# Blade Templates trong Laravel

# -Hiện nay cũng có khá nhiều các PHP framework nổi tiếng như [Codeigniter](http://toidicode.com/codeiginter-16),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](http://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 timhieu.blade.php trong thư mục resources/views

<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 bạn đến với nhóm của chúng tôi</h1>

</body>

</html>

## 2, Sử dụng blade template.

-Trong [PHP](http://toidicode.com/danh-muc/php-4) 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 {{}}.

{{ ‘ Biến ’ }}

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

{{ $a=’<b>hello</b>’ }}

🡪KQ:

<b>hello</b>

#### -Sử dụng

{ !! ‘Biến’ !! }

Cũng với ví dụ như trên là {{ $a=’<b>hello</b>’ }}

🡪KQ:

hello

### 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 love 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!

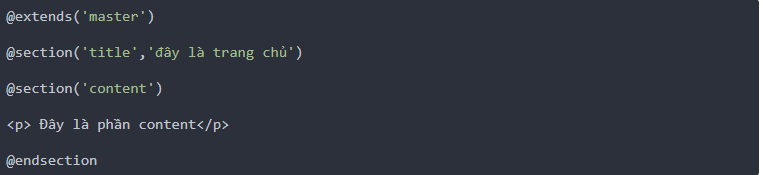
@else

I don’t have any records!

@endif

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

VD1: -Đầu tiên mình sẽ tạo ra 1 blade template có tên **master.blade.php** ở đường dẫn resoures\views\master.blade.php.

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

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

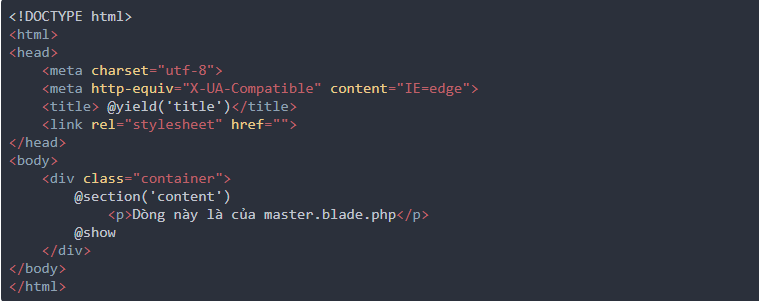
**=>Chú thích:**

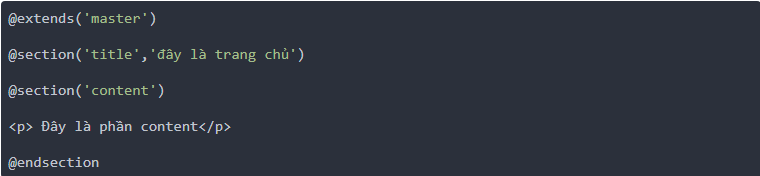
-Như ở trên mình có sử dụng các từ khóa lạ như @yield(),@section(),@extends().

-**@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

VD 2: -Mình sẽ thay đổi code trong file master ở VD1 một chút.

Và file home.php

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

+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.

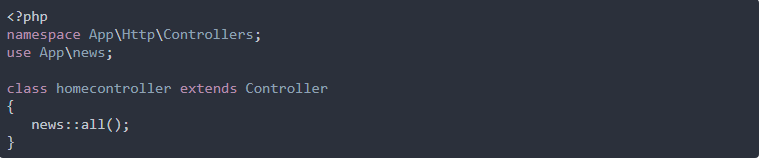
->Chú ý: @Parent có thể đặt trước nội dung muốn thêm vào (nếu muốn phần được thêm sẽ ở sau phần khởi tạo) và ngược lại.

## Eloquent ORM trong Laravel

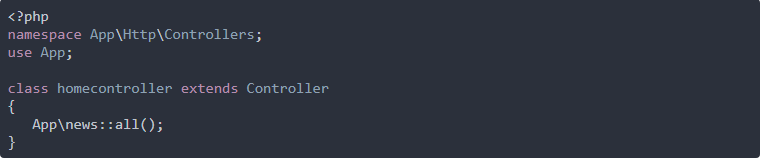
Eloquent ORM lầm việc trên cơ sở ActiveRecord trong CSDL của bạn. Mỗi bảng CSDL sẽ được ánh xạ qua “Model”, và model này được sử dụng để tương tác với bảng.

## ****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**

**Hoặc**



-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 lấy News model Làm mẫu

### 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](http://toidicode.com/query-builder-trong-laravel-11.html" \l "load-h3-2))

**-Chọn cột dữ liệu.**

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

**-Đếm dữ liệu.**

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

### Thêm dữ liệu.

-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.



**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();

.

# Tìm hiểu về pagination trong laravel

# 1.Giới thiệu

# Trong Laravel pagination kết hợp với ‘query builder’ và ‘eloquent ORM’ và cung cấp phân trang của dữ liệu trong database một cách thuận tiện. Laravel có thể nhanh chóng tạo một khoảng thông minh của các links dựa trên trang hiện tại và mã HTML sinh ra thì tương thích với ‘Bootstrap CSS framework’.

2.Sử dụng cơ bản

Phân trang kết quả từ Query Builder  
Có vài cách để phân trang. Đơn giản nhất là sử dụng hàm paginate ở trong query builder hoặc trong Eloquent query. Hàm paginate cung cấp bởi Laravel sẽ tự động xử lý việc tạo ra limit và vị trí trang dựa trên trang hiện tại đang được xem bởi người dùng. Mặc định, trang hiện tại được nhận biết thông qua giá trị ?page trên query string trên HTTP request. Dĩ nhiên là giá trị này được tự động nhận biết bởi Laravel, và cũng được tự động thêm vào các link sinh ra bởi paginator.

Trước tiên, hãy cùng nhau xem việc gọi hàm paginate trên một query. Trong ví dụ này, đối số duy nhất tryền vào hàm paginate là số items bạn muốn hiển thị trên từng page. Ở đây, chúng ta ví dụ đặt chỉ số là 15 item trên một page:

<?php

namespace App\Http\Controllers;

use DB;

use App\Http\Controllers\Controller;

class UserController extends Controller

{

*/\*\**

*\* Show all of the users for the application.*

*\**

*\* @return Response*

*\*/*

public function index()

{

$users = DB::table('users')->paginate(15);

return view('user.index', ['users' => $users]);

}

}

Chú ý: Hiện tại, việc phân trang sử dụng groupBy chưa thể thực thi hiệu quả bởi Laravel. Nếu bạn cần sử dụng groupBy với một tập kết quả phân trang, thì khuyến khích các bạn thực hiện query database và tạo một paginatorthủ công.

* **Phân trang đơn giản**

Nếu bạn chỉ cần hiển thị hai link đơn giản "Next" và "Previous" trên pagination view, bạn có thể sử dụng hàm simplePaginate để thực hiện một query hiệu quả hơn. Cách này rất hữu dụng với một tập dữ liệu lớn nếu bạn không cần hiển thị một link cho mỗi số trang khi thực hiện render:

$users = DB::table('users')->simplePaginate(15);

* **Phân trang kết quả từ Eloquent**

Bạn cũng có thể phân trang kết quả từ Eloquent. Trong ví dụ này, chúng ta sẽ phân trang User model với 15 item một trang. Như bạn thấy, cú pháp gần như giống hệt với phân trang kết quả từ query builder:

$users = App\User::paginate(15);

Bạn cũng có thể gọi paginate sau khi thiết lập rằng buộc trên query, ví dụ như mệnh đề where:

$users = User::where('votes', '>', 100)->paginate(15);

Bạn cũng có thể sử dụng simplePaginate khi phân trang với Eloquent:

$users = User::where('votes', '>', 100)->simplePaginate(15);

* **Tạo thủ công một Paginator**

Đôi khi bạn muốn tạo một đối tượng xử lý phân trang riêng, truyền vào cho nó một mảng các items. Bạn có thể thực hiện bằng cách tạo một đối tượng từ Illuminate\Pagination\Paginator hoặc từ Illuminate\Pagination\LengthAwarePaginator, phục thuộc vào yêu cầu của bạn.

Class Paginator không quan tâm tổng số items trên tập kết quả; tuy nhiên, chính vì thế mà class không có phương thức để lấy được index của trang cuối cùng. Class LengthAwarePaginator nhận đối số tương tự với Paginator, nhưng lại cần biết tổng số items có trong tập kết quả.

Nói một cách khác, Paginator tương ứng với hàm simplePaginate trên query builder và Eloquent, trong khi LengthAwarePaginator lại tương ứng với hàm paginate.

Khi tự tạo một đối tượng paginator thủ công, bạn nên tự "cắt" mảng của tập kết quả truyền vào cho paginator. Nếu bạn không chắc làm như thế nào, hãy tham khảo hàm array\_slice của PHP.

* **Hiển thị kết quả lên views**

Khi bạn gọi hàm paginate hay simplePaginate trên query builder hay Eloquent, bạn sẽ nhận được một đối tượng paginator. Khi gọi hàm paginate, bạn sẽ nhận được một đối tượng của Illuminate\Pagination\LengthAwarePaginator. Khi gọi hàm simplePaginate, bạn sẽ nhận được một đối tượng của Illuminate\Pagination\Paginator. Những đối tượng này cung cấp vài phương thức mô tả tập kết quả. Ngoài những phương thức này, các đối tượng paginator đều là các iterators và có thể được lặp như một mảng.

Khi đã nhận được kết quả, bạn có thể thực hiện hiển thị kết quả và render các link vào page sử dụng Blade:

<div class="container">

@foreach ($users as $user)

{{ $user->name }}

@endforeach

</div>

{!! $users->links() !!}

Hàm links sẽ render các link cho tới hết các trang trong tập kết quả. Mỗi link này đều chứa sẵn một tham số ?page với giá trị đúng. Hãy nhớ là, mã HTML sinh ra bởi hàm links tương thích với Bootstrap CSS framework.

* **Tuỳ chọn The Paginator URI**  
  Hàm setPath cho phép bạn tuỳ chọn URI sử dụng bởi paginator khi sinh ra links. Ví dụ, nếu bạn muốn paginator sinh ra links theo kiểu này <http://example.com/custom/url?page=N>, bạn chỉ cần truyền vào custom/url vào hàm setPath:

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

$users = App\User::paginate(15);

$users->setPath('custom/url');

*//*

});

* **Thêm vào link phân trang**

Bạn có thể thêm vào query string của link phân trang sử dụng hàm appends. Ví dụ, để thêm vào &sort=votes vào mỗi link, bạn nên thực hiện thế này:

{!! $users->fragment('foo')->links() !!}

-\*\* Các phương thức helper bổ sung\*\*

Bạn có thể truy cập các thông tin khác trong phân trang thông qua các phương thức sau trong paginator:

$results->count()

$results->currentPage()

$results->firstItem()

$results->hasMorePages()

$results->lastItem()

$results->lastPage() (Not available when using simplePaginate)

$results->nextPageUrl()

$results->perPage()

$results->previousPageUrl()

$results->total() (Not available when using simplePaginate)

$results->url($page)

* **Chuyển kết quả sang JSON**

Các lớp kết quả phân trang của Laravel triển khai từ contract Illuminate\Contracts\Support\JsonableInterface và mở ra hàm toJson, do đó, rất dễ dàng để có thể chuyển kết quả thành JSON.

Bạn cũng có thể convert một đối tượng paginator sang JSON bằng cách return nó từ một route hay controller action:

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

return App\User::paginate();

});

JSON tạo ra từ paginator sẽ chứa các thông tin meta như total, current\_page, last\_page, và nhiều nữa. Các đối tượng kết quả đều có trong khoá data của mảng JSON. Đây là một ví dụ về JSON tạo bởi paginator từ một route:

{

"total": 50,

"per\_page": 15,

"current\_page": 1,

"last\_page": 4,

"next\_page\_url": "http://laravel.app?page=2",

"prev\_page\_url": null,

"from": 1,

"to": 15,

"data":[

{

*// Result Object*

},

{

*// Result Object*

}

]

}

# Migrations trong Laravel

# -Bạn có thể hiểu Migrations trong Laravel 5 giống như một version control của database, 1 tính năng giúp bạn quản lý các version của database. Migrations trong Laravel 5 cho phép bạn chỉnh sửa scheme database, cập nhật trang thái hiện tại của database. Migrations giúp cho việc quản lý database của bạn trở nên đễ 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**

## Tạo migrations.

# Tạo Migrations bằng lệnh thì các bạn mở cmd lên và trỏ tới thư mục chứa project  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.

## 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. |

## 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 |

# Seeding trong Laravel

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

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

## 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.

### 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).