

# 驗證與訊息

范聖佑 Shengyou Fan  
新北市樹林國小 (2015/07/11)

```
1  /**
2   * Display the specified resource.
3   *
4   * @param int $id
5   * @return Response
6   */
7  public function show($id)
8  {
9      $post = Post::with('comments')->where('id', $id)->first();
10
11      return View::make('post.show')->with('post', $post);
12  }
```

# 單元主題

- 資料驗證對網路應用程式的重要性
- 在 Laravel 裡如何實作資料驗證？有哪些方式可供我們選擇使用？
- 如何透過跳轉處理資料驗證失敗時的動作？以及如何在頁面上顯示錯誤訊息提供使用者回饋機制？
- 示範如何在專案內實作驗證及跳轉功能

# 資料驗證

# 千萬別相信任何人！

- 撰寫 PHP 程式來接受輸入並將結果輸出是很容易的，但卻無法保證所有來源輸入的資料都是安全的
- 常常聽到有網站被惡意攻擊或是不小心將個資洩漏就是因為沒有做好安全性防護機制
- 因此，為了確保應用程式運作正常且安全，就是絕對不要相信任何從使用者端取得的資料！
- 網路應用程式安全的三大守則：
  - 對輸入消毒
  - 驗證資料
  - 跳脫輸出內容

# 為什麼需要驗證資料？

- 由於無法確保使用者送來的資料一定是正確且符合格式的，甚至無法確定送資料來的使用者都是善良的使用者
- 驗證的重點在於確認輸入的資料是符合預期，並確保最終儲存的資料是精確且格式正確的
- 假如接收到的資料是無法通過驗證的，就可以暫停原訂的動作並將錯誤訊息顯示給使用者

# 傳統的驗證方式

- 條件組合大法：
  - 檢查是不是空值 `empty()`、`is_null()`、`isset()`
  - 檢查字串長度 `strlen()`、`mb_strlen()`、`sizeof()`
  - 上網找「`php email/name/phone validate regex`」 然後用 `preg_match()`、`ereg()`

```
// 檢查 username 是否為空值、長度超過 3 個字元且只有大小寫字母
if(!empty($_POST['username']) &&
    strlen($_POST['username']) > 3 &&
    preg_match("/^[a-zA-Z]*$/", $_POST['username']))
{
    // 通過驗證！
}
```

# PHP 內建的驗證函式

- PHP 提供 `filter_var()` 驗證函式並搭配數個以 `FILTER_VALIDATE_*` 開頭的指標供我們做資料驗證
- 雖然 `filter_var()` 提供了數種驗證方式，但它無法驗證所有東西

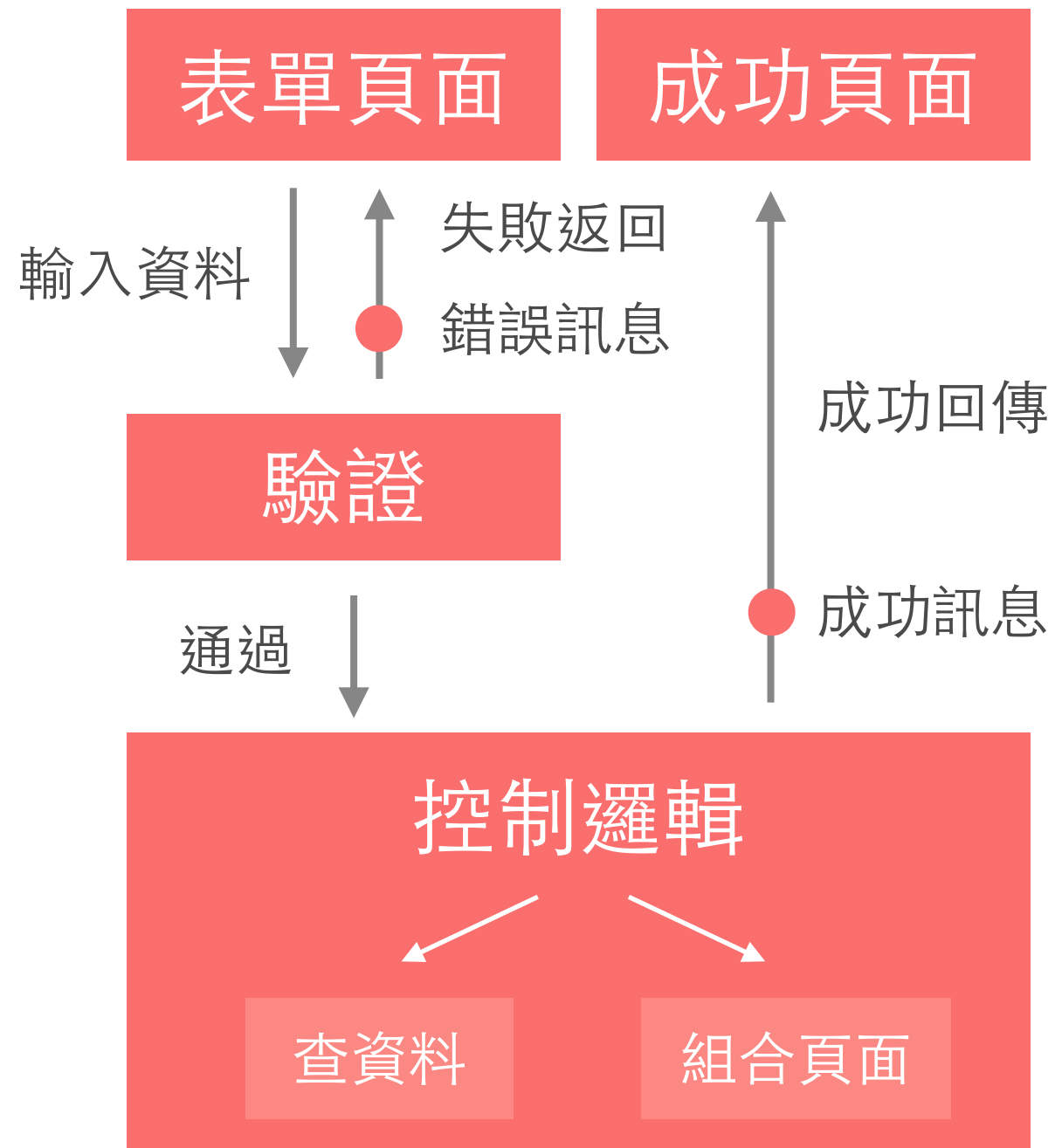
```
// 使用 filter_var() 驗證 Email
$email = $_POST['email'];
$isEmail = filter_var($email, FILTER_VALIDATE_EMAIL);
if ($isEmail !== false) {
    /* 假如驗證通過則會回傳原本的變數內容 */
    /* 假如驗證錯誤則會回傳 false */
    // 通過驗證！
}
```

# Laravel 的驗證機制

- Laravel 內建提供了三種方式供我們驗證資料，可依不同的需求選擇：
  - Validator Facade
  - Controller Validation
  - Form Request Validation



# 表單驗證流程



# Validator Facade

- Laravel 內建提供 Validator 類別，可直接透過 Facade 呼叫，就可以快速的實作驗證機制

```
// 使用 Validator Facade 進行驗證
$validator = Validator::make(
    [
        'username' => 'tester',
        'email' => 'tester@example.com'
    ],
    [
        'username' => 'required|min:3|max:20',
        'email' => 'required|email|unique:users'
    ]
);
if ($validator->fails())
{
    // 沒有通過驗證，跳轉回前一頁，並把錯誤訊息帶過去
    return redirect()->back()->withErrors($validator->errors());
}
```

# Controller Validation

- Laravel 的 BaseController 已經幫我們加好 ValidatesRequests 的 Trait，只要直接使用 validate 方法就可以進行驗證
- 若驗證失敗，就會自動跳轉回前一頁，並把錯誤訊息帶過去，完全自動化！

```
// 在 Controller 裡儲存資料時驗證資料
public function store(Request $request)
{
    $this->validate($request, [
        'username' => 'required|min:3|max:20',
        'email' => 'required|email'
    ]);

    // 以下略
}
```

# Form Request Validation

- 嚴格來說，表單驗證不是 Controller 要做的事，更好的作法是將其獨立成一個類別來專責處理
- 除了單純的資料驗證外，還需要更複雜的身份驗證
- Laravel 5 提供了新的 Form Request 驗證方式，讓驗證動作可以完全與 Controller 分開，而流程和動作一樣自動化完成
- artisan 也新增了 `make:request` 指令，讓產生 Form Request 類別檔案也能自動化

```
$ [php] artisan make:request \
    {name}
```

產生 form request 檔案

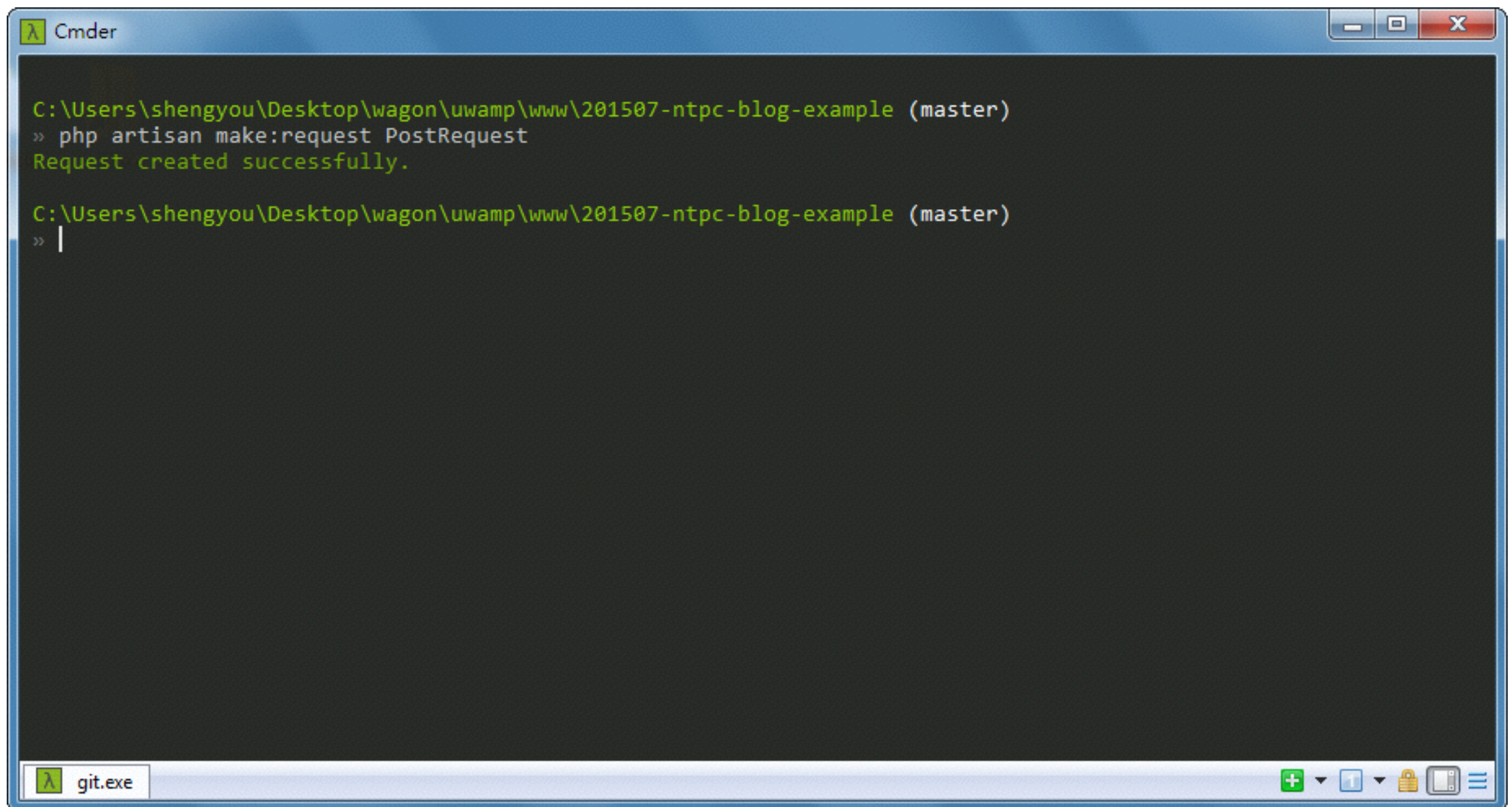
# artisan make:request

- 透過 `artisan` 產生 Form Request 類別
  - `artisan` 會依照給予的名稱，產生類別檔案
- 範例：

```
$ php artisan make:request PostRequest
```

# 產生 Form Request 檔案

使用 artisan 產生 Form Request 檔案



```
C:\Users\shengyou\Desktop\wagon\uwamp\www\201507-ntpc-blog-example (master)
>> php artisan make:request PostRequest
Request created successfully.

C:\Users\shengyou\Desktop\wagon\uwamp\www\201507-ntpc-blog-example (master)
>> |
```

The screenshot shows a Windows Command Prompt window titled 'Cmder'. The command prompt is at the directory `C:\Users\shengyou\Desktop\wagon\uwamp\www\201507-ntpc-blog-example (master)`. The user has entered the command `php artisan make:request PostRequest`, and the output is `Request created successfully.`. The prompt is now waiting for the next command, indicated by `>> |`. The taskbar at the bottom shows the 'git.exe' application is running.

# 命名與指令慣例

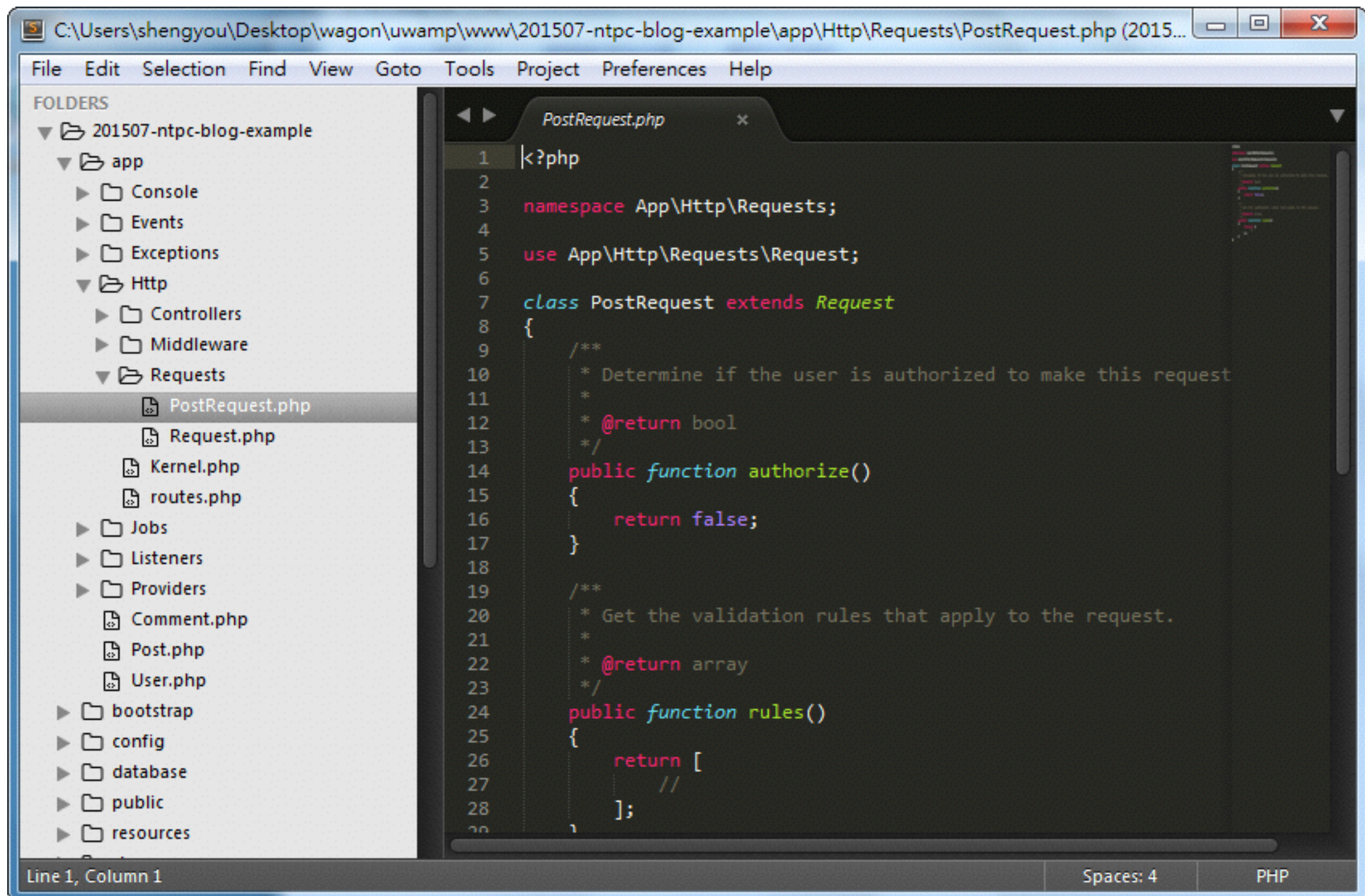
- 一般慣例會將 Form Request 以 {大駝峰資源單數}Request 來命名，如 PostRequest
- 預設 Form Request 檔案會放置在 `app/Http/Requests/` 資料夾底下



# Form Request 檔結構

- 一個 Form Request 檔裡一定實作兩個 method：
  - `authorize` (授權)：在這個 method 裡要回傳 `true` 或 `false`，至於如何判定回傳值為何？就可以在這個 method 裡實作
  - `rules` (驗證規則)：在這個 method 裡回傳一個陣列，這個陣列就跟 `Validator::make` 的第二個參數一樣，把想要驗證的規則寫在其中即可

# Form Request 檔範例



The screenshot displays a code editor window with the following content:

**File Explorer (Left Sidebar):**

- 201507-ntpc-blog-example
  - app
    - Console
    - Events
    - Exceptions
    - Http
      - Controllers
      - Middleware
      - Requests
        - PostRequest.php
        - Request.php
        - Kernel.php
        - routes.php
      - Jobs
      - Listeners
      - Providers
        - Comment.php
        - Post.php
        - User.php
      - bootstrap
      - config
      - database
      - public
      - resources

**Code Editor (Main Area):**

```
1 <?php
2
3 namespace App\Http\Requests;
4
5 use App\Http\Requests\Request;
6
7 class PostRequest extends Request
8 {
9     /**
10      * Determine if the user is authorized to make this request
11      *
12      * @return bool
13      */
14     public function authorize()
15     {
16         return false;
17     }
18
19     /**
20      * Get the validation rules that apply to the request.
21      *
22      * @return array
23      */
24     public function rules()
25     {
26         return [
27             //
28         ];
29     }
30 }
```

**Status Bar (Bottom):**

- Line 1, Column 1
- Spaces: 4
- PHP

# authorize() 實作方式

- 可以在 method 內實作自己的授權邏輯，確認使用者取得授權才能再執行 validate
- 預設為 `false`，若暫時不需要身份認證，可以強制回傳 `true`

// 在 form request 裡驗證身份

```
public function authorize()  
{
```

```
    // 假設 app/Http/routes.php 裡寫
```

```
    // Route::post('comment/{id}');
```

```
    // 則可從 route 拿到 comment id 的值
```

```
    $id = $this->route('id');
```

```
    // 檢查這個 comment 是不是該 user 新增的？
```

```
    return Comment::where('id', $id)
```

```
        ->where('user_id', \Auth::id())->exists();
```

```
}
```

# rules() 實作方式

- 在 `rules()` 裡依照 Laravel 提供的內建驗證規則設定

```
// 在 form request 裡設定規則
public function rules()
{
    return [
        'username' => 'required|min:3|max:20',
        'email' => 'required|email',
        'title' => 'required|max:255',
        'body' => 'required',
    ];
}
```

# Laravel 提供的驗證規則

- Laravel 提供了為數不少的內建驗證規則，以下列出常使用的數個範例：
  - `required, accepted, same`
  - `string, boolean, integer, numeric, array, date`
  - `min, max, between`
  - `email, url, ip`
  - `unique, exists`

# Controller 內的實作

- 由於表單驗證的事情全部由 Form Request 類別處理，因此在 Controller 裡只需要透過 **type-hinting** 把對應的 Form Request 呼叫進來，剩下的事全自動化！

```
// 在 Controller 裡呼叫 Form Request
<?php namespace App\Http\Controllers;

use App\Http\Requests\PostRequest;

public function store(PostRequest $request)
{
    // 只要能夠執行到這一行就表示通過驗證了！
}
```

# 表單錯誤訊息

# Laravel 的表單錯誤訊息

- 在 Laravel 的預設動作裡，執行完驗證後，只要沒有通過，Laravel 就會返回上一個頁面，並把錯誤訊息用 Session 傳遞過去
- 而在任一頁面載入的時候，Laravel 會自動的去檢查 Session 裡有沒有錯誤訊息？只要有錯誤訊息，Laravel 會自動把錯誤訊息放在 `$errors` 這個變數裡給 View 使用 (也就是說不用手動在 Controller 裡把錯誤訊息從 Session 拿出來再送到 View 去)
- `$errors` 這個變數是 `MessageBag` 類別的實體，只要呼叫這個類別對應的方法，就可以將錯誤訊息印出來



# 查詢是否有錯誤訊息？

- 查詢目前的頁面是否有錯誤訊息？有的話就印 Bootstrap 的 Alert Component

```
// resources/views/*.blade.php
// 在 View 裡確認是否有錯誤訊息？
@if ($errors->any())
<div class="alert alert-danger alert-dismissible" role="alert">
    <button type="button" class="close"
        data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
    <strong>錯誤！</strong> 請修正您輸入的資料
</div>
@endif
```

# 取得欄位的錯誤訊息

- 取得欄位的所有錯誤訊息

```
// resources/views/*.blade.php
// 取得 email 的錯誤訊息
{{ $errors->get('email') }}
```

- 取得欄位的第一個錯誤訊息

```
// resources/views/*.blade.php
// 印出 email 的第一個錯誤訊息
{{ $errors->first('email') }}
```

```
// 印出 email 的第一個錯誤訊息，並依格式輸出
{!! $errors->first('email', '<p>:message</p>') !!}

```

# 將錯誤訊息印出來

- 把欄位所有的錯誤訊息印出來

```
// resources/views/*.blade.php
// 印出 email 的所有錯誤訊息
@foreach($errors->get('email') as $error)
    {{ $error }}
@endforeach
```

- 把表單所有的錯誤訊息印出來

```
// resources/views/*.blade.php
// 印出表單的所有錯誤訊息
@foreach($errors->all() as $error)
    {{ $error }}
@endforeach
```

# Model 錯誤處理

# 回傳 404

- 先檢查資料存不存在？假如不存在的話就顯示 404  
Page not found

```
// app/Http/Controllers/PostsController.php
public function show($id)
{
    $post = \App\Post::find($id);

    if (is_null($post))
    {
        abort(404);
    }

    // 以下略
}
```

# 使用 findOrFail()

- Eloquent 有一個方法叫 `findOrFail`，若 Eloquent 找不到對應的資料的話，會拋出 `ModelNotFoundException`

```
// app/Http/Controllers/PostsController.php
public function show($id)
{
    $post = \App\Post::findOrFail($id);

    // 以下略
}
```

# 使用 Session

- 也可以自行處理該錯誤發生時的動作，增加額外的提示給使用者

```
// app/Http/Controllers/PostsController.php
public function show($id)
{
    $post = \App\Post::find($id);

    if (is_null($post))
    {
        return redirect()->route('posts.index')
            ->with('message', '找不到該文章');
    }

    // 以下略
}
```

# 在 View 裡顯示 Session 訊息

- 在 Controller 裡定義好要放入 Session 的訊息內容後，就可以在 View 裡將其取回並顯示在頁面上

```
// resources/views/*.blade.php
@if (session('message'))
<div class="alert alert-danger alert-dismissible"
role="alert">
    <button type="button" class="close"
        data-dismiss="alert" aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
    <strong>錯誤！</strong> {{ session('message') }}
</div>
@endif
```



# 實作專案驗證機制

# 建立 Form Request

- 依照專案實作需求，透過 `make:request` 建立兩個 Form Request 類別
- 在這兩個 Form Request 類別內
  - 強制將 `authorize()` 設為 `true`
  - 設定 `rules()` 裡對應的規則
- 設定 Controller 引入對應的 Form Request

# 設定頁面顯示訊息

- 將以下兩種結果的訊息顯示在對應的頁面上：
  - 表單驗證失敗
  - 資料儲存成功
  - 資料更新成功

# 單元總結

- 在這個單元裡我們學到了些什麼？
  - 透過三種方式在 Laravel 裡實作資料驗證
  - 運用跳轉機制來重導頁面，並在頁面顯示錯誤訊息提示使用者
  - 完成專案應用程式內的驗證及跳轉機制



歡迎提問討論