

單元 4 Artisan & Laravel 目錄結構簡介

Step 01: 如何啟動 Laravel 開發伺服器 (使用 Artisan)

- 要先切換到 Laravel 專案底下
- 這裡使用 Atom 編輯器來開啟
- ⇒ `php artisan`
 - 檢查 artisan
- ⇒ `php artisan serve`
 - 以下參考用，有需要再下指令
- `php artisan serve --host=0.0.0.0`
- `php artisan serve --port=3000`
- ⇒ `localhost:8000`
 - 如果安裝失敗，確定初始環境正確？
打開 Laravel 專案目錄夾，檢查 `.env` 設定檔是否存在？
否，執行 `cp.env.example .env`
 - 開啟 Laravel server 後出現錯誤，執行下列指令
⇒ `php artisan key:gen`
 - 產生金鑰 (在 `.env` 裡面的 `APP_KEY` 可以看到)

Laravel 的入口: Artisan

Artisan 工具可以幫助你完成許多事情，包括：

- 遷移(migrate)資料庫
- 開啟維修模式
- 檢視 Route
- 自動產生程式碼

Laravel 目錄結構

<code>app</code>	<ul style="list-style-type: none">· <code>App\</code><ul style="list-style-type: none">· <u><code>Model.php</code></u> (底線表示資料庫物件名稱 ex: <code>user.php</code>)· <code>Http\</code><ul style="list-style-type: none">· <code>Controller\</code>· <code>Middleware\</code>· 防護或者權限檢查的時候會用到的
<code>bootstrap</code>	<ul style="list-style-type: none">· 啟動應用程式用，並非前端 CSS 套件
<code>config</code>	<ul style="list-style-type: none">· 所有設定檔，常用包括：<ul style="list-style-type: none">· <code>app.php</code> 設定時區<ul style="list-style-type: none">⇒ <code>config/app.php</code>⇒ <code>'timezone' => 'UTC'</code>, 改成 <code>'timezone' => 'Asia/Taipei'</code>,

單元 2 安裝 PHP & Laravel & MySQL (Mac OSX)

- Mac OSX: El Capitan 10.11

Step 00: Terminal 安裝 Oh-My-Zsh

Step 01: 安裝 Xcode 套件 (Command line tool)

- ⇒ `xcode-select -install`
 - 安裝 xcode command line tool (安裝完先打開 xcode 確認)
- ⇒ `xcode-select -p`
 - 檢查是否安裝成功

Step 02: 安裝 Homebrew (OSX 套件管理系統，類似 apt-get)

- <http://brew.sh/>
- 注意：可能會跟 macports 衝突
- ⇒ `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`
- ⇒ `brew install wget`
- ⇒ `brew`
 - 檢查是否安裝成功

Step 03: 升級 PHP 7

<http://php-osx.liip.ch>

- ⇒ `curl -s https://php-osx.liip.ch/install.sh | bash -s 7.0`
- ⇒ `php --version`
 - 自動安裝 script 執行完成後，如果 PHP 仍是 5.6，則需手動修改\$PATH 設定環境目錄
- ⇒ `vim .zshrc`
 - 游標移到最下面，打下列兩行
- ⇒ `# PHP 7`

```
export PATH=/usr/local/php5-7.0.8-20160626-123411/bin:$PATH
# Tinker
alias tinker="php artisan tinker"
```
- ⇒ `which php`
 - 檢查路徑是不是正確

Laravel 專案根目錄檔案

.env	· 環境設定檔
.env.example	· .env 不會被 git 紀錄，用以存放資料庫密碼等環境設定 · 執行期間使用 env() 函數可取得設定值
composer.json	· composer 會自動讀取該檔案 · 該檔案紀錄所有必須被 composer install 安裝的相依套件 · composer.lock 為 composer 自動產生的進度檔，如果選擇讓 git 紀錄該檔案，則可保證每次 composer install 的結果都一模一樣
gulpfile.js	· 前端處理工具 gulp 設定檔案，用以自動化處理 js, css
package.json	· nap (node package manager) 設定檔案 · 定義所有 npm install 需要安裝的相依套件 · 通常使用 npm 來安裝 gulp 等前端處理工具
phpunit.xml	· PHP 單元測試設定檔
server.php	· 用以啟動 server · 通常不必更改

常用 HTTP 動詞

- RESTful API 實作方法
 - ⇒ routes/api.php
 - ⇒ Route::get('/user') 直接拿東西
 - ⇒ Route::post('/user') 用來新增一個東西
 - ⇒ Route::put('/user') 用來更新一個東西
 - ⇒ Route::delete('/user') 用來刪除一個東西

處理參數

- ⇒ routes/web.php
- ⇒ Route::get('/user/{id}', function(\$id) {
 - ⇒ return 'Hi, User ' . \$id;
 - ⇒ });
- ⇒ localhost:8000/user/300
- ⇒ localhost:8000/user/apple

傳送給 Controller

- ⇒ Route::get('/hello/{id}', 'HelloController@hello');
- ⇒ HelloController 中的 hello 函數就要帶一個 id 參數進去

將參數帶入 view

- ⇒ views/welcome.blade.php
- ⇒ <div class="title m-b-md">
- ⇒ {{ \$title }}
- ⇒ </div>
- ⇒ routes/web.php
- ⇒ Route::get('/title/{str}', function(\$str) {
 - ⇒ return view('welcome', [
 - ⇒ 'title' => \$str
 - ⇒]);
 - ⇒]);
- ⇒ localhost:8000/title/Hello

單元 7 資料庫基礎: Migration

- 使用 Laravel 的 Migration 來進行資料表的建立

Why Migration?

- Migration 資料庫遷移
- 優點
 - 將資料表結構寫入 Laravel PHP 檔案內
 - 可用 git 追蹤變更
 - 確保團隊所有資料庫結構同步

Step 00: 先在 MySQL Workbench 建立資料庫，在`.env` 設定資料庫資料，後面 migration 連線測試才不會出現錯誤

- `.env` 設定 mysql 資料庫，預設 `DB_DATABASE=(自己建立一個), DB_USERNAME=root, DB_PASSWORD=(如果前面沒有設定密碼，就是空的，不用打任何東西)`

- 這邊可以在`.env` 中使用先前建立的範例資料庫

`DB_DATABASE=Starting`

`DB_USERNAME=root`

`DB_PASSWORD=`

Step 01: 產生 migration file

⇒ `php artisan make:migration create_post_table --create=posts`

- 第一個底線是 migration 名稱(可以是任何想要的名稱)，第二個底線是 create 成哪一個 table 的名稱

⇒ 進到專案目錄，

`database/migrations/2016_09_15_085513_create_post_table.php` 這個檔案就是要撰寫資料庫 table 結構的地方，它可以代替我們執行 MySQL 指令，包括 `create table`

- 如果做錯了怎麼辦？如果後悔，不應該 `migrate` 這個檔案的話...
 - ⇒ `php artisan migrate:rollback`
// 把上一步復原的意思
 - ⇒ `mysql -u root -p`
 - ⇒ `use Starting;`
 - ⇒ `show tables; //就會發現所有 table 都不見了`
 - ⇒ `quit;`
- ⇒ `php artisan migrate`
// 把 table 再弄回來

如果想要看更多，到 Laravel 的文件查詢

- <https://laravel.com>
- Documentation 查詢關鍵字 `Migration` 可以看到各種資料型態的列表 (Available Column Types)

修飾字：因為 MySQL 預設全都是不允許你有空值 `null`，所以修飾字的意思是幫它修改這些東西，直接寫在資料型態後面

- `->default($value)` 預設值
- `->nullable()` 允許空值
- `->unsigned()` 數字恆正

範例

```
⇒ database/migrations/2016_09_15_085513_create_post_table.php
⇒ public function up()
⇒ {
⇒     Schema::create('posts', function (Blueprint $print) {
⇒         $table->increments('id');
⇒         $table->string('title')->nullable(); //會讓 title 允許變成 null
⇒         $table->string('text')->default('Hello'); //預設值就是 Hello
⇒         $table->integer('author')->unsigned(); //數字恆正，可以讓儲存的空間變大
⇒         $table->timestamps();
⇒     });
⇒ }
```

如果有錯誤，檢查是否正確 `migrate` (可能忘記設定 `.env` 資料庫資訊或密碼)

單元 8 資料庫物件: Model 及關聯

- 配合 Model 就可以用 Laravel 的 Model ORM 來操作資料庫

Laravel 的資料庫介面

- 稱為 Eloquent ORM
- ORM = Object Relational Mapping
- 與一般傳統直接在程式碼中執行 SQL 不同，ORM 採用物件的方式存取資料庫

差異比較：一般 SQL

```
$db = new PDO(資料庫設定);  
$count = $db->exec("select * from posts");
```

差異比較：ORM

```
⇒ Post::all(); //顯示全部筆數 = select * from posts
```

Step 01: 產生一個 post model file

```
⇒ php artisan make:model Post  
⇒ App/Post.php
```

設定 Model 屬性

- MySQL Table 實際名稱
 - protected \$table = 'post';
- Mass-Assigment 大量新增許可欄位
 - protected \$fillable = ['title', 'text', 'author'];
- 保護機密欄位(API 中可用)
 - protected \$guarded=['password'];
- 從 App\User.php 範例中直接複製到 App\Post.php，第一行不用複製
- 檢查 database/migrations/2016_09_15_085513_create_post_table.php，確認資料表是 posts

- 進到 MySQL 確認一下
 - ⇒ mysql -u root -p
 - ⇒ use Starting
 - ⇒ show tables;
 - ⇒ select * from posts;
- 回到 tinker 繼續新增第二筆文章資料
 - ⇒ \$post->create([
 - ⇒ 'title' => 'Post title 2',
 - ⇒ 'note' => 'Something',
 - ⇒ 'author' => 2
 - ⇒]); //create 是 ORM 的 function，新增第二筆資料
 - ⇒ \$post->all();
- 可回到 MySQL 中再次確認是否多出第二筆資料

Step03: 如何在 Model 中建立關聯

一對一 hasOne

- 使用時機舉例：
 - 每個 User 只能有一組密碼

一對多 hasMany

- 使用時機舉例：
 - 每個 User 有[很多]文章

範例：每個 User 可以發很多文章，但是一個文章只能有一個 User

測試前需要先有 User，所以先建立 User 資料

先到 MySQL 查看 User 有哪些欄位

- ⇒ select * from users; //確認是空的
- ⇒ describe users; //顯示 users table 有哪些欄位

單元 9 撰寫後端功能: Controller & Request

單元目標

- 撰寫 posts 資料表功能
 - 顯示全部資料
 - 顯示單一筆資料
 - 新增
 - 刪除
- 練習使用 Validation 檢查資料
- 使用 Postman 進行測試

Step 01: 產生 Controller

⇒ `php artisan make:controller PostController --resource`
//底線是 controller name，大小寫是命名規則，--resource 是幫我們自動產生
controller 中會用到的 function 程式碼
⇒ `app\Http\Controllers\PostController.php`

顯示全部資料

Step 02: 要寫 controller 之前先寫 route

⇒ `routes\web.php`
⇒ `Route::get('/posts', 'PostController@index');` //PostController 中的 index function
⇒

Step 03: 到 controller

⇒ `app\Http\Controllers\PostController.php`

- 先測試有沒有連到 route

⇒ `public function index()`
⇒ {
⇒ `return 'post index';`
⇒ }
⇒ `php artisan serve`
⇒ `localhost:8000/posts`

Step 08: 到 controller 新增找不到資料的訊息

⇒ app\Http\Controllers\PostController.php

· 做法一

```
⇒ public function show($id)  
⇒ {  
⇒     $post = Post::find($id);  
⇒     if( $post == null)  
⇒         abort(404);  
⇒     return $post;  
⇒ }  
⇒ 重複 Step 07 到 Postman 做測試
```

· 做法二 (Laravel 提供的簡潔做法)

```
⇒ public function show($id)  
⇒ {  
⇒     return Post::findOrFail($id);  
⇒ }  
⇒ 重複 Step 07 到 Postman 做測試
```

新增

Step 09: 新增資料，要寫 controller 之前先寫 route

⇒ routes\web.php

⇒ Route::post('/posts', 'PostController@store'); //PostController 中的 store function

Step 10: 先知道資料有沒有送進來，正常是用 HTML 來送，這邊直接用 Postman 來做測試

· 選 Post → localhost:8000/posts → 選擇 Body(確認是 form-data) → 填入下列資料

- title My New Post
- note Something interesting
- author 1

· 這邊先不要按 Send，再回到 controller 繼續寫完

在 Controller 使用 view

- 路徑舉例：

- `view('welcome')`可以存取 `resources/views/welcome.blade.php`
- `view('errors/503')`可以存取 `resources/views/errors/503.blade.php`

Step 02: 來到 PostController.php

```
⇒ app\Http\Controllers\PostController.php
⇒ public function index()
⇒ {
⇒   //return Post::all();
⇒   return view('post');
⇒ }
⇒ localhost:8000/posts
```

Step 03:

```
⇒ resources\views\post.blade.php
⇒ <h1> {{ $title }} </h1>
⇒ app\Http\Controllers\PostController.php
⇒ public function index()
⇒ {
⇒   //return Post::all();
⇒   return view('post', [
⇒     'title' => 'List all my posts',
⇒   ]);
⇒ }
⇒ localhost:8000/posts
```

主版 Extends 的概念，要解決甚麼問題？

- 前端網頁總是有重複的程式碼，例如：
 - Navbar 導覽列按鈕
 - Footer 頁面尾部
- 最大的重複就是頁面本身`<html>`, `<head>`等設定

過去的解法：直接 `include` 的邏輯，概念是把重複且需要一直改的地方分離出來

```
<?php
  include('navbar.php');
?>
```

Blade 樣板語言

- 顯示變數

- 一般變數(自動執行 htmlspecialchars)

- ⇒ {{ \$var }}

- 含有 HTML 的變數(可能造成 XSS 跨站腳本攻擊)

- ⇒ {{!! \$var_html !!}}

// 駭客可能在文章裡面輸入一段惡意的 HTML 和 Javascript 的程式碼，因為用這樣的方法，使用者可能會執行到惡意程式碼，所以不建議這樣用

@if 語法

- ⇒ @if(\$var == true)

- ⇒ {{ \$user->name }}

- ⇒ @endif

@foreach

- 顯示資料庫內容非常好用

- ⇒ @foreach(\$users as \$user)

- ⇒ <tr>

- ⇒ <td>{{ \$user->name }}</td>

- ⇒ </tr>

- ⇒ @endforeach

Step 05: 示範 blade 樣板語言

- php artisan tinker

- ⇒ \$post = new App\Post;

- ⇒ \$post->all()

- app\Http\Controllers\PostController.php

- ⇒ public function index()

- ⇒ {

- ⇒ //return Post::all();

- ⇒ return view('post', [

- ⇒ 'title' => 'List all my posts',

- ⇒ 'posts' => Post::all()

- ⇒]);

- ⇒ }

單元 11 專題實作示範：會員登入及簡易會員資料管理(Blade 做法)

- 本單元先用 Blade 的做法，API 的作法在下一個單元做示範
- 課程目標

- 使用 make:auth 產生登入介面
- 新增一個 User 管理介面，包括
 - 顯示 User(使用 Blade)
 - 新增 User(傳統 Form Post)
 - 刪除 User

Step 01: 產生登入介面

```
⇒ php artisan make:auth  
⇒ git status  
⇒ localhost:8000  
// 頁面右上方有多了 LOGIN, REGISTER
```

```
⇒ php artisan tinker  
⇒ $user = new App\User;  
⇒ $user->all();  
用介面註冊一個新使用者  
⇒ $user->all(); // 確認新的使用者資料有沒有進資料庫
```

Laravel 已經幫我們把登入原始碼做好

放在 vendor\laravel\framework\src\Illuminate\Auth

Login 的原理：在網頁中，打開 Chrome 的開發人員工具，選擇 Application -> Cookies -> laravel_session -> Value (Key, 登入網頁關鍵，輸入帳號時，Laravel 先檢查帳號是否正確，如果正確就會把 session key 記錄下來，下次重新整理時就不用重新登入；如果把 session key 刪掉，就會被強制登入，因為它無法判別我們是登入還是登出

Step 02:User 管理介面[選單](View)

```
⇒ resources\views\layouts\app.blade.php  
⇒ @if(Auth::guest())  
⇒ <li><a href="{{ url('/login') }}>Login</a></li>  
⇒ <li><a href="{{ url('/register') }}>Register</a></li>  
⇒ @else  
⇒ <li><a href="/user">User 管理</a></li>
```

```
use Illuminate\Http\Request;
use App\User;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Hash;
```

```
class UserController extends Controller
{
    public function index()
    {
        $users = User::all();
        return view('user.index', compact('users'));
    }

    public function create()
    {
        return view('user.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email',
            'password' => 'required|confirmed'
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return redirect('/user');
    }

    public function show(User $user)
    {
        return view('user.show', compact('user'));
    }

    public function edit(User $user)
    {
        return view('user.edit', compact('user'));
    }

    public function update(Request $request, User $user)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email',
            'password' => 'nullable|confirmed'
        ]);

        $user->update([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return redirect('/user');
    }

    public function destroy(User $user)
    {
        $user->delete();
        return redirect('/user');
    }
}
```

```
Step 01: 完成 user 控制器
- 在 views/user/index.blade.php 中添加一个删除按钮
- 在 routes/web.php 中添加路由
- 在 UserController 中实现 delete 方法
- 在 User -> destroy 不支持，要写 Query 来完成，逻辑更复杂，所以先在 route 通过 get
```

```
use Illuminate\Http\Request;
use App\User;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\Hash;
```

```
class UserController extends Controller
{
    public function index()
    {
        $users = User::all();
        return view('user.index', compact('users'));
    }

    public function create()
    {
        return view('user.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email',
            'password' => 'required|confirmed'
        ]);

        $user = User::create([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return redirect('/user');
    }

    public function show(User $user)
    {
        return view('user.show', compact('user'));
    }

    public function edit(User $user)
    {
        return view('user.edit', compact('user'));
    }

    public function update(Request $request, User $user)
    {
        $request->validate([
            'name' => 'required',
            'email' => 'required|email',
            'password' => 'nullable|confirmed'
        ]);

        $user->update([
            'name' => $request->name,
            'email' => $request->email,
            'password' => Hash::make($request->password)
        ]);

        return redirect('/user');
    }

    public function destroy(User $user)
    {
        $user->delete();
        return redirect('/user');
    }
}
```

```
⇒ <button type="submit" class="btn btn-primary">  
⇒   Save  
⇒ </button>
```

Step 11: 新增 User 資料---UserController & route

```
routes\web.php  
⇒ Route::post('/user', 'UserController@store');  
  
app\Http\Controllers\UserController.php  
//可以複製 PostController.php 中的 store function  
⇒ use Validator;  
⇒ public function store(Request $request)  
⇒ {  
⇒   $validator = Validator::make($request->all(), [  
⇒     'name' =>'required',  
⇒     'email' =>'required|unique:users',  
⇒     'password' =>'confirmed|required'  
⇒   ]);  
⇒   if ($validator->fails()) {  
⇒     return redirect('/user')  
⇒       ->withErrors($validator)  
⇒       ->withInput();  
⇒   }  
⇒   User::create($request->all());  
⇒   return redirect('/user');  
⇒ }  
  
⇒ localhost:8000/user
```

Step 03: 定義 API route

```
⇒ route\web.php 複製 posts 的 route 到 api.php  
⇒ Route::get('/posts', 'PostController@index');  
⇒ Route::get('/posts/{id}', 'PostController@show');  
⇒ Route::post('/posts', 'PostController@store');
```

```
⇒ route\web.php  
⇒ Route::get('/posts', function() {  
    ⇒ return view('post');  
    ⇒ });  
  
⇒ php artisan route:list  
⇒ 放到 api.php 那邊就會自動接上 api (不會和 web.php 的 view post 衝突)  
⇒ localhost:8000/posts (把<h1>的 title 改成 All my posts , foreach 的部分先註解)  
⇒ localhost:8000/api/posts  
  
⇒ resources\views\post.blade.php  
⇒ $getJSON('/api/posts', function(data) {//把 ajax 加上 api 路徑  
⇒ 開瀏覽器->開發人員工具->Console->localhost:8000/posts 看看有沒有抓到  
posts 的 data  
  
Step 04: 用 jQuery 填表格內容  
⇒ resources\views\post.blade.php  
⇒ <tbody id="tbody">  
  
⇒ @extends('layouts/app')  
⇒ @section('script')  
⇒ <script>  
⇒ $(function() {  
⇒     $getJSON('/api/posts', function(data) {  
⇒         //console.log(data);  
⇒         for(var index in data) {  
⇒             var obj = data[index];  
⇒             console.log(obj.title);  
⇒         }  
⇒     });  
⇒ });  
⇒ </script>
```

Step 06: 實作 view

·複製 post.blade.php 變成 post-single.blade.php

```
⇒ @extends('layouts/app')
⇒ @section('script')
⇒ <script>
⇒ $function() {
⇒   $.getJSON('/posts', function(data) {
⇒     ⇒ });
⇒   });
⇒ </script>
⇒ @endsection

⇒ <div class="col-md-8 col-md-offset-2">
⇒   <div class="panel panel-default">
⇒     <div class="panel-heading">Title</div>
⇒     <div class="panel-body">
⇒       </div>
⇒       </div>
⇒     </div>
⇒   </div>
⇒ localhost:8000/posts/1
```

Step 07: 傳入 id 與製作單一筆資料的介面

- 在 post-single.blade.php
- ⇒ <script>
- ⇒ var post_id = {{\$id}}; //直接用 blade 去把 id 印出來
- 在 web.php 把 id 的值傳給他
- ⇒ Route::get('/posts/{id}', function(\$id) {
⇒ return view('post-single', [
⇒ 'id' => \$id
⇒]);
⇒ });

Step 09: 顯示前 10 筆資料

```
· 到 post.blade.php  
⇒ $getJSON('/posts', function(resp) {  
⇒   for(var index in resp.data) {  
⇒     var obj = resp.data[index];  
⇒     $('#tbody').append('<tr><td>' + obj.id + '</td><td>' + obj.title +  
⇒       '</td></tr>');  
⇒   }  
⇒ });
```

⇒ localhost:8000/posts

Step 10: 製作分頁

```
· 在 post.blade.php  
⇒ </table>  
⇒ <a class="btn btn-primary" id="btn-pre">Previous</a>  
⇒ <a class="btn btn-primary" id="btn-next">Next</a>  
⇒  
⇒ $getJSON('/api/posts', function(resp) {  
⇒   for(var index in resp.data) {  
⇒     var obj = resp.data[index];  
⇒     $('#tbody').append('<tr><td>' + obj.id + '</td><td>' + obj.title +  
⇒       '</td></tr>');  
⇒   }  
⇒   if( resp.next_page_url == null ) {  
⇒     $('#btn-next').hide();  
⇒   } else if( resp.prev_page_url == null ) {  
⇒     $('#btn-pre').hide();  
⇒   }  
⇒   $('#btn-next').attr('href', resp.next_page_url.replace('api/', ''));  
⇒   $('#btn-pre').attr('href', resp.prev_page_url.replace('api/', ''));  
⇒ });
```

單元 16 專題實作示範：商品列表、購物車

課程目標

- 製作一個 AJAX 基礎的簡單商品列表後，嘗試製作購物車
- 簡介製作結帳系統的思路

購物車思路

- 商品顯示頁面
- 加入購物車：將 `product_id` 加入至 `Session`
- 結帳按鈕：連至 `cart` 頁面，印出所有 `session` 內的商品

Session

- 暫存於伺服器的資訊
- 與 `Cookie` 不同：`Cookie` 儲存於瀏覽器端
- PHP 會自動給予客戶端一個 `Session_ID` 存放於 `Cookie` 用以辨識身分
- `Session` 常用於短期記住使用者登入狀態

Session 注意事項

- 如果使用 `$request->session` 出現錯誤訊息：`Session store not set on request`
- 代表你將 `route` 放置於 `api.php` 中
- 唯有 `middleware web` 才能使用 `session`

單元 20 額外補充: vue.js 入門及整合 Laravel

- 本單元將簡介 Vue.js 與 Laravel 5.3 的整合，同時示範如何使用 gulp 編譯 .vue 檔案

安裝相依套件

- 確定已經安裝 npm
- sudo npm install -g gulp-cli
- npm install

甚麼是 Vue.js ? 前端版本的 Blade Template 引擎。

Laravel 5.3 必須使用 Vue.js ?

- 不，Laravel 5.3 只有建議開發者使用 Vue.js
- 透過 npm 安裝後，Laravel 已經幫你把 gulp file 寫好
- 可以透過 webpack 把所有程式碼編譯成 app.js 到前端執行

使用 gulp 打包 vue.js

直接在 html 中撰寫 vue.js

Core Concepts

Service Container

Service Providers

Facades

Contracts

Mail

Notifications

The HTTP Layer

Routing

Middleware

CSRF Protection

Controllers

Requests

Responses

Session

Validation

View & Templates

Views

Blade Templates

Security

Authentication

Authorization

Events

Artisan
Tinker

General Topics

Cache

Events

File Storage

Jobs

Queues

Broadcasting

Scheduler

Database

Query Builder

Pagination

Migrations

Seeding

Redis

Eloquent ORM

Relationships

Collections

Serialization

* 建立專案

```
Composer create-project --prefer-dist laravel/laravel projectName
```

* 建立 Controller

```
php artisan make:controller TaskController --resource
```

* 設立連線資料庫 .env

* 1. 建立 migration file

```
php artisan make:migration create_post_table --create=posts
```

* 2. 進到 migration file 建立欄位

* 3. 實行 migration

```
php artisan migration
```

* 1. 產生 -> post model file (產生 Eloquent 檔案)

```
php artisan make:model Post
```

* 2. 設定 Model 屬性 & 建立關聯

* 3. 測試 Model

```
php artisan tinker
```

* 查看所有 route

```
php artisan route:list
```

1. Create project
 2. Create DB Connection (.env)
 3. Create Table & Migrations
 4. Migration file (coding) → run migration
 5. Create Authentication
 - edit

 6. Create new model
 7. Create new controller → coding
 8. Add Routes
 9. Adding Bootstrap templates
- b. Model 和 route 也可以先都寫好
7. Controller → view → loop
(Displaying data, insert, detail page, update, deleting data, pagination...)
8. Adding Bootstrap templates