PHP Laravel 5.6 Framework

07 - 09.07.2018

Surawut Sodasak
Software Engineer

LARAVEL ตอนที่ 1 รู้จัก PHP Laravel Framework

PHP Laravel Framework มีดียังไง ทำไมถึงได้รับความนิยมอย่างแพร่หลายและถูกใช้ในการพัฒนา Software ระบบใหญ่ๆ ในระดับโลก

ข้อดีทาง Programing

1. Code and File Organization

PHP Framework ที่มีชื่อเสียงทุกตัว ได้รับออกแบบจากนักพัฒนาที่มีประสบการณ์ระดับ Enterprise นั่นหมายความว่าการออกแบบโครงสร้างของโปรแกรม ต้องมีความยืดหยุ่นสูง การจัดวาง folder และ file ต่างๆ นั้นได้ถูกออกแบบและจัดหมวดหมู่เป็นอย่างดี เพื่อให้ทีมพัฒนาซอฟแวร์สามารถเข้าใจโครงสร้างโปรแกรม และต่อเติมได้ง่าย รวมทั้งง่ายต่อการดูแลรักษา

2. Routing

การจัดการเส้นทางของ URL ให้ยืดหยุ่นและมีประสิทธิภาพมากขึ้น สามารถควบคุมความสัมพันธ์ระหว่าง URL ที่ Users ใช้งาน กับโค๊ดของเราได้ง่ายขึ้น อีกทั้งยังสามารถนำ Security เข้ามาควบคุมในส่วนนี้ได้

3. Helper and Library

ติดตั้ง Library ที่จำเป็นและต้องใช้บ่อยๆมาให้พร้อมกับ framework เช่นการ validate ฟอร์ม, การสร้างและแก้ไข session, cookie, ระบบส่ง email, ปฏิทิน, pagination, การจัดการ user / role / permission, และอย่างอื่นอีกมากมาย

4. External Library

สามารถนำ php library ประมาณ 7000 กว่าตัวบน packagist.org มาสร้างเป็น package ใหม่ได้อย่างง่ายดายผ่านระบบ Composer ที่คอยจัดการเรื่อง Dependency ต่างๆ ของ Library ที่ Import เข้ามาใช้ ผ่าน composer.json เช่นต้องการนำ Library A เข้ามาใช้ แต่ Lib A ต้องใช้ Lib B ตัว composer จะดึง Lib A และดึง Lib B หรือ Lib ทุกตัวที่ Lib A ต้องใช้มาด้วยอัตโนมัติ

5. Database Driver

มีทั้งการ Query แบบธรรมดา และใช้งาน ORM ทำให้เลือกใช้ฐานข้อมูลได้หลากหลาย เช่น MySQL, MS SQL, ProgreSQL, SQLite และอื่นๆ สามารถเปลี่ยนฐานข้อมูลได้เลย ผ่าน DB Driver ได้โดยไม่ต้องแก้ไขโค๊ด ทำให้มีความยืดหยุ่นสูงในการเปลี่ยนฐานข้อมูล

6. Security

การจัดการเรื่อง Security เองที่เรื่องที่ยุ่งยาก ซับซ้อน และต้องมีความรู้เป็นอย่างมาก เพราะเราไม่สามารถอุดช่องโหว่จำนวนมาก นับร้อย นับพันที่อาจจะเกิดขึ้น แต่ถ้าใช้ framework การจัดการเรื่อง security จะเป็นไปอย่างง่ายดาย มีข้อผิดพลาดน้อย เพราะ framework ต่างๆ มีเครื่องมือพื้นฐานที่ออกแบบโดยทีมผู้เชี่ยวชาญเรื่อง security โดยเฉพาะเตรียมไว้ให้นักพัฒนาใช้อยู่แล้ว การป้องกัน security พื้นฐานก็เป็นไปแบบอัตโนมัติ เช่น sql injection, xss filter, cookie encryption

7. Less Code & Faster Development

เนื่องจาก PHP Framework ได้เตรียมความพร้อมมาให้ทั้งการวาง Code and File Organization, Library, Security, Routing มาให้เรียบร้อยแล้วทำให้ Developer สามารถเริ่มงานได้เลย และ Focus ไปยังส่วนระบบที่ต้องพัฒนาเท่านั้น ทำให้สามารถพัฒนาระบบได้รวดเร็วและมีคุณภาพได้มากขึ้น

8. Community Support

สิ่งสำคัญสิ่งหนึ่งในการตัดสินใจเลือก Framework สักตัวคือ Community Support เนื่องจากเวลาเราติดปัญหาต่างๆ สามารถค้นหาข้อมูลได้ง่ายดาย ว่าปัญหาที่เราเจอเคยมีคนเจอมาแล้ว และมี Community ช่วยหรือหาวิธีแก้ปัญหาให้แล้ว ทำให้เราไม่ต้องเสียเวลาในการแก้ปัญหาเอง

9. Job Opportunity

บริษัทใหญ่ๆ ส่วนมากใช้ framework กันทั้งสิ้น ดังนั้นถ้าเราสามารถใช้ framework ได้คล่องแคล่ว รู้ลึกรู้จริง โอกาสทางก้าวหน้าในอาชีพการงานก็มีสูง เราสามารถย้ายงาน ไปหาบริษัทที่พร้อมรับโปรแกรมเมอร์มือดี พร้อมให้ค่าตอบแทนสูงได้โดยง่าย

ข้อดีทางการพัฒนาระบบ

1. Suitable for Teamwork

web framework มักจะบังคับให้การออกแบบโปรแกรมเป็นแบบ MVC ดังนั้นเราสามารถแบ่งงานได้ชัดเจนและแยกจากกันอย่างเด็ดขาด ระหว่าง frontend developer / backend developer / database developer หรือแบ่งงานกันเป็น Modules เช่น Login, Register เป็นต้น ทำให้การพัฒนาโปรเจคเป็นไปได้โดยเร็วและพร้อมๆ กันได้

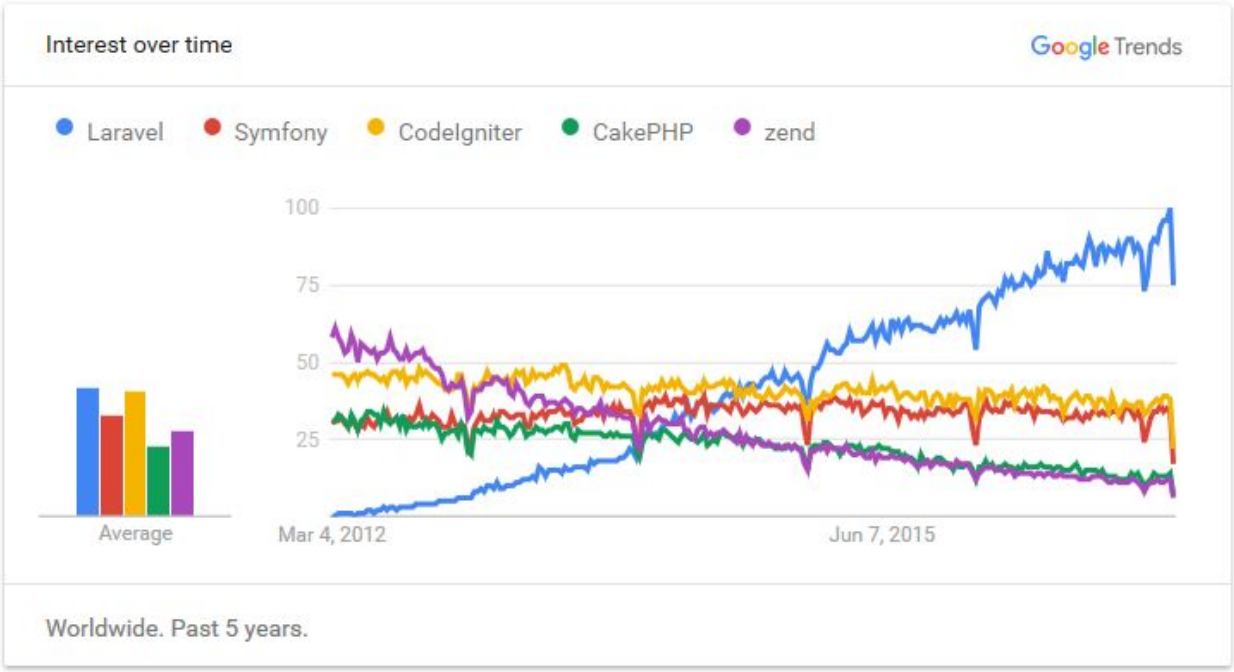
2. Standard Coding

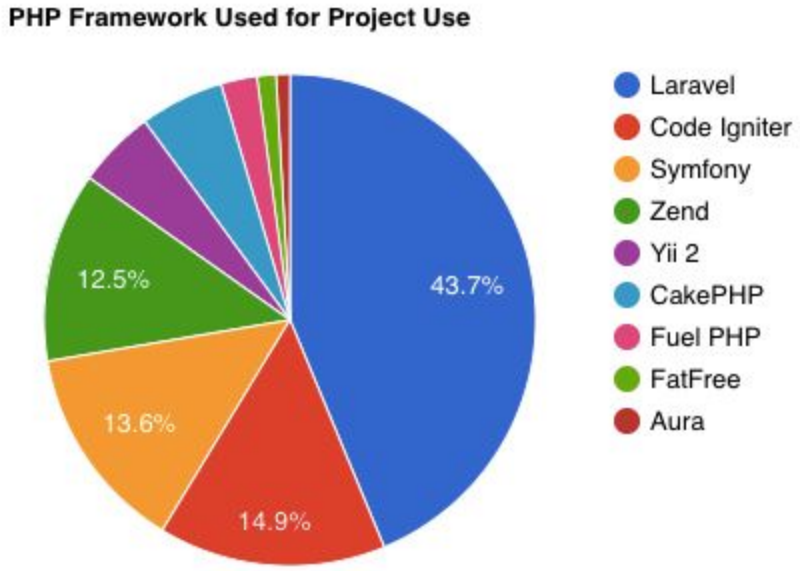
มีมาตรฐานในการเขียนโค๊ดไปในทิศทางเดียวกัน ทำให้เข้าใจโค๊ดของเพื่อนร่วมงาน และ ง่ายในการทำงานร่วมกันหรือพัฒนางานต่อจากเพื่อน

3. Shared Library

เนื่องจาก Framework ได้ถูกออกแบบมาให้ Developer มีการเขียนโปรแกรมร่วมกัน อย่างเป็นระบบและเหมาะสมกับการทำระบบขนาดใหญ่มากขึ้น ทำให้โค๊ดส่วนไหนที่มีความซ้ำซ้อน ได้จับยุบรวมกัน ไปไว้ในส่วน Library กลางของระบบ ที่ทุกคนสามารถใช้ร่วมกันได้

PHP Framework Trends





What is MVC Concept (Model, View, Controller)

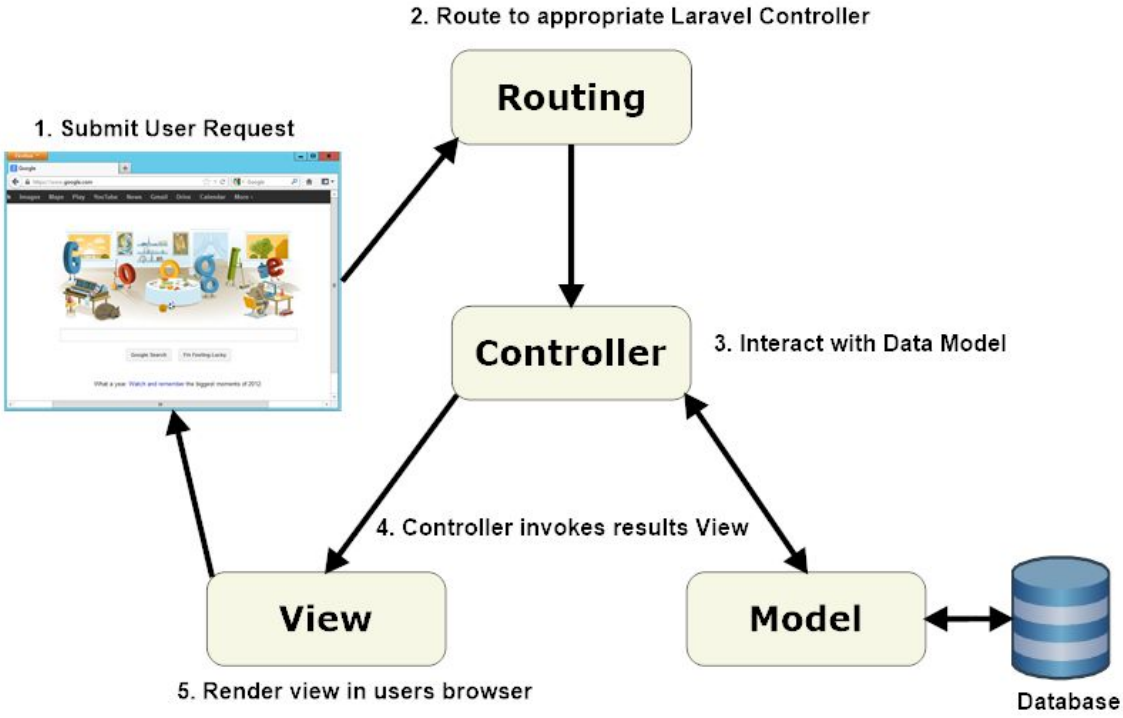
MVC คือ สถาปัตยกรรมการออกแบบซอฟแวร์ (software architecture) ที่ออกแบบให้มีการแยกการทำงาน ออกเป็น 3 ส่วนหลักๆ คนละหน้าที่อย่างชัดเจน และสามารถทำงานเชื่อมต่อกัน เพื่อสามารถรับส่งข้อมูลระหว่างกันได้ โดยมีดังนี้

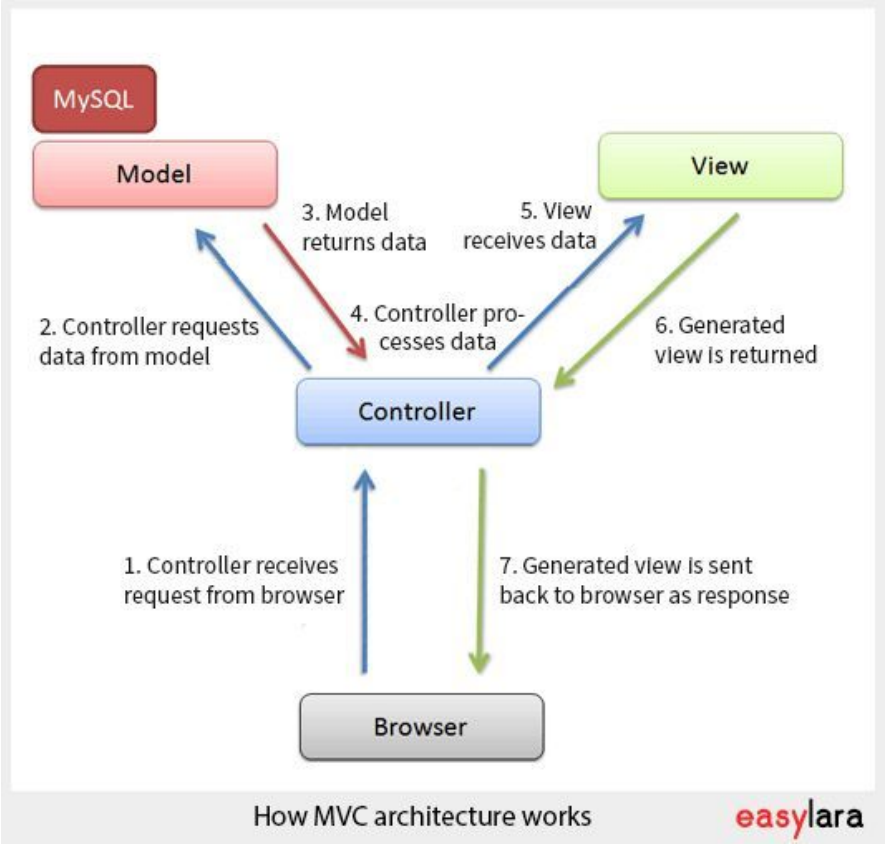
- **Model** ดูแลในส่วนของการทำงานที่ต้องติดต่อ Database จะประกอบด้วย class ที่เชื่อมต่อกับ RDBMS โดย จะมี Eloquent (Laravel) ดูแลในเรื่องของการติดต่อสื่อสารระหว่าง Object และ Database โดยที่ผู้พัฒนาไม่ต้องยุ่งยากกับการใช้ SQL command ทำให้สามารถเปลี่ยน Database ได้โดยที่ไม่ต้องมีการแก้โค้ด

- **View** ดูแลในส่วนที่ต้องแสดงผลผ่าน Browser จัดการเรื่อง Template สำหรับแสดงผล HTML, CSS, JS

- **Controller** เป็นส่วนของการควบคุมการทำงานหลัก และ Logic ของระบบ โดยจะ Interactive กับ Users ผ่าน URl หรือ Links ที่ใช้งาน และเป็นตัวกลางในการเชื่อมต่อระหว่าง View และ Model ในการรับส่งข้อมูลจาก Database มาแสดงผลที่หน้า browser

โดยมีภาพรวมของการทำงานดังนี้


2. Route to appropriate Laravel Controller
Routing
1. Submit User Request
Controller
3. Interact with Data Model
4. Controller invokes results View
View
Model
Database
5. Render view in users browser


MySQL
Model
View
3. Model returns data
5. View receives data
2. Controller requests data from model
4. Controller processes data
6. Generated view is returned
Controller
1. Controller receives request from browser
7. Generated view is sent back to browser as response
Browser
How MVC architecture works
easylara

LARAVEL ตอนที่ 2 ติดตั้ง Web Server และ Composer

Server Requirements

- XAMPP/MAMP/Other Web Server สำหรับ Apache, PHP 7.1.3, MySql
- เปิด Extensions ของ PHP ดังนี้
    - PHP >= 7.1.3
    - OpenSSL PHP Extension
    - PDO PHP Extension
    - Mbstring PHP Extension
    - Tokenizer PHP Extension
    - XML PHP Extension
    - Ctype PHP Extension

Install Composer

Composer เป็นเครื่องมือ ของ PHP ใช้จัดการ library ที่ต้องการใช้ในโปรเจ็ค ลักษณะการใช้งานคือ ให้เราระบุ library ที่โปรเจ็คของเราต้องการไว้ในไฟล์ composer.json จากนั้น composer จะทำการติดตั้งหรืออัพเดท library ที่เราต้องการให้เลย ช่วยให้เราจัดการกับ library ได้ง่ายขึ้น

ประโยชน์หลัก

- ลดเวลาการค้นหาไลบรารี่ที่เราต้องการ เพราะไลบรารี่ของ php ที่ใช้มาตรฐาน psr แทบจะทุกตัวใช้ packagist.org ในการขึ้นทะเบียนว่าสามารถใช้ร่วมกับ composer ได้
- ถ้าไลบรารี่ตัวนั้นมีการอัพเดท เราสามารถติดตามได้โดยไม่ต้องเสียเวลาเข้าไปตรวจที่หน้าเว็บ
- ลดการสร้างไลบรารี่ซ้ำซ้อน เมื่อเราต้องการสร้างไลบรารี่ขึ้นมาใช้เอง ก็เข้าไปค้นดูก่อน ถ้าไม่มีค่อยสร้าง
- ต่อไปถ้าไลบรารี่ของเราจะสามารถใช้งานได้กับทุกๆ framework ที่ใช้ composer

วิธีติดตั้ง

Website : https://getcomposer.org/download

ติดตั้งบน window : http://www.thaicreate.com/community/php-composer.html

ติดตั้งบน Mac OSX, Linux :

To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'544e09ee996cdf60ece3804abc52599c22b1f40f4323403c44d44fdfdd586475ca9813a858088ffbc1f233e9b180f061') { echo 'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"

sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
sudo chmod +x /usr/local/bin/composer
composer
```

Set php path in window environment (Window)

Set Environment :ถ้าใช้ Windows ไปที่ My Computer > Advanced > Environment Variables... > path > Edit > แล้วสังเกตว่ามี path ของ php ที่ถูกต้องไหม? C:\xampp\php; ถ้าไม่มี ก็สามารถใส่ C:\xampp\php; ได้เลย **หลังจากนั้นปิด cmd/PowerShell แล้วเปิดใหม่** เพื่อ Reload ค่าใหม่

LARAVEL ตอนที่ 3 ติดตั้ง LARAVEL

การติดตั้ง Laravel ใน Document ของ Official Website แนะนำไว้ 2 วิธี วิธีที่ง่ายที่สุดคือติดตั้งผ่าน composer โดยเข้าไปใน Terminal หรือ Command Line และทำตามขั้นตอนดังนี้

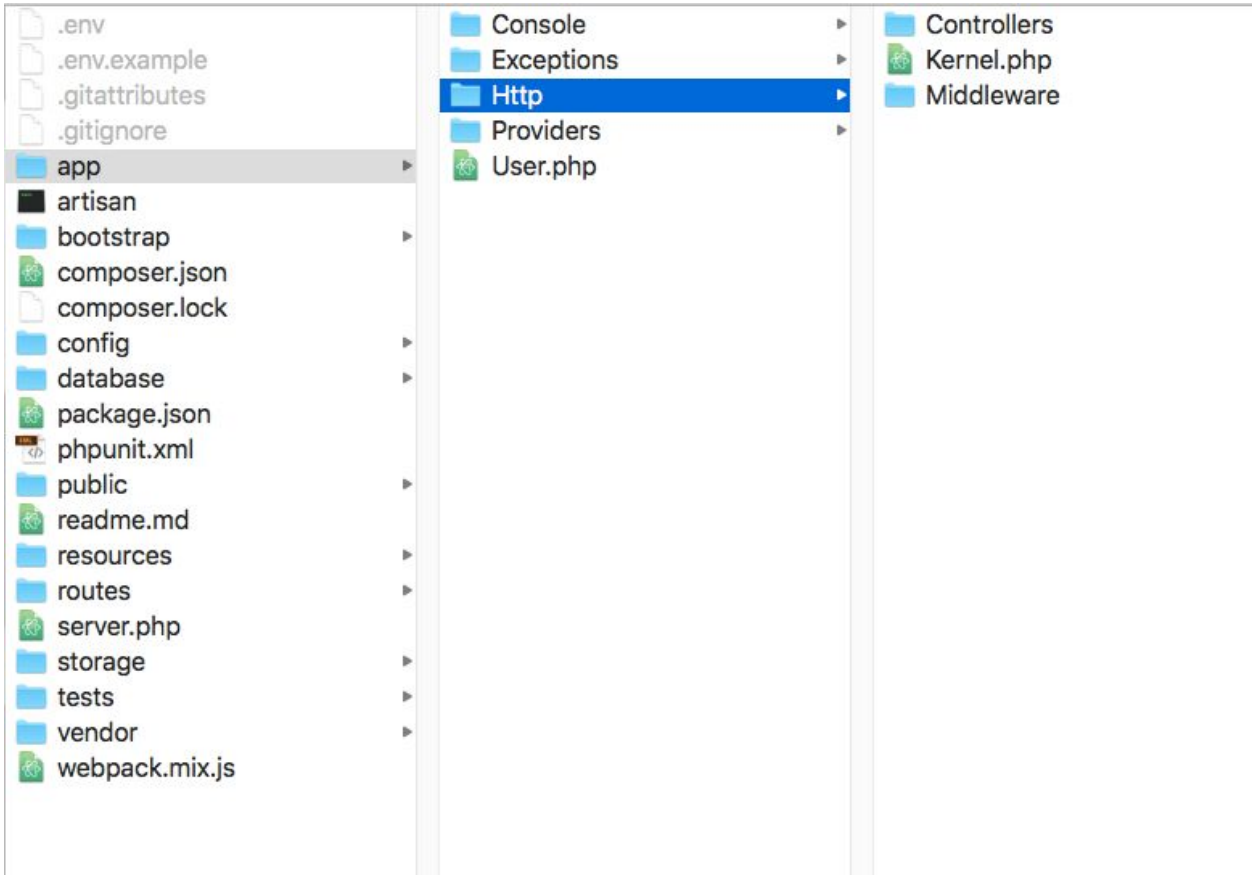
สร้าง Laravel First Project ผ่าน Composer โดยจะทำการดาวโหลด Laravel ลงที่ folder ปัจจุบันที่เราอยู่

```
composer create-project --prefer-dist laravel/laravel training
```

Update composer เพื่อให้ Update Software ให้เป็น Version ล่าสุด

```
composer update
```

Laravel 5.6 Structure



รายละเอียดโครงสร้างของ Laravel ที่ควรรู้จัก

app – โฟลเดอร์นี้ใช้เก็บไฟล์ที่เป็นคอร์หลักของโปรเจ็กเรา Model กับ Controller เราจะเขียนกันในนี้

bootstrap – เห็นว่า Laravel มันไว้ใช้เก็บโค้ดส่วนของการ boot ระบบ

config – โฟลเดอร์นี้ใช้เก็บค่าคอนฟิกทั้งหมดในโปรเจ็ก

database – โฟลเดอร์นี้ใช้เก็บไฟล์ migration, seed, factory (ไม่เก็บ Model ไว้ในนี้)

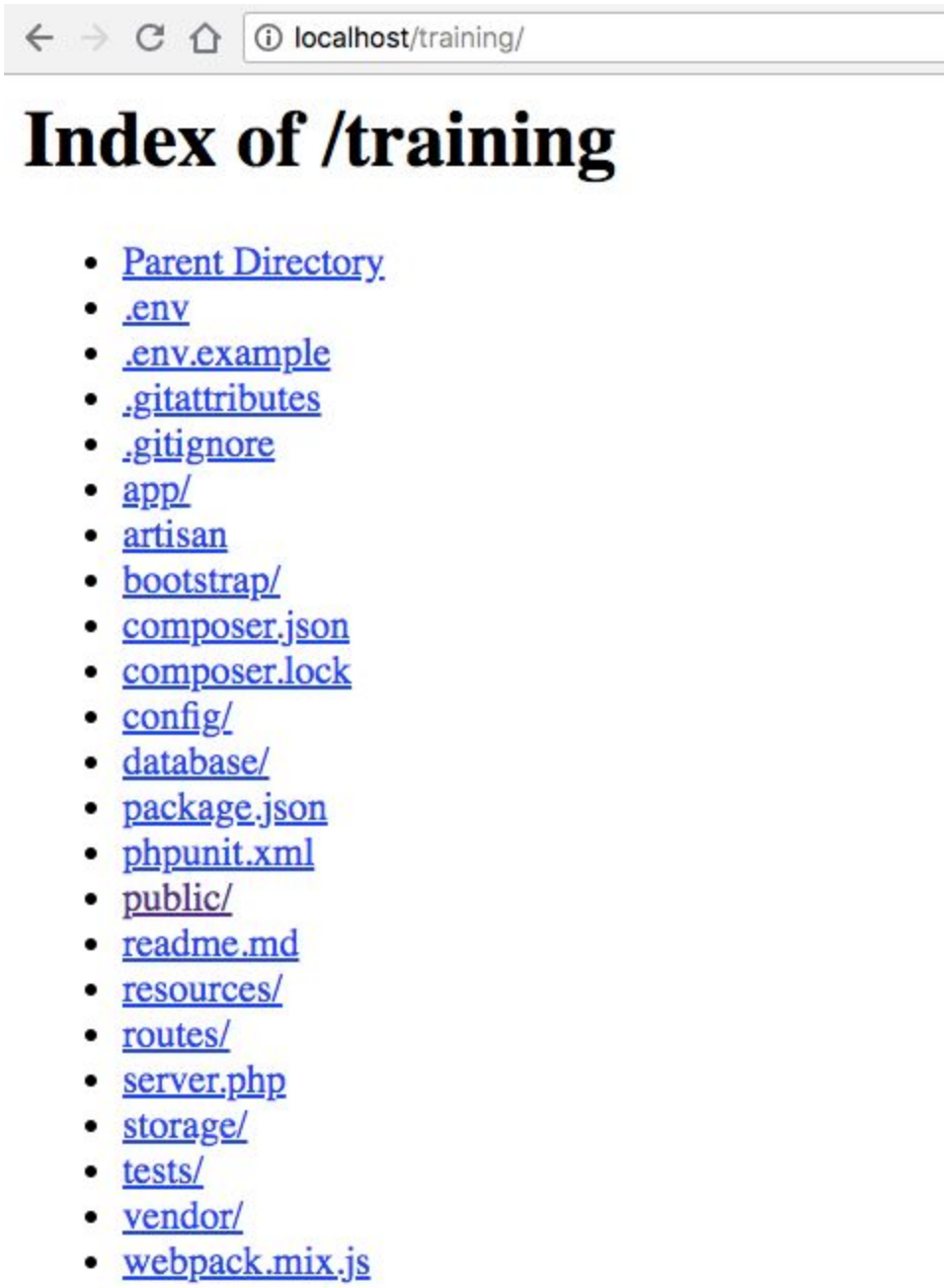
public – เป็นโฟลเดอร์ที่เข้าถึงได้ผ่าน Browser ได้

resources – โฟลเดอร์นี้ใช้เก็บไฟล์ View

storage – โฟลเดอร์นี้ใช้เก็บไฟล์ที่ถูก Laravel คอมไพล์ (ต้องถูกเซ็ตเป็น 777)

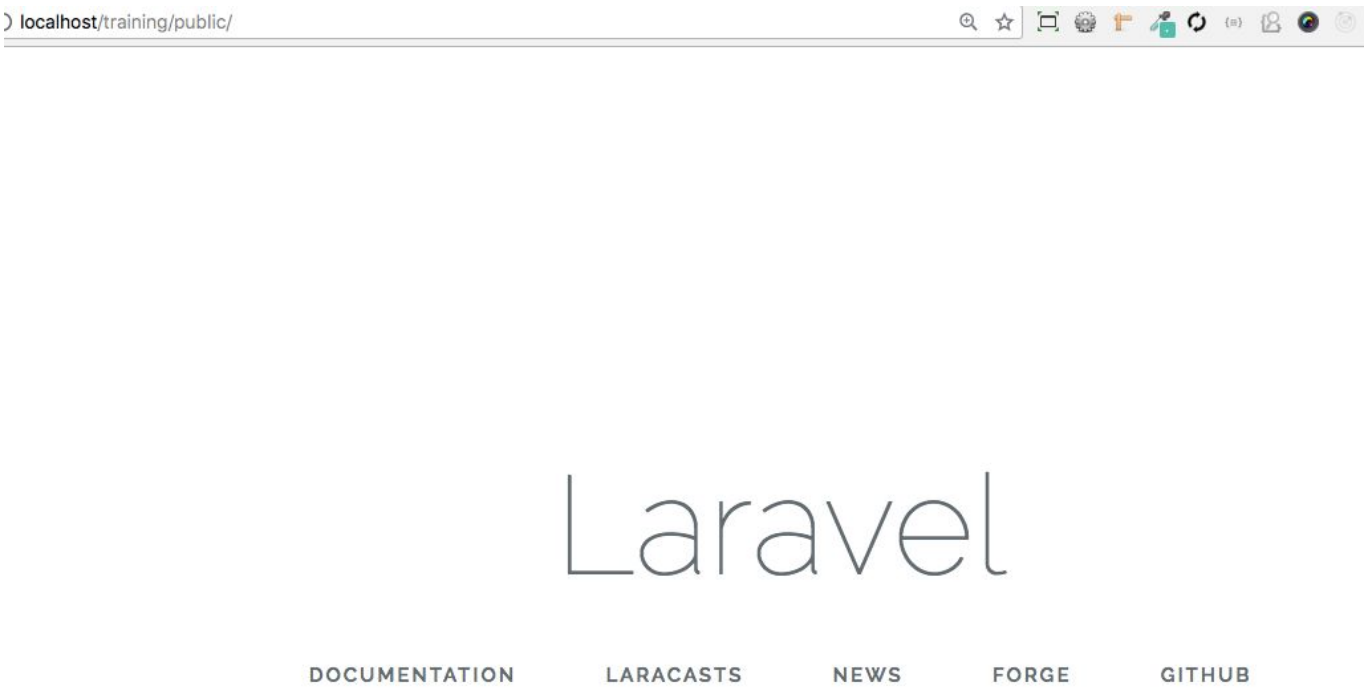
vendor – โฟลเดอร์นี้ใช้เก็บไฟล์คลาสจากภายนอกที่เราจะเอามาใช้ในโปรเจ็ก เมื่อติดตั้ง Library ผ่าน composer จะดาวน์โหลดคลาสมา มันก็จะเอามาเก็บไว้ที่นี่

เปิดหน้าเวป Laravel ครั้งแรก จะเป็นดังนี้



ซึ่งการ Access Codes ตรงๆ ผ่าน Browsers จะสามารถ Access Code ได้แค่ที่อยู่ใน Folder public เท่านั้น เนื่องจากเป็น Security ของ Framework ที่ต้องทำงานตาม Architecture Design เพื่อควบคุมความปลอดภัย ของระบบ ดังนั้นเราจะสามารถ Access เข้าหน้า HTML ใน public folder ดังนี้

URL : http://localhost/training/public/index.php



Setup Virtual Host เพื่อทำให้ URL สวยงาม และตัด /public ออกไป

MAMP เข้า Terminal

```
sudo vi /Applications/MAMP/conf/apache/httpd.conf
```

แก้ไข บรรทัดที่ 575 เพื่อทำการเปิดให้สามารถ config virtual host ของ apache ได้

```
# Virtual hosts
#Include Include /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf
Include /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf
```

เริ่ม Setup Virtual Host

```
#### MAMP on MacOSX ####
sudo vi /Applications/MAMP/conf/apache/extra/httpd-vhosts.conf

#### XAMPP on Window ####
C:\xampp\apache\conf\extra\httpd-vhosts.conf
```

เพิ่มข้อมูล vhost ใหม่เพิ่มไปบรรทัดล่างสุด

```
<VirtualHost *:80>
   ServerAdmin admin@training.me
   DocumentRoot "C:/xampp/htdocs/training/public"

   ## MacOSX ##
   #DocumentRoot "/Applications/MAMP/htdocs/training/public"

   ServerName www.training.me
   ErrorLog "logs/www.training.me-error_log"
   CustomLog "logs/www.training.me-access_log" common
</VirtualHost>

<VirtualHost *:80>
   ServerAdmin admin@training.me
   DocumentRoot "C:/xampp/htdocs"

   ## MacOSX ##
   #DocumentRoot "/Applications/MAMP/htdocs"

   ServerName localhost.me
   ErrorLog "logs/localhost.com-error_log"
   CustomLog "logs/localhost.com-access_log" common
</VirtualHost>
```

ทำการ Map Host เพื่อ Map Domain ที่เราต้องการ กับ Vhost (Localhost) ของเรา

```
#### MAMP on MacOSX ####
sudo vi /etc/hosts

#### XAMPP on Window ####
C:\Windows\System32\drivers\etc\hosts
```

เพิ่มข้อมูล hostname (Domain ที่เราตั้งชื่อตาม ServerName ใน vhost ข้างบน)

```
127.0.0.1 www.training.me
```

เข้า cmd/PowerShell หรือ Terminal ทดสอบการ map host ด้วยการ ping จะได้ผลลัพธ์ ดังนี้

```
ping www.training.me

#### Result ####
PING www.training.me (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.060 ms
```

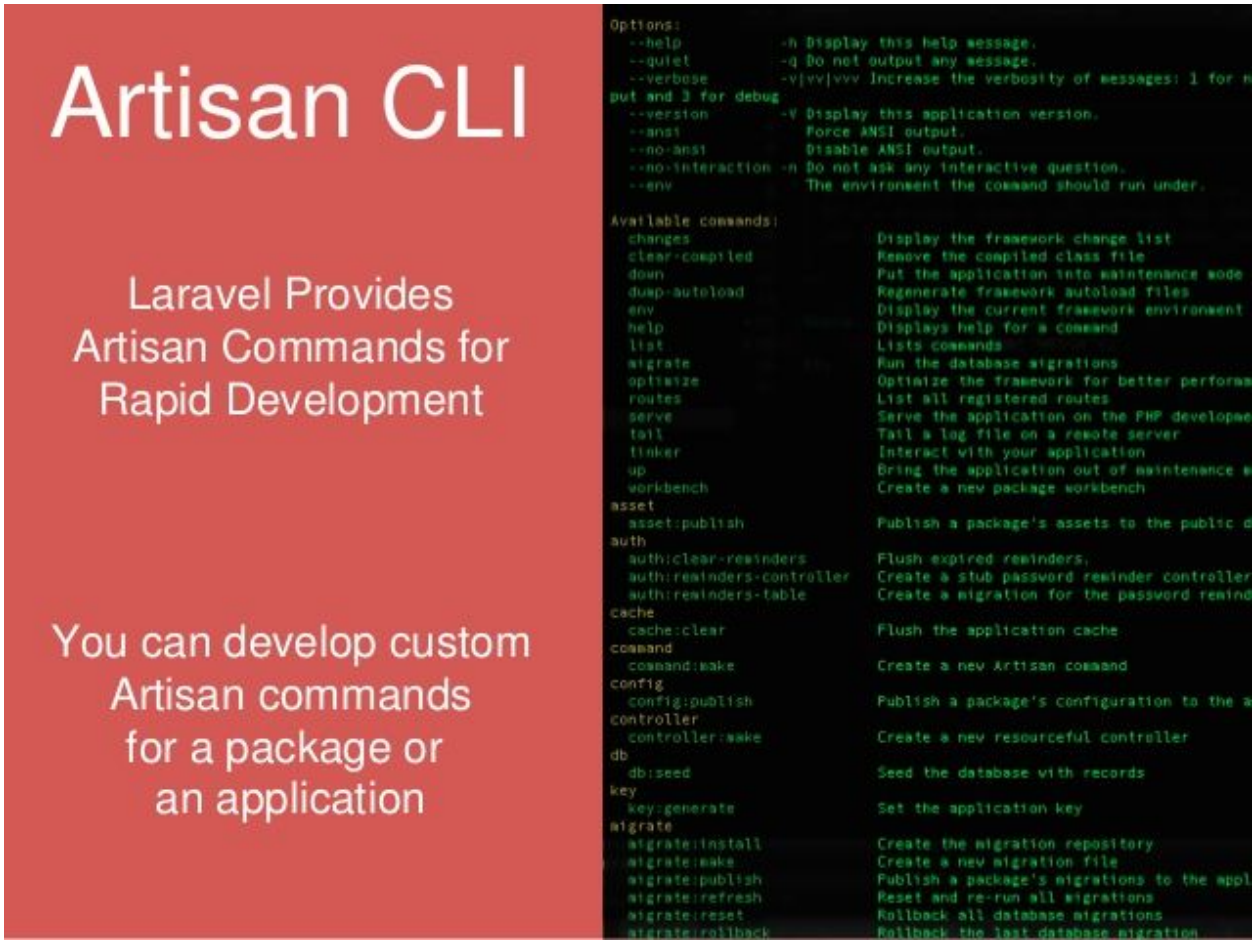
Restart Web Server for Reload new configuration vhost

สามารถเข้าผ่าน domain url ได้ดังนี้ (http://www.training.me)

Laravel

OCUMENTATION LARACASTS NEWS FORGE GITHUB

LARAVEL ตอนที่ 4 ARTISAN CLI



Artisan คือ ชุดคำสั่งที่ใช้เรียกงานผ่านทาง command line เพื่อช่วยให้จัดการงานต่างให้ง่าย รวดเร็วขึ้น เช่นการสร้าง Controller, Model, Migrate, Seed database เป็นต้น

เรียกดูคำสั่งทั้งหมดใช้คำสั่ง list

```
php artisan list
```

ทุกคำสั่งจะมีวิธีการใช้ให้เราดู โดยเพิ่ม parameter help เข้าไปแต่ละคำสั่งที่ต้องการ

```
php artisan help make:controller

php artisan help migrate
```

สร้าง Controller ใหม่ โดยใช้ Artisan CLI ผ่าน Command Line

```
php artisan make:controller DemoController
```

Artisan CLI จะทำการสร้าง Controller ให้อัตโนมัติ ที่ (app/Http/controllers/DemoController.php)

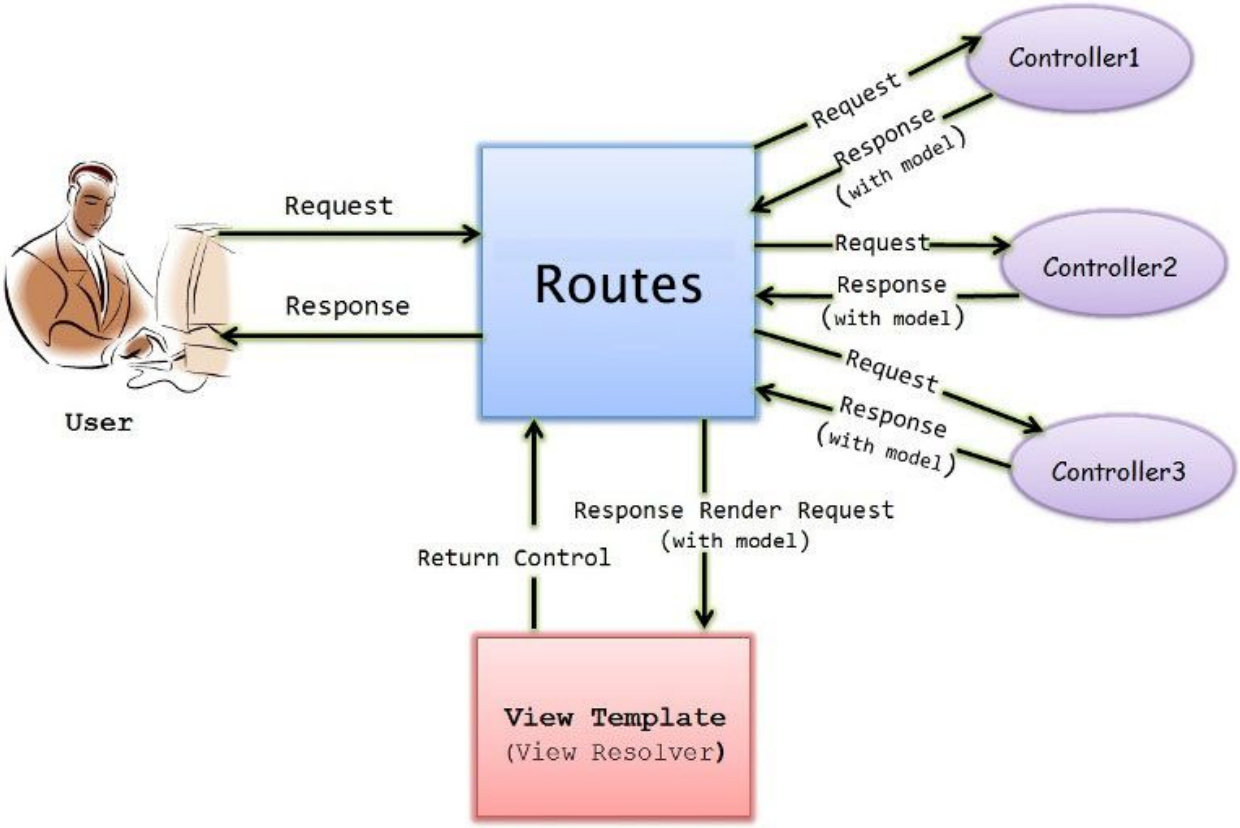
```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

class DemoController extends Controller
{
   //
}
```

ทดลองเขียน Controller เพื่อใช้ในการทดสอบ Routing ในหัวข้อถัดไป

```
class DemoController extends Controller
{
   public function index()
   {
      return "Method GET: Index";
   }

   public function demotwo()
   {
      return "Method POST: demotwo";
   }
```

```
    public function demothree()
    {
        return "Method GET, POST : demothree";
    }

    public function demofour()
    {
        return "Method GET, POST, PUT/PATCH, DELETE : demofour";
    }
}
```

LARAVEL ตอนที่ 5 Routing



Routing คือส่วนที่ใช้จัดการ url ที่ใช้ติดต่อกับ Users ผ่าน Browser ซึ่งจะการจัดการ url ว่า path/slash นี้ จะให้วิ่งไปทำงานที่ Controller ที่ Class และ Function ตัวไหน และควบคุมได้ว่าอนุญาติ ให้ Web Method (GET, POST, PUT/PATCH, DELETE) ตัวไหนที่สามารถ Access Class และ function นี้ได้ เป็นต้น

1. api วาง route ที่ใช้งานผ่าน api (routes/api.php)
2. console วาง route ที่ ถูกเรียกใช้งานผ่าน command line (routes/console.php)
3. web ใช้วาง route ที่เอาไว้เรียกใช้ผ่านหน้าเว็บ (routes/web.php)

Basic Routing

คือการ Routing แบบง่าย โดยสามารถเรียกผ่าน url ด้วย path /foo exp : http://www.training.com/foo

```
Route::get('foo', function () {
    return 'Hello World';
});
```

The Default Route Files

คือการ Routing แบบเชื่อมความสัมพันธ์ระหว่าง url กับ Class และ function ใน Controller อย่างชัดเจน อีกทั้งยังควบคุม Web Method (GET, POST, PUT/PATCH, DELETE) ที่ต้องการได้

จัดการ Routing โดยเพิ่ม Code ที่ไฟล์ routes/web.php

จากหัวข้อ Artisan เราได้ทำการสร้าง Controller ไฟล์ที่ชื่อว่า app/Http/controllers/DemoController.php และเขียนโค๊ดไว้แล้ว

Route แบบ **get** จะสามารถ Access ได้เฉพาะ GET Method เท่านั้น

http://www.training.com/demoone

```
Route::get('/demoone', 'DemoController@index');
Route::post('/demotwo', 'DemoController@demotwo');
```

Route แบบ **post** จะสามารถ Access ได้เฉพาะ POST Method เท่านั้น

http://www.training.com/demotwo

VerifyCsrfToken

รู้จัก VerifyCsrfToken

เนื่องจาก การใช้ POST/PUT/PATCH/DELETE Method จะต้องผ่าน Security ของ VerifyCsrfToken สำหรับการ POST/PUT/PATCH/DELETE ที่ไม่ผ่านฟอร์ม เพื่อป้องกัน Cross-site Request Forgery (CSRF) เพื่อให้ง่ายในการทดสอบ เราจะปิด VerifyCsrfToken โดยวิธีปิดจะมี 2 วิธี

แบบที่ 1 ปิดทั้งหมด เข้าไปที่ไฟล์ app/Http/Kernel.php ให้ Comment ปิดบรรทัดที่ 36

```
//\App\Http\Middleware\VerifyCsrfToken::class,
```

แบบที่ 2 ปิดบาง URL เช่น URL /demotwo, /demothree โดยเข้าไปเพิ่ม Code ในไฟล์ app/Http/Middleware/VerifyCsrfToken.php

```
protected $except = [
        'demotwo',
        'demothree',
```

```
        'demofour ',
        'demofour/*'
    ];
```

Route แบบ **match** สามารถระบุ web method ได้หลายตัวที่ต้องการพร้อมกัน ยกตัวอย่างเช่น จากโค๊ดด้านล่าง จะ Access ได้เฉพาะ GET และ POST Method เท่านั้น

http://www.training.com/demothree

```
Route::match(['get', 'post'], '/demothree', 'DemoController@demothree');
```

Route แบบ **any** สามารถ Access ได้ทุก Web Method

http://www.training.com/demofour

```
Route::any('/demofour', 'DemoController@demofour');
```

Route Parameters

Route Parameter บางครั้งเราต้องการค่าของ segment จาก url. เช่นต้องการค่า id จาก url

http://www.training.com/demofive/1

http://www.training.com/demofive/2

```
Route::get('demofive/{id}', function ($id) {
    return 'ID: '.$id;
});

//Optional
Route::get('demofive/{id?}', function ($id=123) {
    return 'ID: '.$id;
});
```

ต้องการค่า จาก segment id และ name มาใช้

http://www.training.com/demosix/1/myname

```
Route::get('demosix/{id}/{name}', function ($id, $name) {
    return 'ID: '.$id.' || NAME: '.$name;
});
```

Regular Expression Constraints

Regular Expression Constraints ใช้สำหรับการอนุญาติ url segment เป็นข้อมูลประเภทที่ต้องการ เช่น id ต้องเป็นตัวเลขเท่านั้น, name ต้องเป็นตัวอักษรเท่านั้น เป็นต้น

Example 1

http://www.training.com/demoseven/1 => passed

http://www.training.com/demoseven/mytext => not passed

Example 2

http://www.training.com/demoeight/mytext => passed

http://www.training.com/demoeight/1 => not passed

Example 3

http://www.training.com/demonine/1/mytext => passed

```
Route::get('demoseven/{id}', function ($id) {
    return 'demoseven ID: '.$id;
})->where('id', '[0-9]+');

Route::get('demoeight/{name}', function ($name) {
    return 'demoeight NAME: '.$name;
})->where('name', '[A-Za-z]+');

Route::get('demonine/{id}/{name}', function ($id, $name) {
    return 'demonine ID: '.$id.' || NAME: '.$name;
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Global Constraints

Global Constraints คือการนำ Regular Expression Constraints ที่ใช้บ่อยๆ ไปไว้รวมกันในจุดเดียว และสามารถ access ได้ทุก route ได้เลย เข้าไปที่ app/Providers/RouteServiceProvider.php

```
    public function boot()
    {
        Route::pattern('age', '[0-9]+');
        Route::pattern('school', '[a-z]+');
        parent::boot();
    }
```

รทดสอบโดยการเพิ่ม route ใหม่ ที่ routes/web.php

http://www.training.me/demoten/age/18/school/hogwarts

```
Route::get('demoten/age/{age}/school/{school}', function ($age, $school) {
    return 'demoten age: '.$age.' || SCHOOL: '.$school;
});
```

Route Prefixes

Route Prefixes เป็นการจัดกลุ่มของการ route หลายๆตัวเข้ามาอยู่ในกลุ่มเดียวกัน โดยมีเงื่อนไขที่ url ทุกตัวที่ต้องการจัดกลุ่ม จะต้องเป็น url ที่มี prefix ลำดับเดียวกัน ที่เหมือนกัน เพื่อลดการเขียน route ให้สั้นลง และสามารถควบคุมการเข้าถึงได้เช่น ต้อง login ก่อน เป็นต้น

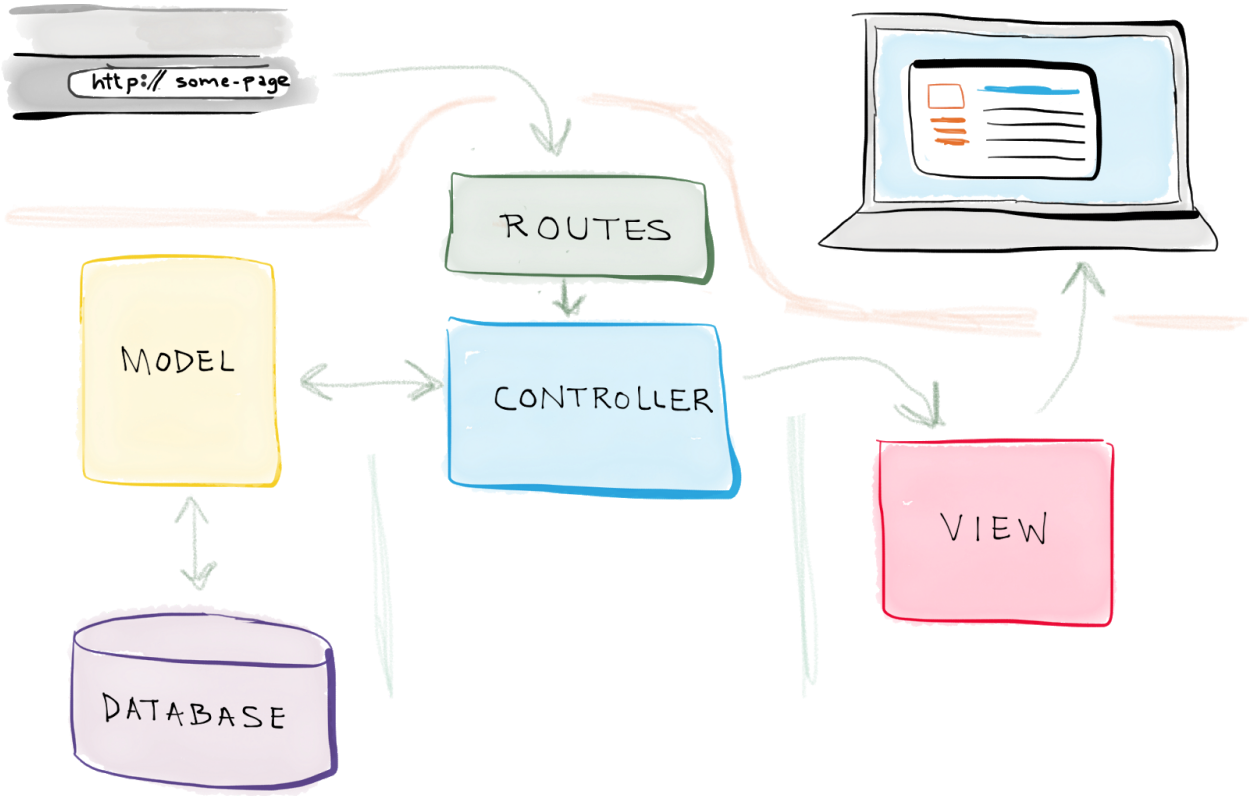
สามารถ access โดยมี prefix คือ admin ใน url ทุกตัว ที่อยู่ใน group นี้ เช่น

http://www.training.com/admin/demothree

http://www.training.com/admin/demofour

```
Route::prefix('admin')->group(function () {
    Route::match(['get', 'post'], 'demothree', 'DemoController@demothree');
    Route::any('demofour', 'DemoController@demofour');
});
```

LARAVEL ตอนที่ 5 Controller



Controllers จะเป็นตัวกลางที่คอยประสานงานรับส่งข้อมูล ระหว่าง Model View Libraries และควบคุม flow, logic การทำงานของระบบตามต้องการ โดยจะมี Routing ที่คอย map ระหว่าง url จาก browers กับ Controller ที่ต้องการ

สร้าง Controller ใหม่ โดยใช้ Artisan CLI ผ่าน Command Line

```
php artisan make:controller BlogController
```

Defining Controllers

```
<?php

namespace App\Http\Controllers;
use Illuminate\Http\Request;

class BlogController extends Controller
{

   public function myaction()
   {
       return 'Hello World';
   }

}
```

namespace คือทำมาแก้ข้อห้ามของ php ที่ว่าด้วยการห้ามใช้ชื่อคลาสซ้ำกัน เช่นในกรณีที่เรามี Class ชื่อ User และมีการนำ Library ที่มีคลาสชื่อว่า User เข้ามาอีก แน่นอนว่าเราไม่สามารถมีชื่อ Class ซ้ำกัน ที่อยู่ในระบบเดียวกันได้ จึงเกิด Concept Namespace เพื่อเข้ามาแก้ปัญหานี้ เพื่อกำหนดขอบเขตที่อยู่ของ Class เพื่อช่วยแก้ปัญหานี้

use คือคำสั่งที่นำเอา class อื่นมาใช้งานใน class Controller ของเรา อย่างเช่น use Illuminate\Http\Request คือนำ class Request (Class ที่จัดการเรื่อง Request จาก form) ที่อยู่ภายใน namespace Illuminate\Http มาใช้งาน

class DemoController extends Controller คือการสืบทอดคลาส จาก class แม่ที่ชื่อว่า Controller (Class DemoController จะมีความสามารถเหมือน class Controller)

function myaction คือ function การทำงานของ Controller ที่จะถูก Map มาจาก Route ตาม URL ที่ Users ใช้ โดยเราจะเข้าไปเขียนโค๊ดเพื่อทำงาน ในการเป็นตัวกลางระหว่าง View, Model และควบคุม Flow, Logic ต่างๆ

Resource Controllers

Resource Controllers คือเป็น Controller ประเภท Restful Controller เมื่อเราทำการสร้าง Resource Controllers ผ่าน Artisan CLI แล้วจะทำการสร้างไฟล์ Controller ขึ้นมา พร้อม function ที่มีชื่อมาตรฐาน โดยหลักการทำงานของ Restful นั้น จะมีความเชื่อมโยงกันระหว่าง Web Method (GET, POST, PUT/PATCH, DELETE), URL Segment และ Function แต่ละตัวของ Controller นั้น

สร้าง Resource Controller ใหม่ โดยใช้ Artisan CLI ผ่าน Command Line โดย Artisan จะสร้าง Resource Controller ตามชื่อที่เราตั้งไว้ที่ app/Http/controllers/PhotoController.php

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

เพิ่ม route แบบ resource เพื่อ Support Controller แบบ Resource Controller ที่ routes/web.php

```
Route::resource('photos', 'PhotoController');

// Case controller in folder
Route::resource('photos', 'Admin\PhotoController');
```

ในกรณีที่ต้องการ Route แบบ Resource Controller หลายๆตัว สามารถทำได้

```
Route::resources([
    'photos' => 'PhotoController',
    'posts' => 'PostController'
]);
```

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit

PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Specifying The Resource Model

เราสามารถสร้าง Controller ที่ทำการผูกกับ Model ที่เราต้องการโดยใช้ option --model=<Modalname> โดย Artisan CLI จะทำการสร้าง Code ในการ import Class Model มาให้ใน Controller อัตโนมัติ และถ้าในกรณียังไม่มี Class Model นี้อยู่ โดย Default จะอยู่ที่

สร้าง **Class Controller** ที่ Path : app/Http/Controllers/PhotoController.php

สร้าง **Class Model** ที่ Path : app/Photo.php

```
php artisan make:controller PhotoController --resource --model=Photo
```

ในกรณี ต้องการสร้าง Controller หรือ Model ให้อยู่ใน Folder (ถ้าไม่มี Folder จะสร้างให้อัตโนมัติ) เพื่อความเป็นระเบียบ จะสามารถสร้างได้โดย โดยในตัวอย่าง

สร้าง **Class Controller** ที่ Path : app/Http/Controllers/Admin/PhotoController.php

สร้าง **Class Model** ที่ Path : app/Model/Photo.php

```
php artisan make:controller Admin/PhotoController --resource --model=Model/Photo
```

Partial Resource Routes

เราสามารถระบุได้ว่า Resource Routes ตัวนี้ จะอนุญาติให้เข้าถึง Function ใน Class ของ Controller

Route file : routes/web.php

```
Route::resource('photos', 'PhotoController')->only([
    'index', 'show'
]);

Route::resource('photos', 'PhotoController')->except([
    'create', 'store', 'update', 'destroy'
]);
```

API Resource Routes

เป็น Route ที่สร้างมาให้เป็น Route แบบ Restful ที่เหมาะกับงานประเภท APIs มีความสามารถของการ Route เหมือน Resource Routes แต่ตัดการ access function **create** และ **edit** ออกไป เนื่องจาก APIs จะไม่มีในส่วนของการสร้างฟอร์ม html ในการ create และ edit

Route file : routes/web.php

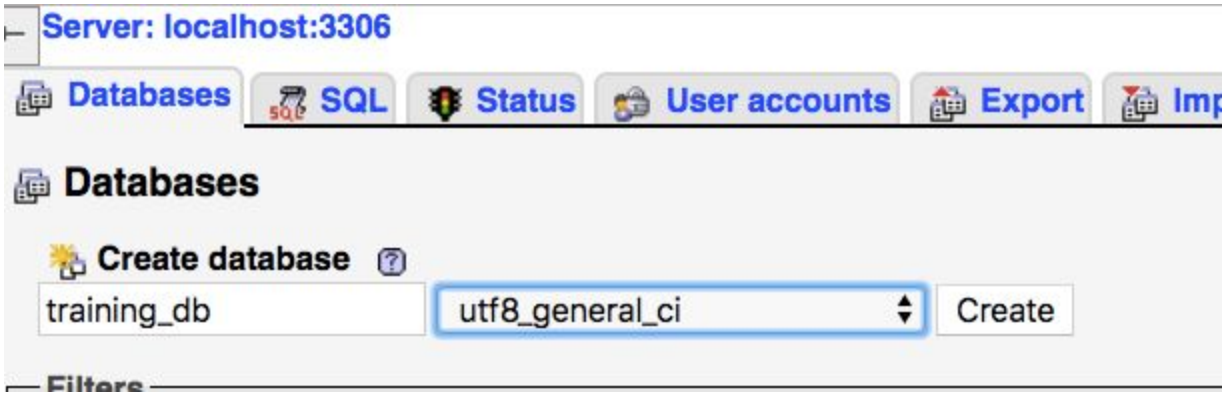
```
Route::apiResource('photos', 'PhotoController');

// multiple apiResources
Route::apiResources([
    'photos' => 'PhotoController',
    'posts' => 'PostController'
]);
```

LARAVEL ตอนที่ 6 Database: Migrations & Seeding

Introduction

Migrations คือเครื่องมือที่ช่วยในการจัดการเกี่ยวกับ Tables ใน Database ประเภท RDBMS เช่น MySQL หรือ MariaDB โดยสามารถ Create Table ใหม่, Alter Table, และ Delete Table เป็นต้น ง่ายในการระบุชื่อ และ ประเภทของ Fields ในแต่ละ Table อีกทั้งยังสามารถ ย้อนกลับ Rollback การจัดการ Table ไปยังก่อนหน้าได้

ก่อนเริ่มต้น เราจะต้องเข้าไป สร้าง Database ใหม่ ชื่อ training_db แบบ utf8_general_ci ใน MySQL ก่อน โดยใช้ PHP MyAdmin ที่ติดตั้งมาแล้วพร้อมกับ Web Server ของเรา



และหลังจากนั้นเราจะทำการ Configuration Database ที่จะใช้ที่ ไฟล์ config/database.php

```
'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'strict' => true,
        'engine' => null,
    ],
```

ใน Laravel Version ใหม่ๆ นี้ ได้มีการเพิ่ม DotEnv Files ขึ้นมา เพื่อใช้ในการรองรับ Projects เดียวกัน ที่ต้องรันอยู่ใน Environtments หลายๆที่ เช่น Local, Alpha, Staging, Production Environtments เป็นต้น โดย Default ไฟล์ .env จะอยู่ที่ Root ของ Folder นั้น (เป็น hidden files ต้องดูจาก Text Editor)

File : .env

```
APP_ENV=local
APP_URL=http://http://www.training.com
DB_HOST=127.0.0.1
DB_DATABASE=training_db
DB_USERNAME=root
DB_PASSWORD=root
```

Trick : Switching .env files by Environment

ในกรณี เรามี Environtments หลายๆที่ เช่น Local, Alpha, Staging, Production Environtments เราสามารถรองรับหลาย Environment ที่มี Configuration ต่างๆ ไม่เหมือนกัน เช่น Database, URL, Cache, Queue เป็นต้น เราสามารถ ทำได้โดยให้เทคนิค DotEnv ดังนี้

ตัวอย่างเช่น สร้างไฟล์ .env.production แล้ว Copy config จาก .env มาใส่ แล้วแก้ไข Config ใน .env.production ให้ตรงกับ Production

File : .env.production

```
APP_ENV=production
APP_URL=http://http://www.training-prod.com
DB_HOST=127.0.0.1
DB_DATABASE=training_db
DB_USERNAM
E=root
DB_PASSWORD=
```

หลังจากนั้น เข้าไปเพิ่มโค๊ดสำหรับการ switch .evn ที่ไฟล์ bootstrap/app.php ก่อน return $app;

File : bootstrap/app.php

```
$httpHost = ( isset($_SERVER['HTTP_HOST']) ) ? $_SERVER['HTTP_HOST'] : 'local.dev';

switch($httpHost)
{
   case 'www.training-prod.com':
      $envFile = '.env.production';
   case 'www.training-staging.com':
      $envFile = '.env.staging';
   default:
      $envFile = '.env';
}

// change $envFile conditionally here
$app->loadEnvironmentFrom($envFile);
```

Database: Migrations

Migrations คืออะไร ??

ขออธิบายแบบง่ายๆ มันก็คือ Code หรือไฟล์ไว้สำหรับสร้างตารางใน Database

การสร้าง Migrations

ให้เราเปิดโปรแกรม Command Line เข้ามาแล้วเข้าไปยัง Path ของ Project เราสามารถสร้าง Migrations ได้อย่างรวดเร็วผ่าน Artisan CLI ดังนี้

File **Migrations** จะถูกสร้างที่

Path : database/migration/xxx_xxx_xxx_xxx_create_shop_table.php

โดยมี Pattern ชื่อไฟล์ คือ <year>_<month>_<date>_<time>_create_users_table

.php

```
$httpHost = ( isset($_SERVER['HTTP_HOST']) ) ? $_SERVER['HTTP_HOST'] : 'local.dev';

use Illuminate\Support\Facades\Schema;

class AppServiceProvider extends ServiceProvider
{
  public function boot()
    {
       Schema::defaultStringLength(191);
    }
```

```
php artisan migrate
php artisan migrate:fresh
```

```
php artisan make:migration create_shop_table
php artisan make:migration add_votes_to_users_table --table=users
```

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
```

```
    */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

- **CreateUsersTable extends Migration** คือคลาส CreateUsersTable สืบทอดคลาส มาจากคลาสแม่คือคลาส Migration ที่มจาก Framework ทำให้มีความสามารถในการ Migration

- **function up()** คือ function ที่ใช้ในการสร้าง field Schema ของ Table ผ่านการเขียน Code แทนการสร้างใน phpmyadmin เมื่อเรามีการรัน migrate ระบบจะมาทำงานที่ function up() และทำการสร้าง Table ตาม Code ที่เราเขียนไว้

- **function down()** คือ function ที่ใช้ในการ Drop Table จะทำงานก็ต่อเมื่อ มีการ rollback หรือ refresh

เราสามารถเพิ่ม หรือ ลด Fileds ใน Table ที่ต้องการได้ เช่น

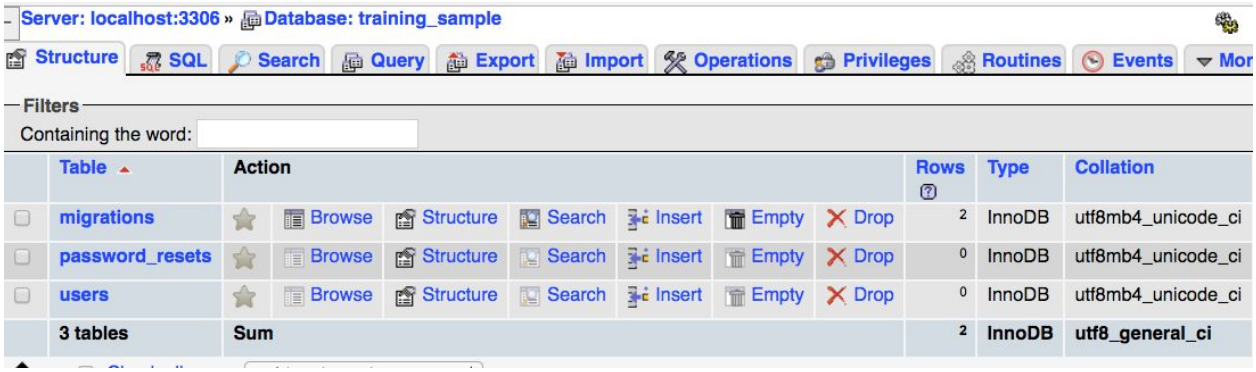
```
public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('surname');
            $table->string('email')->unique();
            $table->string('password');
            $table->integer('age')->default(0);
            $table->string('address')->nullable();
            $table->string('city')->nullable();
            $table->string('mobile')->nullable();
```

```
            $table->integer('active')->default(1);
            $table->rememberToken();
            $table->timestamps();
        });
    }
```

Run migrate in command line by ArtisanCLI

```
#### Migrate ####
php artisan migrate

#### Migrate Refresh ####
php artisan migrate:refresh

#### Rollback 1 step ####
php artisan migrate:rollback

#### Refresh is Reset all & migrate schema table ####
php artisan migrate:refresh

#### Reset all & migrate schema table & insert data ####
php artisan migrate:refresh --seed
```

เมื่อ Migrate เสร็จแล้ว ระบบจะทำการสร้าง Table ตาม Files migration ที่เราสร้างไว้



โดยจะได้ Table **users** ตาม Code ที่เราเขียนไว้ เช่น

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments
1	id	int(10)		UNSIGNED	No	None	
2	name	varchar(191)	utf8mb4_unicode_ci		No	None	
3	surname	varchar(191)	utf8mb4_unicode_ci		No	None	
4	email	varchar(191)	utf8mb4_unicode_ci		No	None	
5	password	varchar(191)	utf8mb4_unicode_ci		No	None	
6	age	int(11)			No	0	
7	address	varchar(191)	utf8mb4_unicode_ci		Yes	NULL	
8	city	varchar(191)	utf8mb4_unicode_ci		Yes	NULL	
9	mobile	varchar(191)	utf8mb4_unicode_ci		Yes	NULL	
10	active	int(11)			No	1	
11	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL	
12	created_at	timestamp			Yes	NULL	
13	updated_at	timestamp			Yes	NULL	

โดยระบบจะทำการเก็บข้อมูลไฟล์ ที่ทำการ Migrate ไป ที่ Table migrations

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

Showing rows 0 - 1 (2 total, Query took 0.0005 seconds.)

```
SELECT * FROM `migrations`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refr

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	id	migration	batch
Edit Copy Delete	5	2014_10_12_000000_create_users_table	1
Edit Copy Delete	6	2014_10_12_100000_create_password_resets_table	1

Check all With selected: Edit Copy Delete Export

Database: Seeding

Seeds คืออะไร ??

คือ โค้ดไว้สำหรับสร้างข้อมูลในตารางหลังจากเราสร้างตารางแล้ว ข้อดีของมันทำให้เราสร้างข้อมูลต้นแบบหรือข้อมูลตายตัวไว้ก่อนได้เลย

การสร้าง Seeds

ให้เราเปิดโปรแกรม Command Line เข้ามาแล้วเข้าไปยัง Path ของ Project เราสามารถสร้าง Migrations ได้อย่างรวดเร็วผ่าน Artisan CLI ดังนี้

```
php artisan make:seeder UsersTableSeeder
```

File **Migrations** จะถูกสร้างที่ Folder : database/seeds/

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('users')->insert([
            'name'     => 'MyUser',
            'email'    => 'MyUser@gmail.com',
            'password' => bcrypt('password')
        ]);
    }
}
```

- **UsersTableSeeder extends Seeder** คือคลาส UsersTableSeeder สืบทอดคลาส มาจากคลาสแม่ คือคลาส Seeder ที่มาจาก Framework ทำให้มีความสามารถในการ Seeder
- **function run()** คือ function ที่ใช้ในการ Insert ข้อมูลเข้า Table ตาม Field และ Data ที่กำหนดไว้
- **DB::table('users')->inser**t คือการใช้ class DB ที่ทำงานด้าน database ในการ insert ข้อมูลเข้า Table เช่น DB::table('users') คือ insert เข้า table users เราสามารถเขียนโปรแกรม เพื่อทำการวนลูปการ Insert ข้อมูลตามเงื่อนไขข้อมูลตามที่ต้องการได้ เช่น

```
public function run()
    {
        $cities = array("bangkok", "nakornpathom");

        for($i=1; $i<=10; $i++) {
           $key   = array_rand($cities);
```

```
            $city = $cities[$key];

            DB::table('users')->insert([
                'name'     => 'user'.$i,
                'surname'  => 'surname'.$i,
                'email'    => 'user'.$i.'@gmail.com',
                'password' => bcrypt('password'),
                'age'      => rand(18,22),
                'address'  => 'my_address'.$i,
                'city'     => $city,
                'mobile'   => '089-9999999',
                'active'   => rand(0,1)
            ]);
        }
    }
```

Run Seeding in command line by ArtisanCLI

```
#### Run Seeding ทั้งหมด ####
php artisan db:seed

#### Run Seeding ระบุ Seeding Class (Table ที่ต้องการ) ####
php artisan db:seed --class=CategoryTableSeeder

#### Migrate and Seeding ####
php artisan migrate:refresh --seed
```

LARAVEL ตอนที่ 7 Model

Introduction

Model ดูแลในส่วนของการทำงานที่ต้องติดต่อ Database จะประกอบด้วย class ที่เชื่อมต่อกับ RDBMS โดยจะมี Eloquent (Laravel) ดูแลในเรื่องของการติดต่อสื่อสารระหว่าง Object และ Database โดยที่ผู้พัฒนาไม่ต้องยุ่งยากกับการใช้ SQL command ทำให้สามารถเปลี่ยน Database ได้โดยที่ไม่ต้องมีการแก้โค๊ด

Path : app/Blog.php

Create Model by ArtisanCLI

```
####  Create Model ####
php artisan make:model Blog
```

```
#### Create Model in folder ####
php artisan make:model Model/Shop

#### Create Model in folder and create migration file ####
php artisan make:model Model/Category --migration
```

ในการสร้าง Model แบบปกติ จะมีรายละเอียด Code คือ

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    //
}
```

ใน Model จะมีรายละเอียด Code คือ

```
<?php

namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Shop extends Model
{
    //
}
```

Table Names

โดยปกติหาก Model เราชื่อว่า Shop laravel ก็จะใช้ Model นี้ในดึงข้อมูลและจัดเก็บข้อมูลจากตารางที่ชื่อว่า shops แต่ถ้าเราอยากจะตั้งชื่อเป็นอย่างอื่นก็ให้ใส่ตัวแปล protected $table = 'ชื่อตารางที่ต้องการ'

```
<?php
```

```
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Shop extends Model
{
   //protected $table = 'my_tablename';
}
```

Primary Keys

โดยปกติ laravel จะกำหนดว่า column ชื่อว่า id เป็น primary key แต่ถ้าอยากกำหนดเป็นอย่างอื่นก็ให้กำหนดค่าในตัวแปล $primaryKey

โดยปกติ Primary Key จะเป็น incrementing integer value (เป็นตัวเลขและรันค่า id อัตโนมัติ) ถ้าหากอยากกำหนดเป็น non-incrementing or a non-numeric ก็ให้กำหนดในตัวแปล public $incrementing กำหนดเป็นfalse.

Timestamps

โดยค่าเริ่มต้น laravel กำหนดให้มีคอลัมน์ created_at และ updated_at อยู่ในตาราง แต่ถ้าหากไม่ต้องการใช้ก็ให้ใส่ตัวแปล $timestamps เป็น false

```
class Shop extends Model
{
   /**
    * Indicates if the model should be timestamped.
    *
    * @var bool
    */
   public $timestamps = false;
}
```

หากต้องการเปลี่ยนชื่อของ columns ก็ให้กำหนดในตัวแปลค่าคงที่ CREATED_AT และ UPDATED_AT

```
class Shop extends Model
{
   const CREATED_AT = 'creation_date';
   const UPDATED_AT = 'last_update';
}
```

LARAVEL ตอนที่ 8 Controllers binding Model (Eloquent)

Retrieving Models In Controller with Eloquent

Model ดูแลในส่วนของการทำงานที่ต้องติดต่อ Database จะประกอบด้วย class ที่เชื่อมต่อกับ RDBMS โดยจะมี Eloquent (Laravel) ดูแลในเรื่องของการติดต่อสื่อสารระหว่าง Object และ Database โดยที่ผู้พัฒนาไม่ต้องยุ่งยากกับการใช้ SQL command ทำให้สามารถเปลี่ยน Database ได้โดยที่ไม่ต้องมีการแก้โค๊ด

```
php artisan make:controller Admin/UsersController --resource
```

```
namespace App\Http\Controllers;

// Namespace for admin folder in controller
//namespace App\Http\Controllers\Admin;

//use App\User;
// Alias name class
use App\User as UserMod;

class UserController extends Controller
{
    public function index()
```

```
    {
        $mods = UserMod::all();

        // Using alias name
        //$mods = UserMod::all();

        foreach ($mods as $item) {
            echo $item->name;
        }
    }
}
```

ใส่เงื่อนไขใน Query

```
$mods = UserMod::where('active', 1)
            ->orderBy('name', 'desc')
            ->take(10)
            ->get();
```

Chunking Results

คือการแบ่ง Query ออกเป็นช่วงๆ จะช่วยประหยัดหน่วยความจำ(memory) เมื่อทำงานกับชุดผลลัพธ์ขนาดใหญ่ ตัวอย่างข้างล่างคือ ระบบจะ query ครั้งล่ะ 200 rows ไปจนกว่าจะสิ้นสุด

```
UserMod::chunk(200, function ($users) {
    foreach ($users as $item) {
        //
    }
});
```

Retrieving Single Models / Aggregates

นอกจากการดึงข้อมูลทั้งหมดของตารางที่ระบุ แล้วคุณยังอาจเรียกดูเรคคอร์ดเดียว โดยใช้ find หรือ first ก็ได้

```
// Retrieve a model by its primary key...
$mod = UserMod::find(1);

// Retrieve the first model matching the query constraints...
$mod = UserMod::where('active', 1)->first();
```

find สามารถเรียกดูข้อมูลจาก array ของ primary key ได้

```
$mods = UserMod::find([1, 2, 3]);
```

Not Found Exceptions

ใช้ **findOrFail** และ **firstOrFail** หากหาข้อมูลไม่เจอระบบจะแจ้ง หน้า error 404 อัตโนมัติ

```
$model = UserMod::findOrFail(1);
$model = UserMod::where('legs', '>', 100)->firstOrFail();
```

Retrieving Aggregates

สามารถใช้ count, sum, max และ aggregate methods ได้ตามตัวอย่างด้านล่าง

```
$count = UserMod::where('active', 1)->count();
$max = UserMod::where('active', 1)->max('age');
```

Inserting & Updating Models (Eloquent)

Inserting data to table users

```
<?php
namespace App\Http\Controllers;

// Namespace for admin folder in controller
//namespace App\Http\Controllers\Admin;

use App\User;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
   /**
    * Create a new flight instance.
    *
    * @param  Request  $request
    * @return Response
    */
   public function store(Request $request)
   {
      // Validate the request...
```

```
        $mod = new User;
        $mod->name = $request->name;
        $mod->save();
    }
}
```

โดยกำหนดค่าให้กับ name โดยค่าส่งมาจากพารามิเตอร์ HTTP request to (ที่อาจจะถูกส่งมาจากฟอร์ม) column created_at และupdated_at จะถูกอัพเดทให้อัตโนมัติ

Updating data to table users

```
$mod = UserMod::find(1);
$mod->name = 'New Flight Name';
$mod->save();
```

Mass Updates

```
UserMod::where('active', 1)
        ->where('city', 'bangkok')
        ->update(['address' => 'new address']);
```

ระบบจะทำการอัพเดท column address ให้เท่ากับ "new address" ทุก Users ที่ Active=1 และ city='bangkok'

Mass Assignment

เราสามารถสร้างข้อมูลได้ในบรรทัดเดียวโดยใช้ create method แต่ก่อนที่จะใช้งาน ต้องกำหนด attribute fillable หรือ guardedใน Model ของเราก่อน

fillable คือกำหนดชื่อของ column ที่อนุญาติให้เข้าถึง

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
```

```
    protected $fillable = ['name'];
}
```

เราสามารถที่จะสร้างข้อมูลให้กับ column name ได้แล้ว

```
$mod =UserMod::create(['name' => 'New User']);
```

Guarding Attributes

guarded คือกำหนดชื่อของ column ที่ไม่อนุญาติให้เข้าถึง นอกเหนือจากนั้นเข้าถึงได้

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that aren't mass assignable.
     *
     * @var array
     */
    protected $guarded = ['city'];
}
```

หากกำหนด **$guarded** เป็นค่าว่าง ก็แสดงว่าทุก column สามารถเข้าถึงได้

```
/**
 * The attributes that aren't mass assignable.
 *
 * @var array
 */
protected $guarded = [];
```

ตัวอย่างการใช้งาน โดยรับค่าจากฟอร์มด้วย **HTTP request**

```
/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(StoreRentPost $request)
{
            Flight::create($request->all());
```

```
}
```

Other Creation Methods

firstOrCreate/ firstOrNew

firstOrCreate จะค้นหาข้อมูลโดยใช้คอลัมน์ / ค่าที่ระบุ ถ้าไม่พบจะทำการสร้างข้อมูลขึ้นมาด้วยแอตทริบิวต์ที่ระบุ

firstOrNew จะค้นหาข้อมูลโดยใช้คอลัมน์ / ค่าที่ระบุ ถ้าไม่พบจะทำ new model instance ขึ้นมา แต่ยังไม่ได้สร้างข้อมูลใหม่ เราต้องเรียกใช้ save() ด้วยตัวเองเพื่อให้ระบบทำการสร้างข้อมูล

```
// Retrieve mod by name, or create it if it doesn't exist...
$mod = User  ::firstOrCreate(['name' => 'My Name']);

// Retrieve mod by name, or create it with the name and delayed attributes...
$mod = User  ::firstOrCreate(
   ['name' => 'My Name'], ['delayed' => 1]
);

// Retrieve by name, or instantiate...
$mod = User  ::firstOrNew(['name' => 'My Name']);

// Retrieve by name, or instantiate with the name and delayed attributes...
$mod = User  ::firstOrNew(
   ['name' => 'My Name'], ['delayed' => 1]
);
```

updateOrCreate

ระบบจะทำการอัพเดทข้อมูลหากข้อมูลมีอยู่ แต่หากข้อมูลไม่มี ระบจะสร้างขึ้นมาใหม่ (ไม่ต้องเรียกใช้save())

```
// If there's a user is active == 2 and set name, email, password
// If no matching model exists, create one.
$mod = UserMod::updateOrCreate(
   ['name' => 'myname1', 'email' => 'myname1@gmail.com', 'password' =>
bcrypt('password')],
   ['active' => 2]
```

```
);
```

Deleting Models

ระบบจะทำการลบข้อมูล User ที่มี id=1

```
$mod = UserMod::find(1);
$mod->delete();
```

Deleting An Existing Model By Key

ในตัวอย่างด้านบนเราจะทำการดึงข้อมูลที่จะลบขึ้นมาก่อนค่อยลบ แต่หากเราจะระบุ primary key ลงไปเลยโดยไม่ต้องดึงข้อมูลก็ทำได้ตามด้านล่าง

```
UserMod::destroy(1);
UserMod::destroy([1, 2, 3]);
UserMod::destroy(1, 2, 3);
```

Deleting Models By Query

ตัวอย่างด้านล่างคือลบทุก User ที่มี column active=0

```
$deletedRows = UserMod::where('active', 0)->delete();
```

Soft Deleting

คือการลบข้อมูลโดยที่ไม่ได้ลบออกจากฐานข้อมูลจริงๆ แต่จะไปเพิ่มวันเวลาที่ลบไว้ใน column deleted_at แทน และระบบจะไม่ดึงข้อมูลที่ deleted_at ไม่เท่ากับ null มาแสดง วิธีใช้งาน Soft Deleting ใน Model ให้กำหนดค่าตามด้านล่าง

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model
{
    use SoftDeletes;

    /**
     * The attributes that should be mutated to dates.
     *
     * @var array
     */
    protected $dates = ['deleted_at'];
}
```

LARAVEL ตอนที่ 9 Eloquent: Relationships

One To One

ความสัมพันธ์แบบ หนึ่งต่อหนึ่ง เช่น ผู้ใช้หนึ่งคนมีโทรศัพท์หนึ่งเครื่อง เป็นต้น

```
php artisan make:migration create_shops_table

php artisan migrate:fresh
php artisan db:seed
```

```
Schema::create('shops', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned();
            $table->string('name');
            $table->string('desc');
            $table->timestamps();
        });
```

```
php artisan migrate

//create seeder
php artisan make:seeder ShopsTableSeeder
```

```
for($i=1; $i<=100; $i++) {
            DB::table('shops')->insert([
        'user_id'   => $i,
```

```
        'name'      => 'ShopID:'.$i.'-UserID:'.$i,
        'desc'      => 'desc',
    ]);
}
```

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

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function shop()
    {
        return $this->hasOne('App\Model\Shop');
    }
}
```

หากจะเข้าถึงข้อมูล Shop ผ่าน Model User

```
$shop = UserMod::find(1)->shop;
echo $shop->name;
```

หรือ

```
$user = UserMod::find(1);
echo $user->shop->name;
```

ก็จะได้ข้อมูลเหมือนกัน

* User::find(1) คือการค้นหาข้อมูลจาก Primary Key ตาม ตัวอย่างนี้คือ ค้นหา id ของตาราง users ที่มีค่าเท่ากับ 1

Foreign Key

การกำหนด foreign key ใน laravel จะกำหนดโดยใช้ “ชื่อตารางหลัก(ไม่มี s) _id” อัตโนมัติ ในกรณีตาราง shops Foreign Key ก็จะเป็น “user_id”

แต่ถ้าอยากจะกำหนดเป็นชื่ออื่นก็ให้ใช้

```
return $this->hasOne('App\Model\Shop', 'foreign_key');
```

แต่ถ้าอยากจะกำหนดเป็นชื่ออื่นก็ให้ใช้

```
return $this->hasOne('App\Model\Shop', 'foreign_key');
```

หากตาราง users Primary Key ไม่ได้กำหนดชื่อเป็น id อาจจะกำหนดเป็นชื่ออื่น เราก็ต้องเชื่อมความสัมพันธ์ระหว่าง Primary Key และ Foreign Key เอง

```
return $this->hasOne('App\Model\Shop', 'foreign_key', 'local_key');
```

เข้าถึงค่าในทางกลับกันใน One To One

หากต้องการเชื่อมความสัมพันธ์กับตาราง users ก็ให้ใช้ belongsTo

```
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Shop extends Model
{
    /**
     * Get the user that owns the shop.
     */
    public function user()
    {
        return $this->belongsTo('App\User');
    }
```

```
}
```

ในการณี foreign key ไม่ใช่ user_id

```
public function user()
{
   return $this->belongsTo('App\User', 'foreign_key');
}
```

ในการณีตาราง users Primary Key ไม่ใช่ id

```
/**
 * Get the user that owns the phone.
 */
public function user()
{
   return $this->belongsTo('App\User', 'foreign_key', 'other_key');
}
```

กำหนดค่า Default หากค่าใน Model ที่เชื่อมความสัมพันธ์มีค่า null

เช่น หาก user (record) โดนลบไป ใน model Post เวลาเรียกใช้งาน User ก็ให้ return ค่าเป็นค่าว่าง

```
/**
 * Get the author of the post.
 */
public function user()
{
   return $this->belongsTo('App\User')->withDefault();
}
```

หรือหากจะ return ค่า Default เป็น attributes ก็ตามตัวอย่างด้านล่าง

```
/**
 * Get the author of the post.
 */
public function user()
{
    return $this->belongsTo('App\User')->withDefault([
        'name' => 'Guest Author',
    ]);
}

/**
 * Get the author of the post.
 */
public function user()
{
    return $this->belongsTo('App\User')->withDefault(function ($user) {
        $user->name = 'Guest Author';
    });
}
```

One To Many

ความสัมพันธ์แบบ หนึ่งต่อกลุ่ม เช่น บทความหนึ่งบทความมีหลายความคิดเห็น เป็นต้น

```
php artisan make:migration create_products_table
```

Migration folder: app/database/migrations/xxx_products_table.php

```
Schema::create('products', function (Blueprint $table) {
        $table->increments('id');
```

```
            $table->integer('user_id')->unsigned();
            $table->integer('shop_id')->unsigned();
            $table->string('name');
            $table->string('desc');
            $table->integer('status')->default(1);
            $table->timestamps();
        });
```

```
php artisan migrate
```

Create Seeder File

```
php artisan make:seeder ProductsTableSeeder
```

Seed folder: app/database/seeds/ProductsTableSeeder.php

```
public function run()
    {
        for($i=1; $i<=100; $i++) {
            for($product=1; $product<=5; $product++) {
                DB::table('products')->insert([
                    'user_id'   => $i,
                    'shop_id'   => $i,
                    'name'      => 'produc-Shop:'.$i.'-'.$product,
                    'desc'      => 'desc',
                    'status'    => rand(0,1)
                ]);
            }
        }
    }
```

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

// Create Product Model
php artisan make:model Model/Product
```

```
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Shop extends Model
{
    /**
     * Get the products for the shop.
     */
    public function products()
    {
        return $this->hasMany('App\Model\Product');
    }
}
```

การเข้าถึง products

```
//$products = \App\Model\Shop::find(1)->products;
$products = ShopMod::find(1)->products;

foreach ($products as $product) {
    //
}
```

สามารถฟิวเตอร์ ด้วยเงื่อนไขเข้าไปอีกได้ เช่น ดึงมาเฉพาะ products ที่มี name ตามต้องการ

```
$products =  App\Model\Shop::find(1)->products()->where('name', 'foo')->first();
```

เข้าถึงค่าในทางกลับกันใน One To Many

หากต้องการเชื่อมความสัมพันธ์กับตาราง Shop ก็ให้ใช้ belongsTo

```php
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
   public function shop()
   {
      return $this->belongsTo('App\Model\Shop');
   }
}
```

Many To Many

ความสัมพันธ์แบบ กลุ่มต่อกลุ่ม เช่นสินค้า 1 ชิ้น (Product) มีได้หลายหมวดหมู่ (Category) ตัวอย่างเช่น Notebook ยี่ห้อ Apple รุ่น MacBook Air อยู่ในสินค้าหมวดหมู่ Computer, Notebook, อุปกรณ์ไฟฟ้า เป็นต้น

แต่ หมวดหมู่ Notebook ก็สามารถมีสินค้าได้หลายรุ่น เช่น Dell Notebook, Asus Notebook, MacBook Air เป็นต้น

เพราะฉนั้น Relation แบบ Many To Many จะต้องมี Table ที่ 3 ที่ใช้เป็น Table ที่เชื่อมความสำพันธ์ระหว่าง Table **categories** กับ Table **products** คือ Table **category_product**

```
// products:
        id    - INTEGER
        name - VARCHAR

// categories:
```

```
        id   - INTEGER
        name - VARCHAR

// category_product (relation table)  :
        id      - INTEGER
        product_id - INTEGER
        category_id - INTEGER
```

ใน Model User

```
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
   /**
    * The categories that belong to the product.
    */
   public function categories()
   {
      return $this->belongsToMany('App\Model\Category');
   }
}
```

เข้าถึง categories ผ่าน Product ใน Controller

```
$product = App\Model\Product::find(1);

foreach ($product>categories as $category) {
   //
}
```

```
$categories =  App\Model\Product::find(1)->categories()->orderBy('name')->get();
```

ปกติตารางที่ใช้เชื่อมระหว่าง model product และ model category ก็จะเป็น category_product (Eloquent will join the two related model names in alphabetical order) หากไม่ใช่ก็สามารถระบุ ชื่อ table ที่เป็น table relation ระหว่าง 2 table คือ

```
return $this->belongsToMany('App\Model\Category, 'category_product');
```

key เชื่อมก็สามารถกำหนดได้เช่นกัน

```
returnr  $this->belongsToMany('App\Model\Category', 'category_product', 'product_id', 'category_id');
```

key เชื่อมก็สามารถกำหนดได้เช่นกัน

```
returnr  $this->belongsToMany('App\Model\Category', 'category_product', 'product_id', 'category_id');
```

หากเข้าถึงผ่าน Model Category

```
<?php
namespace App\Model;
use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    /**
     * The users that belong to the role.
     */
    public function products()
    {
        return $this->belongsToMany('App\Model\Product');
    }
}
```

LARAVEL ตอนที่ 10 Controllers binding View

View คือส่วนการแสดผลของเว็บไซต์ของเรานะครับ เราแยกออกมาเพื่อทำให้ง่ายต่อการจัดการได้ง่ายนะครับ ใน Laravel นี้จะมี template คือ blade engine ในการจัดการแสดงผล หน้า HTML, CSS, JS อีกทั้งยัง รับข้อมูลที่ส่งมาแสดงผล จาก Controller ได้

เราสร้างขึ้นไว้ตรงที่ **resources/views** และตั้งชื่อไฟล์ที่ เช่น เราจะใช้ชื่อไฟล์ว่า template ที่ใช้เรียกจาก Controller เราจะต้องตั้งชื่อไฟล์ View ใน path **resources/views/template.blade.php** จะต้องมี xxx.blade.php เพื่อทำให้ Laravel ทราบว่า ไฟล์ view ตัวนี้คือ เป็น View ที่ใช้ blade template engine

สร้างไฟล์ View ใหม่ File : resources/views/template.blade.php

```
<!doctype html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Laravel</title>
      <link href="https://fonts.googleapis.com/css?family=Raleway:100,600"
rel="stylesheet" type="text/css">
   </head>

   <body>
      <h1> Hello, World.</h1>
   </body>
</html>
```

ทำการเชื่อมต่อระหว่าง Controller กับ View จากที่เราสร้างไปข้างบน โดยจะเข้าไปเพิ่มคำสั่งที่ใช้ในการเชื่อมต่อที่ Controller File ดังนี้

```
public function index()
{
    return view('template');
 }
```

วิธีการส่งค่าจาก Controller ไปยัง View แบบต่างๆ

แบบที่ 1

Controller File

```
public function index()
{
    return view('template')->with('name', 'My Name');
 }
```

View File : resources/views/template.blade.php

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

แบบที่ 2 สามารถ Chain parameter with ได้มากกว่า 1 ตัว

Controller File

```
public function index()
{
    return view('template')
                ->with('name', 'My Name')
                ->with('surname', 'My SurName')
                ->with('email', 'myemail@gmail.com');
 }
```

View File : resources/views/template.blade.php

```
{{ $name.' : '.$surname }}
{{ $email }}
```

แบบที่ สามารถ ส่งเป็นแบบ array

Controller File

```
public function index()
{
    $data = [
```

```
        'name' => 'My Name',
        'surname' => 'My SurName',
        'email' => 'myemail@gmail.com'
    ];

    return view('template', $data);
}
```

View File : resources/views/template.blade.php

```
{{ $name.' : '.$surname }}
{{ $email }}
```

แบบที่ สามารถ ส่งเป็นแบบ array ซ้อน array

Controller File

```
public function index()
{
    $data = [
        'name' => 'My Name',
        'surname' => 'My SurName',
        'email' => 'myemail@gmail.com'
    ];

    $item = [
        'item1' => 'My Value1',
        'item2' => 'My Value2'
    ];

    $results = [
        'data' => $data,
        'item' => $item
    ];

    return view('template', $results);

}
```

View File : resources/views/template.blade.php

```
{{ print_r($data) }}
{{ print_r($item) }}
```

แบบที่ สามารถ ส่งเป็นแบบ Object ที่ได้จาก Model แบบ Single โดยใช้ Compact Function

Controller File

```
public function index()
{
     $mods = UserMod::all();
       return view('template', compact('mods'));
 }
```

View File : resources/views/template.blade.php

```
@if($mods)
   <ul>
   @foreach($mods as $item)
      <li>{{ $item->name.' : '.$item->surname }}</li>
   @endforeach
   </ul>
@endif
```

แบบที่ สามารถ ส่งเป็นแบบ Object ที่ได้จาก Model, Array, String แบบ Multiple โดยใช้ Compact Function

Controller File

```
public function index()
{
     $data = [
         'name' => 'My Name',
         'surname' => 'My SurName',
         'email' => 'myemail@gmail.com'
      ];

       $user = UserMod::find(1);
       $mods = UserMod::all();

       return view('template', compact('data', 'user', 'mods'));
 }
```

View File : resources/views/template.blade.php

```
{{ print_r($data) }}

<h1> Show User Single Object from Model </h1>
{{ $user->name.' : '.$user->surname }}
```

```
<h1> Show mods Multiple Objects From Model</h1>
@if($mods)
   <ul>
   @foreach($mods as $item)
      <li>{{ $item->name.' : '.$item->surname }}</li>
   @endforeach
   </ul>
@endif
```

Controller File :

```
public function index()
{
   $mods = UserMod::paginate(10);
   return view('admin.user.lists', compact('mods') );
 }
```

View File :

```
    <tbody>
         @if($mods)
         @foreach($mods as $item)
          <tr>
            <td>{{ $item->name }}</td>
            <td>{{ $item->mobile }}</td>
            <td>{{ $item->email }}</td>
            <td>{{ $item->city }}</td>
            <td>
               @if($item->active == 1)
                  <span class="badge badge-success">Active</span>
               @else
                  <span class="badge badge-danger">Inactive</span>
                @endif
            </td>



           </tr>
         @endforeach
         @endif
```

Publish Pagination :

```
php artisan vendor:publish --tag=laravel-pagination
```

View file :

```
<th colspan="6" style="justify-content: right;">{{ $mods->links('vendor.pagination.bootstrap-4') }}</th>
```

Create Custom My Request :

File Location : app/Http/Request/UserRequest

```
php artisan make:request UserRequest
```

Create Custom My Rules :

File Location : app/Rules/FiveCharacters

```
php artisan make:rule FiveCharacters
```

LARAVEL Login

Create Controller:

```
php artisan make:controller LoginController
```

Adding code in LoginController :

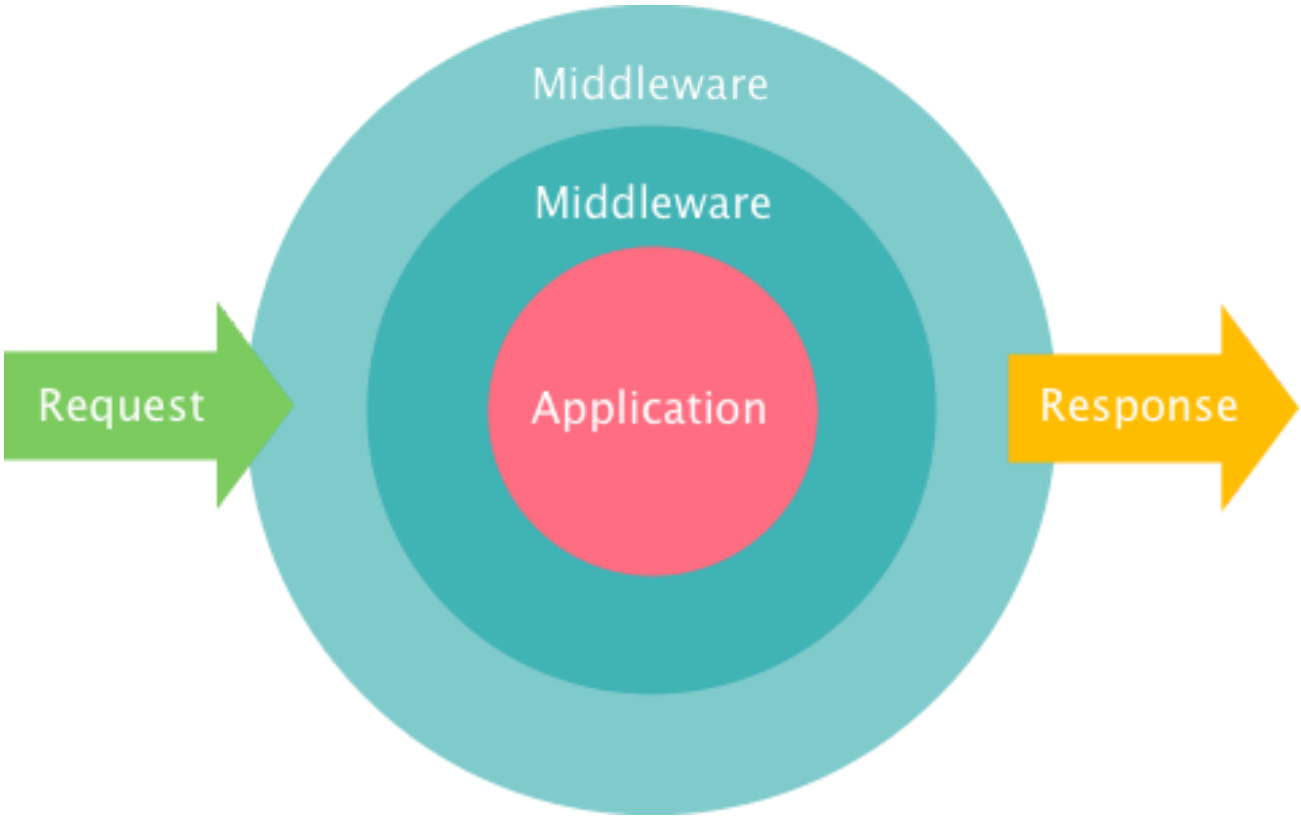
```
use Illuminate\Support\Facades\Auth;
```

```
 //assume this method is loginform
   public function index()
   {
      return view('login.formlogin');

   }


public function authenticate(Request $request)
   {
      $credentials = $request->only('email', 'password');

      if (Auth::attempt($credentials)) {
         // Authentication passed...
         // Get the currently authenticated user...
         //$user = Auth::user();
         return redirect('admin/user');
      } else {
         return redirect('login')
                  ->with('error', 'Invalid User Or Password.');
      }
   }

public function logout()
   {
      Auth::logout();
      return redirect('login');
   }
```

Adding route in web.php :

```
Route::get('login', 'LoginController@index')->name('login');
Route::get('logout', 'LoginController@logout');
Route::post('login', 'LoginController@authenticate');


//Route::resource('admin/user', 'Admin\UsersController');
Route::prefix('admin')->middleware('auth')->group(function () {
   Route::resource('user', 'Admin\UsersController');
});
```

LARAVEL Middleware



Middleware คืออะไร

Middleware คือ กลไกสำหรับคัดกรอง request ที่เข้ามาภายในแอปของเรา โดยแอปของเราสามารถมี middleware ได้หลายตัว แต่ละตัวก็มีหลักการทำงานแตกต่างกันไป เช่น ถ้าจะเข้า url นี้ต้องล็อคอินแล้ว หรือเป็น Admin เท่านั้น เป็นต้น