

Laravel

8

Saac Sornchai

Copyright © 2021 Saac Sornchai

Copying prohibited

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, without the prior written permission of the publisher.

Art. No xxxxx

ISBN xxx-xx-xxxx-xx-x

Edition 1.0

Cover design by Saac Sornchai

Published by -

Printed in Bangkok, Thailand



การพัฒนาเว็บแอปพลิเคชัน ด้วย **Laravel 8** เป็นหนังสือที่จัดทำขึ้นจากประสบการณ์ของผู้เขียนที่หลงใหลในการพัฒนาเว็บแอปพลิเคชัน และบอกเล่าเป็นบันทึกให้ผู้ที่หลงทางมาอ่าน และหวังเป็นอย่างยิ่งว่าใครก็ตามที่ได้อ่านหนังสือเล่มนี้จะหลงใหลการพัฒนาเว็บแอปพลิเคชันเช่นเดียวกับผู้เขียน

การเรียงลำดับเนื้อหาในหนังสือเล่มนี้จะสอดคล้องกับการเรียนการสอนในวิชา **01418442 Web Technology and Web Services** ซึ่งเป็นหนึ่งในวิชาเฉพาะเลือกของหลักสูตรวิทยาศาสตรบัณฑิต พ.ศ. 2560 สาขาวิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยเกษตรศาสตร์ โดยคาดหวังไว้ว่าเนื้อหาในหนังสือจะสมบูรณ์ ณ ภาคต้น ปีการศึกษา 2564

Saac Sornchai

May 2021



1	ภาษาของ Laravel	7
1.1	PHP Standard Recommendation (PSR)	7
1.2	รู้จักกับ Composer	7
1.3	การใช้งาน Composer เบื้องต้น	8
2	Laravel	12
2.1	การติดตั้ง Laravel Project	12
2.2	ข้อควรจำ	13
2.3	การกำหนดค่าเริ่มต้นของ Laravel Project	14
3	Laravel Routing	15
3.1	สิ่งที่ต้องคำนึงในการกำหนด Routing	15
3.2	Route: Required Parameters	17
3.3	Route: Optional Parameters	18
4	Laravel Blade Templates	20
4.1	Template Inheritance	20
4.2	การแสดงผลข้อมูลใน Blade Templates	24
4.3	Blade Control Structure	28

5	Laravel Controller	30
5.1	การสร้าง Controller	30
6	Laravel Migration	35
6.1	การสร้างไฟล์ Migration	35
6.2	การรัน Migrate	37
6.3	การปรับแก้ Migration	37
6.4	การแก้ไขข้อผิดพลาดของการ Migrate	38
7	Laravel Model	41
7.1	การสร้าง Model	41
7.2	Laravel Tinker	42
8	CRUD	44
8.1	การใช้ Model เพื่อเพิ่มข้อมูลในตาราง (Create)	44
8.2	การใช้ Model เพื่อเรียกดูข้อมูลในตาราง (Retrieve)	45
8.3	การใช้ Model เพื่อแก้ไขข้อมูลในตาราง (Update)	46
8.4	การใช้ Model เพื่อลบข้อมูลในตาราง (Delete)	46
9	Laravel Seeder (Database Seeding)	48
9.1	การสร้างคลาส Seeder	48
9.2	การ seed ข้อมูล	49
9.3	Laravel Factory	50
9.4	การใช้งาน Factory	51
10	Laravel Resource Controller	54
10.1	การสร้าง Resource Controller	55
10.2	Validation	63
10.3	การแสดงความแข็งแรงความผิดพลาดจากการตรวจสอบความถูกต้อง	64
11	Laravel Relationship	66
11.1	การเพิ่ม Foreign Key ใน Migration	66

11.2	การกำหนดความสัมพันธ์ One to Many	67
11.3	การเพิ่มข้อมูลในความสัมพันธ์ One to Many	68
11.4	การดึงค่าจากความสัมพันธ์ One to Many	69
11.5	การกำหนดความสัมพันธ์ Many to Many	69
11.6	การเพิ่มข้อมูลในความสัมพันธ์ Many to Many	73
12	Laravel Authentication	75
12.1	การใช้งาน Laravel Breeze	75
12.2	การใช้งาน Auth	76
13	Laravel Authorization	79
13.1	การกำหนด Gate	79
13.2	การใช้งาน Gate	80
13.3	การกำหนด Policy	81
13.4	การใช้งาน Policy	82
14	Laravel RESTful Service	84
14.1	การสร้าง API Controller	84
14.2	API Resource	85
14.3	JSON Web Token	87
14.4	Laravel JWT Authentication	90

1. ภาษาของ Laravel

1.1	PHP Standard Recommendation (PSR)	7
1.2	รู้จักกับ Composer	7
1.3	การใช้งาน Composer เบื้องต้น	8

ในบทนี้จะแนะนำภาษาที่ใช้ในการพัฒนา *Laravel Framework* และเครื่องมือจัดการแพ็คเกจที่ชื่อ *Composer* แบบเบื้องต้น

1.1 PHP Standard Recommendation (PSR)

PSR เป็นมาตรฐานการเขียนโค้ดภาษา PHP ที่กำหนดขึ้นโดย PHP Framework Interop Group (PHP-FIG) เพื่อให้การเขียนโค้ดระหว่างโปรเจกต์หรือระหว่างแพ็คเกจทำงานร่วมกันได้

PSR-4 เป็นหัวข้อลำดับหมายเลข 4 ในมาตรฐาน PSR ซึ่งนำ Autoload มาช่วยในการเรียกใช้คลาสที่เป็น namespace ด้วยคำสั่ง `use namespace\YourClass` จากเดิมที่ต้องเรียกใช้คลาสด้วยคำสั่ง `require "dir/YourClass.php"` และการเขียนโค้ดตามมาตรฐาน PSR-4 ต้องใช้ PHP เวอร์ชัน 5.3 ขึ้นไป (ปัจจุบัน พฤษภาคม 2564 เป็นเวอร์ชัน PHP 7.4¹ และ Composer

PSR หัวข้ออื่น ศึกษาเพิ่มเติมได้ที่ <https://www.php-fig.org/psr/>

1.2 รู้จักกับ Composer

Composer เป็นเครื่องมือช่วยจัดการแพ็คเกจหรือไลบรารีของภาษา PHP รวมถึงช่วยจัดการโปรเจกต์ที่เขียนขึ้นเองได้ด้วย การติดตั้ง Composer ให้ดูที่ <https://getcomposer.org/download>

¹PHP: Supported Versions. Retrieved from <https://php.net/supported-versions.php>

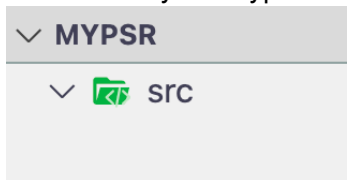
สำหรับผู้ที่ใช้ระบบปฏิบัติการ Windows แนะนำให้ลงโปรแกรม Laragon โดยสามารถดาวน์โหลดได้ที่ <https://laragon.org/download> เลือก Edition Laragon Full ซึ่งใน Laragon มี Terminal ที่สามารถใช้ Composer ได้



Laragon เป็นโปรแกรมจำลองสภาพแวดล้อมการทำงานของเครื่องคอมพิวเตอร์เป็น *development environment* ซึ่งรองรับการเขียนเว็บแอปพลิเคชันเพราะมี *Web Server*, *Database Server* และรองรับภาษาในการเขียนเว็บแอปพลิเคชันหลายภาษา รวมถึงมี *Composer* ติดตั้งมาด้วย

1.3 การใช้งาน Composer เบื้องต้น

1. สร้าง directory ชื่อ mypsr ใน Root Document และสร้าง directory ย่อยชื่อ src



2. สร้างไฟล์ composer.json ใน directory mypsr (ไม่ใช่ใน directory src) โดยมีโค้ดดังนี้

```

1 {
2     "autoload": {
3         "psr-4": {
4             "App\\": "src/"
5         }
6     }
7 }
```

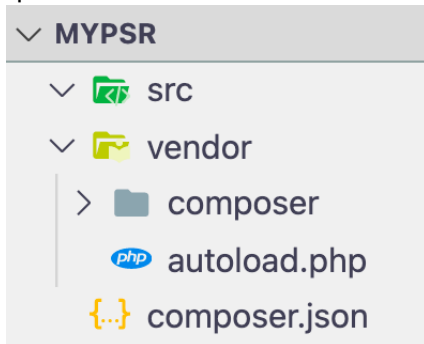
3. ใช้ Command Line Interface Tool (ต่อไปจะเรียกว่า terminal) เปลี่ยน current directory ไปที่ mypsr (เพื่อเข้าถึงไฟล์ composer.json) จากนั้นให้ Composer สร้าง script สำหรับการใช้งาน Autoload ด้วยคำสั่ง

```
composer dump-autoload
```



ทุกครั้งที่มีการแก้ไขไฟล์ *composer.json* (เช่น เพิ่ม *namespace* ให้ *directory* ใหม่) จะต้องสั่ง *composer dump-autoload* ใหม่เสมอ

4. สังเกตว่าจะมี directory ชื่อว่า vendor ซึ่งในนั้นจะมีไฟล์ autoload.php และมี directory ชื่อ composer



ไม่ควรลบไฟล์หรือแก้ไขเนื้อหาใด ๆ ใน directory vendor



directory vendor ควรถูก ignore ในการใช้งาน git ดังนั้น เมื่อ clone repository หรือ pull งานจาก repository มาแล้ว ถ้ามีการเปลี่ยนแปลงใน composer.json ให้ใช้คำสั่ง `composer dump-autoload` ก่อนทำงาน



5. สร้างตัวอย่างคลาส Foo ในไฟล์ src/Foo.php

```
1 <?php
2 namespace App;
3
4 class Foo {
5     public function bar($message) {
6         return "Hello {$message}";
7     }
8 }
```

สามารถสร้าง directory ย่อย ใน directory ที่กำหนด namespace ไว้แล้ว โดยไม่ต้องสั่ง `dump-autoload` เช่น สร้างคลาส FooBar ในไฟล์ src/Utilities/Helper/FooBar.php

โดยกำหนด namespace ของคลาส FooBar เป็น App\Utilities\Helper



6. สร้างไฟล์ index.php ใน directory mypsr

```
1 <?php
2 require 'vendor/autoload.php';
3
4 use App\Foo;
5
6 $foo = new Foo();
7 echo $foo->bar('PSR-4');
```

สังเกตว่า namespace ใช้ backslash (\) เป็นตัวคั่น ไม่ใช่ slash (/) และไม่มี .php



ไฟล์ที่เขียนคลาสในภาษา *PHP* จะต้องมีชื่อเดียวกับชื่อคลาส

7. ทดสอบการรันไฟล์ `index.php` ด้วยคำสั่ง

```
php index.php
```

สังเกตผลลัพธ์ที่ได้ คือ Hello PSR-4

Exercises

Exercise 1.1 หากในไฟล์ `composer.json` มีการกำหนดข้อมูลดังนี้

```
1 {
2     "autoload": {
3         "psr-4": {
4             "App\\": "src/",
5             "Helper\\": "lib/helper/"
6         }
7     }
8 }
```

ให้บอกชื่อไฟล์ของ namespace ต่อไปนี้ (ระบุ path อ้างอิงจาก mypsr)

- (a) `App\Controller`
- (b) `App\Models\Article`
- (c) `App\Http\Controllers\ArticlesController`
- (d) `Helper\Facade\Auth`
- (e) `App\Helper\Middleware\CsrfVerify`

Exercise 1.2 จาก Exercise 1.1 หากคลาส `ArticlesController` (ข้อ c) สืบทอดจากคลาส `Controller` (ข้อ a) ไฟล์ `ArticlesController.php` จะมีโค้ดอย่างไร

Problem

Problem 1.1 กรณีใดบ้างที่จะใช้คำสั่ง `composer dump-autoload`

Problem 1.2 คำสั่งเกี่ยวกับ `composer` ที่น่าสนใจ มีคำสั่งอะไรอีกบ้าง แต่ละคำสั่งใช้ในสถานการณ์ใด

Programming Exercises

Programming Exercise 1.1 เขียนโปรแกรมเพื่อออกแบบคลาสสำหรับการสุ่มค่า โดยมี method สำหรับการสุ่มทั้ง 2 รูปแบบดังนี้

(1) สุ่มตัวเลขจำนวนเต็ม โดยผู้ใช้สามารถกำหนดค่า **argument** ดังนี้

(1.1) หากกำหนด **argument** ค่าเดียว หมายถึง สุ่มเลขตั้งแต่ 0 จนถึงค่า **argument** จำนวน 1 ค่า

(1.2) หากกำหนด **argument** 2 ค่า หมายถึง สุ่มเลขตั้งแต่ค่า **argument** แรก จนถึงค่า **argument** ที่สอง จำนวน 1 ค่า

(1.3) หากกำหนด **argument** 3 ค่า หมายถึง สุ่มเลขตั้งแต่ค่า **argument** แรก จนถึงค่า **argument** ที่สาม ตามจำนวน **argument** ที่สาม โดยถ้า **argument** ที่สามมีค่าเป็น 1 ให้สุ่มเลขเพียงตัวเดียว แต่หาก **argument** ที่สามมีค่ามากกว่า 1 ให้คืนค่าเลขที่สุ่มเป็น **array**

(2) สุ่มข้อความจากตัวอักษร A ถึง Z และ 0 ถึง 9 โดยให้ผู้ใช้กำหนดความยาวของข้อความ และจำนวนข้อความได้ เช่น ผู้ใช้ต้องการข้อความสุ่มขนาด 10 ตัวอักษร จำนวน 10 ข้อความ

2. Laravel

2.1	การติดตั้ง Laravel Project	12
2.2	ข้อควรจำ	13
2.3	การกำหนดค่าเริ่มต้นของ Laravel Project	14

Laravel เป็น Framework ของภาษา PHP สำหรับการพัฒนาเว็บแอปพลิเคชัน ซึ่งพัฒนาโดย Taylor Otwell

เวอร์ชันปัจจุบัน (พฤษภาคม 2564) คือ 8.40.0 และใช้งาน Composer เพื่อจัดการไลบรารีที่ใช้งานใน framework ทั้งหมด

Server Requirements¹

- PHP \geq 7.3
- Extensions: BCMath, CType, Fileinfo, JSON, Mbstring, OpenSSL, PDO, Tokenizer, XML

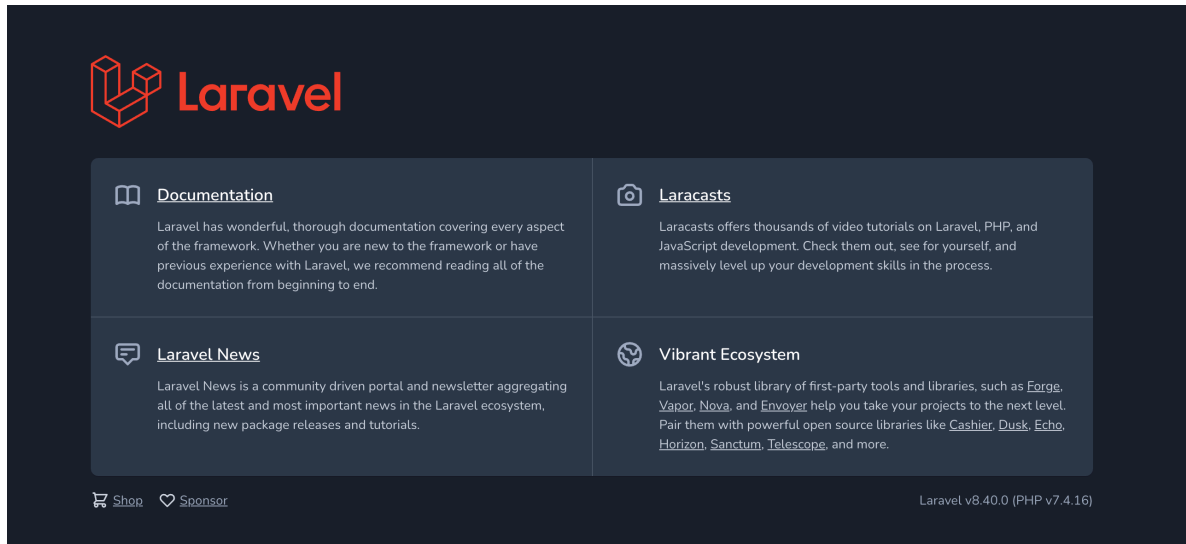
2.1 การติดตั้ง Laravel Project

1. ติดตั้ง Laravel Installer เพื่อใช้งานคำสั่ง `laravel` ได้ โดยใช้ terminal พิมพ์คำสั่ง `composer global require laravel/installer`

2. สร้าง Laravel Project ชื่อ `myapp` ด้วยคำสั่ง `laravel new myapp` แล้วรอให้เสร็จสิ้น

3. ทดลองว่ามี Laravel Project ที่สมบูรณ์ โดยเปลี่ยน `current directory` ไปที่ `myapp` แล้วจำลอง `server` ด้วยคำสั่ง `php artisan serve` จากนั้นใช้ `browser` เข้าไปที่ `http://127.0.0.1:8000`

¹<https://laravel.com/docs/8.x/deployment#server-requirements>



2.2 ข้อควรจำ

1. การใช้งาน **framework** ตัวใดก็ตาม ควรใช้งานตาม **documentation** ที่เป็นทางการ เพื่อให้ผู้ร่วมพัฒนาเข้าใจตรงกัน
2. **framework** เกิดจากการทำงานของไลบรารีหลายตัวร่วมกัน ข้อผิดพลาดส่วนใหญ่จึงเกิดจากการตัวผู้พัฒนาเอง เช่น เขียนโค้ดผิด หรือส่งค่าไปเรียกฟังก์ชันผิด ซึ่งข้อผิดพลาดที่แสดงนั้นมักจะบอกว่าเกิดขึ้นในไฟล์ที่ผู้พัฒนาไม่ได้สร้าง ดังนั้นอย่าลปไฟล์หรือแก้ไขไฟล์ที่ตนเองไม่ได้สร้างเด็ดขาด เพราะไฟล์ของไลบรารีถูกทดสอบด้วย **unit test** แล้ว ควรหาข้อมูลก่อนว่าข้อผิดพลาดเกิดขึ้นจากอะไร และควรรันทดสอบอยู่เสมอ เพื่อให้รู้ว่าครั้งล่าสุดที่ไม่มีข้อผิดพลาดเกิดขึ้น หลังจากนั้นเขียนโค้ดอะไร
3. การ **deploy Laravel Project** ให้แก้ไขไฟล์ **.env** โดยเปลี่ยนค่า **APP_ENV=production** และ **APP_DEBUG=false** การตั้งค่าต่าง ๆ ให้ตั้งค่าที่ไฟล์ **.env** เท่านั้น
4. ไม่ควร **push** ไฟล์ **.env** ไปที่ **git server**
5. **Laravel project** ที่ **clone** จาก **git server** ก่อนจะพัฒนาต่อ ให้สั่ง **composer install** และ **composer dump-autoload**
6. กำหนด **document root** มาที่ **directory public/** เพื่อไม่ให้เข้าถึง **source code** และการตั้งค่าของ **configuration files** หรือ **environment file** ในโปรเจก

2.3 การกำหนดค่าเริ่มต้นของ Laravel Project

การกำหนดค่าต่าง ๆ มักจะกำหนดในไฟล์ `.env` เป็นส่วนใหญ่ เช่น การเชื่อมต่อฐานข้อมูล การตั้งค่า mail driver หรือการตั้งค่า session แต่มีการตั้งค่าบางส่วนที่ต้องไปแก้ไขในไฟล์ `config/app.php` ดังนี้

timezone กำหนดเป็น Asia/Bangkok

provider หากมีการสร้าง Provider ขึ้นเอง

alias หากต้องการกำหนด Facade ของคลาส

Exercises

Exercise 2.1 ให้เปิดดูไฟล์ต่าง ๆ ใน Laravel project และสังเกตว่าจะมีการเขียน comment ไว้ comment ของ Laravel framework จะมีลักษณะเฉพาะคือ มี 3 บรรทัด บรรทัดแรกจะมีจำนวนตัวอักษรมากกว่าบรรทัดที่สองอยู่ 3 ตัว และบรรทัดที่สองจะมีจำนวนตัวอักษรมากกว่าบรรทัดที่สามอยู่ 2 ตัว

Exercise 2.2 Laravel project กำหนด namespace ไว้อย่างไร สังเกตที่ใด

Exercise 2.3 คำสั่ง `php artisan` จะแสดงคำสั่ง `artisan` ทั้งหมด ให้ลองดูว่ามีคำสั่ง `artisan` อะไรบ้าง และอ่านด้วยว่าแต่ละคำสั่งใช้ทำอะไร ในคำอธิบายคำสั่งดังกล่าว รู้จักคำใดบ้าง และไม่รู้จักคำใดบ้าง

Exercise 2.4 การใช้งานคำสั่ง `artisan` สามารถดูความช่วยเหลือได้ด้วย option `--help` เช่น `php artisan serve --help` ถ้าต้องการจะรัน development server ด้วย port 80 จะต้องใช้คำสั่ง `artisan` อย่างไร

Problem

Problem 2.1 การกำหนด document root มาที่ directory `public/` ทำอย่างไร ใน Apache

Problem 2.2 การกำหนด document root มาที่ directory `public/` ทำอย่างไร ใน Nginx

3. Laravel Routing

3.1	สิ่งที่ต้องคำนึงในการกำหนด Routing	15
3.2	Route: Required Parameters	17
3.3	Route: Optional Parameters	18

Routing เป็นการกำหนดวิธีการประมวลผลเมื่อมี *request* (*URL*) เข้ามายัง *server* แล้วส่ง *response* กลับไป ซึ่งต้องกำหนด *route* ในไฟล์ *routes/web.php*

3.1 สิ่งที่ต้องคำนึงในการกำหนด Routing

1. Request ที่เข้ามา เป็น HTTP Method ใด ซึ่งจะกำหนดชื่อฟังก์ชันของ Routing ได้แก่

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);  
Route::options($uri, $callback);
```

2. Request ที่เข้ามา มาจาก URL path อะไร ซึ่งจะกำหนดเป็น parameter ที่ 1 (*\$uri*)

หากเป็นคำที่ประกอบด้วยหลายคำ ให้ใช้ - เป็นตัวคั่น¹ เช่น /about-us

3. จะ response กลับด้วยวิธีใด ซึ่งจะกำหนดเป็น parameter ที่ 2 (*\$callback*)

¹Keep a simple URL structure. Retrieve from <https://developers.google.com/search/docs/advanced/guidelines/url-structure>

ตัวอย่างการ response ด้วย closure ที่ return String

```
Route::get('/hello', function () {  
    return "Hello laravel";  
});
```

← → ↻ ⓘ 127.0.0.1:8000/hello

Hello Laravel

ตัวอย่างการ response ด้วย closure ที่ return Associative Array ซึ่งจะเปลี่ยนเป็น JSON

```
Route::get('/hello/array', function () {  
    return [  
        'message' => 'Hello Laravel',  
        'status' => 'success'  
    ];  
});
```

← → ↻ ⓘ 127.0.0.1:8000/hello/array

{"message":"Hello Laravel","status":"success"}

ตัวอย่างการ response ด้วย closure ที่ return view()

โดยจะต้องระบุชื่อไฟล์ .blade.php ที่อยู่ใน resources/views เป็น parameter ของฟังก์ชัน view()

```
Route::get('/hello/array', function () {  
    return view('welcome');  
});
```


3.2 Route: Required Parameters

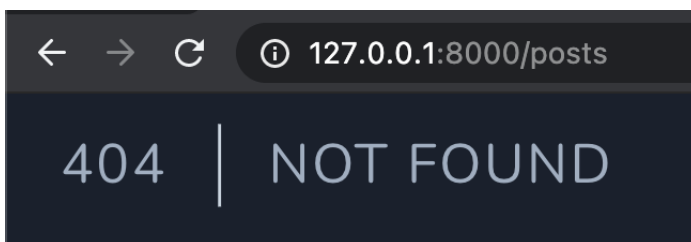
สามารถกำหนด parameter ของ request URL path ได้ โดยระบุชื่อเป็น { } ใน URL path และระบุ parameter name ใน Closure function ด้วย เช่น ต้องการ request /posts/10 หรือ /posts/abc โดย 10 หรือ abc เป็น parameter

```
Route::get('/posts/{id}', function ($id) {  
    return [  
        'id' => $id  
    ];  
});
```



ชื่อที่ระบุใน { } จะต้องเป็นตัวอักษรประเภท alphabetic และไม่ใช่ - ให้เลียงไปใช้ underscore (_)

เมื่อกำหนด required parameter แล้ว แต่ request URL path ไม่ระบุ parameter จะ response 404 Not Found



3.3 Route: Optional Parameters

กรณีที่ parameter ของ request URL path เป็น optional ให้ใส่ ? หลังชื่อที่ระบุใน {} ของ URL path และกำหนด default value ของ parameter ใน Closure function เช่น

```
Route::get('/posts/{id}/{name?}', function ($id, $name = null) {
    return [
        'id' => $id,
        'name' => $name
    ];
});
```

← → ↻ ⓘ 127.0.0.1:8000/posts/123/Saac

```
{"id": "123", "name": "Saac"}
```

← → ↻ ⓘ 127.0.0.1:8000/posts/123

```
{"id": "123", "name": null}
```

Routing รูปแบบอื่น ศึกษาได้ที่ <https://laravel.com/docs/8.x/routing>

คำสั่งตรวจสอบ Routing ทั้งหมด ของ Laravel project

php artisan route:list

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	hello/array		Closure	web
	GET HEAD	posts/{id}		Closure	web
	GET HEAD	posts/{id}/{name?}		Closure	web

Exercises

Exercise 3.1 ให้กำหนด Routing ดังนี้

```
Route::get('/posts/{post_id}/comments/{comment_id}/edit',
    function ($comment_id, $post_id) {
        return [
```

```
'post_id' => $post_id,  
'comment_id' => $comment_id  
];  
});
```

สังเกตว่าตัวแปรใน URL path และ Closure parameter สลับชื่อกัน

(a) จะต้อง request อย่างไร เพื่อให้ response นี้ทำงานได้

(b) อธิบายผลลัพธ์ที่เกิดขึ้น

(c) สรุปข้อสังเกตที่ได้จากการทดลองนี้

Exercise 3.2 ให้ใช้ terminal ดู help ของคำสั่ง `artisan php artisan route:list` แล้วทดลอง option ต่าง ๆ ของคำสั่งนี้ สังเกตและอธิบายผลลัพธ์ที่เกิดขึ้น

Problem

Problem 3.1 การกำหนด Routing ที่ให้ URL path แรกสุดเป็น optional parameter จะมีผลดี และผลเสียต่อการพัฒนาเว็บอย่างไรบ้าง

Problem 3.2 การกำหนด Routing ที่ให้ URL path เป็น optional parameter ก่อน required parameter เช่น `'/{name?}/{id}'` จะทำได้หรือไม่ หากทำไม่ได้จะมีแนวทางแก้ไขปัญหานี้ อย่างไร

Problem 3.3 หากต้องการให้การ request หน้าเว็บเป็น `/posts/123` โดยที่ 123 จะเป็นหมายเลข ของโพสต์หมายเลขอะไรก็ได้ แต่จะต้องระบุเป็นหมายเลขเท่านั้น (ไม่รับ request `/posts/abc` เพราะ `abc` ไม่ใช่ตัวเลข) จะต้องระบุ Routing อย่างไร

4. Laravel Blade Templates

4.1	Template Inheritance	20
4.2	การแสดงผลข้อมูลใน Blade Templates	24
4.3	Blade Control Structure	28

Blade เป็น PHP Templating Engine ที่พัฒนาโดยผู้พัฒนา Laravel โดย Blade Templates จะถูกคอมไพล์เป็นโค้ดภาษา PHP และถูกเก็บเป็น cached ไว้ จนกว่าจะมีการเปลี่ยนโค้ด Blade ใหม่

ไฟล์ Blade จะมีนามสกุล .blade.php และต้องสร้างไฟล์ Blade ไว้ใน resources/views ดังนั้นจะมองไฟล์ Blade เป็นส่วน View ของ MVC (Model-View-Controller Architectural Pattern)

4.1 Template Inheritance

สิ่งหนึ่งที่ต้องจัดการเสมอในการออกแบบหน้าเว็บ คือ ทุกหน้าควรมีรูปแบบในการทำงานเดียวกัน (Consistency) และคุณสมบัติหนึ่งที่ Blade Templates ช่วยในเรื่องนี้คือ Template Inheritance โดยแบ่งออกเป็น 2 ส่วน คือ Layout และ Child View

1. การสร้าง Layout

สร้างไฟล์ resources/views/layouts/main.blade.php

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1
6     ">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>My App</title>
9     <!-- CSRF Token -->
10    <meta name="csrf-token" content="{{ csrf_token() }}">
11    <!-- Styles -->
```

```

11     <link rel="stylesheet" href="{{ asset('css/app.css') }}">
12 </head>
13 <body>
14     @include('layouts.menu')
15
16     <div class="container mx-auto">
17         @yield('content')
18     </div>
19
20     <script src="{{ asset('js/app.js') }}"></script>
21 </body>
22 </html>

```

@**yield** directive ใช้แสดงเนื้อหาที่กำหนดใน @section directive ชื่อเดียวกันที่จะเขียนใน Child View

ให้ใช้คำสั่ง

```

npm install
npm run dev

```

จะมีไฟล์ CSS และ JavaScript มาให้อยู่ที่ public/css/app.css และ public/js/app.js ซึ่งมาจากการประมวลผล CSS และ JavaScript แบบ pre-processing ซึ่งสามารถนำมาใช้งานได้ (บรรทัดที่ 11, 20)

ตัวอย่างการใช้ Sub View อยู่ในบรรทัด 14 คือ @include('layouts.menu')

2. การเขียน Sub View

สร้างไฟล์ resources/views/layouts/menu.blade.php (ตัวอย่างโค้ดนำมาจาก <https://www.creative-tim.com/learning-lab/tailwind-starter-kit/documentation/css/menus>)

```

1 <nav class="relative flex flex-wrap items-center justify-between px-2
  py-3 bg-teal-500 mb-3">
2     <div class="container px-4 mx-auto flex flex-wrap items-center
  justify-between">
3         <div class="w-full relative flex justify-between lg:w-auto px
  -4 lg:static lg:block lg:justify-start">
4             <a class="text-sm font-bold leading-relaxed inline-block
  mr-4 py-2 whitespace-nowrap uppercase text-white" href="#pablo">
5                 My App
6             </a>
7             <button
8                 class="cursor-pointer text-xl leading-none px-3 py-1
  border border-solid border-transparent rounded bg-transparent
  block lg:hidden outline-none focus:outline-none"

```

```

9         type="button"
10        onclick="toggleNavbar('example-collapse-navbar')"
11    >
12        <span class="block relative w-6 h-px rounded-sm bg-
white"></span>
13        <span class="block relative w-6 h-px rounded-sm bg-
white mt-1"></span>
14        <span class="block relative w-6 h-px rounded-sm bg-
white mt-1"></span>
15    </button>
16 </div>
17 <div class="lg:flex flex-grow items-center" id="example-
collapse-navbar">
18     <ul class="flex flex-col lg:flex-row list-none ml-auto">
19         <li class="nav-item">
20             <a class="px-3 py-2 flex items-center text-xs
uppercase font-bold leading-snug text-white hover:opacity-75" href
="#pablo">
21                 Discover
22             </a>
23         </li>
24         <li class="nav-item">
25             <a class="px-3 py-2 flex items-center text-xs
uppercase font-bold leading-snug text-white hover:opacity-75" href
="#pablo">
26                 Profile
27             </a>
28         </li>
29         <li class="nav-item">
30             <a class="px-3 py-2 flex items-center text-xs
uppercase font-bold leading-snug text-white hover:opacity-75" href
="#pablo">
31                 Setting
32             </a>
33         </li>
34     </ul>
35 </div>
36 </div>
37 </nav>
38
39 <script>
40     function toggleNavbar(collapseID) {
41         document.getElementById(collapseID).classList.toggle("hidden");
42         document.getElementById(collapseID).classList.toggle("flex");
43     }

```

44 `</script>`

การเขียน **Sub View** เป็นการเขียนส่วนย่อยของ **Layout** หรือ **Child View** ก็ได้



Laravel 8 ใช้ **Tailwind CSS** เป็น **default CSS framework** ศึกษาการใช้งานเพิ่มเติมได้ที่

<https://tailwindcss.com/docs> และ <https://www.creative-tim.com/learning-lab/tailwind-starter-kit/documentation/quick-start>

3. การเขียน Child View

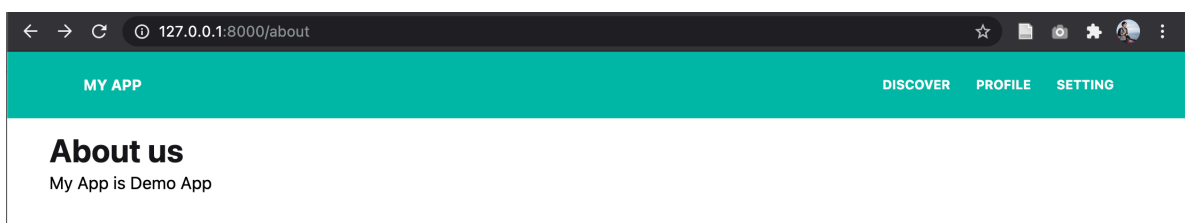
สร้างไฟล์ `resources/views/about.blade.php`

```
1 @extends('layouts.main')
2
3 @section('content')
4     <div class="lg:flex lg:items-center lg:justify-between">
5         <div class="flex-1 min-w-0">
6             <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:
7                 text-3xl sm:truncate">
8                 About us
9             </h2>
10        </div>
11        <div>
12            <p>My App is Demo App</p>
13        </div>
14    @endsection
```

ในไฟล์ **Child View** จะต้องระบุ **Layout** ที่ต้องการใช้งานใน `@extends` directive ซึ่งจะระบุเป็น **Dot Notation** เช่น `layouts.main` หมายถึงไฟล์ `resources/views/layouts/main.blade.php`

ตัวอย่างการระบุ **Routing (routes/web.php)** ให้ **response Child View**

```
Route::get('/about', function() {
    return view('about');
});
```



Tailwind CSS จะมี **default theme** เป็นสีชมพู กรณีที่เปลี่ยนสีเป็นสีอื่น เช่น สี **Teal** ตามตัวอย่างโค้ด จะต้องแก้ไขไฟล์ `tailwind.config.js` ดังนี้

```

1 const defaultTheme = require('tailwindcss/defaultTheme');
2 const colors = require("tailwindcss/colors");
3
4 module.exports = {
5   purge: [
6     './vendor/laravel/framework/src/Illuminate/Pagination/
resources/views/*.blade.php',
7     './storage/framework/views/*.php',
8     './resources/views/**/*.blade.php',
9   ],
10
11   theme: {
12     colors: {
13       ...colors,
14       "current": "current",
15       "transparent": "transparent",
16     },
17     extend: {
18       fontFamily: {
19         sans: ['Nunito', ...defaultTheme.fontFamily.sans],
20       },
21     },
22   },
23
24   variants: {
25     extend: {
26       opacity: ['disabled'],
27     },
28   },
29
30   plugins: [require('@tailwindcss/forms')],
31 };

```

หลังจากแก้ไขไฟล์ ให้ใช้ terminal รันคำสั่ง `npm run dev`

4.2 การแสดงข้อมูลใน Blade Templates

สามารถส่งข้อมูล **Associative Array** เป็น **parameter** ที่ 2 ในฟังก์ชัน `view()` เพื่อนำไปแสดงใน Blade Template เช่น

```

// routes/web.php
Route::get('/about', function() {
    return view('about', [
        'name' => 'Your Name',
    ]);
});

```



```

        'info' => [
            'address' => 'Bangkok, <b>Thailand</b>',
            'email' => 'contact@example.com'
        ]
    });
});

```

โดย key ของ associative array จะเป็นชื่อตัวแปรใน Blade Templates จากตัวอย่างจะทำให้ไฟล์ Blade มีตัวแปร `$name` ที่มีค่า 'Your Name' และมีตัวแปร `$info` ที่มีค่า `['address' => 'Bangkok, Thailand', 'email' => 'contact@example.com']`

การแสดงผลข้อมูลใน Blade Templates จะใช้ double curly braces (`{{ }}`) statement เช่น (ในไฟล์ `resources/views/about.blade.php`)

```

1  @extends('layouts.main')
2
3  @section('content')
4      <div class="lg:flex lg:items-center lg:justify-between">
5          <div class="flex-1 min-w-0">
6              <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:
7                  text-3xl sm:truncate">
8                  About us
9              </h2>
10             </div>
11             <div>
12                 <p>My App is Demo App</p>
13             </div>
14
15             <div class="p-10">
16                 <!--Card 1-->
17                 <div class="w-full lg:max-w-full lg:flex">
18                     <div class="h-48 lg:h-auto lg:w-48 flex-none bg-cover
19                         rounded-t lg:rounded-t-none lg:rounded-l text-center overflow-
20                         hidden" style="background-image: url('https://loremflickr.com
21                         /320/240/mountain');" title="Mountain">
22                     </div>
23                     <div class="border-r border-b border-l border-gray-400 lg:
24                         border-l-0 lg:border-t lg:border-gray-400 bg-white rounded-b lg:
25                         rounded-b-none lg:rounded-r p-4 flex flex-col justify-between
26                         leading-normal">
27                         <div class="mb-8">
28                             <p class="text-sm text-gray-600 flex items-center">
29
30                             <svg class="fill-current text-gray-500 w-3 h-3

```

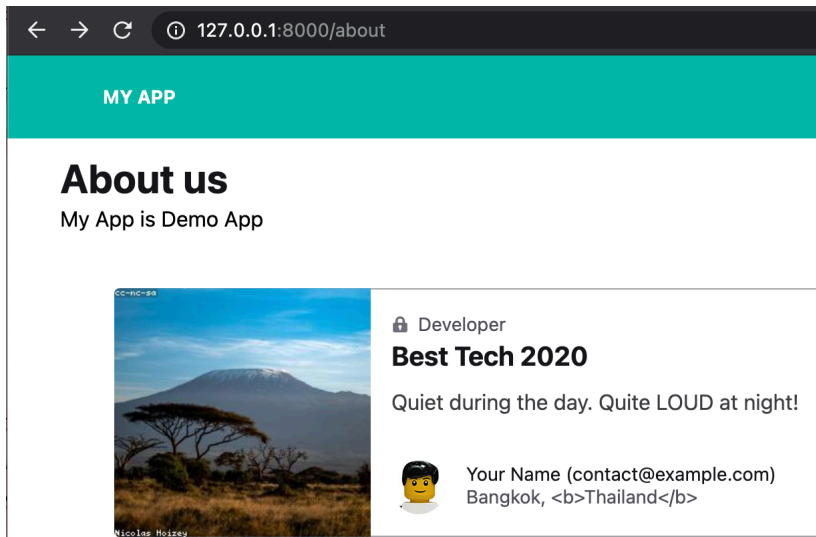
```

24         mr-2" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">
            <path d="M4 8V6a6 6 0 1 1 12 0v2h1a2 2 0 0
1 2 2v8a2 2 0 0 1-2 2H3a2 2 0 0 1-2-2v-8c0-1.1.9-2 2-2h1zm5 6.73
V17h2v-2.27a2 2 0 1 0-2 0zM7 6v2h6V6a3 3 0 0 0-6 0z" />
25         </svg>
26         Developer
27     </p>
28     <div class="text-gray-900 font-bold text-xl mb-2">
Best Tech 2020</div>
29     <p class="text-gray-700 text-base">Quiet during
the day. Quite LOUD at night!</p>
30 </div>
31 <div class="flex items-center">
32     
33     <div class="text-sm">
34         <p class="text-gray-900 leading-none">{{ $name
}} ({{ $info['email'] }})</p>
35         <p class="text-gray-600">{{ $info['address']
}}</p>
36     </div>
37 </div>
38 </div>
39 </div>
40 </div>
41 @endsection

```

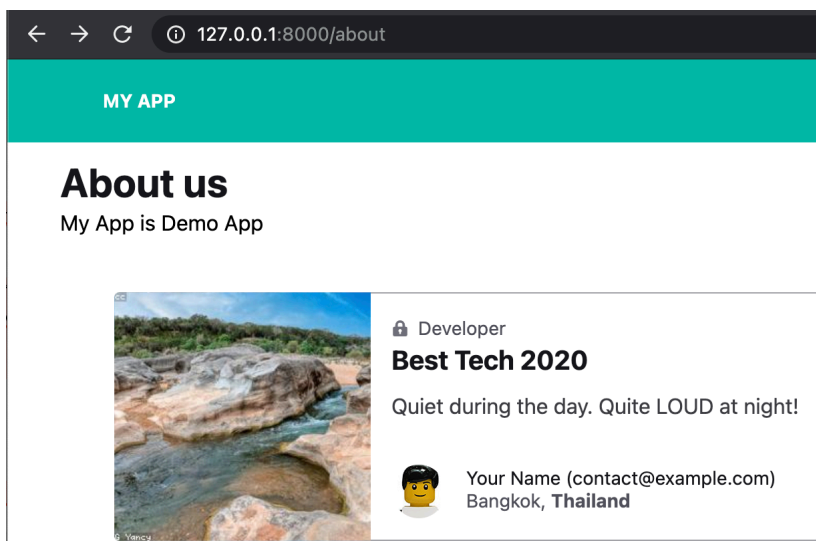
(ดัดแปลงจากหัวข้อ Tailwind CSS Card Example #2 <https://www.ordinarycoders.com/blog/article/17-tailwindcss-cards>)

ซึ่ง `{{ }}` statement จะนำค่าในตัวแปรไปผ่านฟังก์ชัน `htmlspecialchars()` ของ PHP เพื่อป้องกัน XSS Attack หรือเรียกว่าเป็นการแสดงข้อมูลแบบ escaped (ตัวอย่าง tag `` ไม่ถูกประมวลผลเป็น HTML จึงไม่แสดง Thailand เป็นตัวหนา แต่จะแสดงตัวอักษร `` ให้ปรากฏบนหน้า browser)



หากต้องการแสดงข้อมูลแบบ unescaped data ให้ใช้ `{!! !!}` statement แทน เช่น

```
35 <p class="text-gray-600">{!! $info['address']
    {!!}</p>
```



หากไม่ต้องการให้ประมวลผล Blade Syntax ให้ใส่ `@` นำหน้า เช่น

```
@{{ $name }}
```

ซึ่งจะสอดคล้องกับใน JavaScript framework จะใช้ `{ }` ในการแสดงข้อมูลเช่นกัน



4.3 Blade Control Structure

Blade Templates สามารถใช้ if statement โดยใช้ `@if`, `@elseif`, `@else` และ `@endif` directives ซึ่งเมื่อถูกคอมไพล์ จะเปลี่ยนเป็น if statement ของภาษา PHP

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have many records!
@else
    I have no records!
@endif
```

หรือการใช้งาน Loop

```
@for ($i = 0; $i < 5; $i++)
    <p>The current value is {{ $i }}</p>
@endfor

@foreach ($records as $record)
    <p>This is record {{ $record->id }}
@endforeach
```

ศึกษา Blade Templates และ Blade Syntax เพิ่มเติมได้ที่ <https://laravel.com/docs/8.x/blade>

Exercises

Exercise 4.1 สามารถนำ Sub View ไปใช้ (include) ใน Sub View อื่นได้หรือไม่

Exercise 4.2 Comment ที่ใช้ใน Blade Templates จะมี 2 รูปแบบ คือ HTML Comment และ Blade Comment ทั้ง 2 รูปแบบ มีการเขียนเหมือนหรือต่างกันอย่างไร และผลลัพธ์ (หน้าเว็บ) ที่ได้มีข้อแตกต่างกันอย่างไร (คำใบ้ ให้ดู source code ของหน้าเว็บ)

Problem

Problem 4.1 "สิ่งหนึ่งที่ต้องจัดการเสมอในการออกแบบหน้าเว็บ คือ ทุกหน้าควรมีรูปแบบในการทำงานเดียวกัน (Consistency)" คุณเห็นด้วยหรือไม่ และหากมีบางหน้าที่ต้องการ Layout ที่แตกต่างออกไป จะต้องทำอย่างไรใน Laravel

Problem 4.2 XSS Attack คืออะไร และทำไม `htmlspecialchars()` จึงเป็นวิธีหนึ่งป้องกัน XSS Attack ได้

Problem 4.3 ถ้ามีการใช้งาน `@if` statement ใน Layout ที่ต้องมีการอ่านค่าของตัวแปรจาก Child View เช่น การแสดงชื่อของผู้ใช้ใน menu ที่อยู่ใน Layout หากมีตัวแปร `$user` ส่งมาจาก Child View จะต้องเขียน Blade Syntax ใน Layout และ Child View อย่างไร

Problem 4.4 มีวิธีที่ดีกว่า หรือ Best Practice ในการแก้ปัญหา Problem 4.3 หรือไม่ ทำอย่างไร

Problem 4.5 เนื่องจาก Laravel 8 ใช้ Tailwind CSS เป็น default CSS framework แต่หากทีมพัฒนาเห็นร่วมกันว่าจะใช้ CSS framework ตัวอื่น เช่น Bootstrap เป็น CSS framework ของ Laravel project ใหม่ จะต้องตั้งค่าอย่างไรบ้าง

5. Laravel Controller

5.1 การสร้าง Controller

30

การกำหนด Closure ในไฟล์ routes/web.php เพื่อจัดการ Logic ของระบบ จะทำให้ไฟล์ Route นั้น มีจำนวนบรรทัดของโค้ดมากเกินไป ดังนั้นจึงควรรวมการจัดการ Logic ของระบบที่เกี่ยวข้องไว้ด้วยกันใน Controller เดียวกัน เช่น การเพิ่มข้อมูลโพสต์ การแก้ไขโพสต์ การแสดงโพสต์ทั้งหมด การแสดงโพสต์เดียว การลบโพสต์ ควรอยู่ใน PostController เพราะจัดการกับโพสต์ทั้งหมด ซึ่งแยกจากการจัดการข้อมูลผู้ใช้ให้อยู่ใน UserController

Controller Class ทั้งหมด จะอยู่ใน directory app/Http/Controllers และมี namespace เป็น App\Http\Controllers

5.1 การสร้าง Controller

ใช้คำสั่ง `php artisan make:controller <ControllerName>` ผ่าน terminal โดยให้ working directory อยู่ที่ Laravel project ไม่ต้องเข้าไปที่ app/Http/Controllers

การตั้งชื่อ Controller ให้ใช้รูปแบบ UpperCamelCase (ตัวอักษรตัวแรกของแต่ละคำเป็นตัวใหญ่) แล้วลงท้ายด้วยคำว่า Controller เช่น PostController สังเกตคำว่า Post จะเป็นเอกพจน์

ชื่อของ action จะอยู่ในรูปแบบ lowerCamelCase (ตัวอักษรตัวแรกของแต่ละคำเป็นตัวใหญ่ ยกเว้นตัวอักษรแรกสุดเป็นตัวเล็ก)

1. สร้าง Controller ชื่อ PostController ด้วยคำสั่ง

```
php artisan make:controller PostController
```

จะได้ไฟล์อยู่ที่ app/Http/Controllers/PostController.php

2. เพิ่มการทำงานของ PostController ที่เกี่ยวข้องกับการจัดการโพสต์ เช่น index สำหรับการแสดงรายการโพสต์ทั้งหมด หรือ show สำหรับการแสดงโพสต์ตามหมายเลขที่ระบุ

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class PostController extends Controller
8 {
9     public function index()
10    {
11        return 'All Post';
12    }
13
14    public function show($id)
15    {
16        return 'Post ID: ' . $id;
17    }
18 }

```

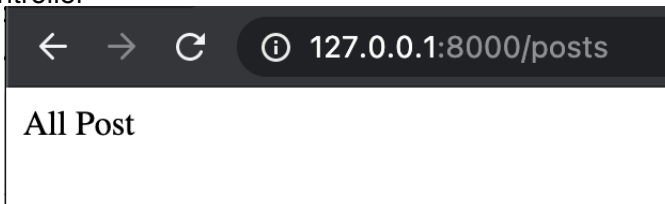
3. กำหนด Route เพื่อเข้าถึง action ของ Controller โดยเปลี่ยน parameter ที่ 2 ของ Route function จาก closure เป็น array ในรูปแบบ [ชื่อ Controller, ชื่อ action]

```

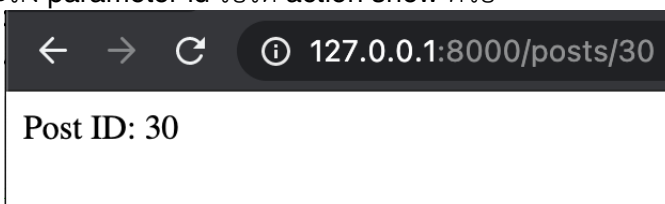
use App\Http\Controllers\PostController;
Route::get('/posts', [PostController::class, 'index']);
Route::get('/posts/{id}', [PostController::class, 'show']);

```

หมายความว่า ถ้ามี request เข้ามาด้วย path /posts ให้ไปประมวลผลที่ action index ของ Post-Controller



แต่ถ้ามี request เข้ามาด้วย path /posts/id ให้ไปประมวลผลที่ action show ของ PostController และส่ง parameter id ไปให้ action show ด้วย



4. สามารถเปลี่ยนเป็นการ return Child View ได้ เช่น ใน action index ของ PostController

```
public function index()
{
    $posts = [
        (object) [
            'id' => 1,
            'title' => 'First Post',
            'detail' => 'First post detail',
            'views' => 350
        ],
        (object) [
            'id' => 2,
            'title' => 'Second Post',
            'detail' => 'Second post detail',
            'views' => 1637
        ]
    ];
    return view('posts.index', ['posts' => $posts]);
}
```

ควรสร้าง directory ย่อย ภายใน directory resources/views เพื่อจัดการ Child View ของแต่ละ Controller แล้วใช้ Dot Notation ในฟังก์ชัน view() เช่น posts.index หมายถึงไฟล์ resources/views/posts/index.blade.php

5. สร้างไฟล์ resources/views/posts/index.blade.php

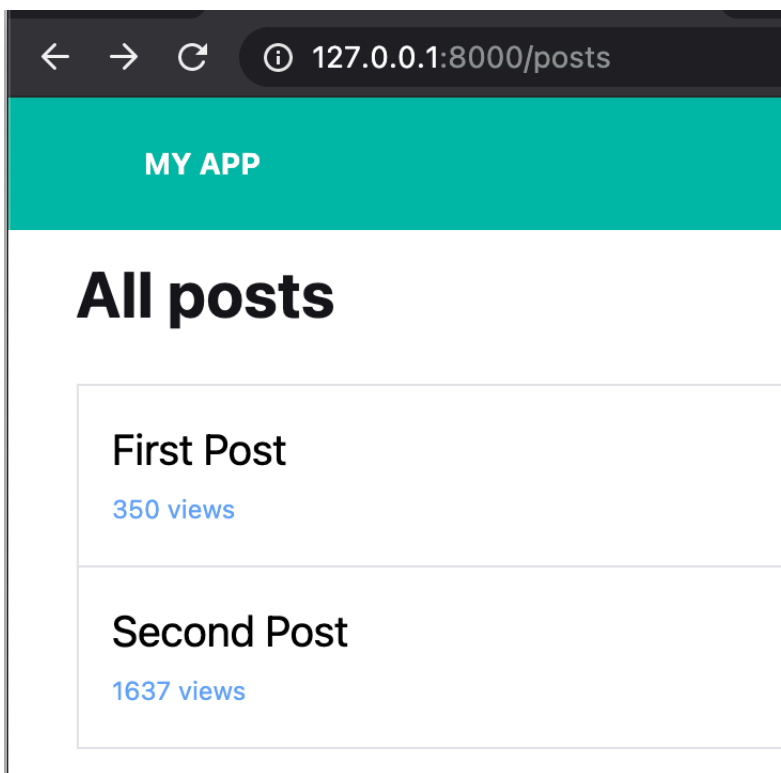
```
1 @extends('layouts.main')
2
3 @section('content')
4     <div class="flex flex-col">
5         <div class="min-w-0">
6             <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:
7             text-3xl sm:truncate">
8                 All posts
9             </h2>
10        </div>
11
12        <div class="mt-6">
13            <ul class="list-reset flex flex-col">
14                @foreach ($posts as $post)
15                    <li class="relative -mb-px block border p-4 border-
16                    grey">
17                        <p class="text-xl">
18                            <a href="{{ url('posts/{<!--
19                                -->id}}") }}">
```



```

18         {{ $post->title }}
19     </a>
20 </p>
21     <span class="text-xs text-blue-400">
22         {{ $post->views }} views
23     </span>
24 </li>
25 @endforeach
26 </ul>
27 </div>
28 @endsection

```



ในไฟล์ Blade สามารถใช้ `function url()` เพื่อสร้าง URL ได้ (บรรทัด 17) และยังมีรูปแบบอื่นในการสร้าง URL สามารถดูเพิ่มเติมได้ที่ <https://laravel.com/docs/8.x/urls>

Exercises

Exercise 5.1 หากใช้คำสั่งต่อไปนี้ในการสร้างคลาส `PostController`

```
php artisan make:controller Admin/PostController
```

(a) คลาส `PostController` ที่สร้างจากคำสั่งนี้ จะอยู่ในไฟล์ใด (ตอบ path เต็ม)

(b) คลาส `PostController` ที่สร้างจากคำสั่งนี้ จะมี namespace ว่าอะไร

Exercise 5.2 ให้กำหนด Routing ดังนี้

```
use App\Http\Controllers\Admin\PostController as
    AdminPostController;
Route::get('/posts/{post_id}/comments/{comment_id}/edit', [
    AdminPostController::class, 'editComment'
]);
```

จงทำให้ Route นี้ ทำงานได้ โดยแสดงผลลัพธ์ว่ารับค่า post_id และ comment_id ได้

6. Laravel Migration

6.1	การสร้างไฟล์ Migration	35
6.2	การรัน Migrate	37
6.3	การปรับแก้ Migration	37
6.4	การแก้ไขข้อผิดพลาดของการ Migrate	38

Migration เป็นเหมือน *Version Control* ของฐานข้อมูล ซึ่งให้ผู้ร่วมพัฒนาระบบร่วมกันปรับแก้และส่งต่อข้อมูล *schema* ของฐานข้อมูลถึงกันผ่านโค้ด โดยไม่ต้องบอกผู้ร่วมพัฒนาให้เพิ่ม *field* ของตารางในฐานข้อมูล แต่เปลี่ยนเป็นการรัน *script* เพื่อแก้ไข *schema* จากโค้ด *Migration* นั้นแทน

6.1 การสร้างไฟล์ Migration

คำสั่งสำหรับการสร้างไฟล์ Migration คือ `php artisan make:migration <migration_name>` ซึ่งจะมี option `--create` สำหรับการเพิ่ม *schema* ของตารางใหม่ และมี option `--table` สำหรับการแก้ไข *schema* ของตารางที่มีอยู่แล้ว เช่น

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

หมายความว่า ให้สร้างไฟล์ Migration ชื่อ `create_posts_table` สำหรับการเพิ่ม *schema* ของตารางใหม่ที่ชื่อตารางว่า `posts` (ระบุใน option `--create`) ชื่อตารางจะอยู่ในรูปแบบ *snake case* (`under_score`) และเป็นพหูพจน์

ไฟล์ Migration ทั้งหมด จะถูกสร้างไว้ที่ directory `databases/migrations` โดยชื่อไฟล์จะมีวันและเวลาที่สร้างไฟล์กำกับอยู่ เพื่อเป็นลำดับของไฟล์ Migration ที่จะทำงาน ซึ่งในขั้นตอนของการ migrate ไฟล์ที่มีชื่อมาก่อน (ตามลำดับพหุนุกรม) จะถูกเรียกให้ทำงานก่อน

ในไฟล์ Migration จะเป็นคลาสที่มีชื่อเดียวกับชื่อ Migration ที่ระบุไว้ในคำสั่ง `make:migration` โดยชื่อ Migration จะเป็น *snake case* แต่ชื่อคลาสจะเป็น *UpperCamelCase*

คลาส Migration จะมี 2 methods คือ up() และ down() โดย up() จะถูกเรียกทำงานเมื่อสั่งรัน migrate และ down() จะถูกเรียกเมื่อสั่งรัน migrate:rollback

ถ้า make:migration ด้วย option --create ใน method up() จะมีคำสั่ง Schema::create() ที่ระบุชื่อตารางไว้แล้ว เพื่อกำหนด schema ของตารางใหม่ แต่ถ้า make:migration ด้วย option --table จะมีคำสั่ง Schema::table() ที่ระบุชื่อตารางที่ต้องการแก้ไข schema ของตารางที่มีอยู่แล้ว และถ้าไม่กำหนด option ตอนที่ make:migration จะต้องจัดการ method up() เอง

การกำหนดฟิลด์หรือคอลัมน์ของตาราง ให้กำหนดใน closure function (Blueprint \$table) ซึ่งเป็น parameter ที่ 2 ของ Schema::create() หรือ Schema::table() ให้ดูวิธีการใช้คำสั่งจาก documentation ที่ <https://laravel.com/docs/8.x/migrations/#columns>

ตัวอย่างฟังก์ชัน up() สำหรับสร้าง schema ของตาราง posts

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('detail');
        $table->bigInteger('views')->default(0);
        $table->timestamps();
    });
}
```

`$table->id()` จะสร้างฟิลด์ชื่อ id ชนิด UNSIGNED BIGINT PRIMARY KEY AUTO INCREMENT

`$table->string('title')` จะสร้างฟิลด์ชื่อ title ชนิด VARCHAR(255)

`$table->text('detail')` จะสร้างฟิลด์ชื่อ detail ชนิด TEXT

`$table->bigInteger('views')->default(0)` จะสร้างฟิลด์ชื่อ views ชนิด BIGINT และมีค่า default เป็น 0

`$table->timestamps()` จะสร้าง 2 ฟิลด์พิเศษ คือ created_at และ updated_at ชนิด TIMESTAMP ซึ่งเป็น 2 ฟิลด์พิเศษ ที่จะถูกกำหนดค่าอัตโนมัติ โดย created_at จะถูกกำหนดเป็น timestamp ที่ record ถูกสร้างขึ้น และ updated_at จะเปลี่ยนค่า timestamp ทุกครั้งที่มีการแก้ไขข้อมูลใน record



Laravel แนะนำให้ทุกตารางมี primary key เพียงฟิลด์เดียว ชื่อว่า id
เมื่อ make:migration ด้วย option --create จะมีคำสั่ง `$table->id()` มาให้

Laravel มีคุณลักษณะหนึ่งจัดการกับการลบข้อมูล โดยไม่ได้ลบจากตาราง เพียงแต่กำหนดค่า *timestamp* (วันและเวลา) ที่ลบข้อมูลนั้นไป เพื่อเป็น *log* ของข้อมูลที่ยังคงอยู่ในตาราง แต่จะไม่ถูกเรียกขึ้นมาแสดงผลหรือคืนค่าเป็นผลลัพธ์จากการค้นคืนข้อมูล โดยกำหนดชื่อฟิลด์ว่า *deleted_at* ชนิด *TIMESTAMP* หรือใช้คำสั่ง `$table->softDeletes()` ในการสร้างฟิลด์นี้ที่ไฟล์ *Migration*



ฟังก์ชัน `down()` จะมีโค้ดที่กำหนดวิธีการ `drop schema` (ถ้า `make:migration` โดยไม่ระบุ `option` ต้องจัดการโค้ดส่วนนี้เอง)

```
public function down()
{
    Schema::dropIfExists('posts');
}
```

6.2 การรัน Migrate

ก่อนรัน *Migrate* ต้องเปิด *database server* ไว้ก่อน โดยต้องมีฐานข้อมูลที่จะทำงานไว้แล้ว โดยต้องกำหนด *collation* ของฐานข้อมูล *MySQL* เป็น *utf8mb4_unicode_ci* สำหรับฐานข้อมูลอื่น ให้ดูการตั้งค่าในไฟล์ `config/database.php`

กำหนดข้อมูลตั้งค่าการเชื่อมต่อฐานข้อมูลในไฟล์ `.env` เช่น ถ้าเชื่อมต่อฐานข้อมูล *MySQL* การตั้งค่าในไฟล์ `.env` ที่เกี่ยวข้อง คือ

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=myapp
DB_USERNAME=myappuser
DB_PASSWORD=myapppassword
```

คำสั่งสำหรับรัน *migrate* คือ `php artisan migrate`

เมื่อ *migrate* ครั้งแรก จะมีตารางชื่อ *migrations* ที่จะบอกชื่อไฟล์ *Migration* ที่ทำงานตามลำดับ และหมายเลข *batch* ของการ *migrate* โดยไฟล์ *Migration* ที่รันในครั้งเดียวกัน จะมีหมายเลข *batch* เดียวกัน นอกจากนี้จะมีตาราง *users*, *password_resets* และ *failed_jobs* ซึ่งมาจากไฟล์ *Migration* ที่มีอยู่แล้วเมื่อติดตั้ง *Laravel project*

หลังจากการรัน *Migrate* ให้ตรวจสอบ *schema* ของตารางในฐานข้อมูลว่าถูกต้องหรือไม่

6.3 การปรับแก้ Migration

หากมีการเพิ่มไฟล์ *Migration* ใหม่เพื่อสร้าง *schema* ของตารางใหม่ สามารถสั่ง *migrate* ต่อได้เลย ด้วยคำสั่ง `php artisan migrate` โดยฟังก์ชัน `up()` ของ *Migration* ใหม่ จะถูกรัน และกลายเป็น

batch หมายเลขถัดไป โดยไม่กระทบหรือไม่ประมวลผลไฟล์ Migration เดิมที่ migrate สำเร็จไปครั้งก่อนหน้าแล้ว

หากมีการปรับแก้ไฟล์ Migration เดิมที่เคย migrate สำเร็จไปแล้ว เช่น เพิ่มบรรทัดคำสั่งเพื่อเพิ่มฟิลด์ใหม่ในไฟล์ Migration เดิม การสั่ง migrate ซ้ำ จะไม่มีการประมวลผล Migration เก่า เพราะมี record ในตาราง migrations ว่าเคย migrate สำเร็จแล้ว ให้ rollback การ migrate ก่อนที่จะ migrate ซ้ำ ซึ่งมี 2 รูปแบบ คือ

1. rollback เฉพาะ batch ล่าสุด ด้วยคำสั่ง `php artisan migrate:rollback`

2. rollback ทุก migration ด้วยคำสั่ง `php artisan migrate:reset`

การ rollback จะส่งผลให้ข้อมูลในตารางถูกลบไปด้วย



คำสั่ง `php artisan migrate:refresh` เป็นคำสั่งที่จะ rollback ทุก migration แล้ว migrate ใหม่ เทียบเท่ากับการใช้คำสั่ง `php artisan migrate:reset` ต่อด้วยคำสั่ง `php artisan migrate`

6.4 การแก้ไขข้อผิดพลาดของการ Migrate

เนื่องจาก PHP เป็น scripting language นั่นคือ การประมวลผลภาษาจะไม่มีการตรวจสอบไวยากรณ์ (syntax) จนกว่าจะประมวลผลถึงบรรทัดคำสั่งนั้น ข้อผิดพลาดที่เกิดขึ้นอาจทำให้สั่ง migrate ซ้ำไม่ได้ เช่น โค้ดส่วนสร้างตารางถูกประมวลผลสำเร็จได้ schema ของฐานข้อมูล แต่โค้ดส่วนเพิ่มฟิลด์นั้นประมวลผลไม่สำเร็จ ทำให้ migrate ซ้ำไม่ได้ และจะมีการแจ้งเตือนความผิดพลาดว่ามีตารางนี้แล้ว

แนวทางการแก้ปัญหา

1. ให้ตรวจสอบตาราง migrations ก่อน ว่าทำงานถึงไฟล์ไหนแล้ว การทำงานที่ถูกต้องคือ ถ้ามี schema ของตารางแล้ว จะมี record ของชื่อไฟล์ Migration ในตาราง migrations แต่หากการทำงานไม่ถูกต้องคือ มี schema ของตารางที่ไม่สมบูรณ์ โดย record ในตาราง migration ไม่มีชื่อไฟล์ Migration ที่เพิ่งแก้ไขไป วิธีแก้ไขคือ ให้ drop schema ของตารางดังกล่าว (จาก Database Management Tools) แล้วส่ง migration ใหม่

2. แก้ไขไฟล์ Migration ให้ถูกต้อง แล้วใช้คำสั่ง `php artisan migrate:fresh` ซึ่งคำสั่งนี้จะลบตารางทั้งหมดในฐานข้อมูล และรัน migration ให้ใหม่

3. ลบฐานข้อมูลทิ้งแล้วสร้างใหม่ จากนั้นสั่ง migrate ใหม่

4. อาจมีประเด็นของปัญหาอื่น เช่น MySQL รุ่นเก่า มีปัญหากับจำนวนตัวอักษรของ VARCHAR ให้ค้นหาวีธีแก้ไขจาก Google หรือ Stackoverflow

การแก้ไขปัญหาดังกล่าว ควรทำเฉพาะใน development environment และต้องระมัดระวังอย่างมากใน production environment ซึ่งควร backup ฐานข้อมูลก่อน

Problem

Problem 6.1 Laravel มี convention (ธรรมเนียมปฏิบัติ) ในการสร้างฟิลด์ที่เป็น foreign key อย่างไรบ้าง และในไฟล์ Migration จะกำหนดฟิลด์ที่เป็น foreign key ได้อย่างไร

Programming Exercise

Programming Exercise 6.1 โรงภาพยนตร์

ให้สร้างไฟล์ Migration ของตาราง cinemas เพื่อเก็บข้อมูลของโรงฉายภาพยนตร์ของโรงภาพยนตร์แห่งหนึ่ง ซึ่งมีฟิลด์ดังนี้

- name เก็บข้อมูลชื่อของโรงฉายภาพยนตร์ เช่น 4DX ซึ่งซ้ำกันไม่ได้
- number เก็บหมายเลขของโรงฉายภาพยนตร์ เช่น (โรงที่) 11 ซึ่งซ้ำกันไม่ได้

ให้สร้างไฟล์ Migration ของตาราง seat_types เพื่อเก็บข้อมูลของประเภทที่นั่ง ซึ่งมีฟิลด์ดังนี้

- name เก็บข้อมูลชื่อของประเภทที่นั่ง เช่น Normal, Premium, Honeymoon ซึ่งซ้ำกันไม่ได้
- price เก็บข้อมูลราคาของประเภทที่นั่ง

ให้สร้างไฟล์ Migration ของตาราง seats เพื่อเก็บข้อมูลของที่นั่งในโรงฉายภาพยนตร์ ซึ่งมีฟิลด์ดังนี้

- row เก็บข้อมูลหมายเลขแถวของที่นั่ง เช่น (แถว) A
- column เพื่อเก็บข้อมูลหมายเลขคอลัมน์ของที่นั่ง เช่น 5
- seat_type_id เพื่อเก็บข้อมูลประเภทที่นั่ง (เป็น foreign key)
- cinema_id เพื่อเก็บข้อมูลโรงฉายภาพยนตร์ (เป็น foreign key)

Programming Exercise 6.2 บัตรสะสมแต้ม

ให้สร้าง Migration ของตาราง members เพื่อเก็บข้อมูลของสมาชิกของร้านค้าแห่งหนึ่ง ซึ่งมีฟิลด์ดังนี้

- name เก็บข้อมูลชื่อสมาชิก - phone เก็บข้อมูลหมายเลขโทรศัพท์ของสมาชิก ซึ่งซ้ำกันไม่ได้ - email เก็บข้อมูลอีเมลของสมาชิก ซึ่งซ้ำกันไม่ได้

ให้สร้าง Migration ของตาราง `member_cards` เพื่อเก็บข้อมูลของบัตรสะสมแต้ม ซึ่งมีฟิลด์ดังนี้

- `name` เก็บข้อมูลชื่อของบัตรสะสมแต้ม - `number` เก็บข้อมูลหมายเลขของบัตรสะสมแต้ม ซึ่งซ้ำกันไม่ได้ - `member_id` เก็บข้อมูลสมาชิกเจ้าของบัตรสะสมแต้ม - `point` เก็บข้อมูลจำนวนแต้มรวมในบัตรสะสมแต้ม

ให้สร้าง Migration ของตาราง `point_logs` เพื่อเก็บข้อมูลประวัติการได้แต้มและการใช้แต้ม ซึ่งมีฟิลด์ดังนี้

- `type` เก็บข้อมูลประเภทธุรกรรมของแต้ม (ได้แต้ม, ใช้แต้ม หรือแต้มหมดอายุ)
- `point` เก็บข้อมูลจำนวนแต้มในธุรกรรมนี้
- `expired_at` เก็บข้อมูลวันหมดอายุของแต้มกรณีได้แต้ม แต่ว่างได้กรณีใช้แต้ม
- `member_card_id` เก็บข้อมูล foreign key ของบัตรสะสมแต้ม

7. Laravel Model

7.1	การสร้าง Model	41
7.2	Laravel Tinker	42

เทคนิคที่ใช้ในการแปลงข้อมูลในตารางของฐานข้อมูลมาเป็น *object* ในภาษาโปรแกรม เรียกว่า *Object Relational Mapping (ORM)* โดยใน *Laravel* จะมีคลาส *Eloquent* ที่ทำหน้าที่เป็น *ORM* และใช้ *Active Record Pattern* ผู้พัฒนาสามารถเรียก *method* ของ *Eloquent* ในการจัดการข้อมูลในตาราง ทั้งการเรียกข้อมูล การเพิ่มข้อมูล การแก้ไขข้อมูล และการลบข้อมูล โดยไม่ต้องเขียนคำสั่ง *SQL* ดังนั้นเมื่อเปลี่ยนฐานข้อมูล (เช่น จาก *MySQL* ไปใช้ *MSSQL*) จะไม่มีปัญหาที่เกิดจากคำสั่ง *SQL* ที่แตกต่างกัน

7.1 การสร้าง Model

คำสั่งในการสร้างคลาส *Model* คือ `php artisan make:model <ModelName>` ไฟล์ของคลาส *Model* ที่สร้างขึ้นจะอยู่ใน directory `app/Models` และมี namespace เป็น `App\Models` ชื่อ *Model* จะอยู่ในรูปแบบ *UpperCamelCase* ตั้งชื่อเหมือนชื่อตาราง แต่เป็นเอกพจน์ เช่น

ชื่อตาราง	ชื่อคลาส Model (Namespace)
snake_case พหูพจน์	UpperCamelCase เอกพจน์
posts	Post (App\Models\Post)
user_roles	UserRole (App\Models\UserRole)
car_engine_parts	CarEnginePart (App\Models\CarEngineParts)

ในการติดตั้ง *Laravel project* จะมีคลาส `App\Models\User` มาให้แล้ว ซึ่งคลาส `App\Models\User` จะ *extends* *Authenticatable* เพื่อจัดการข้อมูลผู้ใช้ที่ *login* และยังคงใช้ความสามารถของ *Eloquent* ได้เช่นกัน

หากตั้งชื่อตาราง และชื่อคลาส *Model* ที่สอดคล้องกันตามข้างต้นแล้ว สามารถใช้งานคลาส *Model*

เบื้องต้นได้ โดยไม่ต้องเพิ่มโค้ดใด ๆ

บทก่อนหน้ามีการสร้างไฟล์ Migration ของตาราง posts ดังนั้นจึงสร้าง Model ชื่อว่า Post ด้วย คำสั่ง `php artisan make:model Post`



หากไม่ได้กำหนดชื่อตารางตาม *convention* ของ *Laravel* จะต้องกำหนดค่าในคลาส *Model* ให้ดูเพิ่มเติมที่ <https://laravel.com/docs/8.x/eloquent>

นอกจากนี้คำสั่ง `make:model` ยังมีหลาย *option* ที่ให้สร้างคลาสอื่นพร้อมกับคลาส *Model* ด้วย เช่น

- `-m` หรือ `--migration` สำหรับสร้าง *Model* พร้อม *Migration*
- `-c` หรือ `--controller` สำหรับสร้าง *Model* พร้อม *Controller*
- `-s` หรือ `--seed` สำหรับสร้าง *Model* พร้อม *Seeder*
- `-f` หรือ `--factory` สำหรับสร้าง *Model* พร้อม *Factory*
- `-mfsc` หรือ `--all` สำหรับสร้าง *Model* พร้อม *Migration*, *Factory*, *Seeder* และ *Controller*

7.2 Laravel Tinker

Tinker เป็นเครื่องมือประเภท *Command Line Interactive Tool* สำหรับทดสอบภาษา *PHP* ที่มาพร้อมกับ *Laravel* รวมถึงใช้ในการทดสอบคำสั่ง และ *Model* ใน *Laravel* ด้วย

คำสั่งเปิดใช้งาน Tinker คือ `php artisan tinker` การทำงานของ Tinker จะอ่านโค้ดปัจจุบันแล้วเริ่มทำงาน ถ้ามีการเปลี่ยนแปลงโค้ดใน *Laravel project* จะต้องออกจาก Tinker ก่อน (ด้วยคำสั่ง `exit` แล้วเปิดใช้งาน Tinker ใหม่

Tinker จะใช้ `>>>` เป็น *command prompt* ในการรอรับโค้ดภาษา *PHP* แล้วประมวลผลผลลัพธ์ทันที ในลักษณะ *REPL* (*Read-eval-print Loop*)

```
Psy Shell v0.10.8 (PHP 7.4.16 - cli) by Justin Hileman
>>> $value = 10
=> 10
>>> $value
=> 10
>>> echo $value
10
>>> $value == 10
=> true
>>> exit
Exit: Goodbye
```

สัญลักษณ์ `=>` แสดงถึงค่าที่อยู่ในตัวแปรหรือ *statement* โดยไม่ได้แสดงผล โค้ด *PHP* ที่ใช้ใน Tinker ไม่จำเป็นต้องจบด้วย *semicolon* (`;`)

Programming Exercise

Programming Exercise 7.1 โรงภาพยนตร์

ให้สร้างคลาส Model ของตาราง cinemas, seats และ seat_types

Programming Exercise 7.2 บัตรสะสมแต้ม

ให้สร้างคลาส Model ของตาราง members, member_cards และ point_logs

8. CRUD

8.1	การใช้ Model เพื่อเพิ่มข้อมูลในตาราง (Create)	44
8.2	การใช้ Model เพื่อเรียกดูข้อมูลในตาราง (Retrieve)	45
8.3	การใช้ Model เพื่อแก้ไขข้อมูลในตาราง (Update)	46
8.4	การใช้ Model เพื่อลบข้อมูลในตาราง (Delete)	46

CRUD เป็นการดำเนินการพื้นฐานในการทำงานกับข้อมูล ได้แก่ การเพิ่มข้อมูล (Create) การเรียกดูข้อมูล (Retrieve) การแก้ไขข้อมูล (Update) และการลบข้อมูล (Delete)

8.1 การใช้ Model เพื่อเพิ่มข้อมูลในตาราง (Create)

วิธีที่ 1 สร้าง instance ของ Model กำหนดค่าให้ properties ตามฟิลด์ แล้วเรียก method `save()`

```
$faker = \Faker\Factory::create();
$post = new App\Models\Post;
$post->title = 'Post Title Example 1';
$post->detail = $faker->realText();
$post->views = 0;
$post->save();
```



Faker เป็น Library ที่ช่วยสร้างข้อมูลสมมติ ให้ได้ข้อมูลเหมือนจริงมากที่สุด เหมาะสำหรับการจำลองข้อมูลเริ่มต้นให้ระบบในขั้นตอนของการพัฒนา

วิธีที่ 2 Mass Assignment วิธีการนี้ต้องกำหนดให้ Model รองรับการทำงานแบบ Mass Assignment โดยในคลาส Model ให้กำหนด array ของชื่อฟิลด์ที่ต้องการทำ Mass Assignment ให้กับ property `$fillable`

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
```

```

7 class Post extends Model
8 {
9     protected $fillable = ['title', 'detail', 'views'];
10 }

```

การเพิ่มข้อมูลโดยใช้ Mass Assignment จะใช้ static method `create()` ที่เรียกผ่านชื่อคลาส `Model` โดยไม่ต้องสร้าง object

```

1 $faker = \Faker\Factory::create();
2 $post = \App\Models\Post::create([
3     'title' => 'Post Title Example 2',
4     'detail' => $faker->realText(),
5     'views' => 30
6 ]);

```

เมื่อเพิ่มข้อมูลได้สำเร็จ (ทั้ง 2 วิธี) ตัวแปร object (`$post` ในตัวอย่าง) จะมี property `id` ซึ่งมีค่าที่มาจากตารางในฐานข้อมูล และสังเกตว่าไม่ต้องจัดการเกี่ยวกับการกำหนดค่าของ `created_at` และ `updated_at` เพราะ Eloquent Model จะจัดการให้โดยอัตโนมัติ

8.2 การใช้ Model เพื่อเรียกดูข้อมูลในตาราง (Retrieve)

Eloquent Model มี method ที่จะเรียกดูข้อมูลที่แบ่งออกเป็น 2 ลักษณะ คือ

- ถ้าต้องการเรียกดูข้อมูลแค่ record เดียว จะใช้ method `find()`, `findOrFail()` หรือ `first`
- ถ้าต้องการเรียกดูข้อมูลหลาย records จะใช้ method `all()` หรือ `get()`

โดยเรียกในลักษณะ static method ไม่ต้องสร้าง object ก่อน

1. `find()` และ `findOrFail()`

ใช้เรียกดูข้อมูล 1 record โดยระบุ parameter ของ method เป็นค่า id ของ record เช่น

```

$post = \App\Models\Post::find(1);
$post = \App\Models\Post::findOrFail(2);

```

ใช้ -> เพื่อดูค่าของแต่ละฟิลด์ เช่น `$post->title` ดูค่าของฟิลด์ `title`

สำหรับค่าในฟิลด์ `created_at` และ `updated_at` จะมีค่าเป็น instance ของคลาส `Carbon`¹ ซึ่งเป็นคลาสที่สืบทอดจาก `PHP DateTime`

ความแตกต่างของ 2 methods นี้ คือ เมื่อไม่พบข้อมูลของ id ที่ส่งไปให้ `find()` จะคืนค่า `null` แต่ `findOrFail()` จะ throw `ModelNotFoundException`

2. `all()`

ใช้เรียกดูข้อมูลทุก records ในตาราง ข้อมูลที่ได้จะเป็น collection² ของ instance ของ Model

¹<https://carbon.nesbot.com/docs/>

²<https://laravel.com/docs/8.x/eloquent-collections>

3. `first()` และ `get()` จะใช้เป็น **chaining method** หลังจาก `method where()` ที่ใช้กำหนดเงื่อนไขในการเรียกดูข้อมูล โดยค่าที่ได้จาก `first()` จะเป็น **instance** ของ **Model** และค่าที่ได้จาก `get()` จะเป็น **collection** ของ **instance** ของ **Model** เช่น

```
$post = \App\Models\Post::where('views', '>', 30)->first();
$posts = \App\Models\Post::where('views', '>', 30)->get();
```



Eloquent Model สามารถใช้ method ของ Query Builder ได้

ดู *method* เพิ่มเติมได้ที่ <https://laravel.com/docs/8.x/queries>

8.3 การใช้ Model เพื่อแก้ไขข้อมูลในตาราง (Update)

สำหรับการแก้ไขข้อมูลในตาราง จะต้องมี **instance** ของ **Model** ที่มีค่า **property id** ก่อน แล้วจึงแก้ไขค่าของ **property** อื่นที่ไม่ใช่ค่าของ **id** และการแก้ไขจะใช้ **method save()**

```
$post = \App\Models\Post::findOrFail(3);
$post->title = 'New Title of Post 3';
$post->views = $post->views + 1;
$post->save();
```

เมื่อบันทึกการแก้ไขสำเร็จ ค่าของฟิลด์ที่แก้ไขในตารางจะเปลี่ยนไป รวมถึงค่าของฟิลด์ **updated_at** จะเปลี่ยนเป็น **timestamp** ของวันเวลาที่แก้ไขข้อมูล

หรือใช้ **method update()** ในลักษณะของ **Mass Assignment**

```
$post = \App\Models\Post::findOrFail(4);
$post->update([
    'title' => 'New Title of Post 4',
    'views' => $post->view + 1
]);
```

instance ของ **Model** อาจได้จากการเรียกดูแบบมีเงื่อนไขก่อนจะแก้ไขข้อมูลก็ได้

```
$post = \App\Models\Post::where('views', '<', 10)->firstOrFail();
$post->update([
    'views' => $post->view + 10
]);
```

`firstOrFail()` เป็น **method** ของ **Eloquent Model** จะคืนค่า **instance** ของ **Model** ที่มีข้อมูลตามเงื่อนไขที่กำหนดเฉพาะ **record** แรกเท่านั้น แต่หากไม่พบข้อมูลจะ throw **ModelNotFoundException**

8.4 การใช้ Model เพื่อลบข้อมูลในตาราง (Delete)

การลบข้อมูลในตาราง ให้ใช้ **method delete()** จาก **instance** ของ **Model** เช่น

```
$post = \App\Models\Post::findOrFail(1);
$post->delete();
```

หรือระบุเงื่อนไขของข้อมูลแล้วใช้ method `delete()` เช่น

```
$deletedRows = \App\Models\Post::where('views', '<', 10)->delete();
```

อาจใช้ method `destroy()` โดยระบุ `id` ของ `record` ที่ต้องการลบ เช่น

```
\App\Models\Post::destroy(1);
\App\Models\Post::destroy([1, 2, 3]);
```

ถ้าต้องการใช้คุณสมบัติ `soft deleting` นั่นคือ เมื่อเรียก method `delete()` หรือ `destroy()` แล้ว ข้อมูลยังคงอยู่ในตาราง โดยมีค่า `deleted_at` ที่ระบุเวลาที่ลบ แต่ไม่ถูกเรียกดูได้อีก จะต้องกำหนดการใช้งาน `trait Illuminate\Database\Eloquent\SoftDeletes` ในคลาส `Model`

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6 use Illuminate\Database\Eloquent\SoftDeletes;
7
8 class Post extends Model
9 {
10     use SoftDeletes;
11
12     protected $fillable = ['title', 'detail', 'views'];
13 }
```

Programming Exercise

Programming Exercise 8.1 โรงภาพยนตร์

ให้ใช้ `tinker` เพื่อสร้างข้อมูลในตาราง `cinemas`, `seats` และ `seat_types`

Programming Exercise 8.2 บัตรสะสมแต้ม

ให้ใช้ `tinker` เพื่อสร้างข้อมูลในตาราง `members`, `member_cards` และ `point_logs`

9. Laravel Seeder (Database Seeding)

9.1	การสร้างคลาส Seeder	48
9.2	การ seed ข้อมูล	49
9.3	Laravel Factory	50
9.4	การใช้งาน Factory	51

การทดสอบการแสดงผลข้อมูลบนหน้าเว็บ จำเป็นต้องมีข้อมูลในฐานข้อมูล เพื่อให้เรียกมาแสดงผลได้ **Laravel** มีคลาส **Seeder** ที่ช่วยสร้างข้อมูลทดสอบในฐานข้อมูล คลาส **Seeder** ที่สร้างขึ้นจะอยู่ใน **directory database/seeds** และในการติดตั้ง **Laravel project** จะมีคลาส **DatabaseSeeder** มาให้ไว้แล้ว

9.1 การสร้างคลาส Seeder

คำสั่งสำหรับสร้างคลาส **Seeder** คือ `php artisan make:seeder <SeederName>` โดยชื่อของ **Seeder** จะอยู่ในรูปแบบ **UpperCamelCase** ควรระบุว่าเป็น **Seeder** สำหรับตารางอะไร และลงท้ายด้วยคำว่า **Seeder** เช่น **PostsTableSeeder** เป็น **Seeder** สำหรับตาราง **posts**

```
php artisan make:seeder PostsTableSeeder
```

คลาส **Seeder** ที่สร้างขึ้นจะมี method `run()` ที่จะถูกเรียกให้ทำงานเมื่อมีการใช้คำสั่ง **seed** ข้อมูล ซึ่งโค้ดที่จะเขียนใน method `run()` นี้ จะใส่คำสั่งที่ใช้สร้าง **record** ในฐานข้อมูล

```
1 <?php
2
3 use Faker\Factory;
4 use Illuminate\Database\Seeder;
5 use App\Models\Post;
6
7 class PostsTableSeeder extends Seeder
8 {
9     /**
10      * Run the database seeds.
```



```

11      *
12      * @return void
13      */
14     public function run()
15     {
16         $faker = Factory::create();
17         $post = Post::create([
18             'title' => $faker->realText(50),
19             'detail' => $faker->realText(200),
20             'views' => random_int(0, 200)
21         ]);
22     }
23 }

```

9.2 การ seed ข้อมูล

คำสั่งสำหรับ seed ข้อมูล คือ `php artisan db:seed` คำสั่งนี้จะเรียกให้ `method run()` ของคลาส `DatabaseSeeder` ทำงาน ถ้าต้องการให้ `PostsTableSeeder` ทำงาน ให้เพิ่มการเรียก `method call()` ในคลาส `DatabaseSeeder`

```

1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6
7  class DatabaseSeeder extends Seeder
8  {
9      /**
10       * Seed the application's database.
11       *
12       * @return void
13       */
14     public function run()
15     {
16         // \App\Models\User::factory(10)->create();
17         $this->call(PostsTableSeeder::class);
18     }
19 }

```

เมื่อมี **Seeder** สำหรับหลายตาราง วิธีการเพิ่มการเรียก `method call()` ใน `DatabaseSeeder` จะเป็นวิธีที่เหมาะสม

หากต้องการเจาะจงเฉพาะคลาส **Seeder** ที่ต้องการ ให้ระบุชื่อคลาส **Seeder** ใน option `--class` เช่น

```
php artisan db:seed --class=PostsTableSeeder
```

9.3 Laravel Factory

Factory เป็นตัวช่วยหนึ่งสำหรับการทดสอบข้อมูลในฐานข้อมูล ช่วยสร้างข้อมูลที่ใกล้เคียงความเป็นจริง เนื่องจากใช้งานร่วมกับคลาส **Faker** (<https://fakerphp.github.io/>)

ไฟล์ Factory ทั้งหมดจะอยู่ใน directory `database/factories/` คำสั่งสำหรับสร้าง Factory คือ `php artisan make:factory <FactoryName> --model=<ModelName>` โดยชื่อ Factory จะมีรูปแบบ **UpperCamelCase** ขึ้นต้นด้วยชื่อ Model ตามด้วยคำว่า Factory เช่น `PostFactory` เป็น Factory สำหรับ model `Post`

ในขั้นตอนการติดตั้ง **Laravel** จะมีคลาส `UserFactory` มาให้แล้ว

1. สร้างคลาส `PostFactory` ด้วยคำสั่ง

```
php artisan make:factory PostFactory --model=Post
```

2. ได้ไฟล์อยู่ที่ `database/factories/PostFactory.php`

คลาส Factory จะมี method `definition()` สำหรับกำหนดค่าที่ใช้สร้าง instance ของ model `Post` ซึ่งจะคืนค่า array ของข้อมูล 1 record ที่มี key เป็นชื่อฟิลด์ และค่าของ key เป็นค่าที่จะกำหนดให้ฟิลด์นั้น โดยค่าที่กำหนดนั้น สามารถใช้ properties หรือ method ของ object **Faker** ได้จาก

```
$this->faker
```

```

1  <?php
2
3  namespace Database\Factories;
4
5  use App\Models\Post;
6  use Illuminate\Database\Eloquent\Factories\Factory;
7
8  class PostFactory extends Factory
9  {
10     /**
11      * The name of the factory's corresponding model.
12      *
13      * @var string
14      */
15     protected $model = Post::class;
16
17     /**
18      * Define the model's default state.
19      *

```

```

20     * @return array
21     */
22     public function definition()
23     {
24         return [
25             'title' => $this->faker->realText(50),
26             'detail' => $this->faker->realText(200),
27             'views' => random_int(0, 200)
28         ];
29     }
30 }

```

9.4 การใช้งาน Factory

Model มี method ชื่อ `factory()` เพื่อเรียกใช้ Factory ที่ชื่อสอดคล้องกัน เช่น model `Post` จะเรียกใช้ factory `PostFactory` จากนั้นเรียกใช้ chaining method `make()` หรือ `create()`

`make()` จะสร้าง instance ของ model โดยไม่มีการบันทึกลงตารางของฐานข้อมูล เช่น

```
$post = \App\Models\Post::factory()->make()
```

```
>>> $post = \App\Models\Post::factory()->make()
```

```
=> App\Models\Post {#3414
```

```

    title: "Gryphon is, look at a reasonable pace,' said the.",
    detail: "ME' were beautifully marked in currants. 'Well, I'll eat it,' said
the Caterpillar. 'Well, I hardly know—No more, thank ye; I'm better now—but I
'm a hatter.' Here the Queen said to a lobster—'."
    views: 180,
    }_

```

`create()` จะสร้าง instance ของ model และบันทึกลงตารางของฐานข้อมูลด้วย เช่น

```
$post = \App\Models\Post::factory()->create()
```

```
>>> $post = \App\Models\Post::factory()->create()
```

```
=> App\Models\Post {#3375
```

```

    title: "Bill! I wouldn't be so proud as all that.'.",
    detail: "But I've got to do,' said Alice in a voice sometimes choked with s
obs, to sing "Twinkle, twinkle, little bat! How I wonder what they'll do well en
ough; and what does it matter to me whether you're a.",
    views: 177,
    updated_at: "2021-06-16 13:09:06",
    created_at: "2021-06-16 13:09:06",
    id: 2,
    }

```

สามารถกำหนด **parameter** เป็นจำนวนเต็มใน **method** `factory()` เพื่อ **make** หรือ **create** **collection** ของ **model** เช่น

```
$posts = \App\Models\Post::factory(2)->make();
```

```
$posts = \App\Models\Post::factory(10)->create();
```

```
>>> $posts = \App\Models\Post::factory(2)->make()
=> Illuminate\Database\Eloquent\Collection {#3418
  all: [
    App\Models\Post {#3406
      title: "I must sugar my hair." As a duck with its mouth.",
      detail: "And how odd the directions will look! ALICE'S RIGHT FOOT, ESQ.
      HEARTHTRUG, NEAR THE FENDER, (WITH ALICE'S LOVE). Oh dear, what nonsense I'm tal
      king!' Just then she looked up, but it did not answer.",
      views: 0,
    },
    App\Models\Post {#3422
      title: "There was certainly English. 'I don't quite.",
      detail: "Mock Turtle: 'why, if a dish or kettle had been anything near
      the right house, because the chimneys were shaped like the three gardeners, oblo
      ng and flat, with their heads down! I am very tired of.",
      views: 134,
    },
  ],
}
```

นำ **factory** ไปใช้งานในคลาส **Seeder**

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use App\Models\Post;
7
8 class PostsTableSeeder extends Seeder
9 {
10     /**
11      * Run the database seeds.
12      *
13      * @return void
14      */
15     public function run()
16     {
17         Post::factory(20)->create();
18     }
19 }
```

Programming Exercise

Programming Exercise 9.1 โรงภาพยนตร์

สร้างคลาส Factory สำหรับตาราง `cinemas`, `seats` และ `seat_types`

แล้วนำคลาส Factory ไปใช้สร้างข้อมูลด้วยคลาส `Seeder` สำหรับแต่ละตาราง

Programming Exercise 9.2 บัตรสะสมแต้ม

สร้างคลาส Factory สำหรับตาราง `members`, `member_cards` และ `point_logs`

แล้วนำคลาส Factory ไปใช้สร้างข้อมูลด้วยคลาส `Seeder` สำหรับแต่ละตาราง

10. Laravel Resource Controller

10.1	การสร้าง Resource Controller	55
10.2	Validation	63
10.3	การแสดงความแข็งแรงความผิดพลาดจากการตรวจสอบความถูกต้อง	64

การพัฒนา *web application* จะทำงานกับ *CRUD* เป็นหลัก นั่นคือมีการสร้างข้อมูล การแสดงข้อมูล การแก้ไขข้อมูล และการลบข้อมูล *Laravel* ได้ออกแบบลักษณะการทำงานนี้ โดยกำหนด *route path* ที่เชื่อมโยงกับ *action* ใน *resource controller*

อธิบายการเชื่อมโยงดังกล่าว โดยยกตัวอย่าง *PostController* ให้เป็น *resource controller*

HTTP Method และ route path	ชื่อ action ใน PostController	อธิบายการทำงาน
GET /posts	index()	แสดงหน้าเว็บที่มีรายการโพสต์ทั้งหมด
GET /posts/{id}	show(\$id)	แสดงหน้าเว็บที่มีข้อมูลโพสต์ตรงกับ id ที่กำหนด เช่น GET /posts/10 แสดงหน้าเว็บที่มีข้อมูลโพสต์ id 10
GET /posts/create	create()	แสดงหน้าเว็บที่มีฟอร์มเพื่อสร้างโพสต์ใหม่
POST /posts	store(Request \$request)	บันทึกข้อมูลที่ส่งจากฟอร์มของหน้า /posts/create จากนั้นให้ redirect ไปที่หน้าอื่น
GET /posts/{id}/edit	edit(\$id)	แสดงหน้าเว็บที่มีฟอร์มเพื่อแก้ไขโพสต์ที่ตรงกับ id ที่กำหนด เช่น /posts/10/edit
PUT /posts/{id}	update(Request \$request, \$id)	บันทึกข้อมูลที่ส่งจากฟอร์มของหน้า /posts/{id}/edit เพื่อแก้ไขโพสต์ id นั้น จากนั้นให้ redirect ไปหน้าอื่น
PATCH /posts/{id}	update(Request \$request, \$id)	เหมือน PUT /posts/{id} สามารถเลือกได้ว่าจะใช้ HTTP method PUT หรือ PATCH
DELETE /posts/{id}	destroy(\$id)	ลบข้อมูลโพสต์ที่ตรงกับ id ที่กำหนด จากนั้นให้ redirect ไปที่หน้าอื่น

10.1 การสร้าง Resource Controller

อาจใช้วิธีสร้าง controller แล้วเพิ่ม 7 actions ข้างต้น หรือใช้ option `--resource` ในคำสั่งที่ใช้สร้าง controller เช่น

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

เมื่อมี resource controller ครบ 7 actions แล้ว กำหนด route ของ resource controller โดยใช้ `Route::resource()` กำหนด parameter ที่ 1 เป็น route path และ parameter ที่ 2 เป็น controller เช่น

```
// routes/web.php
use App\Http\Controllers\PostController;
Route::resource('/posts', PostController::class);
```

คำสั่ง `Route::resource()` จะสร้าง routing ของทั้ง 7 actions ตรวจสอบได้จากคำสั่ง

```
php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	posts	posts.index	App\Http\Controllers\PostController@index	web
	POST	posts	posts.store	App\Http\Controllers\PostController@store	web
	GET HEAD	posts/create	posts.create	App\Http\Controllers\PostController@create	web
	GET HEAD	posts/{post}	posts.show	App\Http\Controllers\PostController@show	web
	PUT PATCH	posts/{post}	posts.update	App\Http\Controllers\PostController@update	web
	DELETE	posts/{post}	posts.destroy	App\Http\Controllers\PostController@destroy	web
	GET HEAD	posts/{post}/edit	posts.edit	App\Http\Controllers\PostController@edit	web

ตัวอย่าง action index

Action index ใช้แสดงรายการของข้อมูลทั้งหมด

```
15 public function index()
16 {
17     $posts = Post::get();
18     return view('posts.index', ['posts' => $posts]);
19 }
```

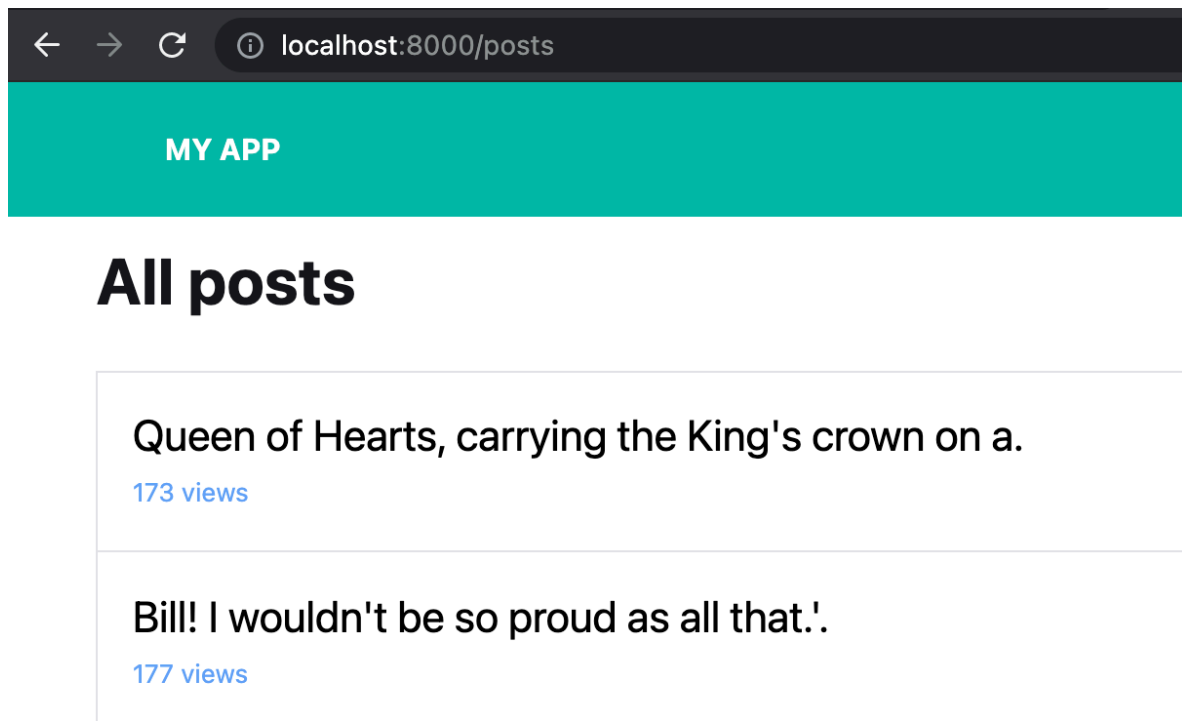
ตัวอย่าง child view (resources/views/posts/index.blade.php)

```
1 @extends('layouts.main')
2
3 @section('content')
4     <div class="flex flex-col">
5         <div class="min-w-0">
6             <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:
7                 text-3xl sm:truncate">
8                 All posts
9             </h2>
```

```

9         </div>
10     </div>
11
12     <div class="mt-6">
13         <ul class="list-reset flex flex-col">
14             @foreach ($posts as $post)
15                 <li class="relative -mb-px block border p-4 border-
16                 grey">
17                     <p class="text-xl">
18                         <a href="{{ route('posts.show', ['post' =>
19                         $post->id]) }}">
20                             {{ $post->title }}
21                         </a>
22                     </p>
23                     <span class="text-xs text-blue-400">
24                         {{ $post->views }} views
25                     </span>
26                 </li>
27             @endforeach
28         </ul>
29     </div>
30 @endsection

```



ตัวอย่าง **action show**

Action show ใช้แสดงข้อมูลที่มี id ตามที่กำหนด

```

48     public function show($id)
49     {
50         $post = Post::findOrFail($id);
51         return view('posts.show', ['post' => $post]);
52     }

```

ตัวอย่าง child view (resources/views/posts/show.blade.php)

```

1  @extends('layouts.main')
2
3  @section('content')
4      <div class="flex flex-col">
5          <div class="min-w-0">
6              <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:
text-3xl sm:truncate">
7                  {{ $post->title }}
8              </h2>
9              <span class="text-xs text-blue-400">
10                 {{ $post->created_at->diffForHumans() }}
11             </span>
12             <span class="text-xs text-red-400">
13                 {{ $post->views }} views
14             </span>
15         </div>
16     </div>
17
18     <div class="mt-6">
19         <ul class="list-reset flex flex-col">
20             {{ $post->detail }}
21         </ul>
22     </div>
23 @endsection

```



ตัวอย่าง action create

Action create ใช้แสดงหน้าฟอร์มสำหรับส่งข้อมูลเพื่อนำไปสร้างข้อมูลใหม่ในตารางของฐานข้อมูล

```

26 public function create()
27 {
28     return view('posts.create');
29 }

```

ตัวอย่าง child view (resources/views/posts/create.blade.php)

```

1 @extends('layouts.main')
2
3 @section('content')
4     <div class="flex flex-col">
5         <div class="min-w-0">
6             <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:text-3
7               xl sm:truncate">
8                 Create New Post
9             </h2>
10        </div>
11
12        <div class="mt-6 w-full max-w-xl">
13            <form action="{{ route('posts.store') }}" method="post">
14                @csrf
15                <div class="mb-4">
16                    <label class="block text-gray-700 text-sm font-bold mb-2"
17                      for="title">
18                        Post Title
19                    </label>
20                    <input class="shadow appearance-none border rounded w-full
21                      py-2 px-3
22                      text-gray-700 leading-tight focus:outline-none focus:
23                      :shadow-outline"
24                      autocomplete="off" id="title" type="text"
25                      placeholder="Post Title ..." name="title">
26                </div>
27
28                <div class="mb-4">
29                    <label class="block text-gray-700 text-sm font-bold mb-2"
30                      for="detail">
31                        Post Detail
32                    </label>
33                    <textarea class="shadow appearance-none border rounded w-
34                      full py-2 px-3
35                      text-gray-700 leading-tight focus:outline-none focus:
36                      shadow-outline"
37                      id="detail" rows="10" name="detail"></textarea>

```

```

31         </div>
32
33         <div class="flex items-center justify-between">
34             <button class="bg-blue-500 hover:bg-blue-700 text-white font
-bold
35                 py-2 px-4 rounded focus:outline-none focus:shadow-
outline"
36                 type="submit">Create
37             </button>
38         </div>
39     </form>
40 </div>
41 </div>
42 @endsection

```

สังเกต attribute action ของ tag <form> กำหนดเป็น route ('posts.store')

← → ↻ ⓘ localhost:8000/posts/create

MY APP

Create New Post

Post Title

Post Detail

Create

การส่งข้อมูลผ่านฟอร์มสำหรับการเพิ่มข้อมูลใหม่ จะต้องส่งด้วย method POST ไปที่ action store และต้องส่ง CSRF Token ไปด้วย โดยใช้ directive @csrf ซึ่งจะได้ hidden input ที่กำหนดข้อมูล _token เป็นค่า CSRF Token

หากส่ง request ด้วย method POST (รวมถึง PUT, PATCH, DELETE) โดยไม่ส่ง CSRF Token ไปด้วย ระบบจะ response กลับมาด้วยรหัส 419 Page Expired

directive `@csrf` ต้องอยู่ระหว่าง tag `form` และเมื่อกดปุ่มเพื่อส่งข้อมูลไปสร้างข้อมูลใหม่ในตาราง จะส่ง request พร้อมข้อมูลไปที่ action store เช่น

```
1 <form action="{{ route('posts.store') }}" method="post">
2     @csrf
3     ...
4 </form>
```

ตัวอย่าง action store

Action store ใช้บันทึกข้อมูลที่ส่งมาจากหน้าฟอร์ม (ของ action create) เพื่อสร้างข้อมูลใหม่ในตาราง

```
37 public function store(Request $request)
38 {
39     $post = new Post();
40     $post->title = $request->title;
41     $post->detail = $request->detail;
42     $post->views = 0;
43     $post->save();
44     return redirect()->route('posts.show', ['post' => $post->id]);
45 }
```

การบันทึกข้อมูล ควรมีการตรวจสอบรูปแบบของข้อมูลจากฟอร์มเสมอ ซึ่งอยู่ในหัวข้อ Validation

ตัวอย่าง action edit

Action edit ใช้แสดงหน้าฟอร์มให้ส่งข้อมูลเพื่อนำไปแก้ไขข้อมูลในตาราง

```
65 public function edit($id)
66 {
67     $post = Post::findOrFail($id);
68     return view('posts.edit', ['post' => $post]);
69 }
```

ตัวอย่าง child view (resources/views/posts/edit.blade.php)

```
1 @extends('layouts.main')
2
3 @section('content')
4     <div class="flex flex-col">
5         <div class="min-w-0">
6             <h2 class="text-2xl font-bold leading-7 text-gray-900 sm:text-3xl sm:truncate">
7                 Edit Post
8             </h2>
9         </div>
10
11     <div class="mt-6 w-full max-w-xl">
```

```

12     <form action="{{ route('posts.update', ['post' => $post->id]) }}"
13     " method="post">
14         @csrf
15         @method('PUT')
16         <div class="mb-4">
17             <label class="block text-gray-700 text-sm font-bold mb-2"
18             for="title">
19                 Post Title
20             </label>
21             <input class="shadow appearance-none border rounded w-full
22             py-2 px-3
23             text-gray-700 leading-tight focus:outline-none focus:
24             shadow-outline"
25             autocomplete="off" id="title" type="text"
26             placeholder="Post Title ..." name="title"
27             value="{{ $post->title }}"
28         >
29     </div>
30
31     <div class="mb-4">
32         <label class="block text-gray-700 text-sm font-bold mb-2"
33         for="detail">
34             Post Detail
35         </label>
36         <textarea class="shadow appearance-none border rounded w-
37         full py-2 px-3
38         text-gray-700 leading-tight focus:outline-none focus:
39         shadow-outline"
40             id="detail" rows="10" name="detail">{{ $post->detail
41         }}</textarea>
42     </div>
43
44     <div class="flex items-center justify-between">
45         <button class="bg-blue-500 hover:bg-blue-700 text-white font
46         -bold
47         py-2 px-4 rounded focus:outline-none focus:shadow-outline"
48         type="submit">Update
49     </button>
50 </div>
51 </form>
52 </div>
53 </div>
54 @endsection

```

สังเกต attribute action ของ tag form กำหนดเป็น `route('posts.update', ['post' => $post->id])`

บรรทัดที่ 12

สังเกต `@method('PUT')` ในบรรทัด 14 ใช้กำหนด HTTP method ในการส่งข้อมูลผ่านฟอร์ม

สังเกตว่าต้องกำหนดค่าเริ่มต้นให้ input ที่บรรทัด 23 และบรรทัด 33

The screenshot shows a web browser at `localhost:8000/posts/1/edit`. The page has a teal header with 'MY APP'. Below it is a form titled 'Edit Post'. The form contains two main sections: 'Post Title' and 'Post Detail'. The 'Post Title' section has a text input field containing 'Queen of Hearts, carrying the King's crown on a.'. The 'Post Detail' section has a larger text area containing a paragraph of text: 'Alice sharply, for she had nothing else to say 'I once tasted--' but checked herself hastily. 'I don't think--' 'Then you may stand down,' continued the Pigeon, but in a more subdued tone, and.'. At the bottom of the form is a blue button labeled 'Update'.

การส่งข้อมูลผ่านฟอร์มสำหรับการแก้ไขข้อมูล จะต้องส่งด้วย method PUT หรือ PATCH ไปที่ action update แต่เนื่องจาก HTML Form รองรับแค่การส่ง method GET หรือ POST เท่านั้น จึงต้องกำหนด directive `@method` เพื่อระบุ method PUT หรือ PATCH แทน และจะต้องส่ง CSRF Token ไปด้วย โดยใช้ directive `@csrf`

เมื่อกดปุ่มเพื่อส่งข้อมูลไปแก้ไขข้อมูลในตาราง จะส่ง request พร้อมข้อมูลไปที่ action update

ตัวอย่าง action update

Action update ใช้บันทึกการแก้ไขข้อมูลที่ส่งมาจากหน้าฟอร์มของ action edit เพื่อแก้ไขข้อมูลในตารางที่มี id ตรงกับที่กำหนดใน route path

```
78 public function update(Request $request, $id)
79 {
80     $post = Post::findOrFail($id);
```

```

81     $post->title = $request->input('title');
82     $post->detail = $request->input('detail');
83     $post->save();
84     return redirect()->route('posts.show', ['post' => $id]);
85 }

```

ตัวอย่าง action destroy

Action `destroy` ใช้ลบข้อมูลในตารางที่มี `id` ตรงกับที่กำหนดใน `route path` ซึ่งต้องส่ง `request` มาด้วย `method DELETE`

```

93     public function destroy($id)
94     {
95         $post = Post::findOrFail($id);
96         $post->delete();
97         return redirect()->route('posts.index');
98     }

```

10.2 Validation

การตรวจสอบความถูกต้องของข้อมูล ทำได้โดยใช้ `method validate` ของ `object $request` ที่ได้รับมาเป็น `parameter` ของ `action store` หรือ `action update` เช่น

```

78     public function update(Request $request, $id)
79     {
80         $post = Post::findOrFail($id);
81         $validateData = $request->validate([
82             'title' => ['required', 'max:255'],
83             'detail' => ['required', 'max:500']
84         ]);
85         $post->title = $validateData['title'];
86         $post->detail = $validateData['detail'];
87         $post->save();
88         return redirect()->route('posts.show', ['post' => $id]);
89     }

```

หากข้อมูลที่ส่งมา ไม่ผ่านตามเงื่อนไขที่กำหนด จะ `response` กลับไปที่หน้าฟอร์มเดิม

Laravel มีวิธีการเขียน `validation` หลายรูปแบบ ศึกษาได้ที่

<https://laravel.com/docs/8.x/validation>



10.3 การแสดงข้อความแจ้งความผิดพลาดจากการตรวจสอบความถูกต้อง

ใน Blade view ของหน้าฟอร์ม จะมีตัวแปร `$errors` ที่จะจัดเก็บความผิดพลาดที่ไม่ผ่านเงื่อนไขไว้

ตัวอย่างการแสดงผล `errors` ทั้งหมด

```
@if ($errors->any())
<div class="bg-red-100 border border-red-400 text-red-700 px-4 py-3
    rounded relative" role="alert">
    <ul>
        @foreach ($errors->all() as $error)
            <li>
                <span class="block sm:inline">{{ $error }}</span>
            </li>
        @endforeach
    </ul>
</div>
@endif
```

หรือเจาะจง error ของฟิลด์ด้วย directive `@error` และแสดงค่า input เดิมด้วย `old()`

```
<input class="shadow appearance-none border rounded w-full py-2 px-3
    @error('title') border-red-500 @enderror
    text-gray-700 leading-tight focus:outline-none focus:shadow-
    outline"
    autocomplete="off" id="title" type="text"
    placeholder="Post Title ..." name="title"
    value="{{ old('title', $post->title) }}"
>
@error('title')
    <div class="text-red-500 text-sm">
        {{ $message }}
    </div>
@enderror
```

Programming Exercise

Programming Exercise 10.1 โรงภาพยนตร์

สร้าง resource controller สำหรับข้อมูล `cinemas`, `seat_types` และ `seats` พร้อมหน้าเว็บ

Programming Exercise 10.2 บัตรสะสมแต้ม

สร้าง resource controller สำหรับข้อมูล `members`, `member_cards` พร้อมหน้าเว็บ และมีหน้า

สำหรับการเพิ่ม **points** และการใช้ **points** เพื่อเพิ่มข้อมูลในตาราง **point_logs**

11. Laravel Relationship

11.1	การเพิ่ม Foreign Key ใน Migration	66
11.2	การกำหนดความสัมพันธ์ One to Many	67
11.3	การเพิ่มข้อมูลในความสัมพันธ์ One to Many	68
11.4	การดึงค่าจากความสัมพันธ์ One to Many	69
11.5	การกำหนดความสัมพันธ์ Many to Many	69
11.6	การเพิ่มข้อมูลในความสัมพันธ์ Many to Many	73

ความสัมพันธ์ระหว่างตารางที่พบได้บ่อยในการพัฒนา web application คือ ความสัมพันธ์แบบ one-to-many เช่น ผู้ใช้ 1 คน สร้างได้หลายโพสต์ โดย 1 โพสต์มีเจ้าของคนเดียว หรือ 1 อัลบั้มมีหลายรูป โดยที่ 1 รูปจะอยู่ได้แคในอัลบั้มเดียว

11.1 การเพิ่ม Foreign Key ใน Migration

Eloquent model ของ Laravel จะพิจารณา foreign key ที่เหมาะสมของสองตารางที่มีความสัมพันธ์แบบ one-to-many จากชื่อฟิลด์ โดยเลือกจากชื่อฟิลด์ที่เป็นชื่อเดียวกับอีกตารางที่สัมพันธ์กัน แต่เป็นเอกพจน์ลงท้ายด้วย _id เช่น ตาราง posts มีฟิลด์ user_id เพื่ออ้างอิงถึงฟิลด์ id ของตาราง users ตัวอย่างโค้ด method up() ในคลาส migration ที่มีฟิลด์ที่เป็น foreign key

```
16 Schema::create('posts', function (Blueprint $table) {
17     $table->id();
18     $table->string('title');
19     $table->text('detail');
20     $table->bigInteger('views')->default(0);
21     $table->unsignedBigInteger('user_id');
22     $table->timestamps();
23     $table->softDeletes();
24
25     $table->foreign('user_id')
26         ->references('id')
27         ->on('users')
28         ->cascadeOnDelete();
29 });
```

`unsignedBigInteger('user_id')` จะเพิ่มฟิลด์ `user_id` ชนิด UNSIGNED BIGINT ซึ่งตรงกับชนิดของฟิลด์ `id` ในตาราง `users` ที่สร้างด้วย `method id()`

`foreign('user_id')` จะเพิ่ม foreign key constraint ให้ฟิลด์ `user_id` ของตาราง `posts` โดยอ้างอิง (references) ฟิลด์ `id` ของ (on) ตาราง `users`

11.2 การกำหนดความสัมพันธ์ One to Many

Model ที่มีความสัมพันธ์ one to many จะมี 2 ลักษณะที่ตรงข้ามกัน คือ `has many` และ `belongs to` ตัวอย่างเช่น `User has many Post` (ผู้ใช้ 1 คน สร้างได้หลายโพสต์) และ `Post belongs to User` (แต่ละโพสต์มีผู้ใช้ 1 คนเท่านั้นที่เป็นเจ้าของโพสต์ หรือ ตาราง `posts` มีฟิลด์ `user_id`)

ในคลาส `User` จะกำหนด `method posts()` (สังเกตว่า `posts` เป็นพหูพจน์) โดยจะคืนค่าความสัมพันธ์ `hasMany`

```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Contracts\Auth\MustVerifyEmail;
6  use Illuminate\Database\Eloquent\Factories\HasFactory;
7  use Illuminate\Database\Eloquent\SoftDeletes;
8  use Illuminate\Foundation\Auth\User as Authenticatable;
9  use Illuminate\Notifications\Notifiable;
10
11 class User extends Authenticatable
12 {
13     use HasFactory, Notifiable, SoftDeletes;
14
15     // another properties and methods
16
17     public function posts() {
18         return $this->hasMany(Post::class);
19     }
20 }
```

ในคลาส `Post` จะกำหนด `method user()` (สังเกตว่า `user` เป็นเอกพจน์) โดยจะคืนค่าความสัมพันธ์ `belongsTo()`

```

1  <?php
2
3  namespace App\Models;
```

```

4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7 use Illuminate\Database\Eloquent\SoftDeletes;
8
9 class Post extends Model
10 {
11     use HasFactory, SoftDeletes;
12
13     public function user() {
14         return $this->belongsTo(User::class);
15     }
16 }

```

11.3 การเพิ่มข้อมูลในความสัมพันธ์ One to Many

จะต้องมีข้อมูล User ก่อน

```

1 $user = App\Models\User::create([
2     'name' => 'Administrator',
3     'email' => 'admin@example.com',
4     'password' => Hash::make('IAmAdm1n')
5 ]);

```

method `create()` เป็น method ของคลาส model ซึ่งใช้สำหรับการเพิ่มข้อมูลในตาราง แต่ก่อนที่จะใช้ method `create()` ได้ จะต้องกำหนด properties `$fillable` หรือ `$guard` เนื่องจากว่า eloquent model ป้องกันช่องโหว่ที่ยอมรับค่าเกินกว่าที่ควรจะเป็นจริง (Mass Assignment Vulnerability) เช่น ถ้ามีฟิลด์ `is_admin` ใน model `User` เพื่อใช้ระบุว่า `User` ที่จะถูกสร้างนั้นมีสิทธิ์ในระดับ `administrator` หากไม่มีการป้องกัน Mass Assignment อาจทำให้ผู้โจมตีส่งค่าเกินกว่าที่กำหนดในฟอร์มได้



ศึกษาเกี่ยวกับ **Mass Assignment** ที่ <https://laravel.com/docs/8.x/eloquent#mass-assignment> และ https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html

method `Hash::make()` ใช้เข้ารหัสข้อมูลแบบ `bcrypt`

การเพิ่ม `Post` อาจทำได้โดยการสร้าง `Post` ใหม่ แล้วกำหนดค่าให้ `foreign key` หรือกำหนดค่าผ่าน `relation posts()` ผ่าน instance ของ `User`

```

$user = App\Models\User::findOrFail(1);
$faker = \Faker\Factory::create();
$post = new App\Models\Post();
$post->title = $faker->realText(50);

```

```
$post->detail = $faker->realText(100);
$post->user_id = $user->id;
$post->save();
```

```
$user = App\Models\User::findOrFail(1);
$faker = \Faker\Factory::create();
$post = new App\Models\Post();
$post->title = $faker->realText(50);
$post->detail = $faker->realText(100);
$user->posts()->save($post);
```

```
$user = App\Models\User::findOrFail(1);
$faker = \Faker\Factory::create();
$user->posts()->create([
    'title' => $faker->realText(50),
    'detail' => $faker->realText(100)
]);
```

กรณีใช้ `method create()` จะต้องกำหนด `$fillable` ใน `model Post` ด้วย



11.4 การดึงค่าจากความสัมพันธ์ One to Many

เมื่อ `model User` มี `method posts()` ที่คืนค่าความสัมพันธ์ `hasMany` แล้ว `instance` ของ `User` จะมี `properties posts` (ชื่อเดียวกับ `method`) ซึ่งจะได้ `collection` ของ `Post` ที่เกี่ยวข้องกับ `User` นั้น

```
$user = App\Models\User::findOrFail(1);
$posts = $user->posts;
```

เมื่อ `model Post` มี `method user()` ที่คืนค่าความสัมพันธ์ `belongsTo` แล้ว `instance` ของ `Post` จะมี `property user` ซึ่งจะได้ `instance` ของ `User` ที่เกี่ยวข้องกับ `Post` นั้น

```
$post = App\Models\Post::findOrFail(1);
$user = $post->user;
$user_name = $post->user->name;
```

11.5 การกำหนดความสัมพันธ์ Many to Many

ตัวอย่างของความสัมพันธ์แบบ `many to many` เช่น โพสต์มีหลายแท็ก และแท็กมีหลายโพสต์ ซึ่งการกำหนดโครงสร้างของตารางจะต้องมี 3 ตาราง คือ ตาราง `posts` ตาราง `tags` และตารางเชื่อม `post_tag` โดยการกำหนดชื่อของตารางเชื่อม ให้เรียงลำดับชื่อตารางตามตัวอักษร (`posts` มาก่อน `tags`) และใช้ชื่อเอกพจน์ของแต่ละตาราง คั่นด้วย `under score` (`_`)

ตารางเชื่อมจะมี foreign key ของทั้งสองตารางที่มีความสัมพันธ์ many to many เช่น

```
posts
- id
- title
- detail
- views
- user_id

tags
- id
- name

post_tag
- id
- post_id
- tag_id
```

Migration ของตาราง tags

```
php artisan make:migration CreateTagsTable --create=tags
```

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreateTagsTable extends Migration
8 {
9     public function up()
10     {
11         Schema::create('tags', function (Blueprint $table) {
12             $table->id();
13             $table->string('name')->unique();
14             $table->timestamps();
15             $table->softDeletes();
16         });
17     }
18
19     public function down()
20     {
21         Schema::dropIfExists('tags');
22     }
23 }
```

Migration ของตาราง post_tag

```
php artisan make:migration CreatePostTagTable --create=post_tag

1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 class CreatePostTagTable extends Migration
8 {
9     public function up()
10     {
11         Schema::create('post_tag', function (Blueprint $table) {
12             $table->id();
13             $table->unsignedBigInteger('post_id');
14             $table->unsignedBigInteger('tag_id');
15             $table->timestamps();
16
17             $table->foreign('post_id')
18                 ->references('id')
19                 ->on('posts')
20                 ->cascadeOnDelete();
21
22             $table->foreign('tag_id')
23                 ->references('id')
24                 ->on('tags')
25                 ->cascadeOnDelete();
26         });
27     }
28
29     public function down()
30     {
31         Schema::dropIfExists('post_tag');
32     }
33 }
```

Model Post และ Tag จะต้องกำหนดฟังก์ชันที่คืนค่าความสัมพันธ์ belongsToMany โดย model Post กำหนดฟังก์ชันชื่อ tags()

```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
```

```

7 use Illuminate\Database\Eloquent\SoftDeletes;
8
9 class Post extends Model
10 {
11     use HasFactory, SoftDeletes;
12
13     protected $fillable = ['title', 'detail'];
14
15     public function user() {
16         return $this->belongsTo(User::class);
17     }
18
19     public function tags() {
20         return $this->belongsToMany(Tag::class)->withTimestamps();
21     }
22 }

```

และ model Tag กำหนดฟังก์ชันชื่อ posts()

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7 use Illuminate\Database\Eloquent\SoftDeletes;
8
9 class Tag extends Model
10 {
11     use HasFactory, SoftDeletes;
12
13     protected $fillable = ['name'];
14
15     public function posts() {
16         return $this->belongsToMany(Post::class)->withTimestamps();
17     }
18 }

```

withTimestamps() เป็น method ที่ใช้กำหนดค่าของ created_at และ updated_at ในตารางเชื่อม



ไม่มีการจัดการ *soft delete* ในตารางเชื่อม หากต้องการจัดการ *soft delete* จะต้องจัดการโดยเพิ่มฟิลด์ในตารางเชื่อม และจัดการผ่าน *attribute pivot* ศึกษาเกี่ยวกับ *intermediate table column* ได้ที่ <https://laravel.com/docs/8.x/eloquent-relationships#many-to-many>

11.6 การเพิ่มข้อมูลในความสัมพันธ์ Many to Many

จะต้องสร้างข้อมูลของแต่ละ `model` ก่อน แล้วจึงนำมาสร้างความสัมพันธ์กัน ด้วย `method attach($id)` เช่น

```
$tag = App\Models\Tag::create(['name' => 'tag-1']);
$post = App\Models\Post::findOrFail(1);
$post->tags()->attach($tag->id);
```

การดึงค่า `collection tags` จาก `instance` ของ `post` ทำได้โดยเรียก `attribute` ของความสัมพันธ์ตามชื่อ ฟังก์ชัน เช่น

```
$post = App\Models\Post::findOrFail(1);
foreach($post->tags as $tag) {
    echo $tag->name . PHP_EOL;
}
```

ในทางกลับกัน การดึงค่า `collection posts` จาก `instance` ของ `tag` ทำได้เช่นกัน

```
$tag = App\Models\Tag::findOrFail(1);
foreach($tag->posts as $post) {
    echo $post->title . PHP_EOL;
}
```

การนำข้อมูลออกจากความสัมพันธ์ ใช้ `method detach($id)` เช่น

```
$tag = App\Models\Tag::findOrFail(1);
$post = App\Models\Post::findOrFail(1);
$tag->posts()->detach($post->id);
```

นอกจากนี้ยังมีรายละเอียดที่น่าสนใจเกี่ยวกับ `Laravel Relationship` อีกมาก ศึกษาได้ที่ <https://laravel.com/docs/8.x/eloquent-relationships>

Programming Exercise

Programming Exercise 11.1 โรงภาพยนตร์

ปรับ migration ให้เพิ่มฟิลด์ที่เป็น `foreign key` แล้ว migrate ใหม่

ปรับ `model` ที่มีความสัมพันธ์กันให้เหมาะสม แล้วปรับ `seeder` ให้ใช้ความสัมพันธ์นั้น จากนั้นทดสอบการ `seed` ข้อมูล

ปรับโค้ดใน `resource controller` ให้เหมาะสม

Programming Exercise 11.2 บัตรสะสมแต้ม

ปรับ migration ให้เพิ่มฟิลด์ที่เป็น `foreign key` แล้ว migrate ใหม่

ปรับ `model` ที่มีความสัมพันธ์กันให้เหมาะสม แล้วปรับ `seeder` ให้ใช้ความสัมพันธ์นั้น จากนั้นทดสอบการ `seed` ข้อมูล

ปรับโค้ดใน resource controller ให้เหมาะสม

12. Laravel Authentication

12.1	การใช้งาน Laravel Breeze	75
12.2	การใช้งาน Auth	76

*Authentication คือการยืนยันตัวตนของผู้เข้าใช้ web application ว่าเป็นใคร ซึ่ง Laravel มี starter kits ให้ 2 รูปแบบ คือ **Laravel Breeze** และ **Laravel Jetstream** ในหนังสือเล่มนี้จะกล่าวถึง **Laravel Breeze** เท่านั้น*

12.1 การใช้งาน **Laravel Breeze**

การติดตั้ง **Laravel Breeze** ใช้คำสั่ง **composer** ดังนี้

```
composer require laravel/breeze --dev
```

เมื่อใช้ **composer** ติดตั้ง **Laravel Breeze** แล้ว ให้ใช้คำสั่ง **artisan breeze:install** ซึ่งจะได้โค้ดติดตั้งสำหรับจัดการเกี่ยวกับ authentication

```
php artisan breeze:install  
  
npm install  
npm run dev  
php artisan migrate
```

เมื่อติดตั้งเสร็จจะได้โค้ดติดตั้งดังนี้

- **app/Http/Controllers/Auth/** เป็น controller ที่จัดการเกี่ยวกับ authentication
- **app/Http/Requests/Auth/** เป็น validation ที่จัดการเกี่ยวกับ authentication
- **resources/views/auth/** เป็น blade views ที่จัดการเกี่ยวกับการแสดงหน้าเว็บที่เกี่ยวข้องกับ authentication

- routes/auth.php เป็น route สำหรับ authentication ซึ่งถูกเพิ่มเข้าไปใน routes/web.php ผ่านคำสั่ง `require __DIR__ . '/auth.php'` ;



บางไฟล์อาจมีการเปลี่ยนแปลงเนื่องจากคำสั่ง `breeze:install` เช่น ไฟล์ `routes/web.php` หรือ `tailwind.config.js` แต่หากใช้การจัดการเวอร์ชันของไฟล์ผ่าน `git` จะสามารถย้อนกลับไปดูโค้ดเก่า และนำมาปรับปรุงกับโค้ดใหม่ได้

หน้าแรกของเว็บจะมีเมนู **Log in** และ **Register** โดยเมนู **Log in** จะไปที่หน้า `/login` และเมนู **Register** จะไปที่หน้า `/register` ซึ่งสามารถดู route ที่เกี่ยวข้องได้จากคำสั่ง `php artisan route:list`



หาก **web application** ไม่มีระบบการลงทะเบียน และต้องการปิดระบบเกี่ยวกับการลงทะเบียน ให้ลบหรือคอมเมนต์โค้ดที่เกี่ยวกับการลงทะเบียนในไฟล์ `routes/auth.php` จากนั้น ในไฟล์ `welcome.blade.php` จะมีโค้ดที่จัดการการแสดงผลเมนูเมื่อมี route จากคำสั่ง `@if (Route::has('register'))`

Controller ที่จัดการระบบ authentication ทั้งหมดอยู่ใน directory `app/Http/Controllers/Auth/` โดย `RegisteredUserController` จัดการเรื่องการลงทะเบียนผู้ใช้ `AuthenticatedSessionController` จัดการเรื่องการเข้าสู่ระบบและออกจากระบบ เป็นต้น แต่ละ controller จะมี method ที่จำเป็น ซึ่งสามารถปรับให้เหมาะสมกับ web application และมี view ที่เกี่ยวข้องอยู่ที่ directory `resources/views/auth/`

12.2 การใช้งาน Auth

Laravel มี Auth Facade ที่นำไปใช้ได้ทั้งใน controller หรือ blade view เพื่อจัดการเกี่ยวกับการยืนยันตัวตนของผู้ใช้

การใช้ Auth Facade ใน controller ให้ import ด้วยคำสั่ง

```
use Illuminate\Support\Facades\Auth;
```

การตรวจสอบว่าผู้ใช้ log in แล้วหรือไม่ ใช้คำสั่ง `Auth::check()` เช่น

```
if (Auth::check()) {
    // The user is logged in...
}
```

สามารถดูข้อมูลของผู้ใช้ที่ log in ได้จาก `Auth::user()` หรือต้องการดู id ของผู้ใช้ที่ log in ได้จาก `Auth::id()` เช่น

```
@if (Auth::check())
    Welcome, {{ Auth::user()->name }}
@endif
```

Parameter Request ใน method ของ controller สามารถดูข้อมูลของผู้ใช้ที่ log in ได้จาก `$request->user()` เช่น method `store()` ของ `PostController` ที่ต้องการเก็บค่าผู้ใช้ที่กำลัง log in ให้เป็นผู้สร้างโพสต์

```

37     public function store(Request $request)
38     {
39         $post = new Post();
40         $post->title = $request->title;
41         $post->detail = $request->detail;
42         $post->views = 0;
43         $request->user()->posts()->save($post);
44         return redirect()->route('posts.show', ['post' => $post->id]);
45     }

```

สามารถกำหนดให้ route ที่ต้องการให้ผู้ใช้ log in ก่อน ก็จะเข้าไปที่หน้านั้นได้ ผ่าน middleware `auth` เช่น

```

Route::get('/posts/create', [PostController::class, 'create'])->
    middleware('auth');
Route::post('/posts', [PostController::class, 'store'])->middleware('
    auth');

```

หรือกำหนดใน constructor ของ controller เพื่อระบุให้ใช้ middleware เพียงบาง method เช่น

```

8 class PostController extends Controller
9 {
10     public function __construct()
11     {
12         $this->middleware('auth')->only([
13             'create', 'store', 'edit', 'update', 'destroy'
14         ]);
15     }
16 }

```

ศึกษาเกี่ยวกับ authentication ของ Laravel เพิ่มเติมได้ที่ <https://laravel.com/docs/8.x/authentication>

Programming Exercise

Programming Exercise 12.1 โรงภาพยนตร์

ทำระบบ authentication สำหรับโรงภาพยนตร์ โดยไม่มีระบบลงทะเบียน (สร้างข้อมูลผู้ใช้ผ่าน tinker)

ผู้ใช้ที่ log in เข้ามา ถือว่าเป็นผู้จัดการข้อมูล cinemas, seat_types และ seats

Programming Exercise 12.2 บัตรสะสมแต้ม

ทำระบบ authentication สำหรับบัตรสะสมแต้ม โดยผู้ใช้ที่ลงทะเบียนจะมีสิทธิ์เป็นสมาชิกทั่วไป เมื่อ log in แล้วให้แสดงข้อมูลบัตรสมาชิกของตนเองพร้อมแต้ม และระบบไม่มีส่วนลงทะเบียนให้ผู้ดูแลระบบ หรือเจ้าหน้าที่ (สร้างผู้ดูแลระบบและเจ้าหน้าที่จาก tinker)
เจ้าหน้าที่จะเห็นข้อมูลบัตรสมาชิกทั้งหมด และมีสิทธิ์ในการเพิ่มแต้มหรือการใช้แต้มของบัตร ผู้ดูแลระบบจะเห็นข้อมูลสมาชิกในระบบว่าใครซื้ออะไร email อะไร และมีสิทธิ์อะไร

13. Laravel Authorization

13.1	การกำหนด Gate	79
13.2	การใช้งาน Gate	80
13.3	การกำหนด Policy	81
13.4	การใช้งาน Policy	82

*Authorization คือการตรวจสอบสิทธิ์ของผู้ใช้ และการอนุญาตให้ผู้ใช้มีสิทธิ์ทำอะไรได้บ้าง โดย Laravel ออกแบบการกำหนดสิทธิ์ของผู้ใช้ในการจัดการข้อมูลในตารางผ่าน **gate** และ **policy** โดย **gate** เป็นเหมือน **route** และ **policy** เป็นเหมือน **controller***

13.1 การกำหนด Gate

ให้กำหนด `Gate::define()` ใน `method boot()` ของคลาส `App\Providers\AuthServiceProvider` คล้ายกับการกำหนด `route` เช่น

```
24 public function boot()
25 {
26     $this->registerPolicies();
27
28     Gate::define('update-post', function ($user, $post) {
29         return $user->id === $post->user->id;
30     });
31
32     Gate::define('create-post', function ($user, $post) {
33         return Auth::check();
34     });
35 }
```

Parameter ที่ 1 ของ `Gate::define()` คือชื่อของ **gate** และ parameter ที่ 2 เป็น closure ที่มี parameter เป็น instance ของ **user** ที่ log in และ instance ของ **model**ที่กำลังจะกำหนดสิทธิ์ในการเข้าถึง ตามลำดับ โดยให้ closure นี้ คืนค่า **true** เมื่อ ผู้ใช้มีสิทธิ์ หรือคืนค่า **false** เมื่อผู้ใช้ไม่มีสิทธิ์ เช่น

คินค่า `$user->id === $post->user->id` หมายความว่า gate ชื่อ `update-post` จะอนุญาตให้ผู้ใช้ที่ `log in` อยู่ (`$user`) มี `id` เดียวกับ `id` ของ `user` ที่เกี่ยวข้องกับโพสต์ (`$post`) เท่านั้น

13.2 การใช้งาน Gate

ตัวอย่างการนำ Gate ไปใช้ใน controller เช่น ใช้ใน method `edit()` และ `update()` ซึ่งจะต้องใช้จาก `Illuminate\Support\Facades\Gate` โดยจะมี `Gate::allows()` ที่ระบุชื่อ gate และ instance ของ model ที่กำหนดสิทธิ์ของ gate ใน `AuthServiceProvider` ไว้แล้ว

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Post;
6  use Illuminate\Http\Request;
7  use Illuminate\Support\Facades\Gate;
8
9  class PostController extends Controller
10 {
11     // other functions
12
13     public function edit($id)
14     {
15         $post = Post::findOrFail($id);
16         if (!Gate::allows('update-post', $post)) {
17             abort(403);
18         }
19         return view('posts.edit', ['post' => $post]);
20     }
21
22     public function update(Request $request, $id)
23     {
24         $post = Post::findOrFail($id);
25         if (!Gate::allows('update-post', $post)) {
26             abort(403);
27         }
28         // The user can update post
29     }
30
31 }
```

ตัวอย่างการนำไปใช้ในหน้า `resources/views/posts/show.blade.php` เพื่อแสดงลิงก์สำหรับไปหน้าแก้ไขเมื่อผู้ใช้มีสิทธิ์แก้ไข


```
@if (Gate::allows('update-post', $post))
    <a href="{{ route('posts.edit', ['post' => $post->id]) }}"
        class="inline-block bg-blue-200 py-2 px-4 hover:bg-blue-300"
    >
        Edit Post
    </a>
@endif
```

กรณีที่การตรวจสอบสิทธิ์ยังไม่มี instance ของ model ที่เจาะจง เช่น หน้าเว็บที่แสดงฟอร์มการสร้างโพสต์ใหม่ หรือหน้าเว็บที่แสดงรายการโพสต์ทั้งหมด ให้ส่ง parameter ที่ 2 ของ method `Gate::allows()` เป็นชื่อคลาสของ Model เช่น

```
@if (Gate::allows('create-post', \App\Models\Post::class))
    <a href="{{ route('posts.create') }}"
        class="inline-block bg-blue-200 py-2 px-4 hover:bg-blue-300"
    >
        Create New Post
    </a>
@endif
```

13.3 การกำหนด Policy

Policy เปรียบเสมือนนโยบายที่กำหนดให้ผู้ใช้งานมีสิทธิ์หรือไม่มีสิทธิ์จัดการข้อมูลในตาราง เช่น การเพิ่มข้อมูล (create) การเรียกดูข้อมูล (retrieve / view) การแก้ไขข้อมูล (update) การลบข้อมูล (delete)

คำสั่งสำหรับสร้างคลาส Policy คือ `php artisan make:policy <PolicyName> --model=<ModelName>` คลาส Policy ที่สร้างขึ้น จะอยู่ใน directory `app/Policies/` โดยชื่อคลาสของ Policy จะเป็น Upper-CamelCase ลงท้ายด้วยคำว่า Policy เช่น การสร้างคลาส Policy สำหรับ model Post

```
php artisan make:policy PostPolicy --model=Post
```

คลาส Policy ที่สร้างขึ้น จะมี method กำหนดมาให้แล้ว 7 method ที่สอดคล้องกับ method ของ resource controller ดังนี้

Policy Methods	Controller Methods
<code>viewAny(User \$user)</code>	<code>index()</code>
<code>view(User \$user, Post \$post)</code>	<code>show(\$id)</code>
<code>create(User \$user)</code>	<code>create()</code> <code>store(Request \$request)</code>
<code>update(User \$user, Post \$post)</code>	<code>edit(\$id)</code> <code>update(Request \$request, \$id)</code>
<code>delete(User \$user, Post \$post)</code>	<code>destroy(\$id)</code>
<code>restore(User \$user, Post \$post)</code>	
<code>forceDelete(User \$user, Post \$post)</code>	

แต่ละ policy methods จะมี parameter แรก เป็น instance ของ User ซึ่งหมายถึงผู้ใช้ที่ log in แล้ว และหากมี parameter ที่ 2 จะหมายถึง instance ของ modelที่กำลังจะกำหนดสิทธิ์ให้ โดยหากผู้ใช้มีสิทธิ์ให้คืนค่า **true** แต่หากผู้ใช้ไม่มีสิทธิ์ให้คืนค่า **false** เช่น

```
public function viewAny(User $user)
{
    return true;
}
```

```
public function create(User $user)
{
    return in_array($user->role, ['CREATOR', 'EDITOR', 'ADMIN']);
}
```

```
public function update(User $user, Post $post)
{
    return $user->role === 'ADMIN' or
           $user->id === $post->user->id;
}
```

ก่อนนำคลาส Policy ไปใช้ จะต้องนำไปลงทะเบียนใน App\Providers\AuthServiceProvider ก่อน โดยระบุชื่อ model และชื่อ policy ใน properties \$policies เช่น

```
16 protected $policies = [
17     \App\Models\Post::class => \App\Policies\PostPolicy::class,
18     \App\Models\Tag::class => \App\Policies\TagPolicy::class,
19 ];
```

13.4 การใช้งาน Policy

ใน controller จะมี method authorize() ให้ใช้งาน โดย parameter ที่ 1 ระบุชื่อ policy method และ parameter ที่ 2 ระบุ instance ของ model หรือชื่อของ model เช่น

```
1 public function create()
2 {
3     $this->authorize('create', Post::class);
4     return view('posts.create');
5 }
```

```
1 public function edit($id)
2 {
3     $post = Post::findOrFail($id);
4     $this->authorize('update', $post);
5     return view('posts.edit', ['post' => $post]);
6 }
```

ใน blade view จะมี directive `@can()` ให้ใช้งาน ซึ่งต้องระบุ parameter 2 ตัวเช่นกัน เช่น

```
1 @can('update', $post)
2     <!-- The current user can update the post... -->
3 @elsecan('create', App\Models\Post::class)
4     <!-- The current user can create new posts... -->
5 @else
6     <!-- ... -->
7 @endcan
```

ผู้ใช้ที่ไม่ได้ `log in` จะถือว่าไม่มีสิทธิ์ตาม *policy* เสมอ หมายความว่า *method* ใน *policy* ที่คืนค่า `true` จะอนุญาตให้ผู้ใช้ที่ `log in` แล้วเท่านั้น



Programming Exercise

Programming Exercise 13.1 บัตรสะสมแต้ม

ใช้ *policy* ในการกำหนดสิทธิ์การเข้าใช้ระบบบัตรสะสมแต้ม

14. Laravel RESTful Service

14.1	การสร้าง API Controller	84
14.2	API Resource	85
14.3	JSON Web Token	87
14.4	Laravel JWT Authentication	90

การเขียน *RESTful Service* ใน *Laravel* จะแยกออกจากการเขียน *web application* โดยกำหนด *route* ในไฟล์ *routes/api.php* และ *URL request path* สำหรับ *RESTful API* จะขึ้นต้นด้วย */api*

ความแตกต่างระหว่าง *routes/api.php* ที่ใช้สร้าง *RESTful Service* กับ *routes/web.php* ที่ใช้เขียน *web application* คือ *routes/api.php* จะใช้ *middleware group api* และ *routes/web.php* จะใช้ *middleware group web*

Middleware เป็นส่วนที่ทำงานก่อน *route* จะเข้าสู่ *controller* ในตอนที่ได้รับ *request* มาจาก *client* โดยลำดับการทำงานของ *middleware* ให้อ่านในคลาส *App\Http\Kernel* ซึ่งเป็นคลาสที่กำหนด *middleware* ทั้งหมดของระบบ

14.1 การสร้าง API Controller

เพื่อแยก *RESTful Service* ออกจาก *web application* การสร้าง *controller* ควรสร้างแยก *namespace* เช่น *App\Http\Controllers\TagController* สำหรับ *web application* และ *App\Http\Controllers\Api\TagController* สำหรับ *RESTful Service* และใส่ *option --api* ในคำสั่ง *make:controller* เช่น

```
php artisan make:controller Api/TagController --api
```

API controller ที่สร้างขึ้นจะมี 5 methods เท่านั้น คือ *index*, *show*, *store*, *update* และ *destroy* โดยจะไม่มี *method create* และ *edit* เนื่องจาก *RESTful Service* ไม่ต้องแสดงหน้าฟอร์มสำหรับการกรอกข้อมูล

การกำหนด *route* สำหรับ *API controller* ทั้ง 5 methods ในไฟล์ *routes/api.php* ให้ใช้คำสั่ง *Route::apiResource()* เช่น

```
use App\Http\Controllers\Api\TagController;
Route::apiResource('tags', TagController::class);
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	api/tags	tags.index	App\Http\Controllers\Api\TagController@index	api
	POST	api/tags	tags.store	App\Http\Controllers\Api\TagController@store	api
	GET HEAD	api/tags/{tag}	tags.show	App\Http\Controllers\Api\TagController@show	api
	PUT PATCH	api/tags/{tag}	tags.update	App\Http\Controllers\Api\TagController@update	api
	DELETE	api/tags/{tag}	tags.destroy	App\Http\Controllers\Api\TagController@destroy	api

ตัวอย่างการเขียน method index ใน Api\TagController

```
public function index()
{
    $tags = Tag::get();
    return $tags;
}
```

จากนั้นทดสอบผ่านโปรแกรม Postman ด้วย GET <http://127.0.0.1:8000/api/tags>

สำหรับการทดสอบ HTTP method POST, PUT หรือ DELETE ผ่าน Postman เพื่อส่ง request ไปยัง RESTful Service นั้น ไม่ต้องส่ง CSRF token เพราะ middleware api ไม่มีการตรวจสอบ CSRFVerify



14.2 API Resource

API Resource เป็นตัวกลางที่เป็น transformation layer ระหว่างข้อมูลจาก eloquent model และ JSON response เพื่อจำกัดรูปแบบการส่งข้อมูลที่เท่าที่จำเป็นในการ response ของ RESTful Service

การสร้างคลาส Resource จะใช้คำสั่ง `php artisan make:resource <ResourceName>` โดยชื่อคลาสจะเหมือนชื่อ model และคลาส Resource ที่สร้างขึ้นจะอยู่ใน directory `app/Http/Resources/` เช่น

```
php artisan make:resource Tag
php artisan make:resource Post
```

การจัดรูปแบบข้อมูลที่จะให้ response ในคลาส Resource ให้ override method `toArray($request)` โดยคืนค่า array ของการจัดรูปแบบข้อมูลที่ต้องการ response เช่น

```
1 <?php
2
3 namespace App\Http\Resources;
4
5 use Illuminate\Http\Resources\Json\JsonResource;
6 use App\Http\Resources\Post as PostResource;
7
```

```

8 class Tag extends JsonResource
9 {
10     /**
11      * Transform the resource into an array.
12      *
13      * @param \Illuminate\Http\Request $request
14      * @return array
15      */
16     public function toArray($request)
17     {
18         return [
19             'id' => $this->id,
20             'name' => $this->name,
21             'created_at' => $this->created_at,
22             'posts' => PostResource::collection(
23                 $this->whenLoaded('posts')
24             ),
25         ];
26     }
27 }

```

`whenLoaded()` ใช้ในกรณีที่ต้องการหลีกเลี่ยงการผูกความสัมพันธ์ที่ไม่จำเป็น โดย `method` นี้จะรับชื่อของความสัมพันธ์ที่กำหนดใน `model` (เช่น `Tag` มี `method posts()` ดังนั้น `Tag` จะมีความสัมพันธ์ชื่อ `posts`)

นำคลาส `Resource` ไปใช้ใน `Api controller` โดยเมื่อต้องการ `response` เป็น `instance` เดียว ให้คืนค่า `instance` ของ `Resource` เช่น

```

1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Tag;
7 use Illuminate\Http\Request;
8 use App\Http\Resources\Tag as TagResource;
9
10 class TagController extends Controller
11 {
12     /**
13      * Display the specified resource.
14      *
15      * @param int $id
16      * @return \Illuminate\Http\Response
17      */

```

```

18     public function show($id)
19     {
20         $tag = Tag::with('posts')->findOrFail($id);
21         return new TagResource($tag);
22     }
23 }

```

เมื่อต้องการ response เป็น collection ให้คืนค่าผ่าน method `collection()` เช่น

```

1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use App\Models\Tag;
7  use Illuminate\Http\Request;
8  use App\Http\Resources\Tag as TagResource;
9
10 class TagController extends Controller
11 {
12     /**
13      * Display the specified resource.
14      *
15      * @param int $id
16      * @return \Illuminate\Http\Response
17      */
18     public function index()
19     {
20         $tags = Tag::get();
21         return TagResource::collection($tags);
22     }
23 }

```

14.3 JSON Web Token

JSON Web Token (JWT) เป็นมาตรฐานเปิด (RFC 7519) ที่ใช้แก้ปัญหาการส่งข้อมูลอย่างปลอดภัยระหว่างการเชื่อมต่อ API โดยที่ถูกออกแบบให้ มีขนาดที่กระทัดรัด และเก็บข้อมูลภายในตัว (<https://jwt.io/>)

JWT แบ่งโครงสร้างออกเป็น 3 ส่วน คือ header, payload และ verify signature

Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC8xMjcucMC4wLjE6ODAwMFwvYXBpXC9hdXRoXC9sb2dpbiIsImIhdCI6MTYyNDE3NTExOCwiZXhwIjoxNjI0MTc4NzE4LCJuYmYiOiJlMjMjQXNzUxMTgsImp0aSI6IjZkaXAzVmhfZFBkVrd0UiLCJzdiI6IjIsInBydiI6IjIzYmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjciOiI0I9RDdGa4pvwPqIgjXh1h6uxSlP7jE-7WUkwh2YAsU
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "iss": "http://127.0.0.1:8080/api/auth/login",
  "iat": 1624175118,
  "exp": 1624178718,
  "nbf": 1624175118,
  "jti": "6dip3VhDdWA6EkWE",
  "sub": 2,
  "prv": "23bd5c8949f600adb39e701c408872db7a5976f7"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

- Header ใช้กำหนดว่า JWT ชุดนี้ถูกเข้ารหัสแบบใด (เช่น SHA256, RSA)
- Payload ใช้เก็บข้อมูล
- Verify Signature ใช้ตรวจสอบความถูกต้องของข้อมูล ถ้าข้อมูลใน payload มีการเปลี่ยนแปลง signature จะไม่ตรงกัน ทำให้ token นี้เชื่อถือไม่ได้



เว็บไซต์ <https://jwt.io/> มี *debugger* ใช้ถอดรหัส JWT ได้ ซึ่งสังเกตว่า JWT ไม่ได้ถูกออกแบบมาให้มีความปลอดภัยในการเก็บข้อมูล แต่ใช้ *verify signature* ในการตรวจสอบว่าข้อมูลถูกต้องและน่าเชื่อถือ (ไม่ถูกเปลี่ยนข้อมูล) หรือไม่ ดังนั้น ไม่ควรเก็บข้อมูลที่เป็นความลับ เช่น รหัสผ่าน เลขบัตรประชาชน เลขบัตรเครดิต หรือกล่าวว่า JWT ใช้เก็บข้อมูลน้อยที่สุดที่สามารถยืนยันตัวตนในการทำ *authorization* และ *authentication*

การใช้งาน JWT ใน Laravel เริ่มจากการติดตั้งผ่าน *composer* ด้วยคำสั่ง

```
composer require tymon/jwt-auth
```

จากนั้นใช้คำสั่งเพื่อสร้าง *configuration file* ซึ่งจะได้ไฟล์ *config/jwt.php*

```
php artisan vendor:publish --provider="Tymon\JWTAuth\Providers\
  LaravelServiceProvider"
```

และให้เพิ่ม *key* `JWT_SECRET` ในไฟล์ *.env* เพื่อกำหนด *secret* สำหรับการสร้าง *verify signature* ซึ่งอาจจะใช้คำสั่งต่อไปนี้ช่วยสร้าง

```
php artisan jwt:secret
```


ขั้นตอนต่อไป จะต้องแก้ไข model User ให้ implements `Tymon\JWTAuth\Contracts\JWTSubject` ซึ่งจะต้อง override 2 methods คือ `getJWTIdentifier()` และ `getJWTCustomClaims()`

```

1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Foundation\Auth\User as Authenticatable;
6  use Tymon\JWTAuth\Contracts\JWTSubject;
7
8  class User extends Authenticatable implements JWTSubject
9  {
10     // other properties and methods
11
12     public function getJWTIdentifier()
13     {
14         return $this->getKey();
15     }
16
17     public function getJWTCustomClaims()
18     {
19         return [
20             'name' => $this->name
21         ];
22     }
23 }
```

สามารถเพิ่มข้อมูลของ User ไปเก็บใน payload ของ JWT ได้ ผ่านการคืนค่าที่ method `getJWTCustomClaims()`

ในไฟล์ `config/auth.php` ให้เปลี่ยนการตั้งค่าเพื่อให้ระบบใช้งาน jwt ในการจัดการข้อมูลผู้ใช้ที่เชื่อมต่อผ่าน API โดยแก้ไข `'guards' 'api'` ให้ใช้ `'driver'` เป็น `'jwt'`

```

38     'guards' => [
39         // ...
40         'api' => [
41             'driver' => 'jwt',
42             'provider' => 'users',
43             'hash' => false,
44         ],
45     ],
```

จากขั้นตอนนี้ จะทำให้ระบบ authentication ของ Laravel ใช้งาน jwt-auth ได้แล้ว

14.4 Laravel JWT Authentication

สร้างคลาส `Api\AuthController` ด้วยคำสั่ง

```
php artisan make:controller Api/AuthController
```

ตัวอย่างโค้ดในคลาส `App\Http\Controllers\Api\AuthController`

```
1  <?php
2
3  namespace App\Http\Controllers\Api;
4
5  use App\Http\Controllers\Controller;
6  use Illuminate\Http\Request;
7  use Illuminate\Support\Facades\Validator;
8  use Tymon\JWTAuth\Facades\JWTAuth;
9  use App\Models\User;
10 use Tymon\JWTAuth\Exceptions\JWTException;
11
12 class AuthController extends Controller
13 {
14     /**
15      * Create a new AuthController instance.
16      *
17      * @return void
18      */
19     public function __construct()
20     {
21         $this->middleware('auth:api', [
22             'except' => ['login', 'register']
23         ]);
24     }
25
26     /**
27      * Get a JWT via given credentials.
28      *
29      * @return \Illuminate\Http\JsonResponse
30      */
31     public function login(Request $request) {
32         $validator = Validator::make($request->all(), [
33             'email' => 'required|email',
34             'password' => 'required|string|min:6',
35         ]);
36
37         if ($validator->fails()) {
38             return response()->json($validator->errors(), 422);
```

```
39     }
40
41     if (! $token = JWTAuth::attempt($validator->validated())) {
42         return response()->json(['error' => 'Unauthorized'], 401);
43     }
44
45     return $this->respondWithToken($token);
46 }
47
48 /**
49  * Register a User.
50  *
51  * @return \Illuminate\Http\JsonResponse
52  */
53 public function register(Request $request) {
54     $validator = Validator::make($request->all(), [
55         'name' => 'required|string|between:2,100',
56         'email' => 'required|string|email|max:100|unique:users',
57         'password' => 'required|string|confirmed|min:8',
58     ]);
59
60     if($validator->fails()){
61         return response()->json($validator->errors()->toJson(),
62 400);
63     }
64
65     $user = User::create(array_merge(
66         $validator->validated(),
67         ['password' => bcrypt($request->password)]
68     ));
69
70     return response()->json([
71         'message' => 'User successfully registered',
72         'user' => $user
73     ], 201);
74
75 /**
76  * Get the authenticated User.
77  *
78  * @return \Illuminate\Http\JsonResponse
79  */
80 public function me(Request $request)
81 {
82     $user = JWTAuth::user();
```

```

83         return response()->json($user);
84     }
85
86     /**
87      * Log the user out (Invalidate the token).
88      *
89      * @return \Illuminate\Http\JsonResponse
90      */
91     public function logout()
92     {
93         auth()->logout();
94         return response()->json(['message' => 'Successfully logged out
95     ']);
96     }
97
98     /**
99      * Refresh a token.
100     *
101     * @return \Illuminate\Http\JsonResponse
102     */
103     public function refresh(Request $request)
104     {
105         return $this->respondWithToken(auth()->refresh());
106     }
107
108     /**
109      * Get the token array structure.
110      *
111      * @param string $token
112      *
113      * @return \Illuminate\Http\JsonResponse
114      */
115     protected function respondWithToken($token)
116     {
117         return response()->json([
118             'access_token' => $token,
119             'token_type' => 'bearer',
120             'expires_in' => config('jwt.ttl') * 60,
121             'user' => auth()->user()
122         ]);
123     }

```

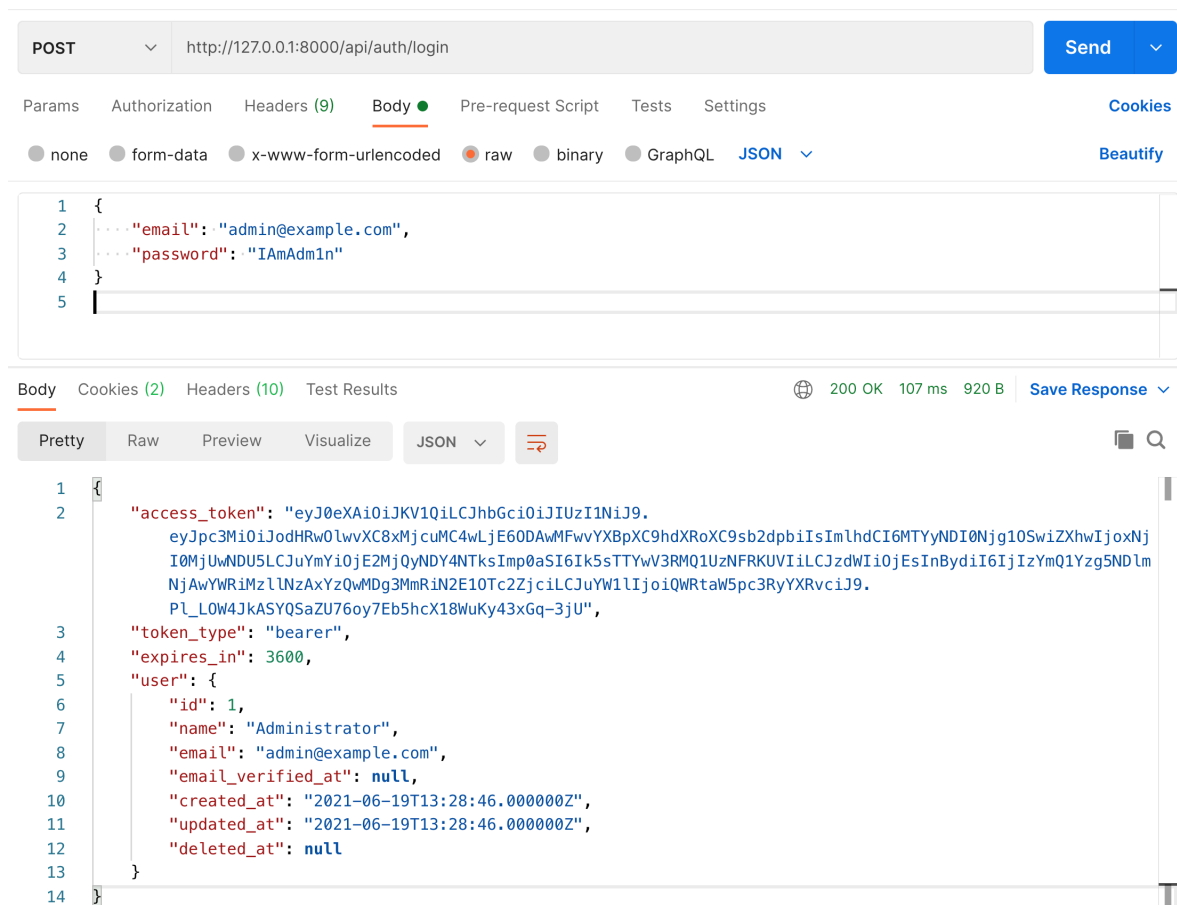
กำหนด route ที่ไฟล์ routes/api.php ดังนี้

```
use Illuminate\Support\Facades\Route;
```

```
use App\Http\Controllers\Api\AuthController;

Route::group([
    'middleware' => 'api',
    'prefix' => 'auth'
], function ($router) {
    Route::post('login', [AuthController::class, 'login']);
    Route::post('logout', [AuthController::class, 'logout']);
    Route::post('refresh', [AuthController::class, 'refresh']);
    Route::post('me', [AuthController::class, 'me']);
});
```

ใช้ Postman ทดสอบการ POST ไปที่ endpoint `http://127.0.0.1:8000/api/auth/login` โดยกำหนด request body ให้มีค่าของ email และ password ของผู้ใช้ในระบบ



JWT token คือค่า `access_token` ที่ `response` กลับไป สามารถนำไปตรวจสอบกับ `debugger` ของเว็บ `https://jwt.io` โดยระบุค่า `JWT_SECRET` จากไฟล์ `.env` ในช่อง `your-256-bit-secret` (อยู่ในส่วน `VERIFY SIGNATURE`) ก่อน แล้วจึงนำค่า `access_token` ไปใส่ที่ช่อง `Encoded` หาก JWT token ถูกต้องและเชื่อถือได้ จะมีข้อความแจ้งว่า `Signature Verified`

Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC8xMjc4wLjE6ODAwMjFwYXBpXC9hdXRoXC9sb2dpbiIsImh0dCI6MTYyNDI0Njg1OSwiZXhwIjoxNjUwNDU5LCJuYmYiOiJlMjMjQmYyNDY4NTksImp0aSI6IjE5sTYyV3RMQ1UzNFRKUVIiLCJzdzIiOiJEsInBydiI6IjIzYmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjciLCJuYmY1IjoiQWRtaW5pc3RyYXRvcjI9.P1_LOW4JkASYQSaZu76oy7Eb5hcX18WuKy43xGq-3jU
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "iss": "http://127.0.0.1:8000/api/auth/login",
  "iat": 1624246859,
  "exp": 1624250459,
  "nbf": 1624246859,
  "jti": "N1M6WtLCU34TJQR",
  "sub": 1,
  "prv": "23bd5c8949f600adb39e701c400872db7a5976f7",
  "name": "Administrator"
}
```

VERIFY SIGNATURE

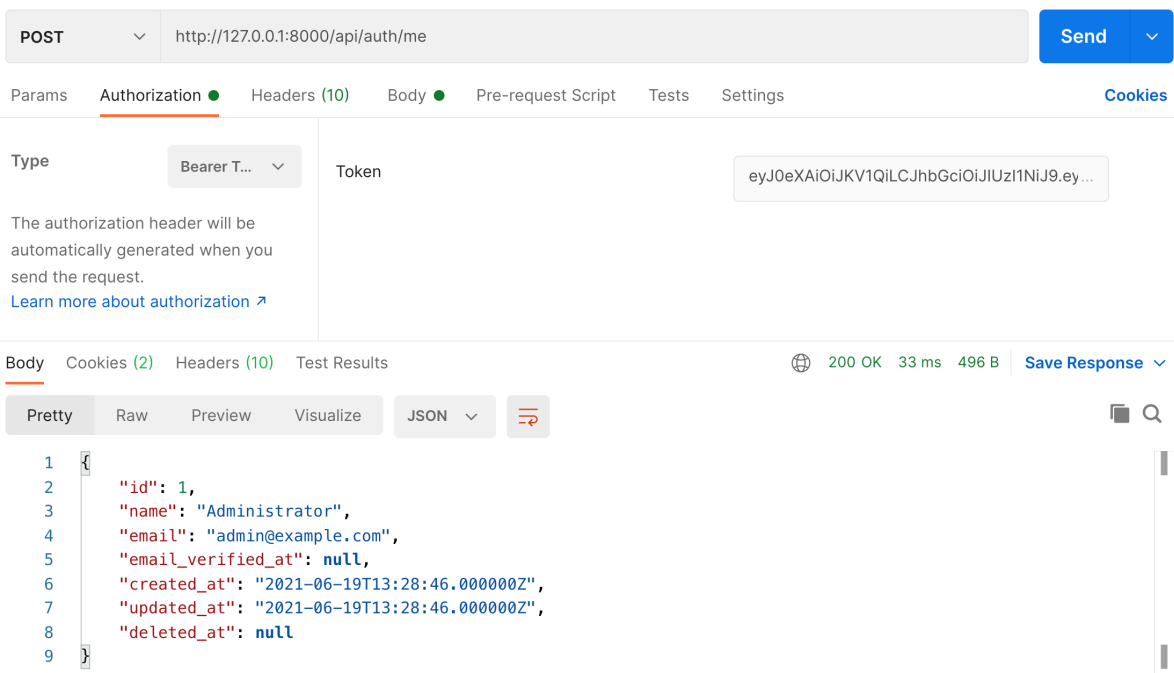
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  E30ZCrYK6Dy9KRv5ry8z
)
```

☐ secret base64 encoded

 Signature Verified

SHARE JWT

ใช้ Postman ทดสอบการ POST ไปที่ endpoint `http://127.0.0.1:8000/api/auth/me` โดยกำหนด request header ผ่านแท็บ Authorization กำหนด Type เป็น Bearer Token และกำหนด Token เป็นค่า `access_token` ที่ได้จาก `/api/auth/login` จะได้ข้อมูลของผู้ใช้ที่ log in



POST `http://127.0.0.1:8000/api/auth/me` Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Type Bearer T... Token `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC8xMjc4wLjE6ODAwMjFwYXBpXC9hdXRoXC9sb2dpbiIsImh0dCI6MTYyNDI0Njg1OSwiZXhwIjoxNjUwNDU5LCJuYmYiOiJlMjMjQmYyNDY4NTksImp0aSI6IjE5sTYyV3RMQ1UzNFRKUVIiLCJzdzIiOiJEsInBydiI6IjIzYmQ1Yzg5NDlmNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjciLCJuYmY1IjoiQWRtaW5pc3RyYXRvcjI9.P1_LOW4JkASYQSaZu76oy7Eb5hcX18WuKy43xGq-3jU`

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies (2) Headers (10) Test Results 200 OK 33 ms 496 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "Administrator",
4   "email": "admin@example.com",
5   "email_verified_at": null,
6   "created_at": "2021-06-19T13:28:46.000000Z",
7   "updated_at": "2021-06-19T13:28:46.000000Z",
8   "deleted_at": null
9 }
```

สร้างคลาส `Api\PostController` ด้วยคำสั่ง

```
php artisan make:controller Api/PostController --api
```

ตัวอย่างโค้ดในคลาส `App\Http\Controllers\Api\PostController`

```
1 <?php
2
3 namespace App\Http\Controllers\Api;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\Post;
7 use App\Models\Tag;
8 use App\Http\Resources\Post as PostResource;
9 use Illuminate\Http\Request;
10 use Illuminate\Support\Facades\Validator;
11 use Tymon\JWTAuth\Facades\JWTAuth;
12
13 class PostController extends Controller
14 {
15     public function __construct()
16     {
17         $this->middleware('auth:api');
18     }
19
20     // other methods
21 }
```

Method `index()`

```
public function index()
{
    $posts = Post::paginate(30);
    return PostResource::collection($posts);
}
```

Method `store()`

```
public function store(Request $request)
{
    $authUser = JWTAuth::user();
    $validator = Validator::make($request->all(), [
        'title' => 'required|max:255',
        'detail' => 'required|max:500',
    ]);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 422);
    }
}
```

```

$post = new Post();
$post->title = $request->title;
$post->detail = $request->detail;
$post->user_id = $authUser->id;
$post->save();
if ($request->has('tags')) {
    $tags = $request->tags;
    $tags = explode(' ', $tags);
    foreach ($tags as $tag_name) {
        $tag = Tag::firstOrCreate([
            'name' => trim($tag_name)
        ]);
        $post->tags()->attach($tag->id);
    }
}
return new PostResource($post);
}

```

Method show()

```

public function show($id)
{
    $post = Post::with(['tags', 'user'])->findOrFail($id);
    return new PostResource($post);
}

```

Method update()

```

public function update(Request $request, $id)
{
    $authUser = JWTAuth::user();
    $post = Post::findOrFail($id);
    if ($authUser->id !== $post->user->id) {
        abort(403);
    }
    if ($request->has('title'))
        $post->title = $request->title;
    if ($request->has('detail'))
        $post->detail = $request->detail;
    $post->save();
    if ($request->has('tags')) {
        $tags = $request->tags;
        $tags = explode(' ', $tags);
        $tag_ids = [];
        foreach ($tags as $tag_name) {
            $tag = Tag::firstOrCreate([
                'name' => trim($tag_name)
            ]);
            $tag_ids[] = $tag->id;
        }
        $post->tags()->sync($tag_ids);
    }
}

```



```

    });
    array_push($tag_ids, $tag->id);
}
$post->tags()->sync($tag_ids);
}
return new PostResource($post);
}

```

Method delete()

```

public function destroy($id)
{
    $authUser = JWTAuth::user();
    $post = Post::findOrFail($id);
    if ($authUser->id !== $post->user->id) {
        abort(403);
    }
    $post->tags()->detach();
    $post->delete();
    return response()->json([
        'success' => true,
        'message' => "Post id {$id} has been deleted"
    ]);
}

```

ตัวอย่างคลาส App\Http\Resources\Post

```

1 <?php
2
3 namespace App\Http\Resources;
4
5 use Illuminate\Http\Resources\Json\JsonResource;
6
7 class Post extends JsonResource
8 {
9     /**
10      * Transform the resource into an array.
11      *
12      * @param  \Illuminate\Http\Request  $request
13      * @return array
14      */
15     public function toArray($request)
16     {
17         return [
18             'id' => $this->id,
19             'title' => $this->title,
20             'detail' => $this->detail,

```

```
21         'views' => $this->views,  
22         'created_at' => $this->created_at,  
23         'user' => [  
24             'id' => $this->user->id,  
25             'name' => $this->user->name  
26         ],  
27         'tags' => $this->tags->pluck('name')  
28     ];  
29 }  
30 }
```



Response จาก API ที่เกิดจาก exception ถ้าต้องการให้คืนค่าเป็น json จะต้องส่ง request header key Accept ที่มี value เป็น application/json

Programming Exercise

Programming Exercise 14.1 โรงภาพยนตร์
ให้สร้าง RESTful Service สำหรับระบบโรงภาพยนตร์

Programming Exercise 14.2 บัตรสะสมแต้ม
ให้สร้าง RESTful Service สำหรับระบบบัตรสะสมแต้ม โดย API จะต้องจัดการสิทธิ์ในการใช้งานอย่างเหมาะสม สอดคล้องกับ role ของผู้ใช้งานด้วย

