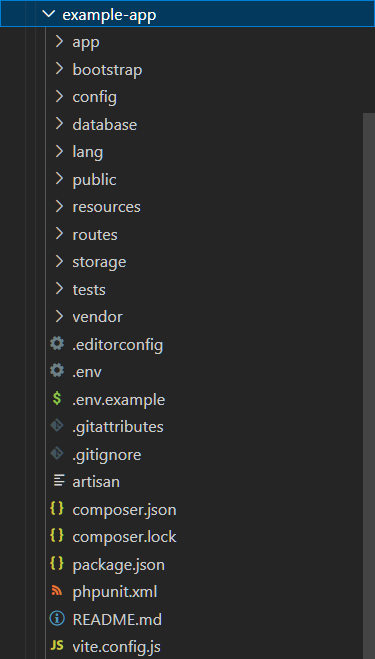
1. Routes
2. Larevel 資料夾架構



建立Laravel時，內部架構如上所示

* App: 包含應用程式的核心程式。 Application的主要資料夾, 內部會放Controllers和Models
* Bootstrap: 包含幾個框架啟動跟自動載入設定的檔案。以及 cache 資料夾，包含一些框架對啟動效能最佳化所產生的檔案。
* Config: 包含所有應用程式的設定檔。有關於設定的內容,都會佈置在此資料夾。
* Database: 資料庫遷移與資料填充檔案。也可以在此資料夾存放 SQLite 資料庫。 建立data、files、tables都可以在資料庫檔案中設置。
* lang: 多國語系資源，每個子目錄對應到一個語系。
* public: 包含前面的控制器和你的資源檔（圖片、JavaScript、CSS，等等）。 可供所有人取用的資源都放置在此資料夾中, 例如: CSS、Image等。
* resources: 包含視圖、原始的資源檔(LESS、SASS、CoffeeScript)，以及語言檔。 需要編譯的資料放置在此資料夾, 像是JS檔案在此資料夾編譯後的結果, 會放置回public的資料夾中做取用。
* routes: 路由參數，設置url對應controller的地方。
* storage: Laravel儲存檔案目錄，含編譯後的Blade樣板、session、log檔案等。
* tests: 單元測試目錄，含PHPUnit測試相關文件。
* vendor: Laravel主程式框架以及相依套件(composer安裝的套件存放在此目錄底下)。

1. Routes介紹

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

    return view('welcome');

});

路徑的內容都放置在routes資料夾內, 其中首頁的設定放置在routes/web.php中。

Route => 為路徑

:: => 兩個冒號代表為static

get => 代表是使用的方法

() => 執行的程式內容

‘/’ => 第一個變數為路徑名稱

function() {} => 當使用者輸入路徑名稱時，會觸發使用的程式邏輯 => 這邊是回傳welcome的內容

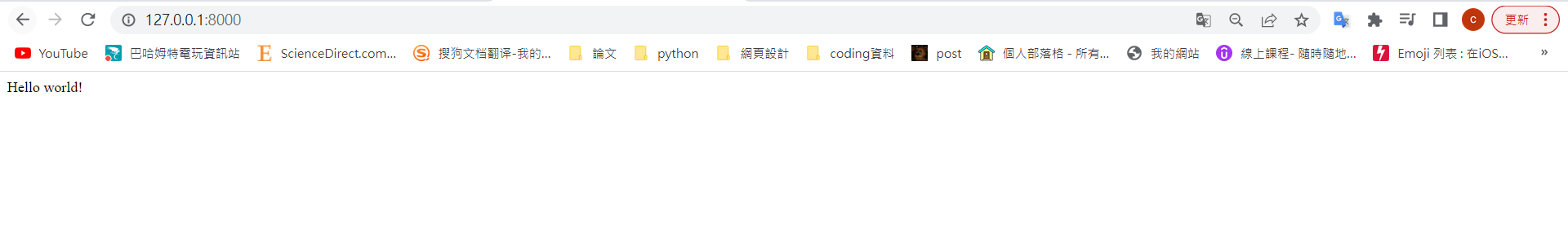
Route::get('/', function () {

    // return view('welcome');

    return "Hello world!";

});

將原先預設的views(視圖)中的welcome檔案給註解掉後，設置自定義的內容，會顯示在網頁中。

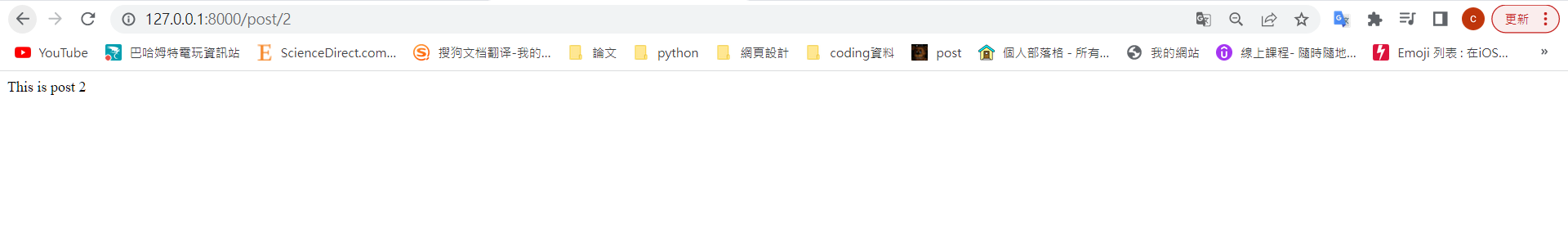


Route::get('/post/{id}', function($id) {

    return "This is post ". $id;

});

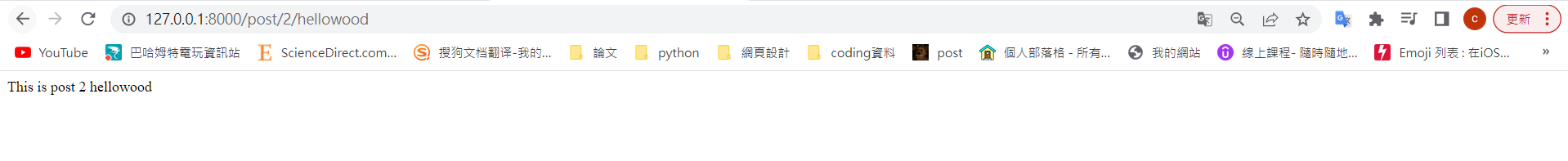
* 獲取變數的方法: 在路徑中設置變數好以取用。



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

    return "This is post ". $id . " ". $name;

});



* 路徑命名

Route::get('admin/posts/example', array('as' =>'admin.home', function() {

    $url = route('admin.home');

    return "This url is ". $url;

}));

定義一個路徑為 localhost:8000/admin/posts/example的路徑

array(‘as’=>’admin.home’) 重新命名路徑名稱為admin.home => admin.home = localhost:8000/admin/posts/example

這邊的過程為

* 1. 觸發url，並且解析 => 2. 解析完畢後，自動將解析後的url存進admin.home中 => 3. 執行內部的程式和邏輯

Route::get('admin/posts/example', function() {

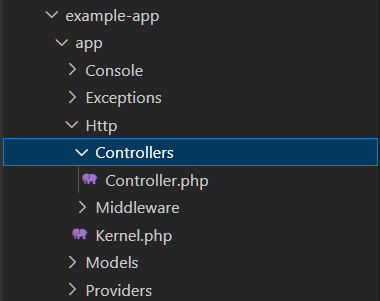
    $url = route('admin.home');

    return "This url is ". $url;

}) -> name('admin.home');

與前面的array()方式一樣，只是用不同的形式表達。

1. Controllers



Controllers的位置在app/Http/Controllers/

1. namespace

在php 程序語言裡邊，語法規則要求同名稱的函數、class、const在一個請求中不允许出現多次。

命名空間(namespace)。空間可以看做一個容器，存儲類、函數、const的容器。



namespace有scope範圍

Test1 涵蓋到namespace Test2;之前

Test2 則是namespace Test2;之後都是

1. use

namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;

use Illuminate\Foundation\Bus\DispatchesJobs;

use Illuminate\Foundation\Validation\ValidatesRequests;

use Illuminate\Routing\Controller as BaseController;

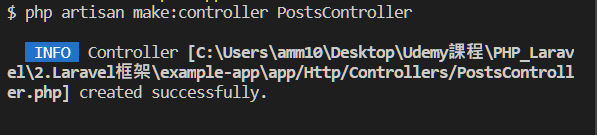
這邊的use為import導入檔案的功能

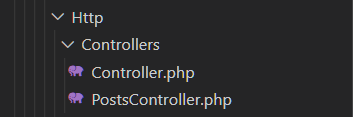
後面綠色的部分為namespace區塊的功能做取用

如前一部分的Test1和Test2，可以直接取用該namespace涵蓋的區塊，並且使用該區塊內部的程式邏輯

1. 建立Controller的三個方法

* 直接在Controllers資料夾底下新增一個php的檔案
* php artisan make:controller PostsController => 直接在git bash中使用php artisan的指令





* php artisan make:controller --resource PostsController => 一樣在git bash中使用php artisan的指令，但此處有使用--resource，會在該建立的php檔案中新增 增(C)查(R)改(U)刪(D)的功能

class PostsController extends Controller

{

    public function index()

    {

        //

    }

    public function create()

    {

        //

    }

    public function store(Request $request)

    {

        //

    }

    public function show($id)

    {

        //

    }

    public function edit($id)

    {

        //

    }

    public function update(Request $request, $id)

    {

        //

    }

    public function destroy($id)

    {

        //

    }

}

1. Routing Controllers

Route::get('/post', '\App\Http\Controllers\PostsController@index');

在routes/web.php中，設置上述內容去連接controllers的內容

後面的controller需要引入該目標controller的位置

藍色的部分為 從該controller中引入的方法是哪一個

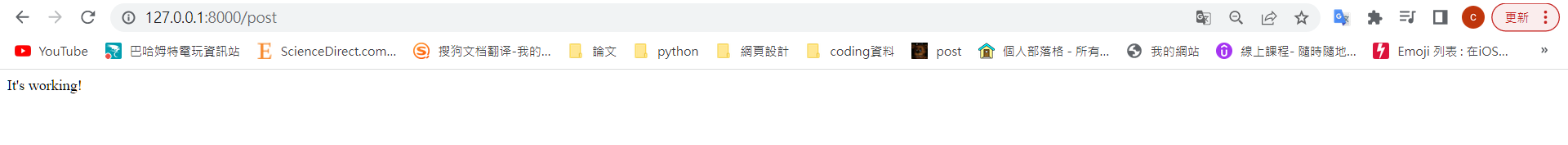
    public function index()

    {

        return "It's working!";

    }

此時回到PostsController.php中，將index()設置些許內容查看是否成功



1. 傳遞變數

Route::get('/post/{id}', '\App\Http\Controllers\PostsController@index');

在routes中設置要傳遞的變數

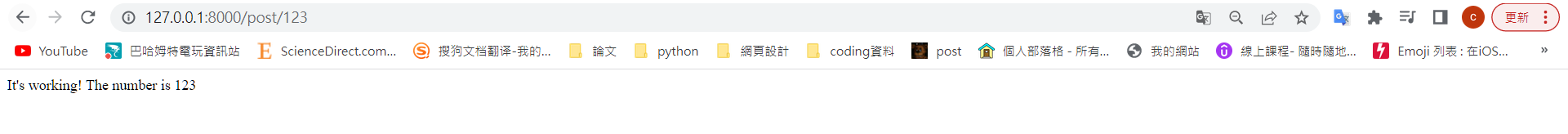
    public function index($id)

    {

        return "It's working! The number is ". $id;

    }

到controller中設置接收的變數，並且回傳至網頁中

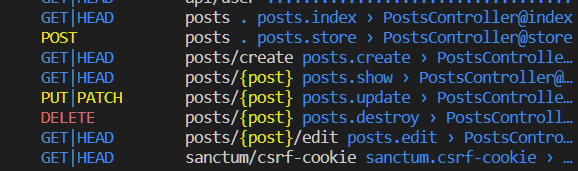


1. Resource

Route::resource('posts', '\App\Http\Controllers\PostsController');

使用resource的方法，可以使用controller中的所有method

回到terminal輸入 php artisan route:list，可以看到目前有哪些route可以使用



可以看到controller中的index()、create()、store()、show()、edit()、update()、destroy()都有顯示在其中

因此不用特地在後面引用controller路徑時，指定需要使用哪個功能

    public function show($id)

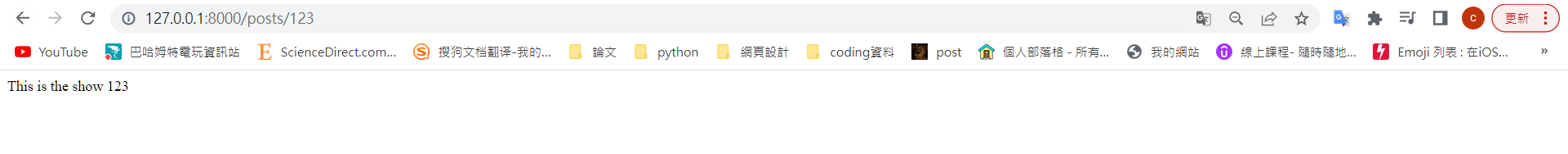
    {

        //

        return "This is the show ". $id;

    }

進行測試，在controller的show()設置上述內容



當輸入/posts/$id時，會觸發show()的功能，並且回傳給使用者查看

    public function create()

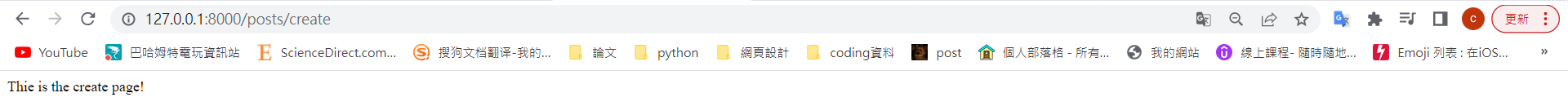
    {

        //

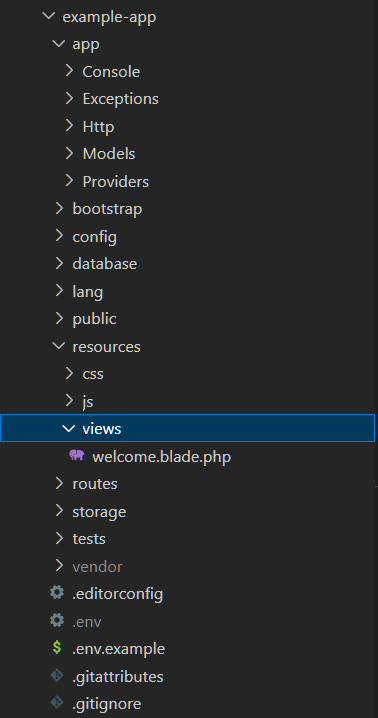
        return "Thie is the create page!";

    }

測試create()



1. Views
2. Views 位置



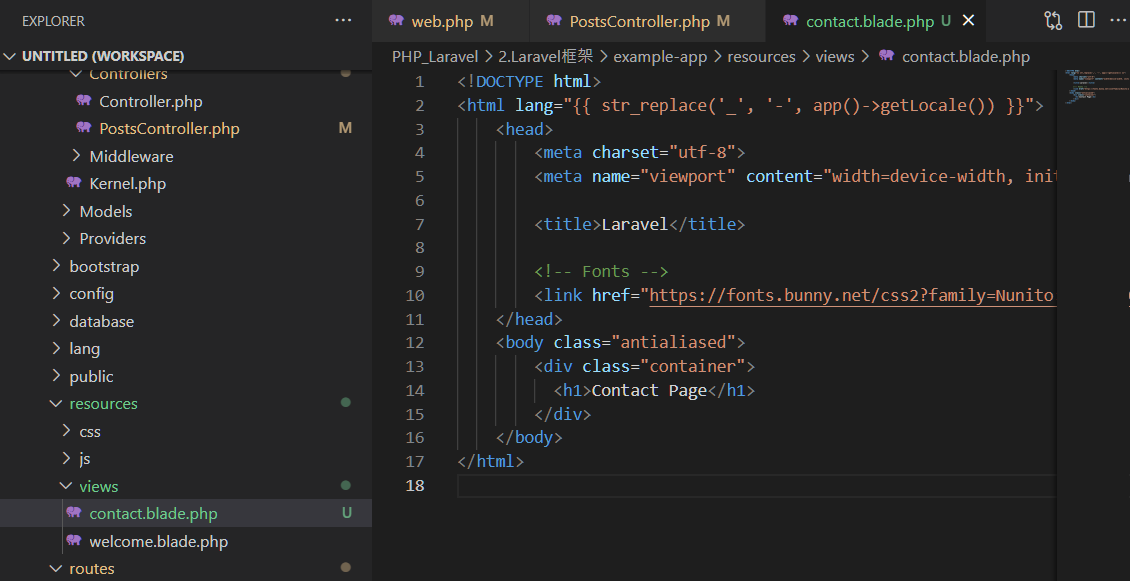
1. 建立views連結

    public function contact($id) {

        return view('contact');

    }

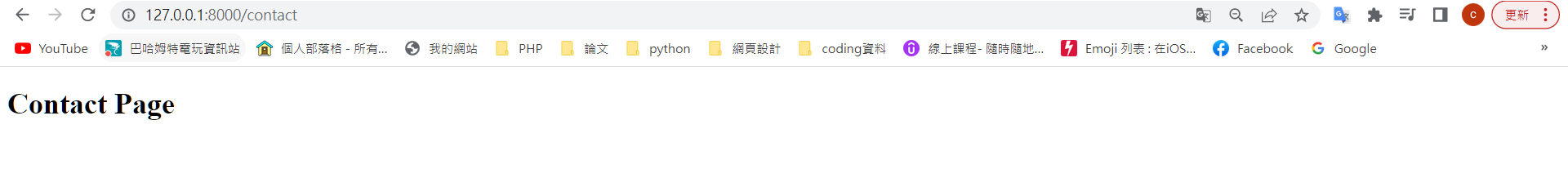
在先前的PostsController.php中新增功能，並且用view()，將新建立的view回傳。



到resources/views中新增欲建立的新view



最後在routes/web.php中新增該view的路由



1. routes傳遞變數給views

* 第一種方式 – 陣列

Route::get('/contact/{id}', '\App\Http\Controllers\PostsController@contact');

先到web.php(routes)中設置要傳遞的變數

    public function contact($id) {

        $new\_id = array(

            'identify'=> $id

        );

        return view('contact', $new\_id);

    }

}

到PostsController.php中接收要傳遞的變數，並且傳遞給view

    <body class="antialiased">

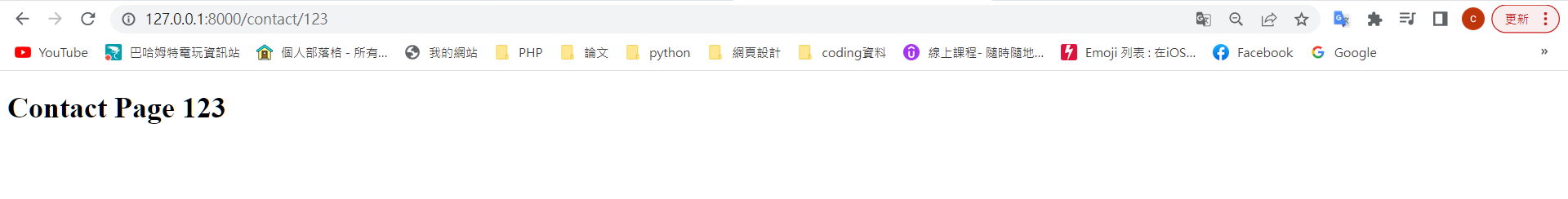
        <div class="container">

          <h1>Contact Page {{$identify}}</h1>

        </div>

    </body>

最後到view接收Controller傳遞的資料



php單箭頭「->」用來參照一個類的屬性和方法或呼叫類中的函數。

雙箭頭「=>」用來定義陣列。

* 第二種方式 - 參照方式(->)

Route::get('post/{id}', '\App\Http\Controllers\PostsController@show\_post');

在routes中定義新的路徑和需要傳遞的變數

    public function show\_post($id) {

        return view('post')->with('id', $id);

    }

在PostsController.php中定義新的function

其中使用到參照方式(Chain)，當使用者呼叫該路徑時，回傳view中的post.blade.php檔案，並且攜帶key=’id’; value=$id的變數到view中。

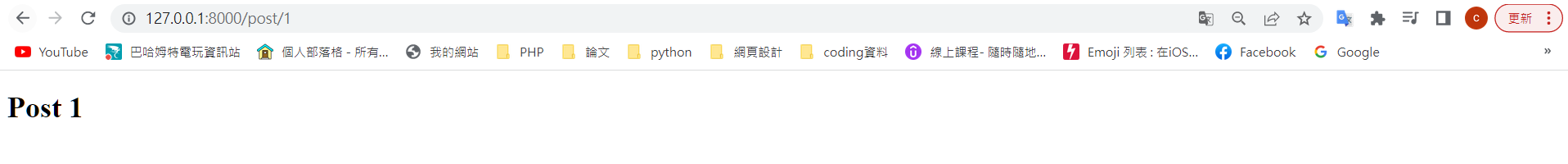
#課程中 => view(‘post’) 開啟post.blade.php該檔案 => ->with(‘id’,$id); 傳遞’id’這個變數攜帶著$id的值到view中

<body>

  <h1>Post {{$id}}</h1>

</body>

到view中接收傳遞的變數



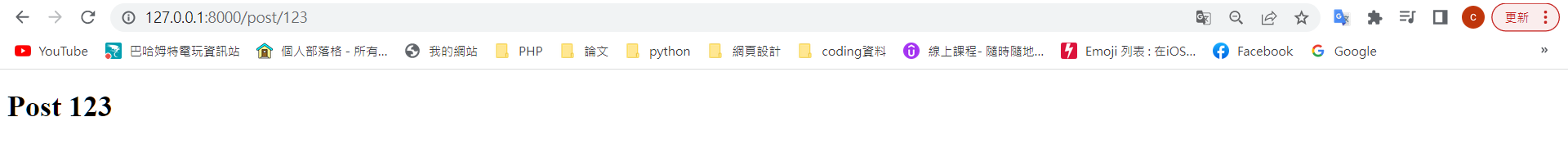
* 第三種方式 – Compact()

    public function show\_post($id) {

        // return view('post')->with('id', $id);

        return view('post', compact('id'));

    }



1. Blade – PHP Template Engine

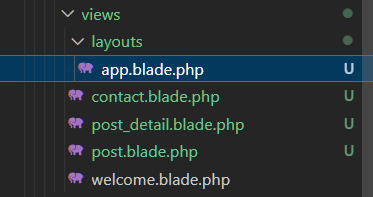
Blade 是 Laravel 框架的 Template Engine。 => 與node.js的ejs Engine類似

Blade 和其他 PHP Template Engine 不同之處在於 Blade 不限制在檔案內使用原生 PHP Code。

**檔案命名**

Blade 檔案通常放在resources/views 下，並使用 .blade.php 作為文件的副檔名後輟名稱。

1. 建立共同Layout



首先到views資料夾中新建layouts的資料夾

將有重複到的內容存進app.blade.php中，之後在其他php檔案引入即可使用

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

</head>

<body>

  <div class="container">

    @yield('content')

  </div>

  @yield('footer')

</body>

</html>

Blade Engine

@section 定義一個內容區塊

@yield 用來顯示給定區塊的內容

@extends('layouts.app')

@section('content')

    <h1>Contact Page</h1>

@endsection

之後前往其他php檔案引入app.blade.php的內容，即可取用

並且可以透過yield => 到目前的檔案中使用section(區塊名稱)進行內容定義

1. Blade傳遞變數

    public function contact() {

        $people = ['Rely', 'Coolbastian', 'LeeJay', 'BoShin', 'Jihe', 'Monzi'];

        return view('contact', compact('people'));

    }

在PostsController中新增$people陣列的變數，並且將變數傳遞給contact.php中。

@section('content')

    <h1 style="text-align: center;">Contact Page</h1>

    @for ($i = 0; $i < count($people); $i++)

        <p style="text-align: center; font-size: 1.5rem;">{{$people[$i]}}</p>

    @endfor

@endsection

透過使用blade引擎內建的邏輯設定，使用for-loop將資料一筆一筆print在網頁中。

#注意: php沒有length()的功能，因此使用的是count()

另一種方式為

@section('content')

    <h1 style="text-align: center;">Contact Page</h1>

    @if (count($people))

        <ul style="text-align: center; padding: 0;">

            @foreach ($people as $person)

                <li style="list-style: none; font-size: 1.5rem;">{{$person}}</li>

            @endforeach

        </ul>

    @endif

@endsection

使用foreach完成目標