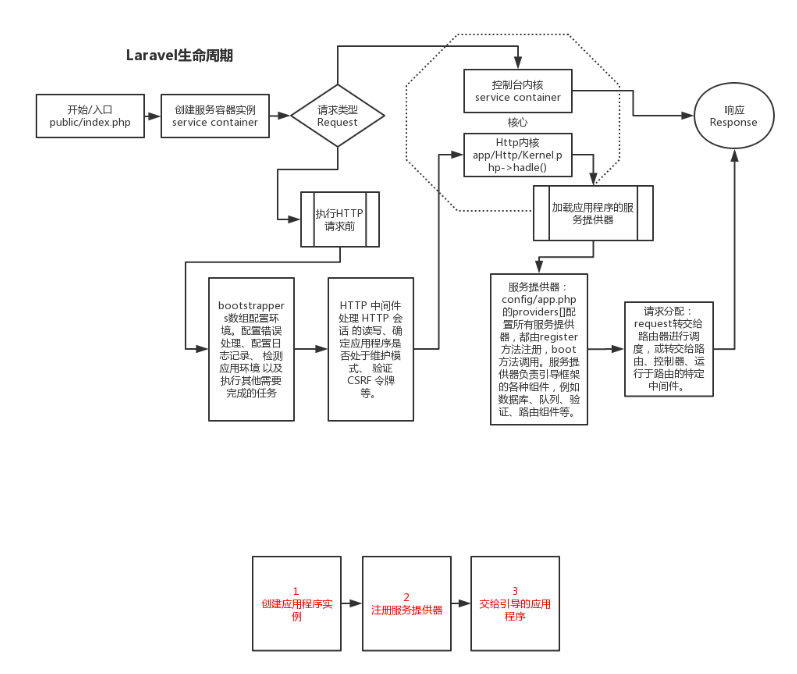
起點:

* 所有Laravel的請求入口都是從public/index.php，此檔案室加載框架的部分起點。
* index.php文件加載Composer生成的自動加載設置，再從bootstrap/app.php腳本中檢索Laravel應用程序的實例。

服務提供者:

* 所有應用下的服務提供者均配置到了config/app.php配置文件中的providers數組中。第一步，所有服務提供者的register方法會被調用，然後一旦所有服務提供者均註冊後，boot方法才被調用。



類、接口、對象:

類、對象:

class Power {

//protected為設定變數只能被此類的函數調用外部不可直接使用

//能力值

protected $ability;

//能力范围或距离

protected $range;

//建構子在創立一個Power的對象時會先執行

public function \_\_construct($ability, $range)

{

$this->ability = $ability; //this指的是這個類

$this->range = $range;

}

}

* 有一個類叫超能力

class Superman

{

protected $power; //有一個power

public function \_\_construct()

{

$this->power = new Power(999, 100);//為power創建一個超能力實體

}

}

* 假設有一個類叫超人他有一超能力叫power，這樣的話，當我們創建一個“超人”實例的時候，同時也創建了一個“超能力”的實例，但是，我們看到了一點，“超人”和“超能力”之間不可避免的產生了一個依賴。

***Note:*** **所謂“依賴”，就是“我若依賴你，少了你就沒有我”。**

* 我們不應該手動在“超人”類中固化了他的“超能力”初始化的行為，而轉由外部負責，由外部創造超能力模組、裝置或者芯片等（我們後面統一稱為“模組” ），植入超人體內的某一個接口，這個接口是一個既定的，只要這個“模組”滿足這個接口的裝置都可以被超人所利用，可以提升、增加超人的某一種能力。這種由外部負責其依賴需求的行為，我們可以稱其為“ 控制反转（IoC）”。

class Flight

{

protected $speed;

protected $holdtime;

public function \_\_construct($speed, $holdtime) {

$this-> speed = $ speed; //this指的是這個類

$this-> holdtime = $ holdtime;

}

}

class Force

{

protected $force;

public function \_\_construct($force) {

$this-> force = $ force;

}

}

class Shot

{

protected $atk;

protected $range;

protected $limit;

public function \_\_construct($atk, $range, $limit) {

$this-> atk = $ atk;

$this-> range = $ range;

$this-> limit = $ limit;

}

}

* 創建好個超能力的類，提供給工廠集合管理

class SuperModuleFactory

{

public function makeModule($moduleName, $options)

{

switch ($moduleName) {

case 'Fight': return new Fight($options[0], $options[1]);

case 'Force': return new Force($options[0]);

case 'Shot': return new Shot($options[0], $options[1], $options[2]);

}

}

}

* 創建一個超能力類統合工廠把所有超能力放進去用選擇的方式給超人選，這種方式又稱”工廠模式”

class Superman

{

protected $power;

public function \_\_construct()

{

// 初始化工廠

$factory = new SuperModuleFactory;

// 通过工廠提供的方法制造需要的模块(單個)

$this->power = $factory->makeModule('Fight', [9, 100]);

/\* 多個

$this->power = array(

$factory->makeModule('Force', [45]),

$factory->makeModule('Shot', [99, 50, 2])

);

\*/

}

}

* 超人類可以使用工廠直接使用多種超能力

class Superman

{

protected $power;

public function \_\_construct(array $modules)

{

// 初始化工厂

$factory = new SuperModuleFactory;

// 通过工厂提供的方法制造需要的模块

foreach ($modules as $moduleName => $moduleOptions) {

$this->power[] = $factory->makeModule($moduleName, $moduleOptions);

}

}

}

// 创建超人

$superman = new Superman([

'Fight' => [9, 100],

'Shot' => [99, 50, 2]

]);

* 更簡潔的超人寫法

接口:

* 因為工廠模式可能壟斷更新，導致新增不夠靈活，但如果讓多方都能創造模組的話會導致沒有統一，所以要使用依賴注入，此方法就是定義接口，讓創造模組的人才們遵循，否則無法使用，這樣自然可以讓所有模組可以正常使用。

interface SuperModuleInterface

{

/\*

\* 任何一个超能力都得有该方法，并拥有一个参数

\*/

public function activate(array $target);

}

class UltraBomb implements SuperModuleInterface

{

public function activate(array $target)

{

protected $atk;

public function \_\_construct($atk) {

$this-> atk = $ atk;

}

}

* 一個超能力implements接口，然後使用接口的函式名稱再重新定義一個函式蓋過去。

class Superman

{

protected $module;

public function \_\_construct(SuperModuleInterface $module)

{

$this->module = $module

}

}

* 從新定義一個超人的類，超人接收一個接口，接收符合接口的超能力。

// 超能力模组

$superModule = new UltraBomb;

$superModule->activate(50);

// 初始化一个超人，并注入一个超能力模组依赖

$superMan = new Superman($superModule);

路由:

基本路由:

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

return 'Hello World';

});

* 最基本的路由由一個URL跟閉包所形成

Route::get('/user', 'UserController@index');

* 和Controller結合的路由，第一個值為URL第二個值為對應的Controller，上面第二個值的意思是，對應到UserController裡的Index function

Route::get($uri, $callback);

Route::post($uri, $callback);

Route::put($uri, $callback);

Route::patch($uri, $callback);

Route::delete($uri, $callback);

Route::options($uri, $callback);

* 路由單一響應的請由方法

Route::match(['get', 'post'], '/', function () { });

Route::any('foo', function () { });

* 路由複合響應請求方法

CSRF保護:

<form method="POST" action="/profile">

@csrf

</form>

* Laravel中有保護機制，POST、PUT 或 DELETE 路由的任何HTML表單都應該包含一個CSRF令牌字段，否則，這個請求將會被拒絕。

重定向路由:

Route::redirect('/here', '/there', 301);

* 如果here碰到error 301則將here導向there

視圖路由:

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);

* 如果你的路由只需要返回一個視圖，其中前兩個參數，分別是URI和視圖名稱。第三個參數選填，傳入一個數組，數組中的數據會被傳遞給視圖

路由參數:

Route::get('posts/{post}/{comment}',function($postId, $commentId){ });

* $postId會自動去跟{post}匹配，$commentId會自動去跟{comment }匹配，不一定名稱要一樣，URL第一個{}會自動跟function第一個變數匹配後面以此類推

可選參數:

Route::get('user/{name?}', function ($name = null) {

return $name;

});

* 可能需要指定一個路由參數，但你希望這個參數是可選的。你可以在參數後面加上 ? 標記來實現，但前提是要確保路由的相應變量有默認值

正則表達式約束:

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

//

})->where('name', '[A-Za-z]+');

* 約束{name}只能為大小寫A~Z

全局約束:

public function boot()

{

Route::pattern('id', '[0-9]+');

parent::boot();

}

* 如果你希望某個具體的路由參數都遵循同一個正則表達式的約束，就使用 pattern 方法在 RouteServiceProvider 的 boot 方法中定義這些模式，上述設定之後所有路由的id都必須為0~9

路由命名:

Route::get('user/profile', function () {. . .})->name('profile');

* 命名此路由為profile

// 生成 URL...

$url = route('profile');

// 生成重定向...

return redirect()->route('profile');

* 為路由指定了名稱後，就可以使用全局輔助函數 route 來生成鏈接或者重定向到該路由

Route::get('user/{id}/profile', function ($id) {...})->name('profile');

$url = route('profile', ['id' => 1]);

* 如果是有定義參數的命名路由，可以把參數作為 route 函數的第二個參數傳入，指定的參數將會自動插入到URL中對應的位置

Route::middleware(['first', 'second'])->group(function () {

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

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

});

* 要給路由組中所有的路由分配中間件，可以在group之前調用middleware方法，中間件會依照它們在數組中列出的順序來運行

隱性路由綁定:

Route::get('api/users/{user}', function (App\User $user) {

return $user->email;

});

* 若是用model來匹配{user}的話名稱要相同，則Laravel 會自動解析定義在路由或控制器行為中與類型提示的變量名匹配的路由段名稱的Eloquent 模型，在這個例子中，由於$user變量被類型提示為Eloquent模型App\User，變量名稱又與URI中的{user}匹配，因此，Laravel會自動注入與請求URI中傳入的ID匹配的用戶模型實例。如果在數據庫中找不到對應的模型實例，將會自動生成404異常。

中間件:

介紹:

* 中間件為過濾進入應用程序的HTTP請求提供了一種方便的機制，最主要還是用來做身分驗證。

創建:

* 在終端機輸入php artisan make:middleware “中間件名稱” ，該命令會在app/Http/Middleware 目錄下創建一個新的中間件

public function handle($request, Closure $next)

{

if ($request->age <= 200) {

return redirect('home');

}

return $next($request);

}

* 上述程式用handle的function它會自動蓋過舊的，然後判斷request的值是否正確在決定下一步，return $next($request);就是指通過過濾，繼續往下執行。

註冊:

* 中間件創立好之後必須到app/Http/Kernel.php中注冊，middleware註冊會套用到所有request，routeMiddleware註冊必須自己去路由定義，middlewareGroups註冊把多個middleware定義成單個名稱。

使用:

Route::get('/', function () {...})->middleware('first', 'second');

* 單個路由在後面加上middleware套用中間件

Route::middleware(['first', 'second'])->group(function () {

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

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

});

* 多個路由可在前面用group套用中間件。

CSRF:

介紹:

* Laravel可以輕鬆地保護你的應用程序免受「cross-site request forgery」（CSRF）攻擊，跨站點請求偽造是一種惡意攻擊，它憑藉已通過身份驗證的用戶身份來運行未經過授權的命令。

使用:

<form method="POST" action="/profile">

@csrf

...

</form>

* 在表單內加上@csrf生成令牌，在web中間件組裡的VerifyCsrfToken「中間件」會自動驗證請求裡的令牌是否與存儲在會話中令牌匹配。

CSRF白名單:

<?php

namespace App\Http\Middleware;

use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;

class VerifyCsrfToken extends Middleware

{

// 这些 URI 将免受 CSRF 验证

protected $except = [

'http://example.com/foo/bar',

'http://example.com/foo/\*',

];

}

* 有時候你可能希望設置一組並不需要CSRF保護的URL，將不需保護的路由填到中間件VerifyCsrfToken中。

控制器:

簡介:

* 控制器並不是強制要求繼承基礎類。但是，如果控制器沒有繼承基礎類，你將無法使用一些便捷的功能，比如 middleware，validate和dispatch方法

單一功能:

<?php

namespace App\Http\Controllers;

use App\User;

use App\Http\Controllers\Controller;

class ShowProfile extends Controller

{

public function \_\_invoke($id)

{

return view('user.profile', ['user' => User::findOrFail($id)]);

}

}

* 若控制器只處理單個行為，則可以在這個控制器中放置一個\_\_invoke方法，這樣路由引用時，就可以不用指定函式只需指定Controller即可

套用中間件:

public function \_\_construct()

{

$this->middleware('auth');

$this->middleware('log')->only('index');

$this->middleware('subscribed')->except('store');

}

* 若需套用中間件，只需在建構子裡面加載即可

http request:

請求獲取:

* 要通過依賴注入的方式來獲取當前HTTP請求的實例，你應該在控制器方法中引入 Illuminate\Http\Request類。傳入的請求實例將通過 服務容器 自動注入

獲取指定輸入值:

$name = $request->input('name', 'Sally');

* 你可以給 input 方法的第二個參數傳入一個默認值。如果請求的輸入值不存在請求上，就返回默認值

$name = $request->input('products.0.name');

* 果傳輸表單數據中包含「數組」形式的數據，那麼可以使用「點」式語法來獲取數組

將輸入閃存至Session:

$request->flash();

$request->flashOnly(['username', 'email']);

$request->flashExcept('password');

* 將請求數據存入session中，這些方法對敏感信息（例如密碼）的保護非常有用

return redirect('form')->withInput();

return redirect('form')->withInput(

$request->except('password')

);

* 可能需要把輸入閃存到session然後重定向到上一頁，這時只需要在重定向方法後加上 withInput 即可
* $username = $request->old('username');
* 若要獲取上一次請求中閃存的輸入，則可以使用 Request 實例中的 old 方法。old 方法會從 Session 取出之前被閃存的輸入數據

視圖:

創建視圖:

* 視圖文件存放於resources/views目錄下，創建時名稱要以”.blade.php”結尾，如:login.blade.php

使用視圖:

return view('admin.login’, $data);

* 用view去訪問視圖，第二個值是將data傳入視圖使用，因為此視圖放在resources/views的admin底下(resources/views/admin)所以view的第一個值必須在前面加上”admin.”

所有視圖共享數據:

* 如果需要共享一段數據給應用程序的所有視圖，你可以在服務提供器的boot方法中調用視圖Facade的share方法。例如，可以將它們添加到AppServiceProvider或者為它們生成一個單獨的服務提供器

<?php

namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider

{

public function boot()

{

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

}

public function register()

{

//

}

}

URL:

簡介:

* Laravel 提供了幾個輔助函數來為應用程序生成URL。主要用於在模板和API 響應中構建URL 或者在應用程序的其它部分生成重定向響應

基本使用:

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

echo url("/posts/{$post->id}");

// http://example.com/posts/1

命名路由的 URL:

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

//

})->name('post.show');

echo route('post.show', ['post' => 1]);

// http://example.com/posts/1

Session:

簡介:

* 由於HTTP驅動的應用程序是無狀態的，Session提供了一種在多個請求之間存儲有關用戶信息的方法。

方法:

* Laravel自帶了幾個不錯而且開箱即用的驅動

file- 將會話存儲在storage/framework/sessions中。

cookie - Sessions被存儲在安全加密的cookie中。

database - Sessions被存儲在關係型數據庫中。

memcached/ redis- Sessions被存儲在基於高速緩存的存儲系統中。

array - Sessions存儲在PHP數組中，但不會被持久化。

***Note:*** **數組驅動一般用於測試並且防止存儲在Session中的數據被持久化。**

Database方法:

Schema::create('sessions', function ($table) {

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

$table->unsignedInteger('user\_id')->nullable();

$table->string('ip\_address', 45)->nullable();

$table->text('user\_agent')->nullable();

$table->text('payload');

$table->integer('last\_activity');

});

* 使用database作為會議驅動時，你需要創建一張包含會話各項數據的表。

上圖是用php artisan session:table 創建的Table

獲取數據:

* Laravel中處理會話數據有兩種主要方法：全局輔助函數session和通過一個Request實例。

public function show(Request $request, $id)

{

$value = $request->session()->get('key','default');//獲取特定變數

$request->session()->put('key','value'); //存入變數

}

* 用$request->session()獲取session後，使用get()取得特定的值，get()的第一個值是你要取的變數，第二個是若找不到此變數的預設值，

$value = session('key', 'default'); //獲取特定變數

session(['key' => 'value']); //存入變數

* 使用輔助函數session來獲取跟存取

閃存數據:

$request->session()->flash('status', 'Task was successful!');

* 使用這個方法保存在會話中的數據，只會保留到下一個HTTP請求到來之前，然後就會被刪除。

刪除數據:

$request->session()->forget('key'); //刪除鍵值為key的

$request->session()->flush(); //刪除全部

表單驗證:

簡介:

* Laravel提供了幾種不同的方法來驗證傳入應用程序的數據。默認情況下，Laravel的控制器基類使用ValidatesRequestsTrait，它提供了一種方便的方法去使用各種強大的驗證規則來驗證傳入的HTTP請求。

編寫驗證器邏輯:

* 我們將使用Illuminate\Http\Request對象提供的validate方法。如果驗證通過，代碼就可以正常的運行。如果驗證失敗，則會拋出異常，並自動將對應的錯誤響應返回給用戶。在典型的HTTP請求的情況下，會生成一個重定向響應，而對於AJAX請求則會發送JSON響應。

public function store(Request $request)

{

$validatedData = $request->validate([

'title' => 'required|unique:posts|max:255',

'body' => 'required',

]);

}

* 規則會按照分配的順序來驗證，unique:posts驗證此title是否在資料庫posts表單內是唯一的

顯示驗證錯誤信息:

* 如果傳入的請求參數未通過給定的驗證規則呢？正如前面所提到的，Laravel會自動把用戶重定向到之前的位置。另外，所有的驗證錯誤信息會被自動存儲到session中。

創建表單請求驗證:

* 面對更複雜的驗證情境中，你可以創建一個「表單請求」來處理更為複雜的邏輯。表單請求是包含驗證邏輯的自定義請求類。可使用Artisan命令make:request來創建表單請求類

php artisan make:request StoreBlogPost

public function rules()

{

return [

'title' => 'required|unique:posts|max:255',

'body' => 'required',

];

}

* 新生成的類保存在app/Http/Requests目錄下。如果這個目錄不存在，運行make:request命令時它會被創建出來，你可以向rules方法傳入所需的任何依賴項。他們會自動被Laravel提供的服務容器自動解析。

public function authorize()

{

$comment = Comment::find($this->route('comment'));

return $comment && $this->user()->can('update', $comment);

}

* 表單請求類內也包含了authorize方法。在這個方法中，你可以檢查經過身份驗證的用戶確定其是否具有更新給定資源的權限。
* 由於所有的表單請求都是繼承了Laravel中的請求基類，所以我們可以使用user方法去獲取當前認證登錄的用戶。

public function messages()

{

return [

'title.required' => 'A title is required',

'body.required' => 'A message is required',

];

}

* 你可以通過重寫表單請求的messages方法來自定義錯誤消息。此方法應該如下所示返回屬性/規則對數組及其對應錯誤消息

public function store(StoreBlogPost $request)

{

$validated = $request->validated();

}

* 如果驗證失敗，就會生成一個讓用戶返回到先前的位置的重定向響應。這些錯誤也會被閃存到session中，以便這些錯誤都可以在頁面中顯示出來。

Blade模板:

簡介:

* Blade是Laravel提供的一個簡單而又強大的模板引擎。和其他流行的PHP模板引擎不同，Blade並不限制你在視圖中使用原生PHP代碼。所有Blade視圖文件都將被編譯成原生的PHP代碼並緩存起來，除非它被修改，否則不會重新編譯，這就意味著Blade基本上不會給你的應用增加任何負擔。Blade視圖文件使用.blade.php作為文件擴展名，被存放在resources/views目錄。

定義佈局:

* 「主」頁面佈局。因為大多數web應用會在不同的頁面中使用相同的佈局方式，因此可以很方便地定義單個Blade佈局視圖

<html>

<head>

<title>App Name - @yield('title')</title>

</head>

<body>

@section('sidebar')

This is the master sidebar.

@show

<div class="container">

@yield('content')

</div>

</body>

</html>

繼承佈局:

* 當定義子視圖時，你可以使用Blade提供的@extends命令來為子視圖指定應該「繼承」的佈局。

@extends('layouts.app')

@section('title', 'Page Title')

@section('sidebar')

@parent

<p>这将追加到主布局的侧边栏。</p>

@endsection

@section('content')

<p>这是主体内容。</p>

@endsection

顯示未轉義數據:

* 默認情況下，Blade {{ }}語句會自動調用PHP的htmlspecialchars函數防止XSS攻擊。不想轉義的話，可以使用{!! !!}

***Note:*** 輸出用戶提供的數據時，千萬要小心。對用戶提供的數據，總是要使用雙花括號進行顯示，防止XSS 攻擊。

顯示原始型態:

@{{ $name }}

* 在前面加上@系統將不會轉譯變數會直接顯示 ”$name”

用戶認證:

簡介:

* Laravel使得實現身份驗證非常簡單。事實上，幾乎所有的配置都是現成的。

身份驗證配置文件位於config/auth.php，其中包含幾個有良好文檔記錄的選項，用於調整身份驗證服務的行為。 在其核心，Laravel的認證設施由“警衛”和“提供者”組成。守衛決定如何對每個請求的用戶進行身份驗證。比如，Laravel帶有一個session保護，它使用會話存儲和Cookies來維護狀態。

提供者決定如何從持久儲存中檢索用戶。Laravel 支持使用Eloquent 和數據庫查詢生成器檢索用戶。但是，你可以根據應用程序的需要來自由定義其他提供者。

注意事項:

* 默認情況下， Laravel包含一個App\User Eloquent model在你的app目錄下。

這個模型可與默認的Eloquent身份驗證驅動程序一起使用。如果你的應用程序沒有使用Eloquent，你可以用database身份驗證驅動程序，它用的是Laravel查詢生成器。

當為App\User模型生成數據庫架構時，確保密碼的長度至少為60個字符。保持默認的字符串長度為255個字符是一個不錯的選擇。

另外，你應該驗證“users”（或等效）表是否包含一個可空的，含有100個字符的remember\_token字符串。此列將用於存儲用戶登錄應用程序時選擇“記住我”選項的令牌。

用戶認證快速指南:

* Laravel附帶了幾個預構建的身份驗證控制器，它們位於App\Http\Controllers\Auth命名空間中。

RegisterController處理新的用戶註冊，

LoginController處理身份驗證，

ForgotPasswordController處理用於重置密碼的電子郵件鏈接，ResetPasswordController包含重置密碼的邏輯。這些控制器中的每一個都使用一個特性來包含它們的必要方法。

使用php artisan make:auth可以快速建構上述路由跟視圖

***Note:***如果應用程序不需要註冊，可以通過刪除新創建的RegisterController並修改路由聲明來禁用它：Auth::routes(['register' => false]);.

路由視圖:

* php artisan make:auth命令將創建認證所需的所有視圖，並將它們放在resources/views/auth目錄中。

make:auth命令還將創建一個包含應用程序基本佈局的resources/views/layouts目錄。

所有這些視圖都使用了Bootstrap CSS框架，但是你可以自由地定制它們。

自定義路徑:

* 在LoginController，RegisterController，ResetPasswordController，還有VerificationController控制器中定義redirectTo屬性來自定義驗證後的重定向位置
* protected $redirectTo = '/';

自定義用戶名:

* Laravel默認使用email字段來認證。如果你想使用其他的字段，可以在LoginController控制器裡面定義一個username方法

public function username()

{

return 'username';

}

自定義看守器:

* 你還可以自定義用戶認證和註冊的「看守器」。要實現這一功能，需要在LoginController, RegisterController和ResetPasswordController中定義guard方法。該方法會返回一個看守器實例

use Illuminate\Support\Facades\Auth;

protected function guard()

{

return Auth::guard('guard-name');

}

* 看守器是用於如果需要2個登入系統時使用，用於辨別是哪個登入系統

自定義驗證/ 存儲:

* 為了修改新用戶在註冊時所需要填寫的表單字段，或者自定義如何將新用戶存儲到數據庫中，你可以修改 RegisterController類。該類負責驗證和創建新用戶。

RegisterController類的validator方法包含了驗證新用戶的規則，你可以隨心所欲地自定義該方法。

RegisterController的create方法負責使用 Eloquent ORM在數據庫中創建新的App\User記錄。你可以根據數據庫的需要自定義該方法。

檢索認證用戶:

* 你可以通過Auth façade或request來訪問已認證的用戶

use Illuminate\Support\Facades\Auth;

// 获取当前通过认证的用户...

$user = Auth::user();

// 获取当前通过认证的用户 ID...

$id = Auth::id();

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProfileController extends Controller

{

public function update(Request $request)

{

$request->user() //返回一个认证用户实例...

}}

保護路由:

* 路由中間件可以用於只允許通過認證的用戶訪問給定的路由。

Laravel自帶了一個auth中間件,它定義在Illuminate\Auth\Middleware\Authenticate中。

由於這個中間件已經在HTTP內核中註冊，你只需把這個中間件附加到路由定義中

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

// 只有认证过的用户可以进入...

})->middleware('auth');

* 當然，如果你使用控制器，你可以在控制器的構造函數中調用 middleware方法來直接將其附加到路由定義中

public function \_\_construct()

{

$this->middleware('auth');

}

重定向未認證的用戶:

* 當auth中間件檢測到一個未認證用戶時，它會把用戶重定向到名為login的命名路由上。

您可以通過修改app/Http/Middleware/Authenticate.php文件中的redirectTo函數來修改此行為

protected function redirectTo($request)

{

return route('login');

}

手動驗證用戶:

* 不一定非要在Lavarel 中使用驗證控制器。如果選擇刪除這些控制器，就需要直接使用Lavarel 驗證類。

可以藉助Auth facade訪問Laravel服務，因此需要在類的開頭導入 Auth。

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class LoginController extends Controller

{

public function authenticate(Request $request)

{

$credentials = $request->only('email', 'password');

if (Auth::attempt($credentials)) {

// 身份验证通过...

return redirect()->intended('dashboard');

}

}

}

if (Auth::attempt(['email' => $email, 'password' => $password, 'active' => 1])) {

// 用户存在，已激活且未被禁用。

}

* 可以自己指定驗證條件

登出:

Auth::logout();

記住用戶:

* 如果想在應用中提供「記住我」功能，可以給attempt方法傳遞一個布爾值作為其第二個參數，這會無限期保持用戶身份驗證，直到用戶手動登出。用戶表需要包含字符串類型的remember\_token列用於存儲令牌。

if (Auth::attempt(['email' => $email, 'password' => $password], $remember)) {

// 用户被记住...

}

用戶授權

簡介:

* Laravel提供了一種簡單的方法來處理用戶的授權動作。

與用戶認證一樣，Laravel的授權方法很簡單，授權操作有兩種主要方式：gates和策略。

Gates:

* Gates是用來決定用戶是否授權執行給予動作的一個閉包函數，並且典型的做法就是在 App\Providers\AuthServiceProvider中使用Gate來定義. Gates總是接收一個用戶實例作為第一個參數，並且可以接收可選參數。

Gate的第一個參數$user會自動去抓當前登入的使用者，所以使用時只需傳入後面參數

public function boot()

{

$this->registerPolicies();

Gate::define('update-post', function ($user, $post) {

return $user->id == $post->user\_id;

});

}

* Gate::define的第一個參數是定義它的名稱，後面為功能

授權動作:

if (Gate::allows('update-post', $post)) {

// 指定当前用户可以进行更新...

}

if (Gate::denies('update-post', $post)) {

// 指定当前用户不能更新...

* 使用allows或denies，第一個參數為要調用的gate第二個為要批授權的參數

策略:

* 策略是在特定模型或者資源中組織授權邏輯的類。例如，你的應用是一個博客，那麼你在創建或者更新博客的時候，你可能會有一個Post模型和一個對應的PostPolicy來授權用戶動作。

可以直接用指令 php artisan make:policy PostPolicy 創建策略

* 策略是在特定模型或者資源中組織授權邏輯的類。例如，你的應用是一個博客，那麼你在創建或者更新博客的時候，你可能會有一個Post模型和一個對應的PostPolicy來授權用戶動作。

php artisan make:policy PostPolicy --model=Post

註冊策略:

* 一旦策略存在，它就需要進行註冊。

新的Laravel應用中包含的AuthServiceProvider有一個policies屬性，可以將各種模型對應到它們的策略中。

protected $policies = [

Post::class => PostPolicy::class,

];

* 在AuthServiceProvider中註冊，表示PostPolicy用於Post

編寫策略:

* 一旦授權策略被註冊，你就可以為授權過後的每個動作添加方法。比如，我們在PostPolicy中定義一個update方法，它會判斷指定的User是否可以更新指定的Post實例。

<?php

namespace App\Policies;

use App\User;

use App\Post;

class PostPolicy

{

public function update(User $user, Post $post)

{

return $user->id === $post->user\_id;

}

}

***Note:*** 如果在Artisan控制台生成策略時，使用--model選項，它會包含進去view，create，update和delete動作方法。

使用策略授權動作:

* Laravel內置的User模型包含兩個有用的方法來授權動作：can和cant。

if ($user->can('update', $post)) {

//

}

* can()第一個參數為策略裡要調用的函數第二個參數為要授權的model

if ($user->can('create', Post::class)) {

// 执行相关策略中的 "create" 方法...

}

* 若不須指定model則第二個參數傳入類即可

通過控制器輔助函數:

* 除了在User模型中提供輔助方法以外，Laravel也為繼承 App\Http\Controllers\Controller這個基類的控制器提供了一個有用的authorize方法。

就像can方法一樣，這個方法需要接收你想授權的動作和相關的模型作為參數。如果這個動作沒有被授權，authorize方法會拋出一個Illuminate\Auth\Access\AuthorizationException的異常，然後Laravel默認的異常處理器會將這個異常轉化成帶有 403狀態碼的HTTP響應。

<?php

namespace App\Http\Controllers;

use App\Post;

use Illuminate\Http\Request;

use App\Http\Controllers\Controller;

class PostController extends Controller

{

public function update(Request $request, Post $post)

{

$this->authorize('update', $post);

$this->authorize('create', Post::class);//不須指定model方法

// 当前用户可以更新博客...

}

}

Email認證

簡介:

* 許多Web 應用要求用戶在使用之前進行Email 地址驗證。Laravel 不會強迫你在每個應用中重複實現它，而是提供了便捷的方法來發送和校驗電子郵件的驗證請求。

Model 預備:

* 開始之前,驗證你的App\User模型是否實現了Illuminate\Contracts\Auth\MustVerifyEmail契約。

此外，你應該使用Illuminate\Auth\MustVerifyEmailtrait

<?php

namespace App;

use Illuminate\Auth\MustVerifyEmail;

use Illuminate\Notifications\Notifiable;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Contracts\Auth\MustVerifyEmail as MustVerifyEmailContract;

class User extends Authenticatable implements MustVerifyEmailContract

{

use MustVerifyEmail, Notifiable;

// ...

}

數據庫注意事項:

* 接下來，你的user需要包含一個email\_verified\_at字段用來存儲Email地址通過驗證的時間。

默認情況下，Laravel框架中users表的數據遷移已經包含了這個字段。所以，您需要做的就只是執行數據庫遷移。

集合

簡介: