Laravel thì nó đều có cách khắc phục cả.

## ****1, Tạo Request với FormRequest****.

-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 :D.

-Cú Pháp tạo Formrequest:

php artisan make:request RequestName

-Trong đó: **RequestName** là Tên Request mà bạn muốn tạo.

VD: Mình tạo một LoginRequest.

php artisan make:request LoginRequest

-Nếu như thành công thì bạn sẽ nhận được thông báo:

created Request successfully

Ngay sau đó bạn truy cập vào **app/Http/Requests** sẽ thấy có một file **LoginRequest.php** có nội dung như sau:

<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class LoginRequest extends FormRequest

{

/\*\*

\* Determine if the user is authorized to make this request.

\*

\* @return bool

\*/

public function authorize()

{

return false;

}

/\*\*

\* Get the validation rules that apply to the request.

\*

\* @return array

\*/

public function rules()

{

return [

//

];

}

}

Chú ý: Tất cả các file Request phải được đặt trong folder Requests.

## 2, Cấu hình Request.

### 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: nếu muốn validation.

+FALSE: nếu không muốn 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 [

//pattern content

];

}

-Và nội dung cú pháp thì hoàn toàn giống với [phần trước](https://toidicode.com/validation-trong-laravel-44.html).

-Chú ý: Tên Hàm là mặc định không được thay đổi.

**VD:**

public function rules()

{

return [

'name'=>'required|max:20',

'age'=>'required|integer|max:20',

];

}

### Thay đổi nội dung báo lỗi.

-Để thay đổi nội dung lỗi hiển thị thì mọi người cần phải tạo một hàm có name **messages()**và dữ liệu trả về là một mảng chứa nội dung báo lỗi vẫn theo cú pháp của [phần trước](https://toidicode.com/validation-trong-laravel-44.html).

VD:

public function messages()

{

return [

'required'=>':attribute Không được để trống',

'max'=>':attribute Không được quá :max ký tự',

];

}

-Chú ý: Thay đổi tên Hàm sẽ không hoạt động.

### Thay đổi tên input.

-Và để thay đổi tên cho input thì mọi người cũng cần phải tạo thêm một hàm có tên **attributes()** và dữ liệu trả về là một mảng chứa tên mới.

VD:

public function attributes(){

return [

'name'=>'Tên',

'age'=>'Tuổi',

];

}

-Chú ý: Thay đổi tên Hàm sẽ không hoạt động.

## 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\RequestName;

+Trong đó:RequestName là tên Class Request mà bạn muốn sử dụng.

VD: Mình muốn gọi LoginRequest ở VD trên:

use App\Http\Requests\LoginRequest;

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

public function name(RequestName $request)

{

//

}

 +**Trong đó:**

* **name:** là tên hàm của các bạn.
* **RequestName**: là tên class Request mà các bạn vừa khai báo namespace của nó.
* **$request:**Là biến mà các bạn muốn ánh xạ vào.

**VD:**

public function store(LoginRequest $request)

{

return $request->all();

}

## 3, Lời kết.

-Phần trên mình đã giới thiệu với mọi người về lớp FormRequest trong Laravel và các hoạt động của nó. Và với cá nhân mình thì mình khuyên các bạn nên validation theo cách này vi cách này nó đáp ứng được quy luật lập trình **DRY.**

# **Bài 22: Authentication Trong Laravel**

Authentication  là một hành động xác  thực một cái gì đó có được hoạt động tiếp hay không hay dừng lại.Và hôm nay chúng ta sẽ cùng tìm hiểu về authentication trong Laravel nhé!

## 1, Authentication là gì?

-Authentication theo Wikipedia:

*là một hành động nhằm thiết lập hoặc chứng thực một cái gì đó (hoặc một người nào đó) đáng tin cậy, có nghĩa là, những lời khai báo do người đó đưa ra hoặc về vật đó là sự thật. Xác thực một đối tượng còn có nghĩa là công nhận nguồn gốc (*provenance*) của đối tượng, trong khi, xác thực một người thường bao gồm việc thẩm tra nhận dạng của họ. Việc xác thực thường phụ thuộc vào một hoặc nhiều nhân tố xác thực (*authentication factors*) để minh chứng cụ thể.*

-Theo như wikipedia định nghĩa thì các bạn cũng đã hiểu sơ qua về nó rồi đúng không? Thì ở trong lập trình khái niện nó cũng như thế, và ứng dụng thực tiễn là chức năng đăng nhập vào hệ thống.

## 2, Authentication trong Laravel.

-Ở phần đầu chúng ta đã hiểu về khái niệm rồi và phần này chúng ta sẽ xem xem Authentication trong Laravel hoạt động như thế nào nhé!

-Đầu tiên mọi người hãy mở thư mục config trong project Laravel và mở tệp tin **auth.php** ra và xem:

<?php

return [

/\*

|--------------------------------------------------------------------------

| Authentication Defaults

|--------------------------------------------------------------------------

|

| This option controls the default authentication "guard" and password

| reset options for your application. You may change these defaults

| as required, but they're a perfect start for most applications.

|

\*/

'defaults' => [

'guard' => 'web',

'passwords' => 'users',

],

/\*

|--------------------------------------------------------------------------

| Authentication Guards

|--------------------------------------------------------------------------

|

| Next, you may define every authentication guard for your application.

| Of course, a great default configuration has been defined for you

| here which uses session storage and the Eloquent user provider.

|

| All authentication drivers have a user provider. This defines how the

| users are actually retrieved out of your database or other storage

| mechanisms used by this application to persist your user's data.

|

| Supported: "session", "token"

|

\*/

'guards' => [

'web' => [

'driver' => 'session',

'provider' => 'users',

],

'api' => [

'driver' => 'token',

'provider' => 'users',

],

],

/\*

|--------------------------------------------------------------------------

| User Providers

|--------------------------------------------------------------------------

|

| All authentication drivers have a user provider. This defines how the

| users are actually retrieved out of your database or other storage

| mechanisms used by this application to persist your user's data.

|

| If you have multiple user tables or models you may configure multiple

| sources which represent each model / table. These sources may then

| be assigned to any extra authentication guards you have defined.

|

| Supported: "database", "eloquent"

|

\*/

'providers' => [

'users' => [

'driver' => 'eloquent',

'model' => App\User::class,

],

/\* 'users' => [

'driver' => 'database',

'table' => 'users',

], \*/

],

/\*

|--------------------------------------------------------------------------

| Resetting Passwords

|--------------------------------------------------------------------------

|

| You may specify multiple password reset configurations if you have more

| than one user table or model in the application and you want to have

| separate password reset settings based on the specific user types.

|

| The expire time is the number of minutes that the reset token should be

| considered valid. This security feature keeps tokens short-lived so

| they have less time to be guessed. You may change this as needed.

|

\*/

'passwords' => [

'users' => [

'provider' => 'users',

'table' => 'password\_resets',

'expire' => 60,

],

],

];

-Nó sẽ có đoạn như trên (chú ý: Mình đang viết bài trên phiên bản mới nhất -5.3 Sau này khi có bản mới hơn có thể nó sẽ khác đi một chút).

-Đoạn code trên khá là 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 là loàng ngoàng lên ở series basic này mình sẽ không trình  bày nhiều về cả đoạn này vì nói nhiều các bạn sẽ loạn. Các bạn chỉ cần 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 thích**:

1. **Driver**: Đây là thiết lập xác định phương thức lấy thông tin người dùng để xác thực, và như các bạn thấy thì  mặc định nó sẽ sử dụng [***eloquent***](https://toidicode.com/eloquent-orm-trong-laravel-13.html)***,*** tuy nhiên còn một sự lựa chọn khác là database và thậm trí bạn cũng có thể tạo thêm một driver của riêng bạn( sẽ nói ở phần nâng cao). Đối với series cơ bản này thì mình khuyên các bạn nên dữ nguyên.
2. **Model**: Đây là thiết lập nguồn dữ liệu lấy ra từ đâu. Ở đây mặc định Laravel chọn là moder User (local: App\User.php). Thông số này bạn có thể chỉnh thành name model của bạn.

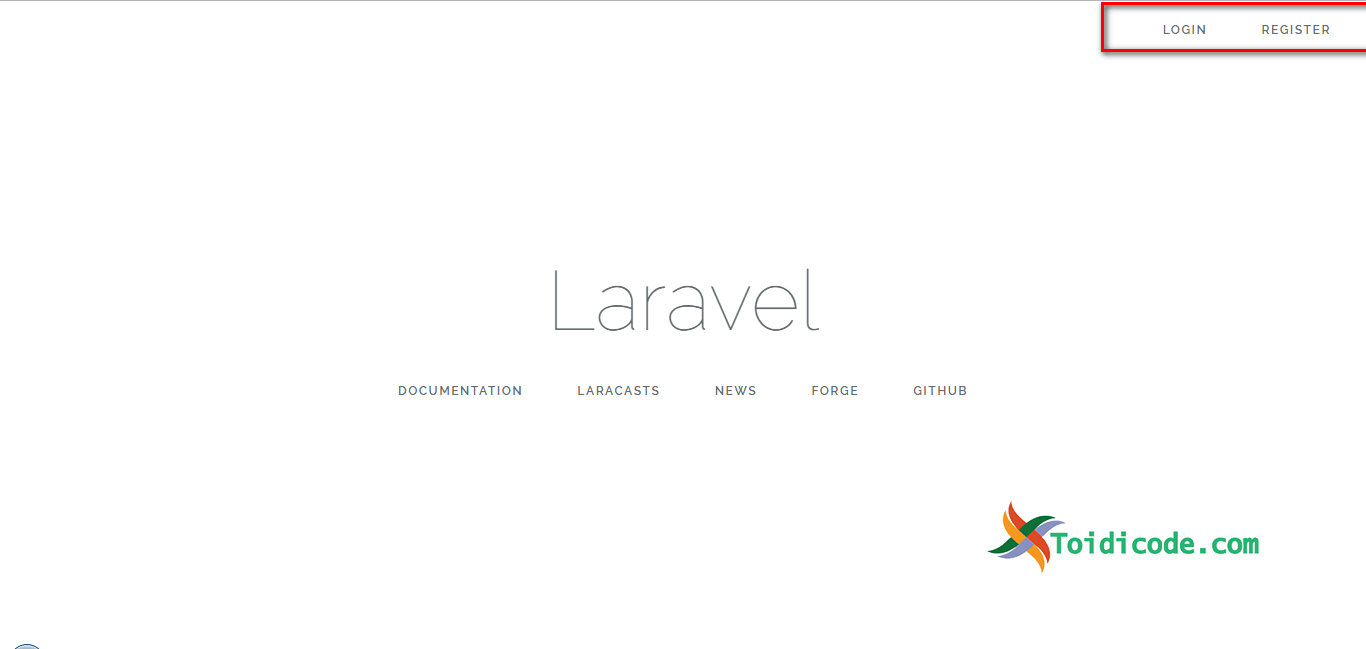
## 3, Tạo Auth Login Trong Laravel.

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

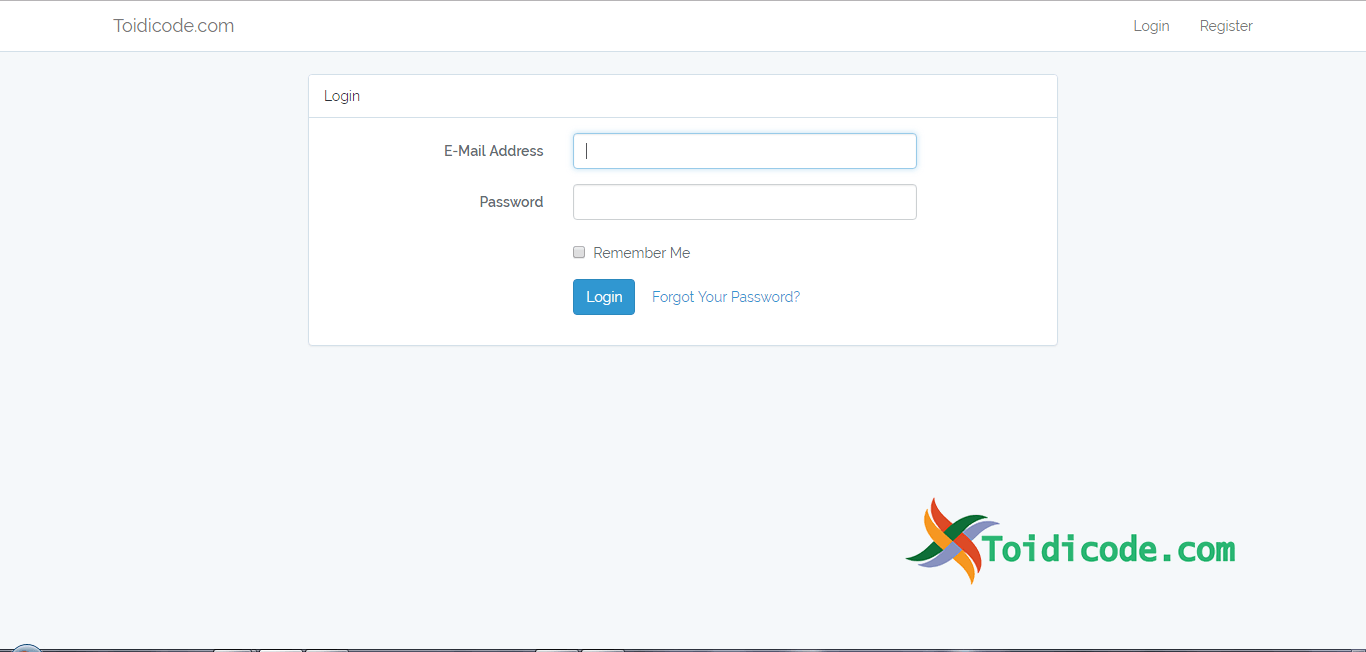
php artisan make:auth

Sau khi chạy lệnh này lên thì Laravel sẽ thêm cho chúng ta một homeController,2 route 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.



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



Login Form



Register form

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

## 3,Lời Kết.

- Phần trên mình đã trình bày về authentication trong Laravel. Nhưng ở đây chỉ là phiên bản mặc định của Laravel thôi còn mình muốn thay đổi một số thứ thì phải làm sao? cùng xem tiếp sau nhé!

Chúc Các bạn học tốt!

# **Bài 23: Authentication Trong Laravel (Register)**

Ở bài trước chúng ta đã tìm hiểu về [authentication default](https://toidicode.com/authentication-trong-laravel-47.html) của Laravel rồi, nhưng giờ đây bảng của bạn không chỉ có bằng ý cột mà nhiều hơn thì chức năng đăng ký của Laravel sẽ không sử dụng được? Sau khi đọc xong bài này bạn sẽ giải quyết được vấn đề đó.

## 1, Tạo database.

- Đầu tiên chúng ta cần phải tạo ra một bảng dữ liệu để thự hiện chức năng đăng nhập. Các bạn truy cập vào **database/migrations**mở tệp **2014\_10\_12\_000000\_create\_users\_table.php**và chỉnh sửa một số thông số như sau:

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('users', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->string('username')->unique();

$table->string('email');

$table->string('password');

$table->integer('level');

$table->rememberToken();

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::drop('users');

}

}

Sau đó các bạn chạy migration (ai không biết xem lại series nhé).

## 2, Chỉnh sửa form.

-Sau Khi đã có database rồi giờ chúng ta cần làm là chỉnh sửa lại các trường trong trong giao diện register của Laravel (Giao diện rigister mặc định sẽ nằm ở resources/views/auth/register.blade.php) như sau:

@extends('layouts.app')

@section('content')

<div class="container">

<div class="row">

<div class="col-md-8 col-md-offset-2">

<div class="panel panel-default">

<div class="panel-heading">Register</div>

<div class="panel-body">

<form class="form-horizontal" role="form" method="POST" action="{{ url('/register') }}">

{{ csrf\_field() }}

<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">

<label for="name" class="col-md-4 control-label">Name</label>

<div class="col-md-6">

<input id="name" type="text" class="form-control" name="name" value="{{ old('name') }}" required autofocus>

@if ($errors->has('name'))

<span class="help-block">

<strong>{{ $errors->first('name') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group{{ $errors->has('username') ? ' has-error' : '' }}">

<label for="email" class="col-md-4 control-label">Username</label>

<div class="col-md-6">

<input id="email" 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>

<div class="form-group{{ $errors->has('email') ? ' has-error' : '' }}">

<label for="email" class="col-md-4 control-label">E-Mail Address</label>

<div class="col-md-6">

<input id="email" type="email" class="form-control" name="email" value="{{ old('email') }}" required>

@if ($errors->has('email'))

<span class="help-block">

<strong>{{ $errors->first('email') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">

<label for="password" class="col-md-4 control-label">Password</label>

<div class="col-md-6">

<input id="password" type="password" class="form-control" name="password" required>

@if ($errors->has('password'))

<span class="help-block">

<strong>{{ $errors->first('password') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group">

<label for="password-confirm" class="col-md-4 control-label">Confirm Password</label>

<div class="col-md-6">

<input id="password-confirm" type="password" class="form-control" name="password\_confirmation" required>

</div>

</div>

<div class="form-group">

<div class="col-md-6 col-md-offset-4">

<button type="submit" class="btn btn-primary">

Register

</button>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

@endsection

Giờ các bạn run project lên và vào register (như [bài trước](https://toidicode.com/authentication-trong-laravel-47.html)) và thấy giao diện đã được thêm input username rồi đúng không? Nhưng khi các bạn đăng ký thì nó sẽ bị lỗi (không tin thử là biết) vì các bạn mới chỉ khai báo giao diện mà chưa báo cho Laravel biết là phải thêm input username vào.

## 3, Chỉnh sửa hệ thống.

### Controllers

-Giờ muốn sử dụng được mọi người  tiếp tục tìm đến controllers **RegisterController**(đường dẫn: **app/Http/Controllers/Auth/RegisterController.php**) và thay đổi 2 function **validator** và **create** như sau:

protected function validator(array $data)

{

return Validator::make($data, [

'name' => 'required|max:255',

'email' => 'required|email|max:255',

'username' => 'required|max:16|unique:users',

'password' => 'required|min:6|confirmed',

]);

}

/\*\*

\* Create a new user instance after a valid registration.

\*

\* @param array $data

\* @return User

\*/

protected function create(array $data)

{

return User::create([

'name' => $data['name'],

'email' => $data['email'],

'username' => $data['username'],

'level'=>0,

'password' => bcrypt($data['password']),

]);

}

Chú thích: Ở đây mình đã khai báo validator thêm cho input username và insert vào username +level(mình không cho hiện ở register nên để mặc định = 0) vào table users.

### Model

Vì bạn có thêm 2 cột username và level vào bảng user nên mình sẽ phải báo cho model user (đườn dẫn : app/User.php) biết như sau:

<?php

namespace App;

use Illuminate\Notifications\Notifiable;

use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable

{

use Notifiable;

/\*\*

\* The attributes that are mass assignable.

\*

\* @var array

\*/

protected $fillable = [

'name', 'email', 'password','username','level',

];

/\*\*

\* The attributes that should be hidden for arrays.

\*

\* @var array

\*/

protected $hidden = [

'password', 'remember\_token',

];

}

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

## 3, Lời kết.

- Phần trên mình đã hướng dẫn mọi người custom lại auth register của Laravel rồi và phần trên chỉ là demo thêm 2 trường, còn các bạn muốn làm bao nhiêu trường cũng được.

- Phần tiếp theo mình sẽ hướng dẫn các bạn custom lại login của Laravel.

# **Bài 24: Auththentication trong Laravel (Login, Logout)**

Bài trước mình đã hướng dẫn mọi người custom lại auth register của Laravel rồi! Hôm nay mình sẽ hướng dẫn mọi người custom phần đăng nhập.

-Lưu ý: Mình vẫn sử dụng database của [bài trước](https://toidicode.com/authentication-trong-laravel-register-48.html) nhé!

## 1, Đăng nhập với Logincontroller của Laravel.

-Trong phần này mình sẽ hướng dẫn cụ thể mọi người đổi email đăng nhập thành **username.**

**-**Đầu tiên chúng ta cần chỉnh sửa lại trường đăng nhập trong giao diện login của Laravel (Đường dẫn: **resources/views/auth/login.blade.php**) như sau:

@extends('layouts.app')

@section('content')

<div class="container">

<div class="row">

<div class="col-md-8 col-md-offset-2">

<div class="panel panel-default">

<div class="panel-heading">Login</div>

<div class="panel-body">

<form class="form-horizontal" role="form" method="POST" action="{{ url('/login') }}">

{{ csrf\_field() }}

<div class="form-group{{ $errors->has('username') ? ' has-error' : '' }}">

<label for="email" class="col-md-4 control-label">Username</label>

<div class="col-md-6">

<input id="email" type="text" class="form-control" name="username" value="{{ old('username') }}" required autofocus>

@if ($errors->has('username'))

<span class="help-block">

<strong>{{ $errors->first('username') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">

<label for="password" class="col-md-4 control-label">Password</label>

<div class="col-md-6">

<input id="password" type="password" class="form-control" name="password" required>

@if ($errors->has('password'))

<span class="help-block">

<strong>{{ $errors->first('password') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group">

<div class="col-md-6 col-md-offset-4">

<div class="checkbox">

<label>

<input type="checkbox" name="remember" {{ old('remember') ? 'checked' : ''}}> Remember Me

</label>

</div>

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

<button type="submit" class="btn btn-primary">

Login

</button>

<a class="btn btn-link" href="{{ url('/password/reset') }}">

Forgot Your Password?

</a>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

@endsection

-Chú thích: Phần trên mình đã thay đổi trường email mặc định của Laravel thành username.

-Tiếp đó mọi người 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 'columnName';

}

-Chú thích **columnName:**Là tên cột bạn muốn sử dụng để đăng nhập.

**VD**: Mình để là đăng nhập bằng username thì sẽ như sau:

public function username()

{

return 'username';

}

-Với Phần cấu hình đăng nhập này chỉ như thế là đã thành công rồi.

## 2, Đăng nhập với attempt().

-Đây là một phần  mở rộng của Laravel giúp chúng ta có thể tạo ra các phiên đăng nhập tùy chỉnh.

-Mình vẫn sử dụng **login.blade.php** của Laravel nhưng sẽ sửa action của form thành:

<form class="form-horizontal" role="form" method="POST" action="{{ url('/logintest') }}">

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

Route::post('logintest','TestController@check');

-Và tạo thêm một TestController để xử lý đăng nhập:

<?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

}

}

}

**-Chú ý:**

* Trong controller nếu muốn sử dụng authentication thì phải khai báo sử dụng namespace của nó: use **Illuminate\Support\Facades\Auth;**
* Auth::attempt(): Đây là phương thức check input data có hợp lệ không, nếu hợp lệ sẽ trả về true và ngược lại. Hoàn toàn có thể thêm điều kiện VD:

$data = [

'username' => $request->username,

'password' => $request->password,

'level' => 1,

];

## 3, Logout Trong Laravel.

-Trong Laravel Để đăng xuất thì khá là đơn giản. Các bạn chỉ cần sử dụng phương thức:

Auth::logout();

## 4, Kiểm tra đăng nhập với check().

-Để kiểm tra xem user đã đăng nhập hay chưa thì Laravel có hỗ trợ chúng ta phương thức **check()** với cú pháp:

if (Auth::check()) {

// Đã đăng nhập.

} else {

//chưa đăng nhập.

}

- Phương  thức check() trả về giá trị true nếu đã đăng nhập và ngược lại.

## 5, Truy vấn thông tin của user đang đăng nhập.

-Để truy vấn các thông tin của người đang đăng nhập thì Laravel có hỗ trợ chúng ta phương thức **user()**,

$user = Auth::user();

**-**Phương thức này trả về giá trị là một **json,** và đương nhiên là các bạn có thể truy vấn vào các phần tử trong json.

VD:

Auth::user()->name;

-Và để lấy id của user đang đăng nhập thì dùng:

Auth::id();

## 4, Lời kết.

-Bài này mình đã hướng dẫn mọi người custom lại chức năng đăng nhập, truy vấn thông tin và đăng xuất trong authentication rồi. Qua bài này các bạn thấy được sức mạnh của Laravel thế nào rồi chứ? Nhưng chưa dừng ở đó Laravel vẫn còn nhiều cái hay lắm! (xem tiếp ở các phần sau nhé).

# **Bài 25: Middleware trong Laravel**

Font size

Tiếp tục với series [tự học Laravel](https://toidicode.com/series/hoc-laravel) của Toidicode.com, hôm nay mình sẽ giới thiệu với mọi người về Middleware trong Laravel và ở đây là phiên bản 5.3.

## 1, Middleware là gì?

-Middleware theo wikipedia:

***Middleware****là phần mềm máy tính với nhiệm vụ kết nối các thành phần phần mềm hoặc các ứng dụng với nhau. Phần mềm loại này bao gồm một tập các dịch vụ cho phép sự tương tác giữa các tiến trình chạy trên một hoặc nhiều máy khác nhau. Công nghệ middleware đã được phát triển để cung cấp khả năng hoạt động tương hỗ, phục vụ cho các kiến trúc phân tán thường được để hỗ trợ và đơn giản hóa các ứng dụng phân tán phức tạp.*

-Thì ở đây đối với Laravel nó cũng hoàn toàn là như thế.

## 2, Tạo Middleware trong Laravel.

-Để tạo một middleware, trong Laravel đã cung cấp cho chúng ta một lệnh:

php artisan make:middleware MiddelwareName

Trong đó: MiddlewareName là tên của middleware các bạn muốn Tạo.

VD:

php artisan make:middleware CheckLevel

-Nếu nhận được thông báo : **Middleware created successfully.**thì đã tạo thành công middleware.

-middleware mặc định sẽ phải nằm trong thư mục: **app/Http/Middleware**.

Tiếp đó các bạn mở file vừa tạo ra sẽ thấy có dạng:

<?php

namespace App\Http\Middleware;

use Closure;

class CheckLevel

{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle($request, Closure $next)

{

return $next($request);

}

}

## 3, Sử dụng Middleware.

-Mọi xử lý trong middleware đều nằm trong handle function. Mọi người xem qua ví dụ cho dễ hiểu nhé:

VD: Mình muốn user vừa đăng nhập phải có level >1 mới được vào admin:

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Support\Facades\Auth;

class CheckLevel

{

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle($request, Closure $next)

{

if(Auth::check()&&Auth::user()->level>1)

{

return $next($request);

}else{

return redirect('/');

}

}

}

Xem VD: trên chắc mọi người cũng hiểu về cách hoạt động của nó rồi nhỉ!.

## 4, Đăng ký middleware.

-Các phần trên mới chỉ là tạo ra middleware và viết code cho nó nhưng chưa sử dụng được, giờ nếu muốn sử dụng được middleware trong project thì chúng ta cần phải đăng ký middleware bằng các kiểu middleware sau đây:

### Global middleware.

-Global middleware là một middel 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: **app/Http/Kernel.php**tìm đến đoạn:

/\*\*

\* The application's global HTTP middleware stack.

\*

\* These middleware are run during every request to your application.

\*

\* @var array

\*/

protected $middleware = [

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,

];

Đăng ký với cú pháp:

\App\Http\Middleware\MiddlewareName::class

Trong đó : MiddlewareName là tên middleware các bạn muốn thêm.

VD: Mình muốn thêm middleware CheckLevel ở VD trên:

protected $middleware = [

\Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,

\App\Http\Middleware\CheckLevel::class,

];

### Route middleware.

-Khác với global middleware Route middleware chỉ sử dụng được khi bạn gọi nó ở trong Route.

Để đăng ký Route middleware thì mọi người cũng cần phải vào app/Http/Kernel.php tìm đến đoạn:

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,

];

Đăng ký với cú pháp:

'Name'=> \App\Http\Middleware\MiddlewareName::class

**Trong đó:**

* **Name:**Là tên các bạn muốn đặt cho middleware.
* MiddlewareName là tên của Middleware các bạn muốn thêm.

VD: Mình sẽ thêm middleware CheckLevel vào Route middleware.

'Checklevel'=>\App\Http\Middleware\CheckLevel::class,

Sau khi đăng ký xong middleware các bạn có thể sử dụng nó với route như sau:

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

//

})->middleware('Checklevel');

Hoặc với nhiều middleware:

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

//

})->middleware('Middleware1','Middleware2');

Hoặc cũng có thể sử dụng tên đầy đủ của Middleware như sau:

use App\Http\Middleware\CheckLevel;

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

//

})->middleware(CheckLevel::class);

### Nhóm middleware.

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

Để khai báo nhóm middleware chúng ta tìm đến đoạn:

protected $middlewareGroups = [

'web' => [

\App\Http\Middleware\EncryptCookies::class,

\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

\Illuminate\Session\Middleware\StartSession::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 sử dụng trong Route với cú pháp:

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

//

})->middleware('web');

Hoặc

Route::group(['middleware' => ['web']], function () {

//

});

## 5, Lời kết.

- Qua bài này mình đã hướng dẫn mọi người làm việc với middleware trong Laravel. Tuy nhiên nó còn có 2 loại middleware nữa mình không giới thiệu ở đây vì rất ít khi sử dụng, ai quan tâm có thể xem [ở đây](https://laravel.com/docs/5.3/middleware#middleware-parameters).

# **Bài 26: Hướng dẫn tích hợp và sử dụng laravel debugbar package**

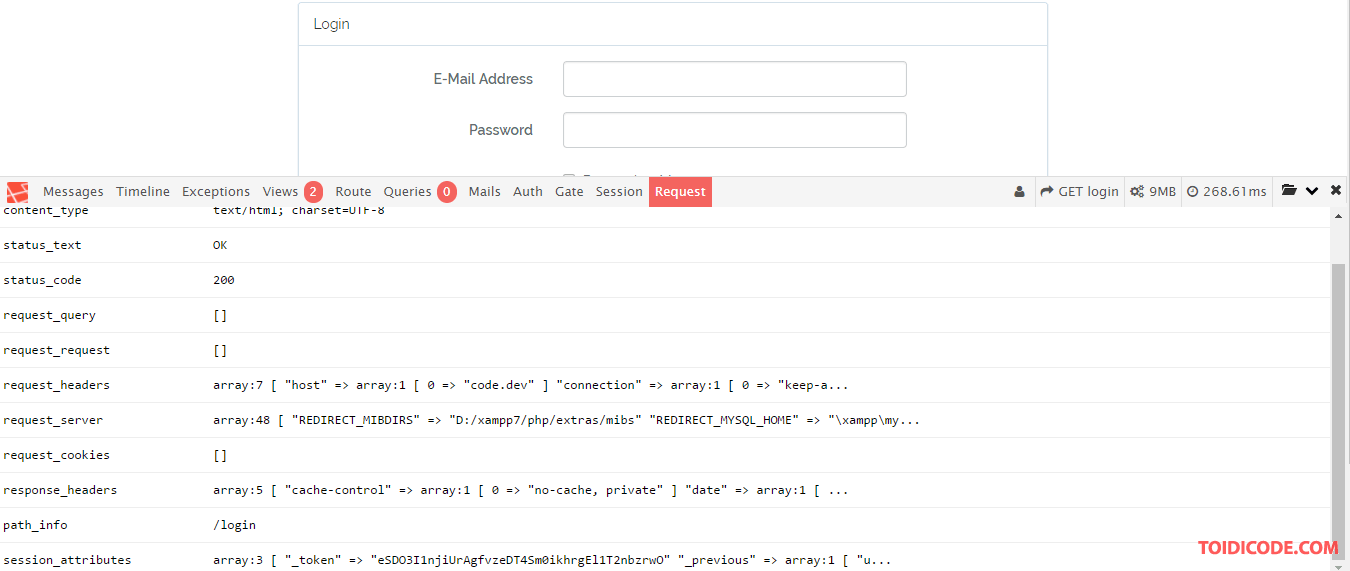
Sau một thời gian dài bỏ qua series này, hôm nay chúng ta sẽ tiếp tục quay lại với series Laravel và tìm hiểu về **package laravel debugbar**...

## 1, Laravel debugbar là gì?

-Laravel debugbar là một gói package được viết ra bởi [Barry vd. Heuvel](https://github.com/barryvdh), với mục đích giúp cho các nhà phát triển trên laravel có thể debug 1 cách nhanh chóng và dễ dàng.

-Package này hỗ trợ chúng ta một số các tính năng nổi bật như:

* Hiện tất cả các câu truy vấn khi load 1 trang đến database.
* Hiện ra tất cả các view cấu thành lên trang hiện tại.
* Hiện thông tin của route hiện tại.
* Hiển thị tất cả các sự kiện trên trang hiện tại.
* Hiển thị toàn bộ các session đang được lưu trữ.
* Hiển thị thông tin của request.
* Hiển thị mức tiêu thụ RAM của ứng dụng.
* ... và còn nhiều chức năng khác nữa...



giao diện của laravel debugbar

## 2, Tích hợp Laravel debugbar vào project.

-Để có thể tích hợp được laravel debugbar bào ứng dụng thì việc đầu tiên của bạn là phải có bộ source của laravel.

-Tiếp đó bạn sử dụng cmd (trên window) hoặc terminal (trên linux) để di chuyển đến project laravel của bạn và chạy lệnh

composer require barryvdh/laravel-debugbar

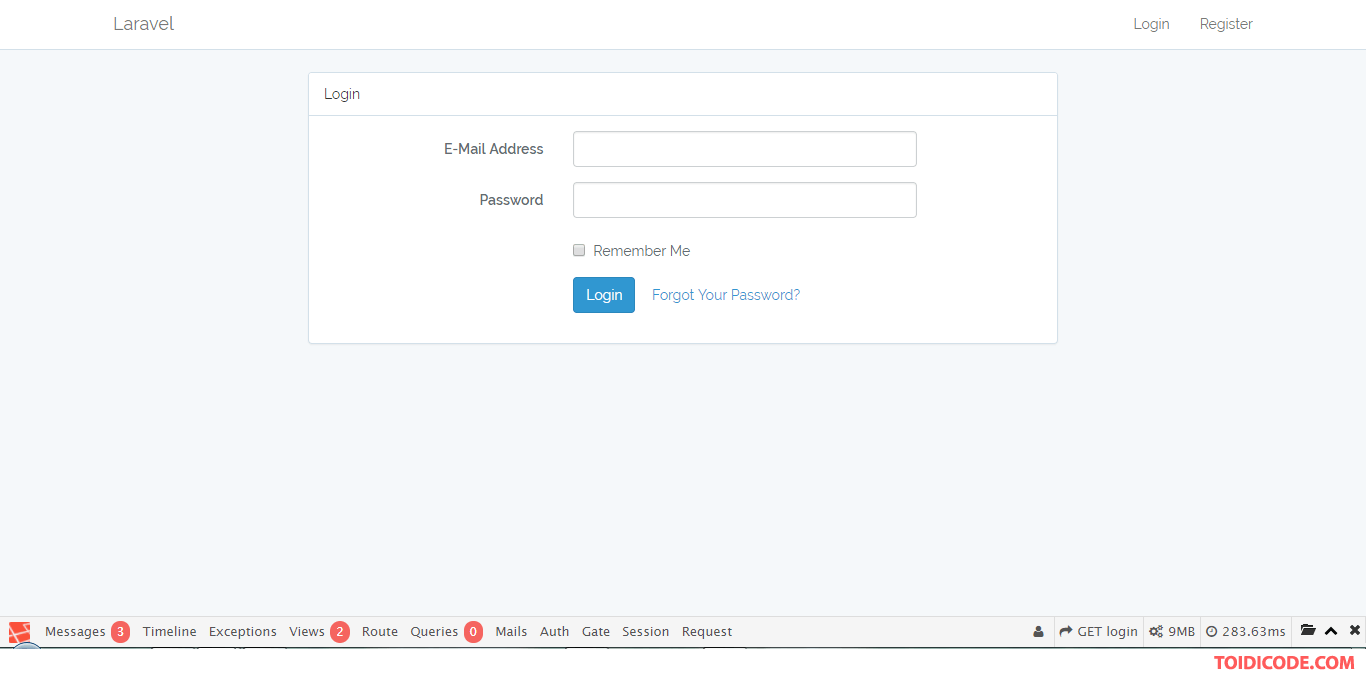
Sau khi đã tải thành công package, thì các bạn vào project laravel của mình và truy cập config/app.php tìm đến key providers và thêm đoạn code sau vào.

Barryvdh\Debugbar\ServiceProvider::class,

-Tiếp đó, nếu như bạn muốn sử dụng **facade** của laravel debugbar thì bạn cần thêm đoạn code sau vào aliases trong file config/app.php.

'Debugbar' => Barryvdh\Debugbar\Facade::class,

-Lúc này bạn quay lại ứng dụng và chạy lại, nếu như ứng dụng không báo lỗi gì và ở phía dưới trang web có một thanh bar dạng như sau thì bạn đã cài đặt thành công.



## 3, Sử dụng laravel debugbar.

-Sau khi đã cài đặt được laravel debugbar rồi thì những chức năng đã được tích hợp sẵn trên đó nhìn vào rất dễ hiểu rồi nên mình không trình bày nữa.

-Và nếu như ở phần trên bạn có tích hợp facede của laravel debugbar thì bạn sẽ sử dụng được các đoạn code dưới đây.

+ Thêm messages:

//Thêm message info

Debugbar::info($object);

//Thêm message error

Debugbar::error('Error!');

//Thêm message warning

Debugbar::warning('Watch out…');

//Thêm message value and name

Debugbar::addMessage('Another message', 'mylabel');

+ Thêm Timeline:

Timeline là công cụ giúp cho bạn có thể check xem trang hiện tại load trong vòng bao nhiêu lâu. Dựa vào đó bạn có thể phân tích xem trên trang có những đoạn code nào gây tốn thời gian nhất và khắc phục.

//Bắt đầu

Debugbar::startMeasure('render','Time for rendering');

//Kết thúc

Debugbar::stopMeasure('render');

Debugbar::addMeasure('now', LARAVEL\_START, microtime(true));

Debugbar::measure('My long operation', function() {

// Do something…

});

 Và còn nhiều chức năng hữu ích khác nữa. Bạn nào muốn tìm hiểu thêm có thể tham khảo tại [đây](https://github.com/barryvdh/laravel-debugbar).

## 3, Lời kết.

-Debugbar quả là một package hữu ích. Nhưng bạn hãy nhớ tắt nó đi khi ứng dụng của bạn đã ở chế độ **production**.

# **Bài 27: Bảy directives hữu dụng trong blade template có thể bạn chưa biết?**

Sau một thời gian im hơi lặng bóng với series [PHP Laravel Framwork](https://toidicode.com/hoc-laravel), thì ngày hôm nay mình đã quay trở lại và giới thiệu với mọi người một số directives hữu dụng trong Blade template mà có thể bạn chưa biết

*Chú ý các directive dưới đây chỉ dùng được cho phiên bản Laravel 5.5 trở lên thôi, riêng phần 6 và 7 chỉ Laravel 5.6 trở lên mới có thể sử dụng.*

## 1,  @includeFirst().

Nếu như trước đây bạn cần phải viết một đoạn code khá là dài để kiểm tra view có tồn tại hay không thì mới include và ngược lại...

**VD1**:

@if(view()->exists('view1'))

@include('view1')

@else

@include('view2')

@endif

Thì thay vì viết dài như thế bạn có thể sử dụng directive @includeFirst() cho ngắn gọn với cú pháp như sau:

@includeFirst(['view1, view2, ..., viewN'], data)

**Trong đó**:

* view1, view2, ... là các view mà chúng ta muốn include. Blade sẽ chỉ include duy nhất một view đúng đầu tiên mà nó check được.
* data là một mảng chứa dữ liệu truyền vào view.

**VD2**: Rút ngắn ví dụ 1 với @includeFirst()

@includeFirst(['view1', 'view2']);

## 2, @includeIf()

Nếu bạn muốn kiểm tra view tồn tại thì mới include thì hãy sử dụng @includeIf() thay vì phải dùng if directive để check.

**VD3**:

@includeIf('view', ['key' => 'data'])

## 3, @includeWhen()

Nếu như bạn muốn có một tham số để check xem view này có nên được include hay không thì @includeWhen() sẽ giúp bạn.

**VD4**:

@include($flag, 'view', ['some' => 'data'])

Trong ví dụ trên nếu như $flag trả về true thì nó mới include view.

## 4, @auth.

Thông thường để check auth trong blade template thì chúng ta thường sử auth() helper để check.

**VD5**:

@if(auth()->user())

// The user is authenticated.

@endif

Nhưng nếu bạn muốn ngắn gọn và minh bạch hơn thì có thể sử dụng directive @auth trong blade template.

**VD5**:

@auth

// The user is authenticated.

@endauth

## 5, @guest.

Tương tự như @auth() directive thì đối với guest blade cũng cung cấp cho chúng ta directive @guest.

**VD6**:

// normal

@if(auth()->guest())

// The user is not authenticated.

@endif

// shortcode

@guest

// The user is not authenticated.

@endguest

Và nó cũng có cơ chế else

@guest

// The user is not authenticated.

@else

// The user is authenticated.

@endguest

## 6, @csrf.

Thay vì phải dùng:

{{ csrf\_field() }}

như cũ thì bạn có thể chuyển đổi nó về dạng directive cho ngắn gọn như sau:

@csrf

## 7, @method().

Tương tự bạn cũng có thể thay thế:

{{ method\_field('DELETE') }}

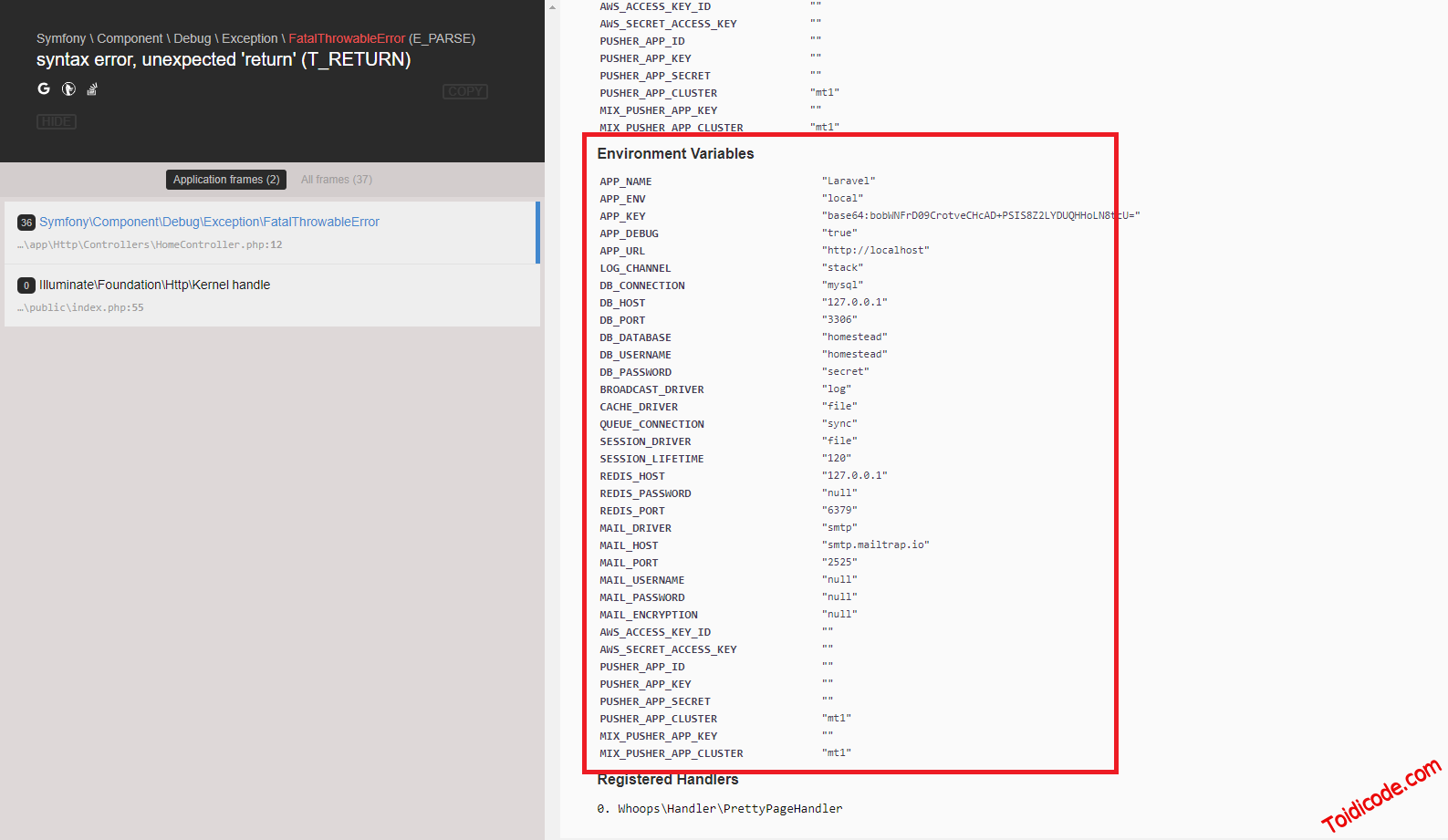
về dạng directive như sau:

@method('DELETE')

# 

# **Bài 28: Ẩn config quan trọng trong chế độ debug Laravel**

Như các bạn đã biết thì trong Laravel có cung cấp cho chúng ta xác định các enviroment (môi trường) mà các chế độ liên quan đến debugging. Và nếu như chúng ta mà bật chế độ debug thì nó Laravel sẽ report lỗi là gì, ở đâu, thông tin request, env,...



Và như các bạn đã thấy thì các thông tin như APP\_KEY, DATABASE,... đều hiện lên. Như thế này thì trong một số trường hợp sẽ hoàn toàn không tốt.

VD như môi trường staging của dự án vẫn cần phải bật debug để test có lỗi còn biết luôn lỗi là gì ở đâu,... Nhưng lại cần ẩn một số thông tin như DB config,... thì lúc này quả thật rất cần thiết.

Và**kể từ Laravel 5.5 chúng ta có thể ẩn những thông tin đó đi được** và config rất đơn giản.

**1. Ẩn Config trong debug mode.**

Để cấu hình ẩn thông tin không muốn hiển thị ra màn hình report bạn chỉ cần thêm vào config/app.php một block có key là debug\_blacklist và giá trị là array chứa các key muốn ẩn

VD: Mình muốn ẩn APP\_KEY với DATABASE\_PASSWORD config thì mình sẽ cấu hình như sau:

'debug\_blacklist' => [

'\_ENV' => [

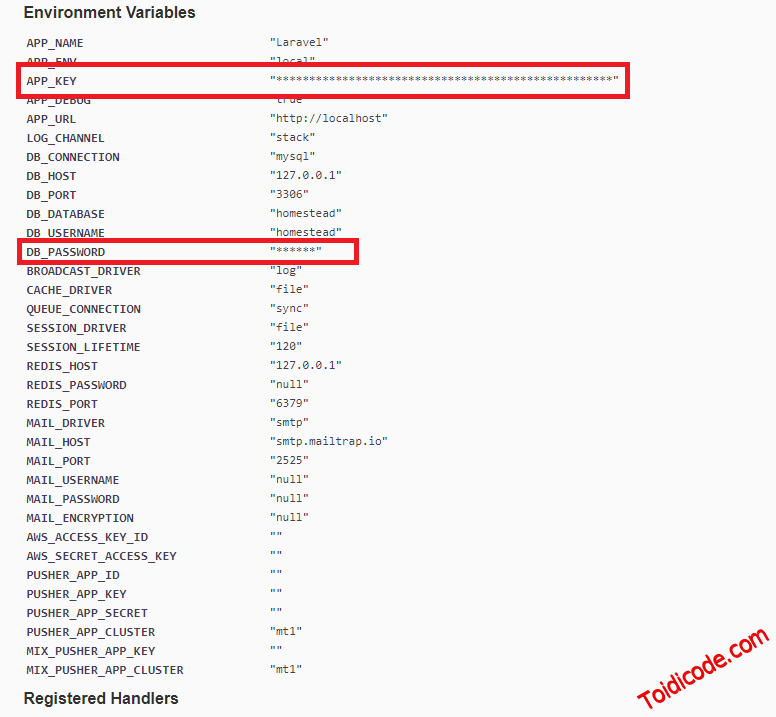
'APP\_KEY',

'DB\_PASSWORD',

]

]

Thì lúc này màn hình report lỗi sẽ tự động convert giá trị của 2 env kia thành **"\*\*\*"**.



Và đây là config mình hay dùng để ẩn những thông tin cơ bản của laravel.

'debug\_blacklist' => [

'\_ENV' => [

'APP\_KEY',

'DB\_PASSWORD',

'REDIS\_PASSWORD',

'MAIL\_PASSWORD',

'PUSHER\_APP\_KEY',

'PUSHER\_APP\_SECRET',

],

'\_SERVER' => [

'APP\_KEY',

'DB\_PASSWORD',

'REDIS\_PASSWORD',

'MAIL\_PASSWORD',

'PUSHER\_APP\_KEY',

'PUSHER\_APP\_SECRET',

],

'\_POST' => [

'password',

],

],

Nếu như các bạn cần ẩn thêm gì thì cứ thêm vào nhé.

**2. Lời kết.**

Đây chỉ là một việc rất nhỏ nhưng trong một số trường hợp nó lại rất quan trọng đó :D. Và cuối cùng hãy nhớ là luôn luôn tắt debug mode trông môi trường production nhé!

**Giáo trình tham khảo nội dung trên trang https://toidicode.com/**