

e-Book

# พัฒนา Modern Web App ด้วย Laravel 9



โค้ชเอก  
[CodingThailand.com](http://CodingThailand.com)

# Web Development with Laravel 9 (step by step)

พัฒนา Modern Web App ด้วย Laravel 9

โค้ชเอก

หนังสือเล่มนี้จัดทำโดยที่ <https://codingthailand.com/laravel9ebook>

เควอร์ชัน 1 ออกจำหน่ายวันที่ 20 มีนาคม 2565

หนังสือเล่มนี้ถูกเขียนตั้งใจจัดทำขึ้นเพื่ออย่างให้มีคนอ่านได้ภาษาไทยสักเล่มเกี่ยวกับ Laravel ที่เน้นเนื้อหาตั้งแต่พื้นฐาน การทำงานกับฐานข้อมูล และการทำรายงานต่างๆ โดยเน้นสรุปประเด็นที่สำคัญ เพื่อให้ผู้อ่านสามารถนำไปต่อยอดพัฒนา Web Application ที่ต้องการได้ หวังว่าหนังสือเล่มนี้จะเป็นประโยชน์ ประยุต์เวลาการเรียนรู้ขอให้สนุกกับการเรียนรู้ครับ

“คงเขียนนี้ไม่รู้ด้วยการพัฒนาตัวเอง และลงมือทำอย่างสม่ำเสมอ”



©2022 โค้ชเอก

## สารบัญ

<b>บทที่ 1 การเตรียมตัวและการติดตั้ง Laravel ด้วย Composer</b>	4
- การติดตั้งต้องเตรียมอะไรบ้าง	
- Extensions ของ PHP ที่ควรเปิดให้	
- การติดตั้ง Laravel ด้วย Composer	
- การตั้งค่าระบบของ Laravel	
- การตั้งค่า timezone	
- การ Debugging ใน Laravel	
<b>บทที่ 2 ทำความรู้จักกับ Laravel, MVC และ Best Practices</b>	12
- ทำไมต้องใช้ PHP Framework	
- ทำความรู้จักกับ Laravel	
- โครงสร้างของ Laravel ที่สำคัญ	
- MVC และ Best Practices	
<b>บทที่ 3 การเขียน และใช้งาน Controllers, Routes, Layout, Views</b>	15
- พื้นฐานการเขียน Controllers, Routes, Views และการส่งค่าของตัวแปรไปแสดงผลที่ Views	
- การสร้างไฟล์ และการจัดการ Layout	
- การเรียกใช้ Layout ใน Laravel	
- การสร้าง Section ในมโดยใช้ @yield	
<b>บทที่ 4 ออกแบบฐานข้อมูลและตารางด้วย Artisan, Database Migrations และการทำ Seeding</b>	26
- การตั้งค่าฐานข้อมูล	
- การสร้างตารางฐานข้อมูลด้วย Migration	
- การเพิ่มข้อมูลเริ่มต้นลงในตารางด้วย Seeding	
<b>บทที่ 5 การทำงานกับฐานข้อมูล การสร้าง Models และ การใช้ Eloquent ORM</b>	33
- การสร้าง Models	
- การใช้งาน Eloquent ORM	
- ตัวอย่างคำสั่งสำหรับการเรียกดูข้อมูล หรือแสดงข้อมูล	
- ตัวอย่างคำสั่งสำหรับกรองข้อมูล (Filtering records) เทียบได้กับ where, order by และ limit	

- คำสั่งสำหรับการเพิ่มข้อมูล และแก้ไขข้อมูล
- คำสั่งในการลบข้อมูล
- แสดงข้อมูลตาราง ประเภทหนังสือ (typebooks)
- การลบข้อมูล ประเภทหนังสือ (typebooks)
- การแบ่งหน้าข้อมูล (Pagination)
- Query scopes
- การสร้าง Accessors
- การสร้าง Mutators
- การกำหนด Eloquent relations
- แสดงข้อมูลตารางหนังสือ (books) ด้วยการทำ relations (join table)

บทที่ 6 การสร้าง Web Forms การตรวจสอบความถูกต้องของข้อมูล และการอัปโหลดไฟล์ 55

- การสร้างฟอร์มใน Laravel
- การติดตั้ง และใช้งาน Laravel Collective
- สร้างฟอร์มเพิ่มข้อมูลหนังสือ (books)
- การตรวจสอบความถูกต้องของข้อมูล (Validation)
- การติดตั้ง Image Library เพื่อเตรียมพร้อมก่อนอัปโหลดไฟล์
- การเพิ่มข้อมูลหนังสือ (books) และอัปโหลดไฟล์ภาพ
- สร้างฟอร์มแก้ไขข้อมูลหนังสือ (books) และแก้ไขภาพที่ต้องการอัปโหลด
- สร้างฟอร์มการลบข้อมูลหนังสือ (books)
- การทำ responsive lightbox

บทที่ 7 การใช้งาน Sessions และการจัดการสิทธิ์ผู้ใช้งาน 93

- การใช้งาน Session
- การใช้งาน Flash Data
- การกำหนดสิทธิ์ให้

บทที่ 8 การสร้างรายงานในรูปแบบ PDF และ Charts (รูปแบบวิดีโอ) 108

บทที่ 9 ใบน้ำพิเศษ 109

- สรุป 39 คำสั่งของ Laravel ที่ใช้งานบ่อย

## บทที่ 1 การเตรียมตัวและการติดตั้ง Laravel ด้วย Composer

### การติดตั้งต้องเตรียมอะไรบ้าง

เนื่องจากหนังสือเล่มนี้ไม่ใช่หนังสือพื้นฐาน การเตรียมตัวอย่างแรกในการอ่านหนังสือเล่มนี้คือ เราจะต้องมีพื้นฐานภาษา PHP และ มีความรู้เกี่ยวกับการเขียนโปรแกรมในรูปแบบของ Object Oriented Programming หรือ OOP มา ก่อน เพื่อให้เกิดประโยชน์สูงสุดในการเรียนรู้ สำหรับคนที่ยังไม่มีพื้นฐานความรู้ดังที่กล่าวมา ผู้อ่านควรให้ศึกษา ก่อนครับ

ในการติดตั้ง Laravel นั้น ก่อนเรียนต้องเตรียมตัวและติดตั้งโปรแกรมต่างๆ ประกอบด้วย

1. XAMPP สำหรับจำลองเครื่องเราให้เป็น Web Server ประกอบด้วย Apache, PHP, MySQL/MariaDB และ phpMyAdmin หรือโปรแกรมจำลอง Web Server อื่นๆ
2. PHP จะต้องเป็นเวอร์ชัน 8 ขึ้นไป เพราะฉะนั้นในข้อ 1 จะต้องดูด้วยว่าใช้ XAMPP ที่มี PHP เวอร์ชัน 8 หรือไม่
3. Visual Studio Code สำหรับใช้เขียนโค้ด หรือจะใช้ IDE หรือ Editor ที่ถนัดก็ได้
4. Composer สำหรับการจัดการกับ PHP Packages และ Library ต่างๆ
5. Node.js เวอร์ชัน LTS สำหรับการจัดการกับส่วนของ frontend

### Extensions ของ PHP ที่ควรเปิดไว้

บางครั้งระหว่างติดตั้ง Composer หรือ พัฒนา Web Application อาจมี errors สำหรับบางคำสั่ง ก่อนติดตั้ง Laravel ควรไปตรวจสอบ หรือเปิด extension ในไฟล์ php.ini ให้เรียบร้อย คือ ให้เปิดไฟล์ php.ini (C:\xampp\php\php.ini) ค้นหา extensions แล้วเอาเครื่องหมาย; (เข็มโคลอน) ข้างหน้าออก เสร็จแล้วบันทึกไฟล์แล้ว Restart Apache สำรวจการ extensions ที่ควรเปิด มีดังต่อไปนี้

```
extension=php_bz2.dll      extension=php_curl.dll      extension=php_mbstring.dll      extension=php_fileinfo.dll
extension=php_gd2.dll       extension=php_openssl.dll      extension=php_intl.dll      extension=php_pdo_mysql.dll
extension=php_mbstring.dll
```

```
990 | extension=php_bz2.dll
991 | extension=php_curl.dll
992 | extension=php_mbstring.dll
993 | extension=php_exif.dll
994 | extension=php_fileinfo.dll
995 | extension=php_gd2.dll
996 | extension=php_gettext.dll
997 | ;extension=php_gmp.dll
998 | extension=php_intl.dll
999 | extension=php_openssl.dll
```

หมายเหตุ extension=php\_openssl.dll ให้ใส่ เครื่องหมาย; (เเชมิคอลอน) ไม่ต้องเอาออกก็ได้ ปกติจะเปิดมาให้แล้ว

## การติดตั้ง Laravel ด้วย Composer

การติดตั้ง Laravel นั้น วิธีที่ง่ายและสะดวก แนะนำติดตั้งผ่าน Composer โดยมีขั้นตอนการติดตั้ง ดังนี้

1. เปิด Command Prompt และพิมพ์ cd C:\xampp\htdocs เพื่อเข้าไปโฟลเดอร์ htdocs

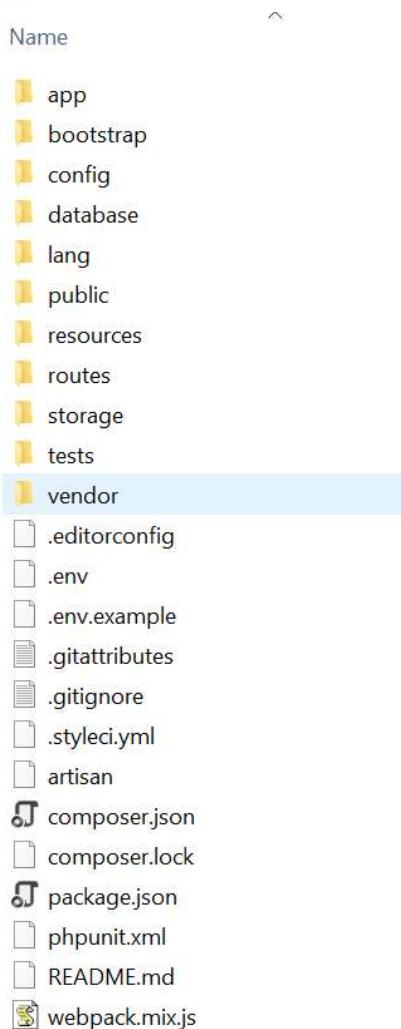
Administrator: Command Prompt

C:\>cd C:\xampp\htdocs

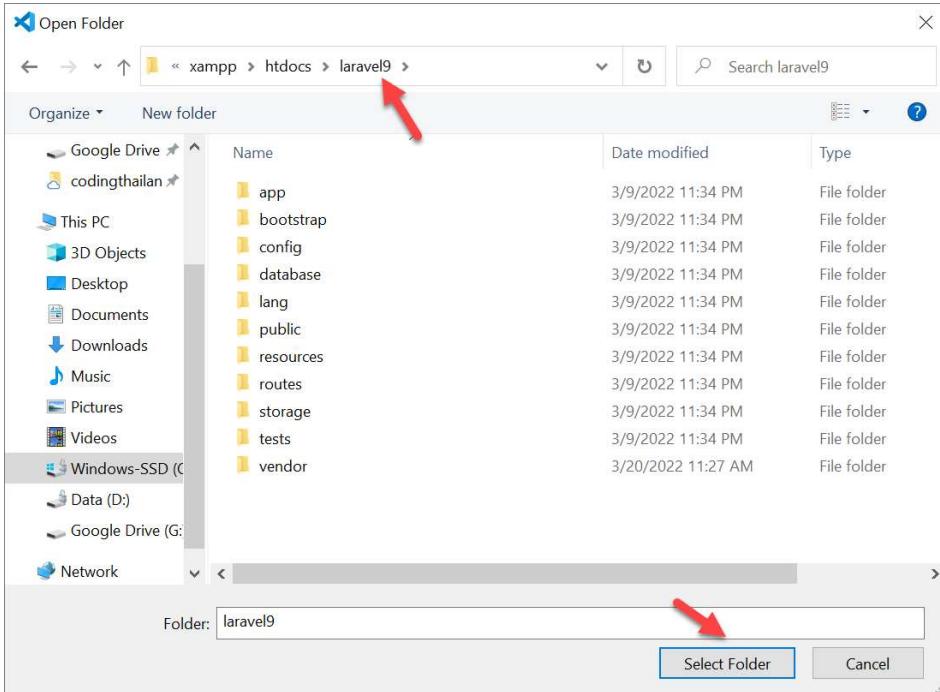
C:\xampp\htdocs>

2. พิมพ์คำสั่ง composer create-project --prefer-dist laravel/laravel laravel9 "9.\*.\*" แล้วกด enter  
(laravel9 คือ ชื่อไฟล์เดอร์ที่เก็บโปรเจคของเรา สามารถตั้งชื่ออื่นได้)

3. รอสักครู่จนการติดตั้งเสร็จเรียบร้อย (โครงสร้างไฟล์เดอร์ของ Laravel หลังติดตั้ง)



4. เปิดโปรแกรม Visual Studio Code คลิกที่เมนู File -> Open Folder... จากนั้นเลือกไฟล์เดอร์ที่เราได้ติดตั้งไว้



5. เปิดไฟล์ .env เพื่อตรวจสอบความเรียบร้อยอีกครั้ง (ไฟล์ .env เป็นไฟล์สำหรับตั้งค่าสภาพแวดล้อมการทำงานของ Laravel)

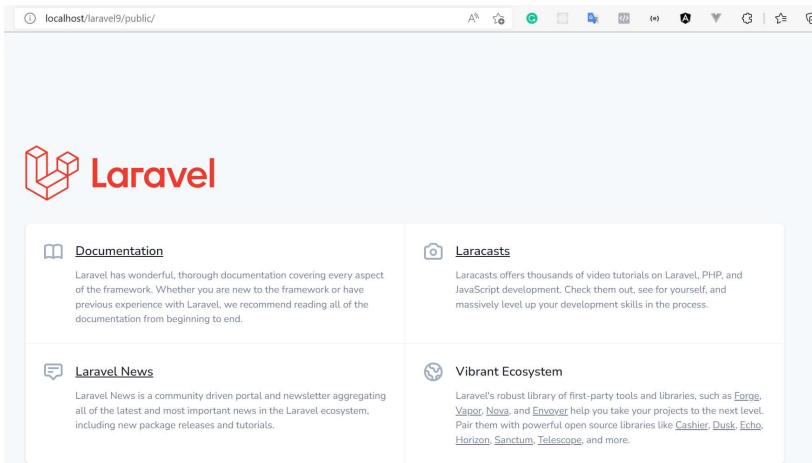
```

1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:/RF+TrMKaUmhxcoNbg0PQCpWdrs1GU6
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=laravel
15 DB_USERNAME=root
16 DB_PASSWORD=
17

```

Note: ไฟล์ .env หากใครเคยใช้ MySQL/MariaDB แล้ว สามารถกรอกรายละเอียดการติดต่อกับข้อมูลฐานข้อมูล ได้ตั้งแต่บรรทัดที่ 11 ถึงบรรทัดที่ 16

6. ทดสอบ Laravel ผ่าน Browser โดยพิมพ์ URL ดังนี้ <http://localhost/laravel9/public/> แล้วก็ติดตั้ง Laravel เรียบร้อย



## การตั้งค่าระบบของ Laravel

หลังจากการติดตั้งแล้ว เราควรทำความสะอาดรู้จักกับการตั้งค่าต่างๆของ Laravel กันก่อน โดยให้เปิดไฟล์เดอร์ config การตั้งค่าสำคัญๆ ได้แก่

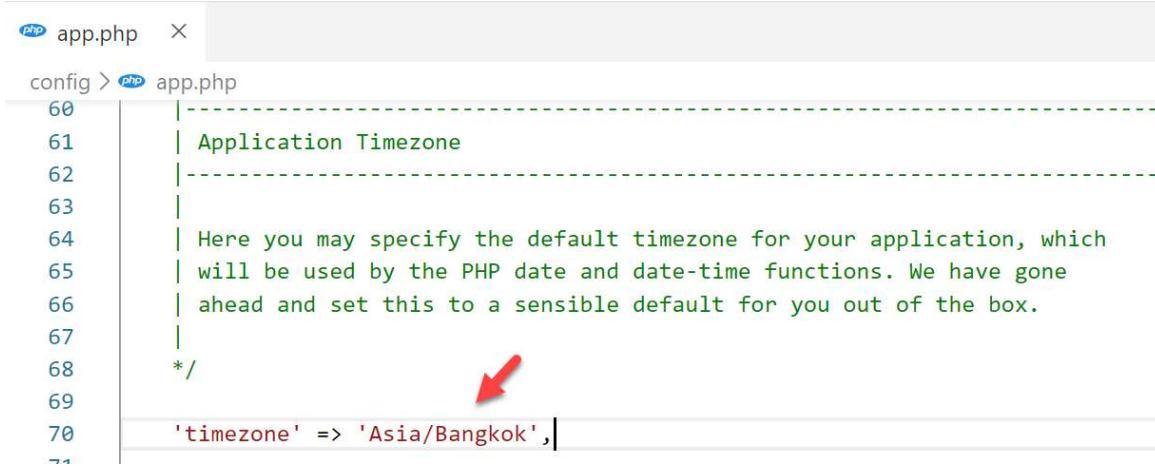
- **app.php:** เป็นรายละเอียดการตั้งค่าภาพรวมของระบบเราทั้งหมด เช่น กำหนดการเปิด-ปิด ของ Debug Mode, การกำหนด timezone ให้กับ Web Application เป็นต้น แนะนำเรื่องอยู่ในประเทศไทย ก็ควรกำหนดเป็น 'timezone' => 'Asia/Bangkok'
- **auth.php:** เป็นรายละเอียดการตั้งค่าเกี่ยวกับการล็อกอิน การรับรอง หรือตรวจสอบผู้ใช้ เช่น การกำหนดตารางผู้ใช้ในฐานข้อมูล, การตั้งค่าเกี่ยวกับการ reset รหัสผ่าน เป็นต้น
- **cache.php:** รายละเอียดการตั้งค่าของ cache โดย Laravel รองรับประเภท cache ได้หลายตัว ได้แก่ filesystem, database, memcached, redis เป็นต้น โดยปกติ Laravel จะกำหนดค่าปริยาย (default) เป็น filesystem
- **database.php:** รายละเอียดการตั้งค่าเกี่ยวกับฐานข้อมูลต่างๆ เช่น กำหนดการเชื่อมต่อให้กับฐานข้อมูล เป็นต้น หลังจากที่เราติดตั้ง Laravel แล้ว ค่าการเชื่อมต่อ default จะเป็น MySQL/MariaDB การตั้งค่าการเชื่อมต่อแนะนำให้กำหนดที่ไฟล์ .env ในส่วนของ DB\_CONNECTION และอื่นๆ
- **filesystems.php:** รายละเอียดการตั้งค่าและกำหนดปลายทางของระบบไฟล์ในโปรเจกของเรา เช่น การจัดการกับไฟล์เมื่อเราอัปโหลดไฟล์ต่างๆ เป็นต้น โดยรองรับทั้งแบบ local disk หรือจะใช้ Amazon S3 ก็ได้ เช่นเดียวกัน
- **mail.php:** รายละเอียดการตั้งค่าการสำหรับการส่งอีเมลของระบบว่าเราจะใช้ driver รูปแบบไหน รองรับได้หลากหลาย ได้แก่ smtp, mail, sendmail, mailgun, mandrill,ses, sparkpost และ log
- **services.php:** รายละเอียดการตั้งค่าและกำหนดบริการของ third-party ต่างๆ เช่น Stripe ใช้เป็น gateway สำหรับจ่ายเงินร้านค้าออนไลน์, การส่งอีเมล เป็นต้น
- **session.php:** รายละเอียดการตั้งค่าระบบ Sessions ของ PHP โดยสามารถกำหนดได้หลายแบบ ได้แก่ file, cookie, database, apc,memcached, redis และ array
- **view.php:** รายละเอียดการตั้งค่าที่อยู่หรือ path สำหรับ view ใน Laravel

Note: การตั้งค่าไฟล์ทุกไฟล์ในโฟลเดอร์ config นั้น หากบรรทัดใดมีคำว่า env อยู่นั่นแปลว่า เราควรกำหนดค่าพกนี้ที่ไฟล์ .env

## การตั้งค่า timezone

สิ่งแรกที่ควรทำหลังการติดตั้งอันต่อมาคือ การตั้งค่าวันที่และเวลาให้ถูกต้องกับ timezone ของประเทศไทย มีขั้นตอนดังนี้

- เปิดไฟล์ config/app.php และแก้ไขค่า timezone จาก UTC เป็น Asia/Bangkok



```

  config > app.php
  config > app.php
  60 |-----|
  61 | Application Timezone
  62 |-----|
  63 |
  64 | Here you may specify the default timezone for your application, which
  65 | will be used by the PHP date and date-time functions. We have gone
  66 | ahead and set this to a sensible default for you out of the box.
  67 |
  68 */
  69
  70 'timezone' => 'Asia/Bangkok',
  71
  
```

- จากนั้นให้ทดสอบเชื่อมต่อเพื่อแสดงวันที่และเวลาว่าถูกต้องหรือไม่ โดยให้เปิดไฟล์ resources/views/welcome.blade.php และเขียนโค้ดเพื่อแสดงวันที่และเวลาปัจจุบัน เสร็จแล้วบันทึกไฟล์ และลองรันดูว่าวันที่และเวลาถูกต้องหรือไม่



```

  welcome.blade.php
  Source History
  79
  80     <div class="content">
  81         <div class="title m-b-md">
  82             Laravel
  83         </div>
  84
  85         <div class="title m-b-md">
  86             {{ date("d/m/Y H:i:s") }}
  87         </div>
  88
  89     <div class="links">
  90         <a href="https://laravel.com/docs">Documentation</a>
  
```

## การ Debugging ใน Laravel

การ Debug โค้ดใน Laravel เราสามารถทำได้หลายวิธีทั้งในรูปแบบของฟังก์ชัน และเครื่องมืออำนวยความสะดวก ดังนี้

- การใช้ฟังก์ชัน dd()

เราสามารถใช้ฟังก์ชัน dd() ในการ debug โค้ดได้ โดยส่วนใหญ่จะใช้ debug ตัวแปรต่างๆ เช่น dd(\$var) ข้อดีของการใช้ฟังก์ชัน dd() คือ ระบบจะหยุดหรือจบการทำงานที่ฟังก์ชันนี้ทันที แต่หากไม่ต้องการก็สามารถใช้ dump() แทนได้ ตัวอย่างการใช้งาน

```
function index() {
    $items = array(
        'items' => [
            'PHP Basic',
            'PHP OOP',
            'PHP Framework'
        ]
    );
    dd($items);
    return view('welcome');
}
```

- การใช้ Laravel Logger

โดยปกติหากระบบพิ่มราชบบพิษErrors เกิดขึ้น Laravel จะสร้างและเก็บ errors เหล่านี้ไว้ในไฟล์ storage/logs/laravel.log เราสามารถเปิดดูได้โดย แต่หากต้องการ custom ข้อความเองก็สามารถทำได้โดยใช้คำสั่ง \Log::debug(\$var) นอกจากนี้เรายังสามารถกำหนดระดับหรือรูปแบบของข้อความที่ต้องการ debug ได้ด้วย ได้แก่ info, warning, error, critical ตัวอย่างการใช้งาน

```
\Log::info('ข้อความเกี่ยวกับ information');
\Log::warning('มีบางอย่างผิดปกติ');
\Log::error('เกิด errors ในส่วนนี้');
\Log::critical('ขันตราย');
```

- การใช้ Laravel Debugbar (แนะนำตัวนี้เพื่อว่าสามารถดูผ่าน Browser ได้เลย) การติดตั้งมีขั้นตอนดังนี้

Note: Laravel Debugbar สามารถเข้ามาได้ หากไม่ต้องการใช้งาน

1. เข้าเว็บ <https://github.com/barryvdh/laravel-debugbar>
2. เปิด Command Prompt แล้วพิมพ์ cd C:\xampp\htdocs\laravel7 เพื่อเข้าไปในไฟล์เดอร์โปรเจค จากนั้นพิมพ์คำสั่ง composer require barryvdh/laravel-debugbar --dev เพื่อติดตั้ง และกด enter
3. เปิด Command Prompt ขึ้นมาอีกครั้ง แล้วพิมพ์
 

```
php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"
```

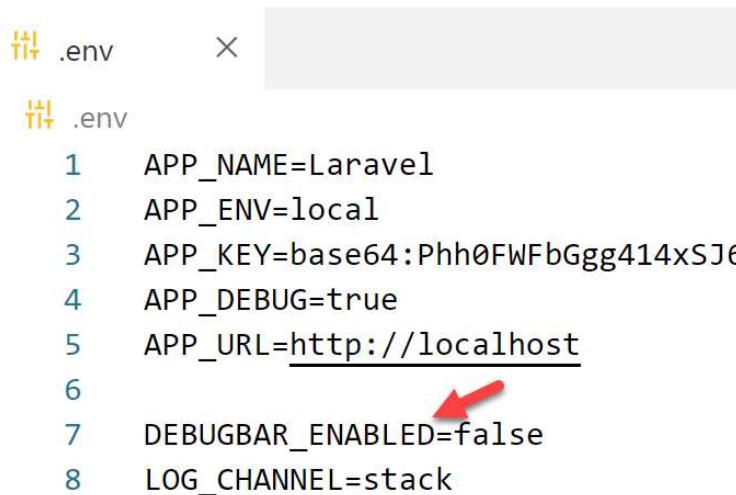
 จากนั้นกด enter เพื่อ publish และ copy ไฟล์ config ของ debugbar ไปยังไฟล์เดอร์ config ถ้าเรียบร้อยจะสังเกตเห็นว่ามีไฟล์ใหม่ชื่อว่า debugbar.php ถูกสร้างขึ้นมาครับ (ในไฟล์เดอร์ config)
4. ตรวจสอบการติดตั้ง Laravel Debugbar โดยเปิดและรีเฟรช Browser อีกครั้ง ต่อไปเราจะสามารถตรวจสอบ errors ได้สะดวกแล้ว



รายละเอียดของ Tab ต่างใน Laravel Debugbar มีดังนี้

- Messages: เข้าไปดู errors หรือข้อความต่างๆจากไฟล์ log
- Timeline: ใช้สำหรับดูเวลารวมในการโหลด page ต่างๆ
- Exceptions: ใช้สำหรับดูข้อผิดพลาด (exceptions) เมื่อเราโยน (thrown) ออกมายังระบบ
- Views: ใช้สำหรับดูรายละเอียดในการ render view รวมถึง layout ด้วย
- Route: ใช้สำหรับดูข้อมูลรายละเอียดการ requested route
- Queries: ใช้สำหรับดูรายละเอียด และรายการของ SQL queries ที่กำลังประมวลอยู่
- Mails: ใช้สำหรับดูรายละเอียดเกี่ยวกับการส่งอีเมล
- Request: ใช้สำหรับดูข้อมูลการ request รวมถึง status code, request headers เป็นต้น

หมายเหตุ หากเราไม่ต้องการใช้งาน Debugbar และ สามารถปิดการใช้งาน Debugbar ได้โดยให้แก้ไฟล์ .env โดยกำหนดค่า DEBUGBAR\_ENABLED เป็น false ดังรูป



```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:Phh0FWFbGgg414xSJ6
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 DEBUGBAR_ENABLED=false
8 LOG_CHANNEL=stack
```

## บทที่ 2 ทำความรู้จักกับ Laravel, MVC และ Best Practices

### ทำไมต้องใช้ PHP Framework

- มีการเขียนโค้ดที่เป็นมาตรฐาน ช่วยลดและกำจัดโค้ดที่ไม่จำเป็น
- ช่วยลดระยะเวลาในการทำงาน เช่น เรื่องความปลอดภัย การสร้างฟอร์ม เป็นต้น
- ช่วยทำให้การทำงานเป็นทีมง่ายขึ้น เพราะต้องเขียนโค้ดเป็นมาตรฐานเดียวกัน
- ช่วยในการบำรุงรักษาโค้ดได้ง่ายขึ้น
- มี community ที่เข้มแข็ง เรากำลังสามารถ แลกเปลี่ยนข้อความและนำได้

### ทำความรู้จักกับ Laravel

Laravel เป็น web application framework ที่มีคุณสมบัติที่ช่วยให้เราเขียน web application ได้ง่ายขึ้น มีคุณสมบัติครบถ้วน มีจุดเด่นตรง การเขียนโค้ดสั้น กระชับ และยังเหมาะสมกับการทำงานร่วมกับด้าน front-end เป็นอย่างมาก

### โครงสร้างของ Laravel ที่สำคัญ

โครงสร้างแต่ละไฟล์เดอร์ของ Laravel มีดังนี้

```

./app/                      # เก็บไฟล์เดอร์ laravel ของเรา
./app/Models                 # เก็บไฟล์ Models
./app/Console/               # เก็บ Commands classes ต่างๆ
./app/Exceptions/
./app/Http/
./app/Http/Controllers/     # เก็บไฟล์ controllers
./app/Http/Middleware/      # เก็บ Filters ต่างๆสำหรับใช้งานกับ requests
./app/Providers              # เก็บคลาส Service provider

./bootstrap/                 # จุดเริ่มต้นของระบบ bootstrapping scripts

./config/                    # เก็บไฟล์การตั้งค่าระบบ

./database/
./database/migrations/       # เก็บส่วนของ Database migration classes
./database/seeds/            # เก็บ Database seeder classes

./public/                     # ไฟล์เดอร์ document root

```

```

./public/.htaccess
./public/index.php          # จุดเริ่มต้นของ Laravel application

./resources/
./resources/sass/           # เก็บไฟล์ของ Sass
./resources/lang/            # เก็บไฟล์ Localization และภาษาต่างๆ
./resources/views/           # เก็บไฟล์ Templates สำหรับ render เป็น html

./storage/
./storage/app/              # เก็บไฟล์ต่างๆของระบบ เช่น ไฟล์ที่ถูกอัปโหลดมาจากฟอร์ม เป็นต้น
./storage/framework/         # เก็บ cache ต่างๆ
./storage/logs/              # เก็บ logs

./tests/                     # เก็บไฟล์สำหรับเขียน Test cases
./vendor/                     # เก็บ Third-party ต่างๆที่ติดตั้งด้วย Composer
./env.example                 # ตัวอย่างไฟล์ env
./artisan                     # Artisan command-line utility
./composer.json               # Project dependencies manifest
./phpunit.xml                 # ตั้งค่า PHPUnit สำหรับ running tests
./server.php                  # development server

```

## MVC และ Best Practices

รูปแบบการเขียนแบบ MVC (Model, View, Controller) นั้น การจะเขียนให้ดี ต้องศึกษาแนวทางกันก่อนที่ได้กันก่อน สุบไปหัดังนี้

### สรุปการเขียน Model ที่ดี

- ประกอบด้วย โค้ดในส่วน business data
- ประกอบด้วย โค้ดในการส่วนของการตรวจสอบความถูกต้องของข้อมูล
- ประกอบ ด้วย เมธอด การทำงานในส่วนของ business logic
- อย่าเขียนโดยเดียวกับการ request, session หรือโดยเดียวกับสภาพแวดล้อมของระบบ
- ระวังหรือหลีกเลี่ยงเขียนโดยเดียวกับ HTML ในส่วนของการแสดงผลใน Model ให้ไปเขียนที่ view แทน

### สรุปการเขียน View ที่ดี

- View จะต้องมีโค้ดเฉพาะ HTML และ PHP ที่เกี่ยวข้องกับการแสดงผล จัดรูปแบบข้อมูลต่างๆเท่านั้น
- จะต้องไม่ได้เกี่ยวกับการ query ฐานข้อมูลต่างๆ
- หลักเดียวของการรับค่า \$\_GET, \$\_POST เพราะเป็นหน้าที่ของ Controller

- ถ้าเรารับค่ามาจากการ request จะต้องไม่ไปแก้ไขค่าที่รับมา
- ใช้คลาสในกลุ่ม Helper เพื่อช่วยในการจัดรูปแบบข้อมูล

### สรุปการเขียน Controller ที่ดี

- มีไวยภาพในการรับ request ข้อมูล
- มีไวยภาพในการจัดรูปแบบข้อมูล
- มีไวยภาพในการจัดรูปแบบผลลัพธ์
- ไม่ควรมีโค้ดการประมวลผลของ Models ถ้ามีให้ไปเขียนที่ Models ดีกว่า
- หลีกเลี่ยงการเขียน HTML และโค้ดที่เกี่ยวข้องกับการแสดงผลข้อมูล ให้ไปเขียนที่ view ดีกว่า

## บทที่ 3 การเขียนและใช้งาน Controllers, Routes, Layout, Views

พื้นฐานการเขียน Controllers, Routes, Views และการส่งค่าของตัวแปรไปแสดงผลที่ Views

สำหรับ Laravel นั้นมีรูปแบบ หรือ paradigm ที่เรียกว่า Model-View-Controller หรือ MVC เพราะฉะนั้นพื้นฐานสำคัญอย่างหนึ่งคือ การสร้าง Controllers, การสร้าง routes และส่งค่าข้อมูลหรือตัวแปรไปแสดงผลที่ Views สำหรับการสร้าง Controllers นั้น Laravel จะมีเครื่องมือช่วยเรา เรียกว่า artisan (command-line) และเพื่อให้ทุกคนมีพื้นฐาน และความเข้าใจกระบวนการทำงานอันนี้ เราจะมาสร้างหน้าเว็บกัน 1 หน้า ได้แก่ หน้าเพจเกี่ยวกับเรา (about) มีขั้นตอนดังนี้

1. เปิดไฟล์ routes/web.php เพื่อสร้างเส้นทาง หรือให้มองว่าเป็น URL ก็ได้ครับ เอียนโค้ดดังนี้

```
Route::get('about', 'SiteController@index');
```



```
routes > web.php
  9 |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15 Route::get('about', 'SiteController@index');
16
17 Route::get('/', function () {
18     return view('welcome');
19 });
20
```

อธิบายโค้ด ในการสร้าง route เราจะให้ชี้ไปยัง Controller ชื่อว่า SiteController และให้ไปทำงานที่ action หรือ เมธอด ชื่อว่า index()

2. เปิด Command Prompt cd เข้าไปที่โฟลเดอร์โปรเจค (cd C:\xampp\htdocs\laravel7) จากนั้น เพื่อพิมพ์คำสั่งสำหรับสร้าง Controller ดังนี้

```
php artisan make:controller SiteController
```

Administrator: Command Prompt

```
C:\xampp\htdocs\laravel7>php artisan make:controller SiteController
Controller created successfully.
```

อธิบาย การสร้าง Controller ในเมจจะใช้คำสั่ง make:controller ตามด้วยชื่อของ controller ที่ต้องการสร้าง (การตั้งชื่อแนะนำให้ขึ้นต้นด้วยตัวพิมพ์ใหญ่และตามด้วยคำว่า Controller)

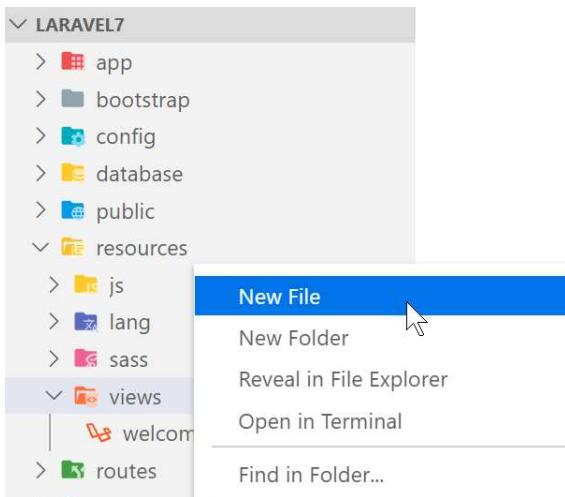
Note: หากต้องการศึกษาคำสั่งทั้งหมดของ artisan ให้พิมพ์ php artisan และกด enter และถ้าหากต้องการรู้วิธีการใช้ในแต่ละคำสั่งให้พิมพ์ --help ต่อท้าย เช่น php artisan make:controller --help

- ไฟล์ SiteController.php จะถูกสร้างที่ไฟลเดอร์ app\Http\Controllers ให้เปิดไฟล์ SiteController.php แล้วเขียน เมนูดู ชื่อว่า index ดังนี้ (เมนูดู ที่ตั้งขึ้นมาจะต้องสอดคล้องกับการเขียน route ในข้อ 1)

```
SiteController.php ×
app > Http > Controllers > SiteController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class SiteController extends Controller
8  {
9      public function index()
10     {
11         return view('about');
12     }
}
```

- จะเห็นว่าตอนนี้เราได้สร้าง route กับ controller เรียบร้อยแล้ว ถ้าสังเกตใน เมนูดู index() จะเห็นว่าเราได้เขียนได้ดีเพื่อสั่งให้ render ไปที่ view ชื่อว่า about นั่นเอง การสร้างไฟล์ view นั้นเราสามารถสร้างไฟล์ได้ที่ไฟลเดอร์ resources\views จะคลิกขวาท

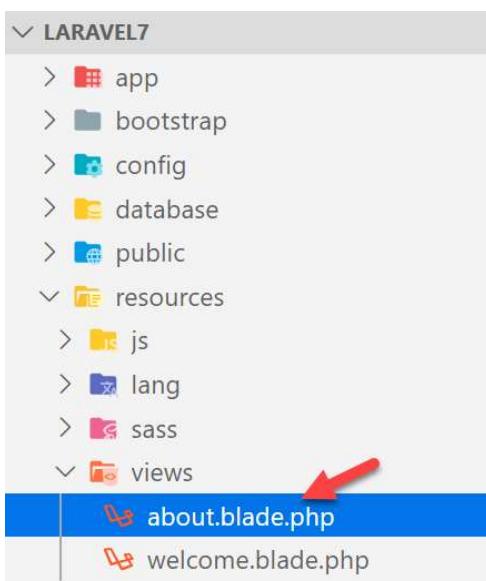
ที่ไฟล์เดอร์ view เพื่อสร้างไฟล์ใหม่ๆได้ ดังรูป



Note: ในไฟล์เดอร์ views นี้เราสามารถสร้างไฟล์เดอร์ชื่อなんก็ได้ครับ หากมีไฟล์เดอร์ชื่อなんก็ให้แก้ไขเด็ดใน Controller

เช่น return view('site.about') หมายถึง ข้างไฟล์ about.blade.php ซึ่งอยู่ในไฟล์เดอร์ site

5. เสร็จแล้วตั้งชื่อ views ให้พิมพ์ about.blade.php (การตั้งชื่อ views ให้พิมพ์ตามด้วย .blade.php เช่นๆ)



6. ทดสอบแก้ไขไฟล์ about.blade.php ดังนี้

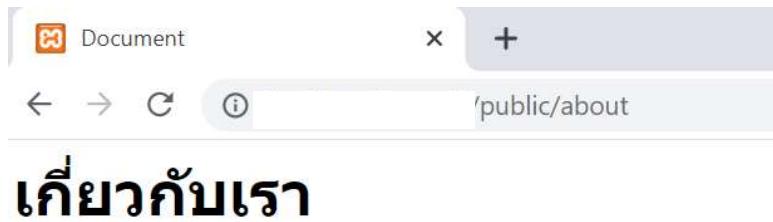


```

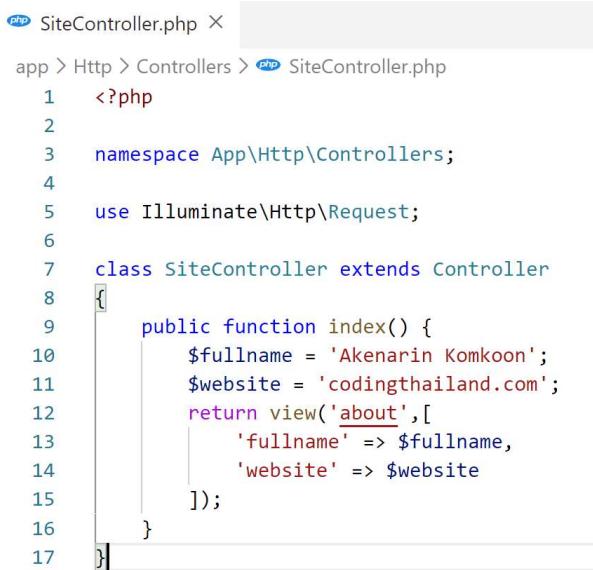
about.blade.php X
resources > views > about.blade.php > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Document</title>
8  </head>
9  <body>
10     <h1>เกี่ยวกับเรา</h1>
11     <p>
12         codingthailand.com
13     </p>
14 </body>
15 </html>

```

7. บันทึกไฟล์แล้วพิมพ์ URL เพื่อทดสอบการทำงาน ดังนี้ <http://localhost/laravel9/public/about>



8. ต่อมา หากเราต้องการส่งข้อมูล (ตัวแปร) มาแสดงผลที่ views สามารถเขียนเพิ่มเติม เมนูดู ชื่อว่า index ดังนี้



```

SiteController.php X
app > Http > Controllers > SiteController.php
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class SiteController extends Controller
8  {
9      public function index() {
10          $fullname = 'Akenarin Komkoon';
11          $website = 'codingthailand.com';
12          return view('about',[
13              'fullname' => $fullname,
14              'website' => $website
15          ]);
16      }
17  }

```

อธิบายโดยดีคือ เราสามารถส่งตัวแปร และข้อมูลที่ต้องการเพื่อไปแสดงผลในหน้า View ได้ หากมีตัวแปรที่ต้องการส่งหลายตัว (Array) ก็ให้คั่นด้วยเครื่องหมายคอมม่า โดยในตัวอย่างจะส่งตัวแปร \$fullname และ \$website เพื่อไปแสดงผลที่หน้า View (about.blade.php)

9. ต่อมาให้ลองแสดงค่าข้อมูลของตัวแปรที่ส่งมาได้แก่ \$fullname และ \$website โดยให้แก้ไขไฟล์ about.blade.php ดังนี้

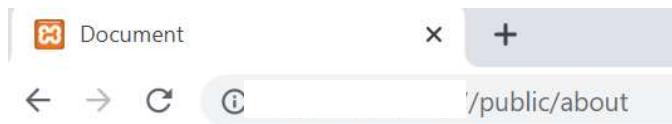


```

about.blade.php ×
resources > views > about.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=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Document</title>
8  </head>
9  <body>
10     <h1>{{ $fullname }}</h1>
11     <p>
12         <a href="https://{{ $website }}"> {{ $website }} </a>
13     </p>
14 </body>
15 </html>

```

10. บันทึกไฟล์ทั้งหมดแล้วรันอีกครั้ง <http://localhost/laravel9/public/about>



## Akenarin Komkoon

[codingthailand.com](http://localhost/laravel9/public/about)

เพียงเท่านี้เราก็สามารถสร้างหน้าเพจ และส่งข้อมูลจาก Controller ไปให้ที่ View ได้เรียบร้อย 😊

Note: ขั้นตอนการเขียน route การสร้าง Controller การสร้าง View และการส่งตัวแปรไปแสดงผลที่ Views ควรฝึกและทำความเข้าใจส่วนนี้เยอะๆ เพราะเป็นพื้นฐานสำคัญและได้ใช้บ่อยมาก อาจทดลองโดยการสร้างหน้าเพจอื่นโดยไม่ต้องอ่านหนังสือ

## การสร้างไฟล์ และการจัดการ Layout

ในกรณีที่เราสร้างหน้าเพจ (View) แล้วมีโค้ด HTML ซ้ำๆ กันในแต่ละหน้า แนะนำให้แยกออกมายเป็นไฟล์ layout ต่างหากดีกว่า เพราะเวลาแก้ไขโค้ดจะได้ไม่ต้องตามเปลี่ยนแก้ที่ลับไฟล์ หากเราใช้ layout เราก็สามารถแก้ไขโค้ดได้เพียงจุดเดียว ทำให้ทุกหน้าที่เรียก layout นั้นๆ เปลี่ยนตามที่แก้ไขในทันที หากเทียบกับการเขียน PHP ปกติ ก็เทียบได้กับคำสั่ง include หรือ require นั่นเอง

ในหนังสือเล่มนี้ เราจะใช้ Bootstrap ซึ่งเป็น CSS Framework ที่ได้รับความนิยม และทำให้โปรเจคของเราสามารถแสดงผลแบบ Responsive ได้โดย ประดิษฐ์ตัวเอง ตั้งแต่ Laravel เวอร์ชัน 5.2 เป็นต้นมา จะมีการสร้างโค้ดอัตโนมัติให้ในส่วนของการล็อกอินเข้าใช้งานระบบ และตรงนี้เองเราไม่ต้องสร้าง layout เองเลย Laravel จะจัดการให้ มาดูขั้นตอน การสร้างระบบล็อกอิน กัน ซึ่งแน่นอนเราจะได้ไฟล์ layout เป็นต้นมาด้วย

1. เข้าไปรูปแบบของเราแล้วเปิด Command Prompt พิมพ์คำสั่ง เพื่อติดตั้ง laravel/ui ด้วยคำสั่ง

```
C:\Administrator: Command Prompt - composer require laravel/ui
```

```
C:\xampp\htdocs\laravel>composer require laravel/ui
```

2. เข้าไปรูปแบบของเราแล้วเปิด Command Prompt พิมพ์คำสั่ง php artisan ui bootstrap --auth และกด Enter

```
C:\Administrator: Command Prompt
```

```
C:\xampp\htdocs\laravel>php artisan ui bootstrap --auth ←  
Bootstrap scaffolding installed successfully.  
Please run "npm install && npm run dev" to compile your fresh scaffolding.  
Authentication scaffolding generated successfully.
```

ต่อด้วยพิมพ์คำสั่ง npm install && npm run dev เพื่อ compile โค้ด laravel/ui

```
C:\Administrator: Command Prompt
```

```
C:\xampp\htdocs\laravel>npm install && npm run dev -
```

3. ต่อด้วยคำสั่ง npm install resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps และคำสั่ง npm run dev อีกครั้ง
4. จากนั้น Laravel จะสร้างโค้ดอัตโนมัติให้เราทั้งในส่วนของ views , HomeController.php และเพิ่มโค้ดในไฟล์ routes\web.php ให้ด้วย และแน่นอนจะมีการสร้างโฟลเดอร์และไฟล์ layout ที่เป็น Bootstrap Framework มาให้เลย ที่ไฟล์ app\resources\views\layouts\app.blade.php

```

routes > web.php
routes > web.php
9 | Here is where you can register web routes for your application. These
10 | routes are loaded by the RouteServiceProvider within a group which
11 | contains the "web" middleware group. Now create something great!
12 |
13 |
14 */
15 Route::get('about','SiteController@index');
16
17 Route::get('/', function () {
18 |     return view('welcome');
19 });
20
21 Auth::routes();
22
23 Route::get('/home', 'HomeController@index')->name('home');
24

```

5. ลองทดสอบและเปิดใน Browser <http://localhost/laravel9/public/> ສังเกตว่าจะมีเมนู Login และเมนู Register มาให้แล้ว!

Note: ตอนนี้ยังไม่สามารถล็อกอินหรือลงทะเบียนได้ เพราะเรา还没有สร้างตารางในฐานข้อมูลซึ่งจะกล่าวถึงในบทต่อๆไป

6. ตอนนี้เราได้ไฟล์ layout มาเรียบร้อยนั่นคือไฟล์ app.blade.php (app\resources\views\layouts\app.blade.php) ลองเปิดแล้วลงแก้ไขเมนูต่างๆได้ ต่อไปหากเราต้องการเพิ่มเมนูต่างๆ ก็สามารถแก้ไขและเรียกใช้ layout นี้ได้เลยครับ



```

    </ul>
  38
  39      <!-- Right Side Of Navbar -->
  40      <ul class="navbar-nav ml-auto">
  41          <!-- Authentication Links -->
  42          @guest
  43              <li class="nav-item">
  44                  <a class="nav-link" href="{{ route('login') }}">เข้าระบบ</a>
  45              </li>
  46              @if (Route::has('register'))
  47                  <li class="nav-item">
  48                      <a class="nav-link" href="{{ route('register') }}">ลงทะเบียน</a>
  49                  </li>
  50              @endif
  51          @else

```

Note: ลองเปิดไฟล์ views ที่เกี่ยวกับระบบดีอกรอนได้ที่ในไฟล์เดอร์ app\resources\views\auth\ จากนั้นให้ลองเปิดไฟล์แต่ละไฟล์แล้วแก้ไขความเป็นภาษาไทยดูครับ

## การเรียกใช้ Layout ใน Laravel

การเรียกใช้ Layout ใน Laravel นั้น ไฟล์ที่เรียกจะต้องใช้คำสั่ง `@extends('ชื่อไฟล์ layout ที่ต้องการ')` วางไว้ตាำแห่งบันสุดของไฟล์ ส่วนเนื้อหาจะใช้คำสั่ง `@section('ชื่อที่ตั้งขึ้นจาก @yield ในไฟล์ layout')` และปิดท้ายด้วย `@endsection` เราสามารถปรับหน้าเกี่ยวกับเราที่เคยสร้างไว้แล้วให้เรียกใช้ layout ดูครับ

1. เปิดไฟล์ app\resources\views\about.blade.php แก้ไขโค้ด ดังนี้

```

about.blade.php X
resources > views > about.blade.php > ...
1   @extends('layouts.app')
2
3   @section('content')
4   <div class="container">
5       <div class="row justify-content-center">
6           <div class="col-md-8">
7               <div class="card">
8                   <div class="card-header">เกี่ยวกับเรา</div>
9
10                  <div class="card-body">
11
12                      </div>
13                  </div>
14              </div>
15          </div>
16      </div>
17  @endsection

```

2. ต่อมา เพื่อความสะดวกให้เราสร้างเมนูเพื่อลิงก์มาที่ about ด้วย ให้เปิดไฟล์ app\resources\layouts\app.blade.php แล้วเพิ่มโค้ด ดังนี้

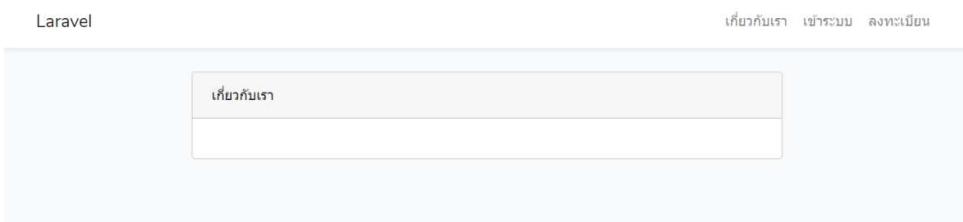
```

app.blade.php X
<!-- Authentication Links -->
@guest
    <li class="nav-item">
        <a class="nav-link" href="{{ route('about') }}">เกี่ยวกับเรา</a>
    </li>
<li class="nav-item">
    <a class="nav-link" href="{{ route('login') }}">เข้าระบบ</a>
</li>
@if (Route::has('register'))
    <li class="nav-item">
        <a class="nav-link" href="{{ route('register') }}>ลงทะเบียน</a>
    </li>
@endif

```

Note: หากต้องการเรียกโค้ดทำลิงก์เมนูเข้าระบบ หรือลงทะเบียน เช่น {{ route('about') }} รีบตั้งเดียวกัน โดยให้เปิดไฟล์ routes\web.php และตั้งชื่อ route ตามนี้ Route::get('/about','SiteController@index')->name('about');

3. เปิด Browser และลองคลิกเมนู เกี่ยวกับเรา ก็เป็นอันเสร็จเรียบปั้บอยครับ สำหรับการนำ layout เข้ามาใช้งาน



Note: การ extend layout มาใช้ด้วยคำสั่ง `@extends('layouts.app')` โดย `layouts.app` หมายถึง การอ้างอิงไฟล์ `app.blade.php` ซึ่งอยู่ในโฟลเดอร์ `layouts` เพื่อระบุชั้นหน้าหากเราสร้างไฟล์ `layout` เองก็อย่าลืมอ้างอิงให้ถูกต้องด้วย

### การสร้าง Section ใหม่โดยใช้ `@yield`

หากเราต้องการสร้าง Section ในหน้า view ใดๆ ที่ไมาระบุให้ layout สามารถกำหนดได้โดยใช้คำสั่ง `@yield('ชื่อที่ตั้งขึ้นมา')` เช่น เราอาจสร้าง `@yield('footer')` ในไฟล์ layout หากหน้า view ใดมีการแทรก JavaScript ก็สามารถเรียกใช้ตรงนี้ได้ การเรียกใช้ก็แค่ใช้คำสั่ง `@section('footer')` และปิดท้ายด้วย `@endsection` มาลองสร้างกันดู

1. เปิดไฟล์ `app\resources\views\layouts\app.blade.php` เขียนโค้ด `@yield('footer')` ไว้ในจุดที่ต้องการ ในตัวอย่างนี้จะกำหนดไว้ล่างสุดหลังโค้ด JavaScript ต่างๆ

```

app.blade.php ×
resources > views > layouts > app.blade.php > html
73     |           |
74     |           |
75     |           |
76     |           |
77     |           |
78     |           |
79     |           |
80     |           |
81     |           |
82     |           |
83     |           |
84     |           |
85     |           |
86     |           |
87     |           |

```

```

73         |           |
74         |           |
75         |           |
76         |           |
77         |           |
78         |           |
79         |           |
80         |           |
81         |           |
82         |           |
83         |           |
84         |           |
85         |           |
86         |           |
87         |           |

```

เปิดไฟล์ views ได้ๆที่ต้องการเรียกใช้ ในที่นี่จะยกตัวอย่างไฟล์ about.blade.php การเรียกใช้ ก็ให้เพิ่มคำสั่ง @section('content') และปิดท้ายด้วย @endsection หากเราต้องการเขียนโค้ด JavaScript ก็สามารถเขียนตรงได้เลยครับ



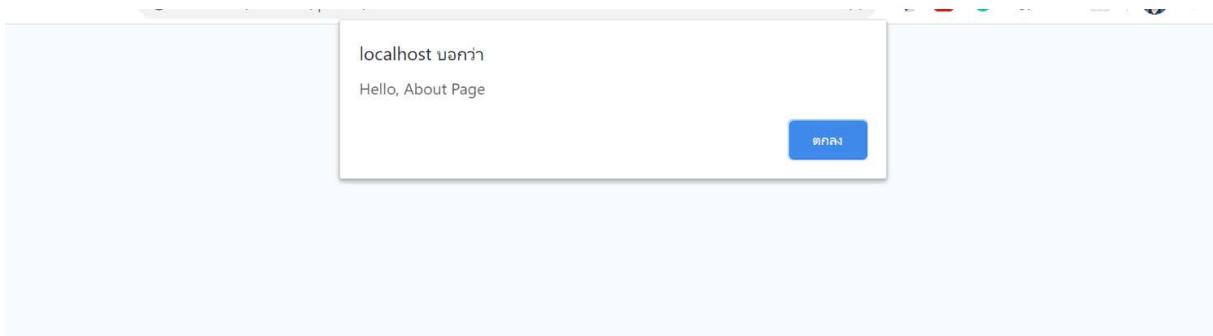
```

about.blade.php ×

resources > views > about.blade.php > ...
7   <div class="card">
8     <div class="card-header">เกี่ยวกับเรา</div>
9
10    <div class="card-body">
11      </div>
12    </div>
13  </div>
14  </div>
15  </div>
16 </div>
17 @endsection
18
19 @section('footer')
20
21 <script>
22   |   alert("Hello, About Page");
23 </script>
24
25 @endsection

```

- ลองทดสอบบันคุณจะพบว่าโค้ด JavaScript บรรทัดนี้ จะมีการทำงานเฉพาะหน้าที่เรียกใช้เท่านั้น ไม่กระทบกับหน้าอื่นๆเลย



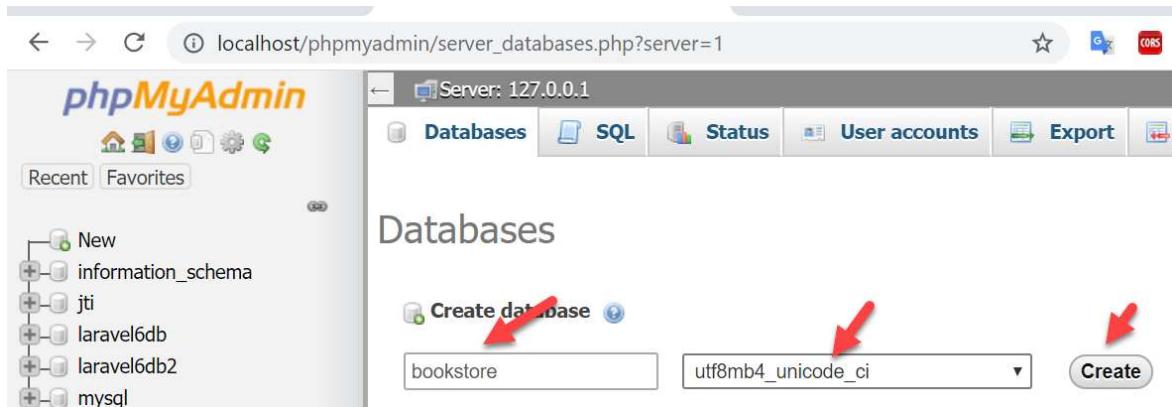
## บทที่ 4 ออกรูปแบบฐานข้อมูลและตารางด้วย Artisan, Database Migrations และการทำ Seeding

ใน Laravel มีคุณสมบัติที่ช่วยให้เราออกแบบและเขียนโค้ดเพื่อกำหนดโครงสร้างของตารางในฐานข้อมูลได้ เรียกว่า Database Migrations เราสามารถใช้ artisan ช่วยในการรันคำสั่งสร้างตาราง (table) ได้เลย นอกจากนั้นเรายังสามารถกำหนดข้อมูลเริ่มต้นของตารางต่างๆ ได้ เรียกว่า การทำ Seeding

### การตั้งค่าฐานข้อมูล

การตั้งค่าฐานข้อมูลเป็นสิ่งที่ควรกำหนดโดย หากเรามีการใช้งานฐานข้อมูลในระบบ เพราะถ้าไม่ตั้งค่า Laravel จะไม่สามารถติดต่อฐานข้อมูลได้ หากเราใช้ MySQL/MariaDB สามารถกำหนดผู้ใช้, รหัสผ่านผู้ใช้, และฐานข้อมูลได้ ในไฟล์ชื่อว่า .env

1. เปิดโปรแกรม phpMyAdmin เปิด Browser และพิมพ์ <http://localhost/phpmyadmin> เพื่อสร้างฐานข้อมูลใหม่ ในหน้าสือเล่นนี้ เราชำใช้ฐานข้อมูลชื่อว่า bookstore พิมพ์ชื่อฐานข้อมูล และกด Create



2. เปิดไฟล์ .env และกรอกข้อมูล ชื่อฐานข้อมูล, ชื่อผู้ใช้, รหัสผ่าน ดังนี้

```
nv ×
env
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=bookstore
DB_USERNAME=root
DB_PASSWORD=
```

Note: ในส่วนของ DB\_PASSWORD โปรแกรม XAMPP จะไม่ได้กำหนดรหัสผ่านมาให้จึงต้องใส่ แต่หากระบบเรามีชื่อผู้ใช้ หรือรหัสผ่านก็อย่าลืมกรอกข้อมูลให้ครบถ้วน

## การสร้างตารางฐานข้อมูลด้วย Migration

หลังจากที่ตั้งค่าฐานข้อมูลเรียบร้อยแล้ว เราจะลองสร้างตารางในฐานข้อมูลโดยใช้ Database Migration ซึ่งต้องใช้คำสั่ง command-line ด้วย artisan นั้นเอง คำสั่งพื้นฐานที่เกี่ยวข้องกับการทำ Database Migration มีดังนี้

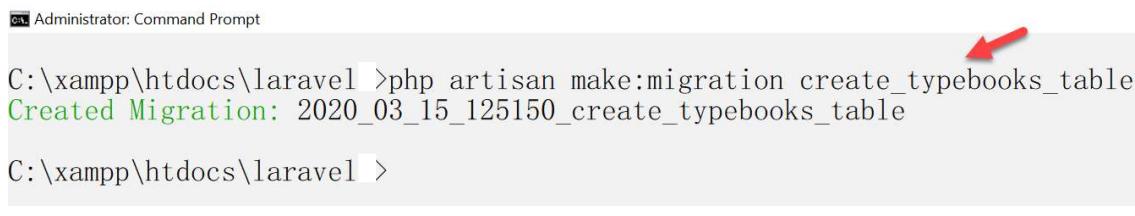
- `php artisan make:migration <ชื่อคลาส>` เป็นคำสั่งสำหรับสร้างไฟล์ Migration ซึ่งต้องระบุชื่อคลาสด้วย
- `php artisan migrate:install` เป็นคำสั่งสำหรับสร้างตาราง migrations ในฐานข้อมูล
- `php artisan migrate` เป็นคำสั่งสำหรับรัน migration
- `php artisan migrate:refresh` เป็นคำสั่งให้ rollback ทั้งหมด และสั่งรัน migrate ใหม่อีกครั้ง
- `php artisan migrate:rollback` เป็นคำสั่งสำหรับใช้ undo การทำงานก่อนหน้านั้น
- `php artisan migrate:fresh` เป็นคำสั่งสำหรับลบตารางทั้งหมด และสั่ง migrate ใหม่อีกครั้ง

**Note:** การใช้งาน Migration ควรออกแบบฐานข้อมูลให้เสร็จเสียก่อนจะได้ไม่เสียเวลา ตามว่าไม่ต้องใช้ migration ได้หรือเปล่า คำตอบคือ ได้ ขึ้นกับเรา อาจออกแบบด้วย phpMyAdmin แบบปกติได้เช่นเดียวกัน

หลังจากเรียนรู้คำสั่งเกี่ยวกับ Migration แล้ว มาลองสร้างตารางกันได้เลย โดยเราจะสร้าง 2 ตาราง ได้แก่ typebooks (ประเภทหนังสือ) และ ตาราง books (หนังสือ) ส่วนตารางเกี่ยวกับการล็อกอินและผู้ใช้นั้น Laravel สร้างมาให้เราเรียบร้อยแล้ว

- สร้างไฟล์ migration ในม'(ตาราง typebooks) เข้าไปที่ไฟล์เดอร์ประจุของเรา แล้วพิมพ์คำสั่งดังนี้

```
php artisan make:migration create_typebooks_table
```



```
C:\xampp\htdocs\laravel >php artisan make:migration create_typebooks_table
Created Migration: 2020_03_15_125150_create_typebooks_table

C:\xampp\htdocs\laravel >
```

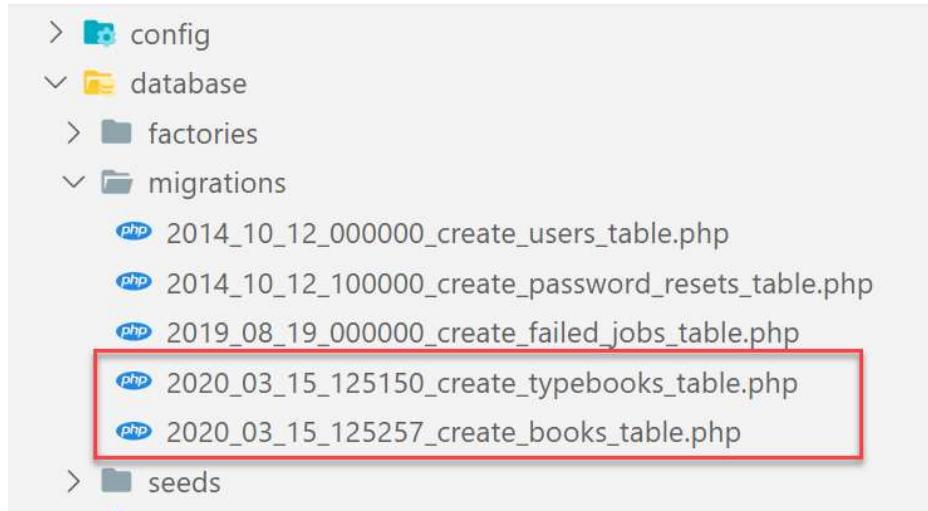
**Note:** หากมีการใช้งาน foreign key (FK) ควรสร้างตาราง parent หรือ master ก่อน

2. สร้างไฟล์ migration ในมุม (ตาราง books) พิมพ์ php artisan make:migration create\_books\_table และ enter อีกครั้ง

```
C:\xampp\htdocs\laravel >php artisan make:migration create_books_table
Created Migration: 2020_03_15_125257_create_books_table

C:\xampp\htdocs\laravel >
```

3. เมื่อเราสร้างไฟล์ migration ไฟล์ทั้งหมดสามารถตรวจสอบได้ที่โฟลเดอร์ database\migrations



Note: ตัวเลขด้านหน้าคือวันที่และเวลาที่สร้างขึ้นมา ไม่มีผลกระทบต่อโค้ดอะไร ซึ่งแต่ละคนจะไม่เหมือนกัน

4. เปิดไฟล์ xxx\_create\_typebooks\_table.php เพื่อเขียนโค้ดในการสร้างโครงสร้างของตาราง โดยต้องสร้าง table ที่ เมธอด ชื่อว่า up() และเขียนเพื่อลบ table ใน เมธอด ชื่อว่า down() ดังนี้

```
<?php

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

class CreateTypebooksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('typebooks', function (Blueprint $table) {
            $table->id(); //รหัสประเภทหนังสือ
            $table->string('name'); //รายละเอียดประเภทหนังสือ
            $table->timestamps();
        });
    }

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

```

    }

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

```

5. เปิดไฟล์ xxx\_create\_books\_table.php เพื่อเขียนโค้ดในการสร้างโครงสร้างของตาราง โดยโค้ดสร้าง table ที่ เมธอด ชื่อว่า up() และเขียนเพื่อลบ table ใน เมธอด ชื่อว่า down() ดังนี้

```

<?php

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

class CreateBooksTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('books', function (Blueprint $table) {
            $table->id();
            $table->string('title'); //ชื่อหนังสือ
            $table->decimal('price',10,2); //ราคา
            $table->unsignedBigInteger('typebooks_id');
            $table->foreign('typebooks_id')->references('id')->on('typebooks');
            $table->string('image'); //รูปหนังสือ
            $table->timestamps();
        });
    }

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

```

6. ขั้นตอนนี้หากใครใช้ฐานข้อมูล MySQL เวอร์ชันต่ำกว่า 5.7.7 ให้เปิดไฟล์ app\Providers\AppServiceProvider.php และเพิ่ม  
ให้ดีที่ method ซึ่งว่า boot ดังรูป

```

    EXPLORE          AppServiceProvider.php ×
    OPEN EDITORS      app > Providers > AppServiceProvider.php
    LARAVEL7
        app
            Console
            Exceptions
            Http
        Providers
            AppServiceProvider.php
            AuthServiceProvider.php
            BroadcastServiceProvider.php
            EventServiceProvider.php
            RouteServiceProvider.php
            User.php
        bootstrap
        config
        database

```

```

1  <?php
2
3  namespace App\Providers;
4
5  use Illuminate\Support\ServiceProvider;
6  use Illuminate\Support\Facades\Schema; ←
7
8  class AppServiceProvider extends ServiceProvider
9  {
10
11     /**
12      * Register any application services.
13      *
14      * @return void
15     */
16     public function register()
17     {
18         Schema::defaultStringLength(191); ←
19     }
}

```

7. เปิด Command Prompt ขึ้นมา พิมพ์ php artisan migrate:install แล้ว enter เพื่อสร้างตาราง migrations ในฐานข้อมูล จากนั้น  
ให้พิมพ์คำสั่ง php artisan migrate เพื่อส่งรันไฟล์ migration ทั้งหมด แค่นี้เราก็จะได้ตารางสำหรับฐานข้อมูลมาใช้แล้วครับ โดย<sup>ๆ</sup>  
สามารถตรวจสอบตารางที่สร้างได้ที่ phpMyAdmin

```

Administrator: Command Prompt
C:\xampp\htdocs\laravel >php artisan migrate:install ←
Migration table created successfully.

C:\xampp\htdocs\laravel >php artisan migrate ←
Migration table created successfully.

```

localhost/phpmyadmin/db\_structure.php?server=1&db=bo

phpMyAdmin

Recent Favorites

bookstore

Structure SQL Search

Containing the word:

Table	Action
books	Browse
failed_jobs	Browse
migrations	Browse
password_resets	Browse
typebooks	Browse
users	Browse

## ตัวอย่างอื่นๆ สำหรับการเขียนโค้ดเพื่อกำหนดโครงสร้างของตาราง สำหรับทำ Migration

- \$table->string('comments')->nullable(); กำหนดค่าลักษณะของอนุญาตค่า null ได้
- \$table->tinyInteger('age')->unsigned(); กำหนดค่าลักษณะให้ไม่ติดเครื่องหมาย
- \$table->tinyInteger('age')->unsigned()->default(0); กำหนดค่าปริยาย (default) เป็น 0

## การเพิ่มข้อมูลเริ่มต้นลงในตารางด้วย Seeding

เราสามารถเพิ่มข้อมูลเริ่มต้นให้กับແກ່ໄນຕາງໄດ້ ເຊັ່ນ ການຕັ້ງຄ່າລະບົບ ອີເວມແຕ່ຊື່ຜູ້ໃຊ້ ອີ່ອຮັສຜ່ານທີ່ເວົາຕ້ອງການເພີ່ມຕອນທຳ Migration ໂດຍ ໄພລື່ສໍາຫຼັກການເຂົ້ານັ້ນຈະອຸປະກອດໃນໂຟຣ໌ database\seeds DatabaseSeeder.php ໃນຕັ້ອຍ່າງນີ້ຈະລອງທົດສອບສ້າງຜູ້ໃຊ້ໃນລະບົບເວົາໃໝ່ນາ 1 ດັນ ມີໜັ້ນຕອນ ດັນນີ້

1. เปิดไฟล์ database\seeds\DatabaseSeeder.php ແລ້ວເຂົ້ານັ້ນໂຟຣ໌ສໍາຫຼັກການເພີ່ມຂໍ້ມູນໃນຕາງ ດັນນີ້

```

EXPLORER
    OPEN EDITORS
        DatabaseSeeder.php
    LARAVEL7
        app
        bootstrap
        config
        database
            factories
            migrations
        seeds
            DatabaseSeeder.php
            .gitignore
            node_modules
            public
            resources
            routes
            storage
            tests
            vendor
        .editorconfig

DatabaseSeeder.php
1 <?php
2
3 use Illuminate\Database\Seeder;
4 use Illuminate\Support\Facades\Hash; ←
5
6 class DatabaseSeeder extends Seeder
7 {
8     /**
9      * Seed the application's database.
10     *
11     * @return void
12     */
13     public function run()
14     {
15         $user = new \App\User();
16         $user->name = 'Akenarin Komkoon';
17         $user->email = 'codingthailand@gmail.com';
18         $user->password = Hash::make('123456');
19         $user->save();
20         // $this->call/UsersTableSeeder::class);
21     }
22 }

```

Note: Hash::make() เป็นคำสั่งສໍາຫຼັບເຂົ້າຈຳລັກຂອງ password ດູ້ເພີ່ມເຕີມໄດ້ທີ່ <https://laravel.com/docs/master/hashing>

2. ในหัวข้อที่แล้วเราได้สร้าง table ไว้แล้ว หากต้องการทำ seeding ให้ใส่คำสั่ง --seed ต่อท้าย เช่น php artisan migrate --seed ถ้าในฐานข้อมูลยังไม่มี Table แต่ถ้ามี table อยู่แล้วสามารถอัปเดตโดยใช้ migrate:refresh ดังนี้

```
php artisan migrate:fresh --seed
```

Laravel จะตรวจสอบ table เก่า แล้วสร้าง table ใหม่พร้อมกับ seeding ให้เลย เมื่อรันคำสั่งแล้วใส่ phpMyAdmin จะสังเกตว่ามี แถวในตาราง users เพิ่มให้เรียบร้อยแล้ว

	<input type="button" value="T"/>	<input type="button" value="←"/>	<input type="button" value="→"/>	<input type="button" value="▼"/>	<b>id</b>	<b>name</b>	<b>email</b>	<b>password</b>
	<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	1	Akenarin Komkoon	codingthailand@gmail.com	\$2y\$10\$I9.hut36nL3

Note: ตอนนี้เราสามารถลงทะเบียนผู้ใช้ได้แล้ว และสามารถล็อกอิน และล็อกเอาท์ออกจากระบบ ฝากทดสอบด้วยนะครับ 😊

## บทที่ 5 การทำงานกับฐานข้อมูล การสร้าง Models และ การใช้ Eloquent ORM

เมื่อเราได้สร้างฐานข้อมูล และตารางเรียบร้อยแล้ว ต่อไปเป็นการสร้าง Models เพื่อกำหนด table ในฐานข้อมูล และการใช้งาน Eloquent ORM สำหรับการจัดการกับฐานข้อมูลที่ง่ายขึ้น เอียนโค้ดสั้นลง โดยไม่ต้องใช้ภาษา SQL ครับ

### การสร้าง Models

เมื่อสร้างตารางในฐานข้อมูลแล้วลำดับต่อมา คือเราควรสร้าง Model ให้กับตารางแต่ละตาราง การสร้าง Model สามารถใช้ artisan ช่วยในการสร้าง มีรูปแบบดังนี้

```
php artisan make:model <ชื่อคลาสของโมเดล>
```

- สร้าง Model ตาราง typebooks เข้าไปที่ไฟล์เดอร์project เปิด Command Prompt แล้วพิมพ์

```
php artisan make:model TypeBooks
```

Administrator: Command Prompt

```
C:\xampp\htdocs\laravel>php artisan make:model TypeBooks
Model created successfully.
```

```
C:\xampp\htdocs\laravel>
```

**Note:** ตอนที่สร้าง Models หากต้องการทำ migration ด้วยสามารถใส่ -m ต่อท้ายคำสั่งได้ เช่น

```
php artisan make:model TypeBooks -m
```

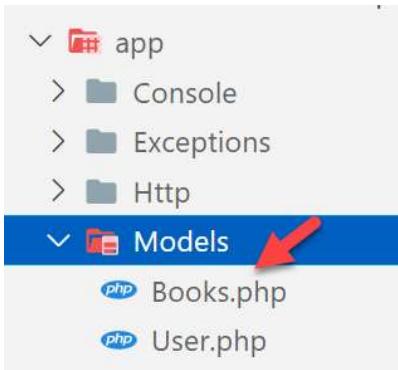
- สร้าง Model ตาราง books พิมพ์คำสั่ง `php artisan make:model Books`

Administrator: Command Prompt

```
C:\xampp\htdocs\laravel>php artisan make:model Books
Model created successfully.
```

```
C:\xampp\htdocs\laravel>
```

3. เมื่อใช้คำสั่งสร้าง Model แล้วไฟล์ Model จะอยู่ในโฟลเดอร์ app/Models



4. เปิดไฟล์ app\Models\TypeBooks.php เพื่อเขียนโค้ดกำหนดชื่อ table ดังนี้

```
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
    use HasFactory;
}
```

5. เปิดไฟล์ app\Models\Books.php เพื่อเขียนโค้ดกำหนดชื่อ table และการกำหนดการทำ Mass Assignment ดังนี้

```
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
    protected $table = 'books'; //กำหนดชื่อตารางในฐานข้อมูล
    protected $fillable = ['title', 'price', 'typebooks_id']; //กำหนดให้สามารถเพิ่ม
    //ข้อมูลได้ในคำสั่งเดียว Mass Assignment
}
```

## การใช้งาน Eloquent ORM

Eloquent เป็นตัวช่วยเราให้สามารถเขียนได้เพื่อจัดการกับฐานข้อมูลได้ง่ายขึ้น โดยใช้คำสั่งเพียงไม่กี่คำสั่ง ไม่ว่าจะเป็นการเรียกดูข้อมูล แสดงข้อมูล การเพิ่ม การแก้ไข หรือลบข้อมูลต่างๆ

### ตัวอย่างคำสั่งสำหรับการเรียกดูข้อมูล หรือแสดงข้อมูล

- \$typebooks = TypeBooks::all(); //ใช้ all() สำหรับแสดงข้อมูลทั้งหมดในตาราง
- \$typebooks = TypeBooks::find(1); //ใช้ find(ค่า Primary Key) สำหรับแสดงข้อมูล 1 แถวโดยมีเงื่อนไขเท่ากับค่า primary key ที่รับเข้ามา (ใช้ในกรณีที่ Primary Key เป็น int หรือตัวเลขเท่านั้น)
- \$person = Person::where('person\_id', '=', '001')->first(); //ใช้ where ร่วมกับ first() สำหรับแสดงข้อมูล primary key ที่ไม่ใช้ตัวเลข (person\_id เป็น Primary Key)
- \$person = Person::where('status', '=', '1')->get(); //ใช้ get() สำหรับเรียกดูข้อมูลในกรณีอื่นๆ

### ตัวอย่างการใช้งานฟังก์ชันที่ใช้บ่อย

- \$bookCount = Books::count(); //นับจำนวนแถวทั้งหมด
- \$maximumTotal = Order::max('amount'); //หาค่ามากที่สุด
- \$minimumTotal = Order::min('amount'); //หาค่าน้อยที่สุด
- \$averageTotal = Order::avg('amount'); //หาค่าเฉลี่ย
- \$lifetimeSales = Order::sum('amount'); //หาผลรวม

### ตัวอย่างคำสั่งสำหรับกรองข้อมูล (Filtering records) เทียบได้กับ where, order by และ limit

- \$person = Person::where('prefix\_id', '=', '01')->get();
- \$customers = Customer::orderBy('id','desc')->limit(2)->get();
- \$person = Person::limit(5)->get(); หรือ \$person = Person::take(2)->get();
- \$customers = Customer::where('firstname','like','%')->get();

### คำสั่งสำหรับการเพิ่มข้อมูล และแก้ไขข้อมูล

- ใช้ save() สำหรับเพิ่มหรือแก้ไขข้อมูล
- ใช้ create() สำหรับเพิ่มข้อมูลแบบบรรทัดเดียวหรือเรียกว่า Mass Assignment แต่ก่อนจะใช้ ต้องไปกำหนดพิลเดท์ที่ต้องการเพิ่มให้กับตัวแปร \$fillable ที่ไฟล์ Model ก่อน

## คำสั่งในการลบข้อมูล

มี 2 วิธี ได้แก่

- ใช้ delete() สำหรับ ลบโดยเรียกดู record ที่ต้องการลบก่อน ค่อยสั่งลบ เช่น

```
$cat = Cat::find(1);
```

```
$cat->delete();
```

- ใช้ destroy() สำหรับลบ แต่ไม่ต้อง find() ก่อน เช่น

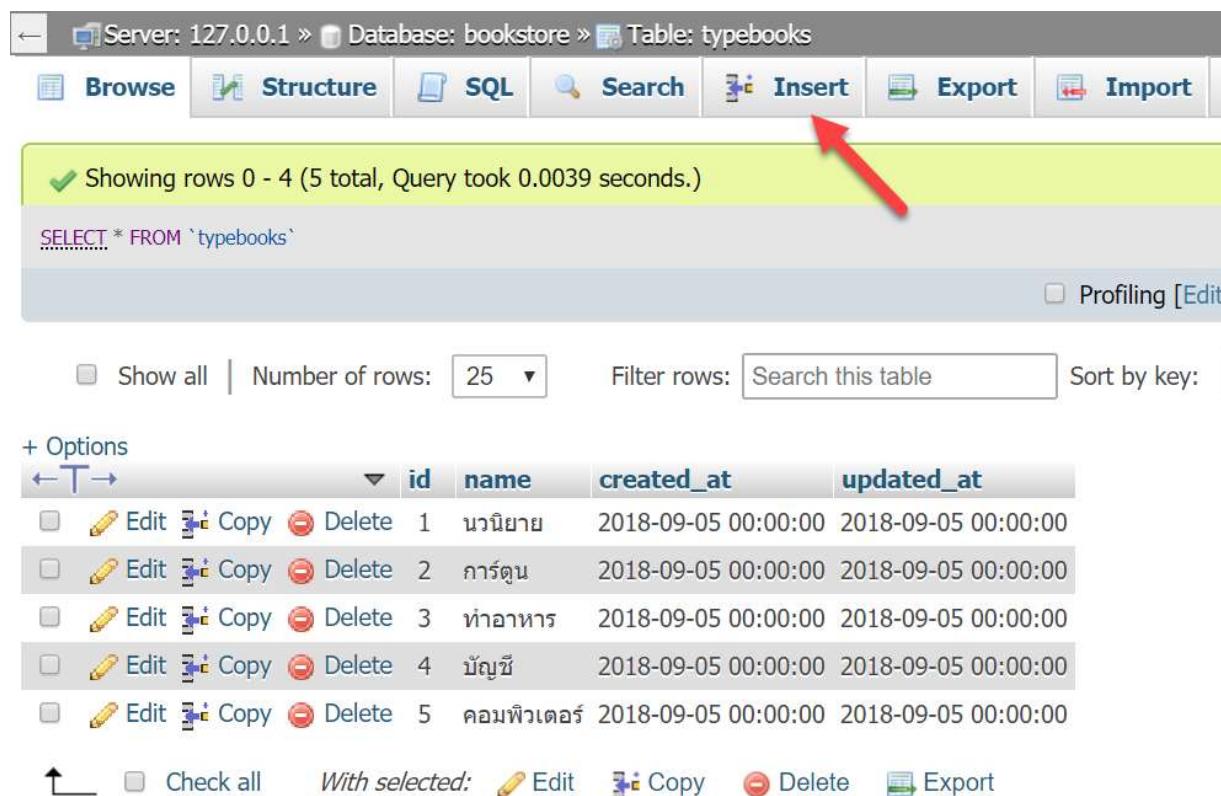
```
Cat::destroy(1);
```

หรือ

```
Cat::destroy(1, 2, 3, 4, 5); // การลบที่ลະหลາຍແດວ
```

## แสดงข้อมูลตาราง ประเภทหนังสือ (typebooks)

หลังจากเรียนรู้คำสั่งของ Eloquent แล้ว เราจะมาทดลองสร้างหน้าเพจสำหรับแสดงข้อมูลประเภทหนังสือ แต่ก่อนอื่นแนะนำให้ phpMyAdmin เพื่อเพิ่มข้อมูล (เมนู Insert) ขั้น 5 แถวในตาราง typebooks ก่อนครับ เพราะจะได้เห็นข้อมูลเมื่อแสดงผลในหน้าเพจ



The screenshot shows the phpMyAdmin interface with the following details:

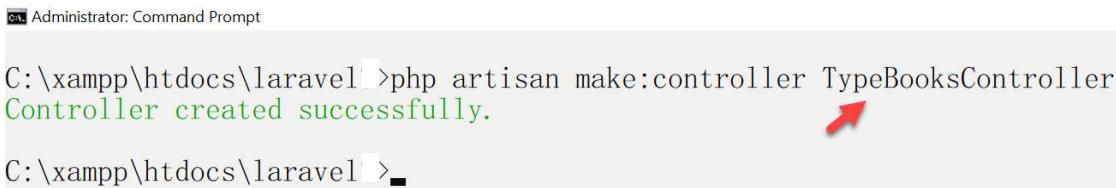
- Server:** 127.0.0.1
- Database:** bookstore
- Table:** typebooks
- Toolbar Buttons:** Browse, Structure, SQL, Search, Insert (highlighted with a red arrow), Export, Import.
- Status Bar:** Showing rows 0 - 4 (5 total, Query took 0.0039 seconds.)
- Query Editor:** SELECT \* FROM `typebooks`
- Table Data:**

	id	name	created_at	updated_at
<input type="checkbox"/>	1	นานิยาย	2018-09-05 00:00:00	2018-09-05 00:00:00
<input type="checkbox"/>	2	การดูน	2018-09-05 00:00:00	2018-09-05 00:00:00
<input type="checkbox"/>	3	ทำอาหาร	2018-09-05 00:00:00	2018-09-05 00:00:00
<input type="checkbox"/>	4	บัญชี	2018-09-05 00:00:00	2018-09-05 00:00:00
<input type="checkbox"/>	5	คอมพิวเตอร์	2018-09-05 00:00:00	2018-09-05 00:00:00
- Bottom Navigation:** Check all, With selected: Edit, Copy, Delete, Export.

ขั้นตอนการแสดงข้อมูลเมื่อดังนี้

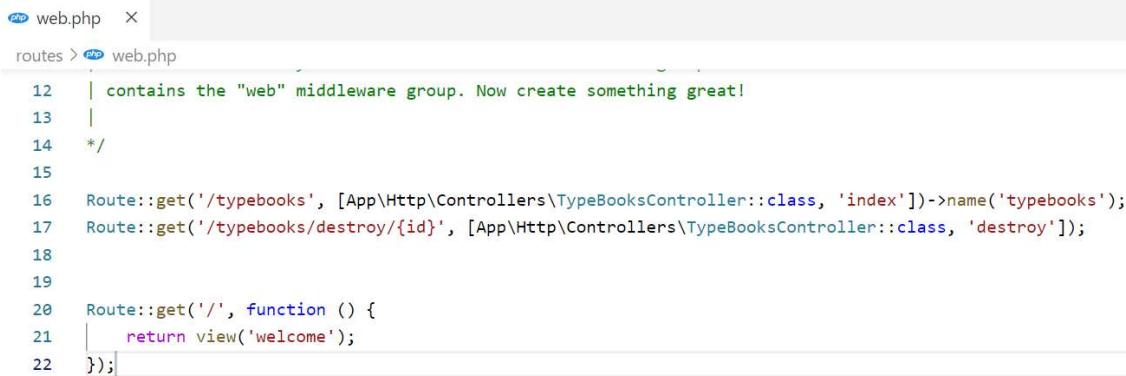
- เปิด Command Prompt เพื่อพิมพ์คำสั่งสำหรับสร้าง Controller ดังนี้

```
php artisan make:controller TypeBooksController
```



```
C:\xampp\htdocs\laravel>php artisan make:controller TypeBooksController
Controller created successfully.
```

- เปิดไฟล์ routes\web.php เพื่อสร้าง route โดยเราจะสร้าง 2 ตัวเพื่อการแสดงข้อมูล และการลบข้อมูล ดังนี้



```
web.php  X
routes > web.php
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/typebooks', [App\Http\Controllers\TypeBooksController::class, 'index'])->name('typebooks');
17 Route::get('/typebooks/destroy/{id}', [App\Http\Controllers\TypeBooksController::class, 'destroy']);
18
19
20 Route::get('/', function () {
21     return view('welcome');
22});
```

- เปิดไฟล์ app\Http\Controllers\TypeBooksController.php จากนั้นเขียน เมธอด (index, destroy) ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\TypeBooks;//นำเข้าโมเดล TypeBooks เท่านำใช้งาน

class TypeBooksController extends Controller
{
    public function index() {
        $typebooks = TypeBooks::all(); //แสดงข้อมูลทั้งหมด
        //$typebooks = TypeBooks::orderBy('id','desc')->get(); //แสดงข้อมูลทั้งหมดเรียงจากมากไปน้อยโดยใช้ id
        $count = TypeBooks::count(); //นับจำนวนและทั้งหมด
        return view('typebooks.index', [
            'typebooks' => $typebooks,
            'count' => $count
        ]); // ส่งไปที่ views ไฟล์ controller typebooks ไฟล์ index.blade.php
    }
}
```

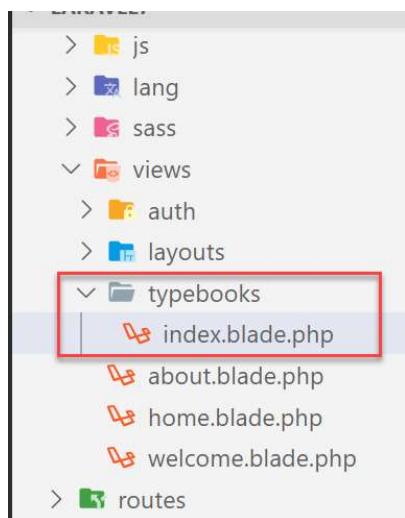
```

public function destroy($id) {
    //TypeBooks::find($id)->delete();
    TypeBooks::destroy($id);
    return back();
}

```

อธิบายโค้ด ในส่วนของ เมธอด index() เราจะใช้ all() สำหรับดึงข้อมูลทั้งหมดมาเก็บไว้ในตัวแปร \$typebooks เพื่อส่งไปให้ view แสดงผล และใช้ count() สำหรับนับจำนวนแถวทั้งหมดในตารางนี้ สงไปแสดงผลที่ view เข่นเดียวกัน ส่วน เมธอด destroy(\$id) เราจะใช้เพื่อรับค่า primary key จาก URL แล้วทำการลบแถวออกจากตารางครับ

4. สร้างไฟล์เดอร์ typebooks และสร้างไฟล์ view ชื่อว่า index.blade.php เพื่อรับตัวแปรต่างๆ จาก TypeBooksController เพื่อแสดงผล ดังรูป



5. จากข้อ 4 เปิดไฟล์ resources\views\typebooks\index.blade.php เขียนโค้ดเพื่อแสดงผล (วนลูปข้อมูล) ดังนี้

```

@extends('layouts.app')

@section('content')


แสดงข้อมูลประเภทหนังสือ [ทั้งหมด {{ $count }} รายการ]



| รหัส                 | ประเภทหนังสือ          | เครื่องมือ                                                                                       |
|----------------------|------------------------|--------------------------------------------------------------------------------------------------|
| {{ \$typebook->id }} | {{ \$typebook->name }} | <a &gt;ลบ&lt;="" a&gt;<="" href="{{ url('/typebooks/destroy/'.\$typebook-&gt;id) }}" td=""> </a> |


```

6. เพิ่มเมนูเพิ่มชื่อว่า ประเภทหนังสือ โดยเปิดไฟล์ resources\views\layouts\app.blade.php ดังนี้

```

app.blade.php ×

d.navbar-light.bg-white.shadow-sm > div.container > div#navbarSupportedContentollapse.navbar-collapse >
38
39      <!-- Right Side Of Navbar -->
40      <ul class="navbar-nav ml-auto">
41          <!-- Authentication Links -->
42          @guest
43
44              <li class="nav-item">
45                  <a class="nav-link" href="{{ route('typebooks') }}">ประเภทหนังสือ</a>
46              </li>
47
48              <li class="nav-item">
49                  <a class="nav-link" href="{{ route('about') }}">เกี่ยวกับเรา</a>
50              </li>
51

```

7. บันทึกไฟล์ทั้งหมดแล้วลองรันดูครับ <http://localhost/laravel9/public/typebooks>

The screenshot shows a Laravel application interface. At the top, there is a navigation bar with links: 'ประเภทหนังสือ' (highlighted with a red box), 'เกี่ยวกับเรา', 'เข้าระบบ', and 'ลงทะเบียน'. Below the navigation bar, the main content area has a title 'แสดงข้อมูลประเภทหนังสือ [ทั้งหมด 5 รายการ]'. A table lists five categories of books:

รหัส	ประเภทหนังสือ	เครื่องมือ
1	นวนิยาย	<a href="#">ลบ</a>
2	การคุณ	<a href="#">ลบ</a>
3	ท่องเที่ยว	<a href="#">ลบ</a>
4	นักเขียน	<a href="#">ลบ</a>
5	คอมพิวเตอร์	<a href="#">ลบ</a>

### การลบข้อมูล ประเภทหนังสือ (typebooks)

จากหัวข้อที่แล้วในส่วนของไฟล์ resources\views\typebooks\index.blade.php เราได้แทรกลิงก์สำหรับให้ผู้ใช้คลิกเพื่อลบข้อมูลออกไป ตามโค้ดนี้ `<a href="{{ url('/typebooks/destroy/'.$typebook->id) }}>ลบ</a>` เมื่อผู้ใช้คลิกลบ เราจะส่งค่า id คือ primary key ไปกับ URL เพื่อส่งไปลบยัง เมธอด destroy(\$id) ของ TypeBooksController กัน

ถ้าเปิดไฟล์ app\Http\Controllers\TypeBooksController.php เราจะเห็นว่าที่ เมธอด destroy(\$id) ได้เขียนโดยคิดสำหรับไปไว้แล้ว ดังนี้

```
public function destroy($id) {
    //TypeBooks::find($id)->delete();
    TypeBooks::destroy($id);
    return back();
}
```

จากนั้นให้ทดสอบบล็อกด้วยค้วบ (เมื่อลบแล้วเราใช้ back() เมื่อย้อนกลับ URL ก่อนหน้านี้)

## การแบ่งหน้าข้อมูล (Pagination)

หากข้อมูลมีปริมาณมาก การแสดงข้อมูลทั้งหมดในหน้าเดียวอาจทำให้ข้อมูลโหลดได้ช้า เราชรทำກารแบ่งหน้าข้อมูล และ Laravel ได้เตรียม เมธอด ให้เราเรียกใช้ไว้แล้วครับ โดยเราจะเขียนโดยคิดแบ่งหน้า ที่ Controller และอีกส่วนจะเขียนที่ views ได้แก่

- การแบ่งหน้าแบบปกติ จะใช้ paginate(จำนวนแถวต่อหน้า) ตัวอย่างเช่น  

```
$persons = Person::paginate(20);
```
- การแบ่งหน้าอย่างง่าย จะใช้ simplePaginate(จำนวนแถวต่อหน้า) ตัวอย่างเช่น  

```
$persons = Person::simplePaginate(15);
```
- และในส่วนของ view ให้เขียนโดยเพื่อ render ดังนี้  

```
{!! $persons->render() !!} // $persons คือ ตัวแปรที่ส่งมาจาก Controller  
และหากต้องการแสดงจำนวนแถวข้อมูลทั้งหมดให้เขียนแบบนี้  
{!! $persons->total() !!} // $persons คือ ตัวแปรที่ส่งมาจาก Controller
```

Note: เราจะเลือกใช้การแบ่งหน้าแบบปกติ หรือ การแบ่งหน้าอย่างง่ายก็ได้ครับ ข้อแตกต่างคือ รูปแบบการแสดงผลโดยการแบ่งหน้าอย่างง่าย จะแสดงในรูปแบบ "Next" และ "Previous"

มาลดลงແປ່ງໜ້າຂໍ້ອມມູນປະເກທໜັງສືອ

1. ເປີດໄຟລ໌ app\Http\Controllers\TypeBooksController.php ໂດຍເພີ່ມໂຄ້ດແປ່ງໜ້າທີ່ເມຮອດ index() ດັ່ງນີ້

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\TypeBooks;//ນໍາເອົາໂມເດລ TypeBooks ເຂົ້າມາໃຊ້ຈານ

class TypeBooksController extends Controller
{
    public function index() {
        // $typebooks = TypeBooks::all();
        // $typebooks = TypeBooks::orderBy('id','desc')->get();
        $count = TypeBooks::count(); //ນັບຈຳນວນແຖງທັງໝົດ

        //ແປ່ງໜ້າ
        // $typebooks = TypeBooks::simplePaginate(3);
        $typebooks = TypeBooks::paginate(3);

        return view('typebooks.index', [
            'typebooks' => $typebooks,
            'count' => $count
        ]); // ສັ່ງໄປທີ່ views ໃຟດເດອກ໌ typebooks ໄຟລ໌ index.blade.php
    }

    public function destroy($id) {
        // TypeBooks::find($id)->delete();
        TypeBooks::destroy($id);
    }
}
```

```

        return back();
    }
}

```

ให้ลองเปิด-ปิด comment และดูข้อแตกต่างได้ และถ้าเราเขียนโค้ดการแบ่งหน้าก็ไม่ต้องเรียก all() อีกครับ

```

//แบ่งหน้า
//$typebooks = TypeBooks::simplePaginate(3);
$typebooks = TypeBooks::paginate(3);

```

2. ต่อมาให้เปิดไฟล์ views ได้แก่ resources\views\typebooks\index.blade.php แล้วแทรกโค้ด  
 `{{ $typebooks->links() }}` ไว้ส่วนท้ายของตาราง ดังนี้

```

@extends('layouts.app')

@section('content')


แสดงข้อมูลประเภทหนังสือ [ทั้งหมด {{ $count }} รายการ]



| รหัส | ประเภทหนังสือ | เครื่องมือ |
|------|---------------|------------|
|------|---------------|------------|


```

```

@foreach ($typebooks as $typebook)

<tr>

<td>{{ $typebook->id }}</td>
<td>{{ $typebook->name }}</td>
<td><a href="{{ url('/typebooks/destroy/'.$typebook->id) }}>ລົບ</a></td>
</tr>

@endforeach

</table>

{{ $typebooks->links() }}

</div>
</div>
</div>
</div>
</div>
</div>
@endsection

```

3. ບັນທຶກໄຟລ໌ແລ້ວກົດລອງຮັນດູກັບ <http://localhost/laravel9/public/typebooks>

The screenshot shows a Laravel application interface. At the top, there is a navigation bar with links: 'Laravel', 'ປະເທດນັ້ນສືບ', 'ເກີຍກົມເຮົາ', 'ເຂົ້າຮັບນິນ', and 'ລົງທະເບຸນ'. Below the navigation bar, there is a title 'ແສດງຂໍ້ມູນປະເທດນັ້ນສືບ [ທັງໝາດ 4 ຮາຍການ]'. A table displays four entries:

ຮັດສ	ປະເທດນັ້ນສືບ	ເຄື່ອງນືອ
1	ນວນີຍາຍ	ລົບ
2	ກາງຊຸນ	ລົບ
4	ນັກປີ້	ລົບ

At the bottom left of the table, there is a navigation bar with icons for back, forward, and search.

## Query scopes

Query scopes เป็นเทคนิคการรวมเอา query ที่มีความซ้ำซ้อนมาเขียนไว้ที่ Models แทนที่จะเขียนที่ Controllers ประยุกต์คือ ทำให้ได้ดั้ง Controller จานง่าย สะดวก และยืดหยุ่นขึ้นครับ โดยข้างหน้าชื่อเมธอดจะต้องขึ้นต้นด้วยคำว่า scope เช่น ตัวอย่างเช่น หากเราต้องการหาผู้ใช้ที่อายุมากกว่า 18 ปี แทนที่เราจะเขียนโค้ดเบօะๆ ที่ Controller ก็ให้มาเขียนที่ Models ดีกว่า

```
class User extends Model {
    public function scopeOver18($query)
    {
        $date = Carbon::now()->subYears(18);
        return $query->where('birth_date', '<', $date);
    }
}
```

เวลาเรียกใช้ที่ Controller ก็เขียนแค่นี้พอ (ตัดคำว่า scope ออกไป) ลองนำไปใช้ดูได้ครับ

```
$userOver18 = User::over18()->get();
```

## การสร้าง Accessors

Accessors หากเรียกง่ายๆ อีกชื่อหนึ่งก็คือ getter นั่นเอง เป็นเมธอดที่มีประยุกต์คือ เราสามารถสร้าง attribute ที่ไม่ใช่ attribute ในฐานข้อมูลได้ โดยให้กำหนดที่ Models นั้นๆ อาจทำการประมวลผล หรือคำนวณค่าข้อมูลจากตาราง เช่น การนำชื่อและนามสกุลมาเชื่อมกัน การคำนวณราคารวัสดุสินค้า หากเราไม่ได้กำหนดตารางในฐานข้อมูล เป็นต้น

ข้อกำหนดของ Accessor คือ ชื่อเมธอดจะต้องขึ้นต้นด้วยคำว่า get และลงท้ายด้วยคำว่า Attribute ดังตัวอย่าง

```
class User extends Model {
    public function getfullnameAttribute()
    {
        return $this->firstname . " " . $this->lastname;
    }
}
```

เวลาเข้าถึงหรือเรียกใช้งาน Accessor ก็ให้ตัด get และ Attribute ออกไปเหลือแค่ fullname (ตัวพิมพ์เล็ก) เช่น

```
$user->fullname;
```

## การสร้าง Mutators

Mutators ก็คือ setter นั่นเอง คล้ายกันกับ Accessors คือ เราสามารถสร้าง attribute ที่ไม่ใช่ attribute ในฐานข้อมูลได้ โดยเมธอดที่สร้างขึ้นนี้จะเป็นการรับค่าพารามิเตอร์เข้ามาเพื่อ set ค่าให้กับ attribute ของ Models

ข้อกำหนดของ Mutators คือ ข้อมูลจะต้องขึ้นต้นด้วยคำว่า set และลงท้ายด้วยคำว่า Attribute ดังตัวอย่าง

```
class User extends Model {
    public function setPasswordAttribute($password)
    {
        return $this->attributes['password'] = Hash::make($password);
    }
}
```

เวลาเข้าถึงหรือเรียกใช้งาน Mutators ก็ให้ตัด set และ Attribute ออกไปเหลือแค่ password (ตัวพิมพ์เล็ก) เช่น

```
$user->password = '123456';
```

## การกำหนด Eloquent relations

การกำหนดความสัมพันธ์ของ Eloquent นั้น เป็นการกำหนดว่ามีตารางใดบ้างในฐานข้อมูลที่มีความสัมพันธ์กันอยู่ วูปแบบของความสัมพันธ์หรือ relations ที่ใช้ปอยๆ มีดังต่อไปนี้

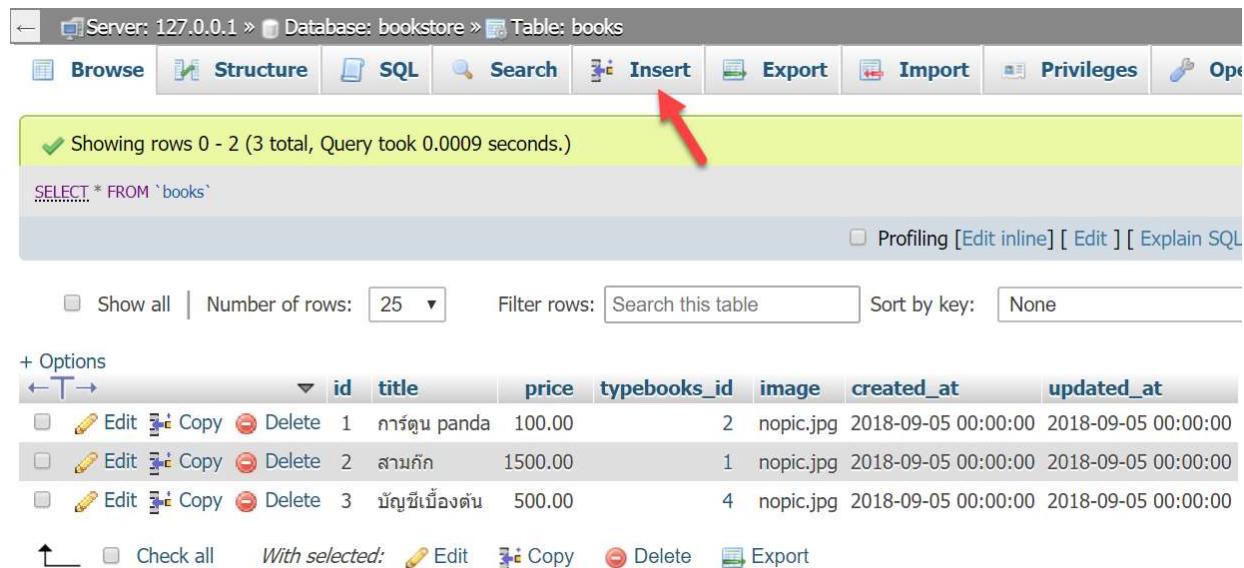
- One To One ความสัมพันธ์แบบหนึ่งต่อหนึ่ง
- One To Many ความสัมพันธ์แบบหนึ่งต่อคุณ หรือเรียกว่า Belongs To Relation ก็ได้
- Many To Many ความสัมพันธ์แบบกลุ่มต่อกลุ่ม

การกำหนด Relation เราจะสร้างเมธอดเพิ่มที่ Models ที่มีความสัมพันธ์กัน เช่น หากตารางใดมีความสัมพันธ์แบบหนึ่งต่อหนึ่งให้เรียกใช้เมธอด hasOne() และอีกดาราที่ที่เรียกไปก็ให้กำหนดเป็น belongsTo() พั้นทั้งระบุ FK ที่ใช้ด้วย เช่นเดียวกันหากมีความสัมพันธ์เป็นแบบ One To Many จะกำหนดเป็น hasMany() และตารางที่ที่เรียกันก็จะใช้ belongsTo() เราเรียก belongsTo() อีกอย่างหนึ่งว่า การ inverse ความสัมพันธ์ได้ ส่วนความสัมพันธ์แบบ Many To Many ให้กำหนดเป็น belongsToMany() ทั้งสองฝ่ายควร

ในหนังสือเล่มนี้จะยกตัวอย่างความสัมพันธ์ที่ใช้ปอยที่สุดได้แก่ One To Many และ Many To One นั่นเอง

## แสดงข้อมูลตารางหนังสือ (books) ด้วยการทำ relations (join table)

ในหัวข้อนี้เราจะสร้างหน้าเพจเพื่อแสดงข้อมูลจากตารางหนังสือ (books) ซึ่งมีความสัมพันธ์กับตารางประเภทหนังสืออยู่ (One To Many) ก่อนอื่นให้เราเปิด phpMyAdmin เพื่อเพิ่มข้อมูลหนังสือ เพื่อเป็นตัวอย่างก่อนนะครับ ตัวอย่างการกรอกข้อมูล ดังนี้



The screenshot shows the phpMyAdmin interface for the 'bookstore' database. The 'books' table is selected. The top menu bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert' (which has a red arrow pointing to it), 'Export', 'Import', 'Privileges', and 'Open'. Below the menu, a green status bar says 'Showing rows 0 - 2 (3 total, Query took 0.0009 seconds.)'. The SQL query 'SELECT \* FROM `books`' is displayed. The table itself has three rows with columns: id, title, price, typebooks\_id, image, created\_at, and updated\_at. The first row has image 'nopic.jpg'. At the bottom of the table, there are buttons for 'Check all', 'With selected:', 'Edit', 'Copy', 'Delete', and 'Export'.

ลังเกตว่าคอลัมน์ image ให้เรากรอกเป็น nopic.jpg ไว้ก่อน

ขั้นตอนการแสดงข้อมูลหนังสือ มีดังนี้

1. เปิดไฟล์ routes\web.php เพื่อสร้าง route แต่ครั้งนี้จะเราจะสร้าง route ในรูปแบบของ resource เราจะให้ Laravel จัดการให้ ดังนี้

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

Route::resource('books', App\Http\Controllers\BooksController::class)->name('index', 'books');
```



2. กำหนดความสัมพันธ์ระหว่างตาราง typebooks และ books ในรูปแบบของ One to Many เปิดไฟล์ app\Models\TypeBooks.php เพิ่มเมื่อตัวชี้หัวรับกำหนด relations ดังนี้

```
<?php
```

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
```

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

```
class TypeBooks extends Model
```

```
{
```

```
protected $table = 'typebooks'; //กำหนดชื่อตารางให้ตรงกับฐานข้อมูล
```

```
protected $fillable = ['title', 'price', 'typebooks_id']; //กำหนดให้สามารถเพิ่มข้อมูลได้ในคำสั่งเดียว Mass Assignment
```

```
public function books() {
```

```
    return $this->hasMany(Books::class); //กำหนดความสัมพันธ์รูปแบบ One To Many ไปยังตาราง books
```

```
}
```

```
}
```

เปิดไฟล์ app\Models\Books.php เพื่อทำการ inverse relation (Many To One) โดยใช้พังก์ชันหรือเมธอดต่อไปนี้

```
<?php
```

```
namespace App\Model;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
```

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

```
class Books extends Model
```

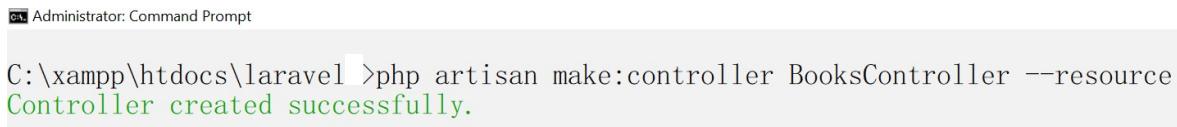
```
{
```

```
protected $table = 'books'; //กำหนดชื่อตารางในฐานข้อมูล
```

protected \$fillable = ['title', 'price', 'typebooks\_id']; //กำหนดให้สามารถเพิ่มข้อมูลได้ในคำสั่งเดียว Mass Assignment

```
public function typebooks() {
    return $this->belongsTo(TypeBooks::class, 'typebooks_id'); //กำหนด FK ด้วย
}
```

3. สร้างไฟล์ BooksController.php ในรูปแบบของ resource หรือเรียกว่า RESTful Controller ก็ได้ ให้เข้าไปโฟลด์ controllers และเปิด Command Prompt ขึ้นมา พิมพ์คำสั่ง php artisan make:controller BooksController --resource และกด enter



```
C:\xampp\htdocs\laravel >php artisan make:controller BooksController --resource
Controller created successfully.
```

4. จากนั้นลองเปิดไฟล์ BooksController.php จะเห็นว่า Laravel ได้สร้างเมธอดต่างๆ ในรูปแบบของ RESTful มาให้เรียบร้อยโดยที่เราไม่ต้องสร้างเอง (แนะนำวิธีนี้)
5. จากนั้นลองเปิดไฟล์ BooksController.php เขียนคำสั่งที่เมธอด index() เพื่อดึงข้อมูลหนังสือโดยใช้เมธอด with() เพื่อเชื่อม relation กับ typebooks และสร้างรายการหนังสือทั้งหมดไปที่ views (ในเดี๋ยวนี้ยังไม่การเรียงลำดับ id จากมากไปน้อย และแบ่งหน้าด้วย)

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use App\Models\Books; //อย่าลืม use โมเดลเข้ามาใช้งาน
```

```
class BooksController extends Controller
{
    /**
     * Display a listing of the resource.
     *
```

\* Display a listing of the resource.

```

*
* @return \Illuminate\Http\Response
*/
public function index() {

    $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
    return view('books/index',[ 'books' => $books]);

}

/** 
 * Show the form for creating a new resource.
 *
* @return \Illuminate\Http\Response
*/
public function create()

{
    //
}

/** 
 * Store a newly created resource in storage.
 *
* @param \Illuminate\Http\Request $request
* @return \Illuminate\Http\Response
*/
public function store(Request $request)

{
    //
}

/** 
 * Display the specified resource.
*

```

```
* @param int $id
* @return \Illuminate\Http\Response
*/
public function show($id)
{
    //



    /**
     * Show the form for editing the specified resource.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //



        /**
         * Update the specified resource in storage.
         *
         * @param \Illuminate\Http\Request $request
         * @param int $id
         * @return \Illuminate\Http\Response
         */
        public function update(Request $request, $id)
        {
            //



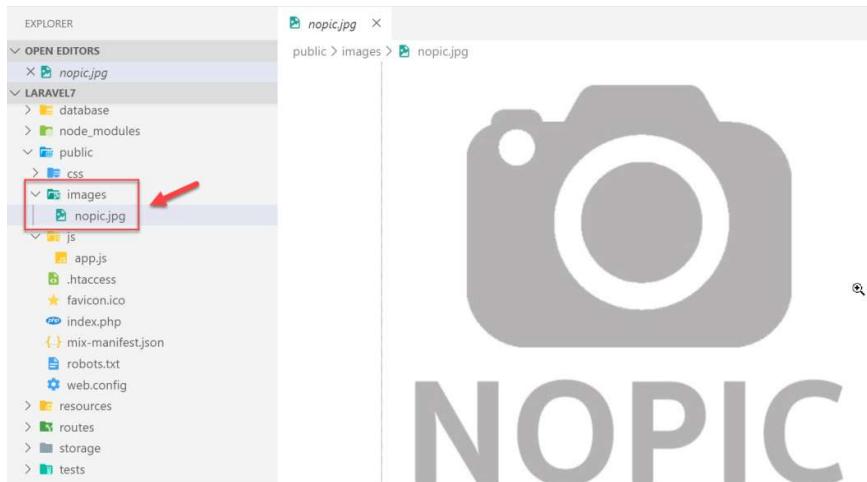
        }
    }
}
```

```

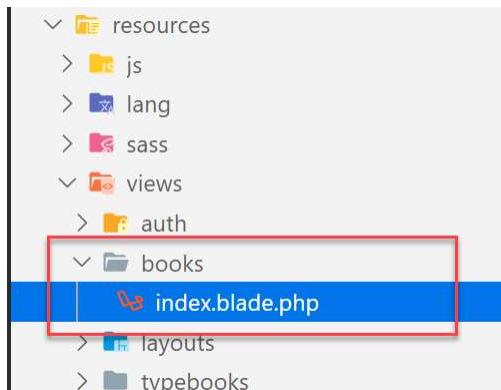
/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    //
}

```

6. เพื่อการแสดงผลที่สวยงามและถูกต้อง แนะนำให้หารูปภาพ nopic.jpg ไปวางไว้ที่โฟลเดอร์ public\images (อย่าลืมสร้างโฟลเดอร์ images ก่อน) ดังนี้



7. มาที่ส่วน views ก็ให้สร้างโฟลเดอร์ books และไฟล์ index.blade.php เพื่อแสดงผลข้อมูลในรูปแบบตาราง ดังนี้



8. เปิดไฟล์ index.blade.php จากข้อ 6 และเขียนคำสั่งเพื่อแสดงผลในรูปแบบตารางดังนี้

```

@extends('layouts.app')

@section('content')


แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{ $books->total() }} เล่ม



| รหัส             | ชื่อหนังสือ         | ราคา                           | หมวดหมู่                | รูปภาพ                                                                                                                           |
|------------------|---------------------|--------------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| {{ \$book->id }} | {{ \$book->title }} | number_format(\$book->price,2) | \$book->typebooks->name | <a href="{{ asset('images/'.\$book-&gt;image) }}">  </a> |

{{ $books->links() }}


```

9. สร้างเมนู หนังสือ เพิ่ม เปิดไฟล์ resources\views\layouts\app.blade.php เขียนโค้ด ดังนี้

```
app.blade.php X
white.shadow-sm > div.container > div#navbarSupportedContent.collapse.navbar-collapse > ul.navbar-nav.
38
39      <!-- Right Side Of Navbar -->
40      <ul class="navbar-nav ml-auto">
41          <!-- Authentication Links -->
42          @guest
43
44              <li class="nav-item">
45                  <a class="nav-link" href="{{ route('books') }}">หนังสือ</a>
46              </li>
47
48              <li class="nav-item">
49                  <a class="nav-link" href="{{ route('typebooks') }}">ประเภทหนังสือ</a>
50              </li>
51
52              <li class="nav-item">
53                  <a class="nav-link" href="{{ route('about') }}">เกี่ยวกับเรา</a>
54              </li>
--
```

10. ลองรันทดสอบ จะเห็นว่า ข้อมูลประเภทหนังสือที่มีความสัมพันธ์กับหนังสือ ได้แสดงขึ้นมาเรียบร้อย 😊

รหัส	ชื่อหนังสือ	ราคา	หมวดหนังสือ	รูปภาพ
3	นักเขียนเด็ก	500.00	นักเขียน	
2	สามก๊ก	1,500.00	นวนิยาย	
1	การ์ตูน panda	100.00	นวนิยาย	

## บทที่ 6 การสร้าง Web Forms การตรวจสอบความถูกต้องของข้อมูลและการอัปโหลดไฟล์

### การสร้างฟอร์มใน Laravel

การสร้างฟอร์มใน Laravel มี 2 วิธี ได้แก่

- เขียนโค้ด HTML เองทั้งหมด
- ใช้ Laravel Collective เป็นคลาสที่ช่วยสร้างฟอร์ม (แนะนำตัวนี้จะประหยัดเวลามากกว่า)

### การติดตั้ง และใช้งาน Laravel Collective

รายละเอียดการติดตั้ง และคู่มือ ให้เราเข้าเว็บ <https://laravelcollective.com/> จากนั้นเลือกเมนู Docs -> HTML

The screenshot shows a browser window with the URL [laravelcollective.com](https://laravelcollective.com/) in the address bar. The page content includes the Laravel Collective logo, a large heading 'Just some', and a subtext 'A collection of packages and tools for the Laravel community.' In the top right corner, there are two links: 'Docs' and 'Tools'. A red arrow points to the 'Docs' link.

### ขั้นตอนการติดตั้ง

ให้เปิดไฟล์ Command Prompt และพิมพ์คำสั่ง `composer require laravelcollective/html` ดังรูป (อย่าลืมใส่คอมม่าด้วย)

The screenshot shows a command prompt window with the title 'Administrator: Command Prompt'. The user has entered the command `C:\xampp\htdocs\laravel >composer require laravelcollective/html` and is awaiting the output of the command.

## สร้างฟอร์มเพิ่มข้อมูลหนังสือ (books)

หลังจากที่เราติดตั้ง Laravel Collective เรียบร้อย เราจึงสามารถสร้างฟอร์ม บุํม หรือลิงก์ต่างๆได้ เพื่อเป็นการทดสอบว่าเราติดตั้ง Laravel Collective สมบูรณ์หรือไม่ ลองสร้างลิงก์ที่อยู่ในรูปแบบบุํม ดังนี้

เปิดไฟล์ resources\views\books\index.blade.php และเพิ่มคำสั่ง เมธอด link\_to() เพิ่มสร้างลิงก์เพิ่มข้อมูล ดังนี้

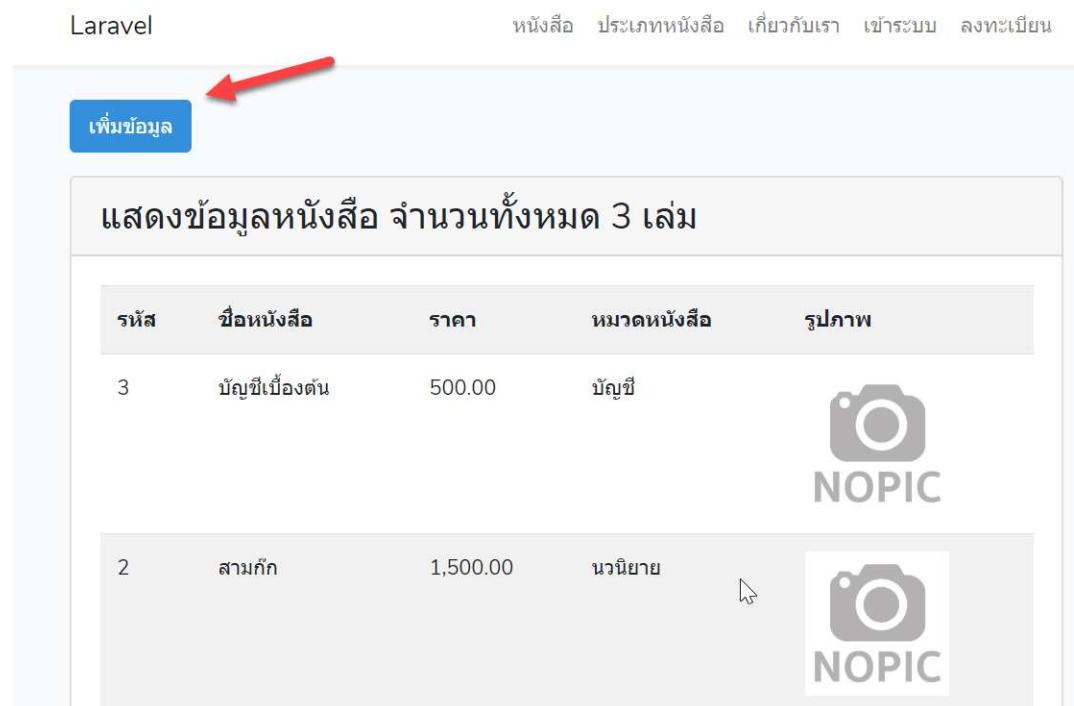
```
<?= link_to('books/create', $title = 'เพิ่มข้อมูล', [class' => 'btn btn-primary'], $secure = null); ?>
```



```
resources > views > books > index.blade.php >
1   @extends('layouts.app')
2
3   @section('content')
4   <div class="container">
5     <div class="row">
6       <div class="col-lg-10 col-lg-offset-1">
7         <?= link_to('books/create', $title = 'เพิ่มข้อมูล', [class' => 'btn btn-primary'], $secure = null); ?>
8
9
10    <div class="card mt-3">
11      <div class="card-header h3">
12        แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{ $books->total() }} เล่ม
13      </div>
14    </div>

```

บันทึกไฟล์แล้วลองรันดู หากมีปุ่มลิงก์เพิ่มเข้ามาแสดงว่าการติดตั้งเรียบร้อยดี ไม่มีปัญหา



Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับเรา เนื้อรัตน์ ลงทะเบียน

รหัส	ชื่อหนังสือ	ราคา	หมวดหนังสือ	รูปภาพ
3	บัญชีเบื้องต้น	500.00	บัญชี	
2	สามก๊ก	1,500.00	นวนิยาย	

เมื่อกดบุํมเพิ่มข้อมูล ต่อไปเราจะมาสร้างฟอร์มเพิ่มข้อมูลหนังสือ โดยเราต้องสร้าง views รองรับ และเขียนเมธอดที่ Controller ให้ตรงกับ เมธอดที่ลิงก์ไปด้วย ดังนี้

1. เปิดไฟล์ BooksController.php ที่เมธอด create() ให้เขียนให้ดีเพื่อ render หน้า views ดังนี้

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

2. มาที่ views ให้สร้างไฟล์ create.blade.php ในโฟลเดอร์ books ดังนี้

```
@extends('layouts.app')

@section('content')


เพิ่มข้อมูลหนังสือ



{!! Form::open(array('url' => 'books', 'files' => true)) !!}


<?= Form::label('title', 'ชื่อหนังสือ'); ?>
<?= Form::text('title', null, ['class' => 'form-control', 'placeholder' => 'ชื่อหนังสือ']); ?>



{!! Form::label('price', 'ราคา'); !!}
{!! Form::text('price', null, ['class' => 'form-control', 'placeholder' => 'ตั้ง 100, 100.25']); !!}



<?= Form::label('typebooks_id', 'ประเภทหนังสือ'); ?>


```

```

        <?= Form::select('typebooks_id',
App\Models\TypeBooks::all()->pluck('name', 'id'), null, ['class' =>
'form-control', 'placeholder' => 'กรุณาเลือกประเภทหนังสือ...']); ?>
    </div>

    <div class="form-group">
        {!! Form::label('image', 'รูปภาพ'); !!}
        <?= Form::file('image', null, ['class' =>
'form-control']); ?>
    </div>

    <div class="form-group">
        <?= Form::submit('บันทึก', ['class' => 'btn
btn-primary']); ?>
    </div>

    {!! Form::close() !!}

```

</div>

</div>

</div>

</div>

</div>

</div>

@endsection

3. ทดสอบโดยการคลิกปุ่ม เพิ่มข้อมูล เราจะได้หน้าเพจสำหรับเพิ่มข้อมูลรูปเบร์ร้อย พื้นที่ห้องเลือกประเภทหนังสือได้ด้วย

Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับเรา

The screenshot shows a Laravel form titled "เพิ่มข้อมูลหนังสือ". The form fields are:

- ชื่อหนังสือ**: An input field containing "เขียน 100, 100.25".
- ราคา**: An input field containing "เขียน 100, 100.25".
- ประเภทหนังสือ**: A dropdown menu showing "กรุณาเลือกประเภทหนังสือ...".
- รูปภาพ**: A file input field with the placeholder "เลือกไฟล์" and the note "ไม่ได้เลือกไฟล์ใด".
- บันทึก**: A blue "บันทึก" button at the bottom of the form.

อธิบายเพิ่มเติม การใช้ฟอร์มนั้นจะมีต้องการเปิด และปิดฟอร์ม เช่น การเปิดฟอร์มจะใช้คำสั่ง

```
{!! Form::open(array('url' => 'books', 'files' => true)) !!}
```

```
และปิดฟอร์มจะใช้คำสั่ง {!! Form::close() !!}
```

หากฟอร์มของเรามีการอัพโหลดไฟล์ด้วย ให้ระบุ 'files' => true ตอนเปิดฟอร์มนั้นเอง

การจัดข้อมูลใส่ใน dropdown list เอาจสามารถเรียกใช้ method pluck() ได้เลย ดังอย่างเช่น

```
<?= Form::select('typebooks_id', App\Models\TypeBooks::all()->pluck('name', 'id'), null, ['class' => 'form-control', 'placeholder' => 'กรุณาเลือกประเภทหนังสือ...']); ?>
```

### การตรวจสอบความถูกต้องของข้อมูล (Validation)

เมื่อสร้างฟอร์มเสร็จเรียบร้อยแล้ว ก่อนกดปุ่มนี้ควรมีความตรวจสอบความถูกต้องของข้อมูลในฟอร์มก่อน เช่น ตรวจสอบว่า

ผู้ใช้กรอกข้อมูลมาหรือไม่ กรอกข้อมูลมาถูกต้องตามรูปแบบหรือเปล่า เป็นต้น ใน Laravel จะมีกฎในการตรวจสอบความถูกต้องของข้อมูล สำหรับมาให้แล้ว สามารถกำหนดได้ตามสะดวก

Note: สามารถดูกฎ (rules) ทั้งหมดได้ที่นี่ <https://laravel.com/docs/master/validation#available-validation-rules>

### ขั้นตอนการสร้าง และตรวจสอบความถูกต้องของข้อมูลจากฟอร์ม

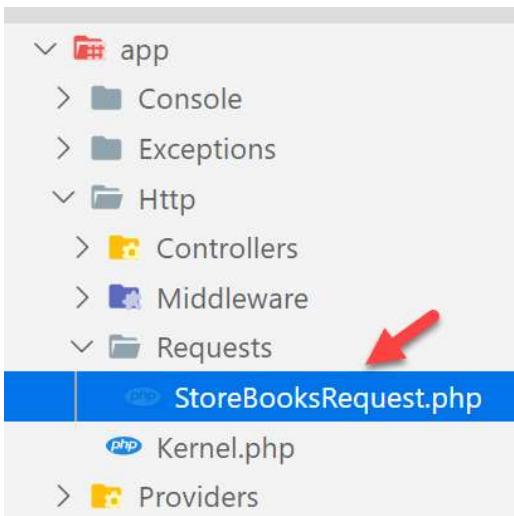
- สร้าง request สำหรับตรวจสอบความถูกต้องของข้อมูล โดยให้เข้าไปในโปรดักชันเรา แล้วเปิด Command Prompt จากนั้น พิมพ์คำสั่ง

```
php artisan make:request StoreBooksRequest แล้วกด enter
```

Administrator: Command Prompt

```
C:\xampp\htdocs\laravel >php artisan make:request StoreBooksRequest
Request created successfully.
```

2. ไฟล์ StoreBooksRequest.php จะถูกสร้างขึ้นที่ไฟลเดอร์ app\Http\Requests\



3. เปิดไฟล์ StoreBooksRequest.php เพื่อเขียนโค้ดກฎการตรวจสอบ และข้อความตัวตอบที่จะแสดงให้กับผู้ใช้ทราบ ดังนี้

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class StoreBooksRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        // return false;
        return true; //หากกำหนดเป็น false จะต้องล็อกอินก่อน
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'title' => 'required',
            'price' => 'required',
        ];
    }
}
```

```

        'typebooks_id' => 'required',
        'image' => 'mimes:jpeg,jpg,png',
    ];
}

public function messages() {
    return [
        'title.required' => 'กรุณากรอกชื่อหนังสือ',
        'price.required' => 'กรุณากรอกราคา',
        'typebooks_id.required' => 'กรุณาเลือกหมวดหนังสือ',
        'image.mimes' => 'กรุณาเลือกไฟล์ภาพนามสกุล jpeg,jpg,png',
    ];
}

```

4. เปิดไฟล์ BooksController.php เพื่อเรียกใช้งาน (use) StoreBooksRequest เข้ามา และกำหนดชนิดของ request ที่เมื่อคัด store จะถูกเปลี่ยนเป็น StoreBooksRequest แทน ดังนี้

```
<?php
```

```
namespace App\Http\Controllers;
```

```

use Illuminate\Http\Request;
use App\Models\Books; //อย่าลืม use โน้ตเดลเข้ามาใช้งาน
use App\Http\Requests\StoreBooksRequest;
```

```

class BooksController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index() {
        $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
    }
}
```

```
        return view('books/index',['books' => $books]);  
    }  
  
    /**  
     * Show the form for creating a new resource.  
     *  
     * @return \Illuminate\Http\Response  
     */  
    public function create()  
    {  
        return view('books.create');  
    }  
  
    /**  
     * Store a newly created resource in storage.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @return \Illuminate\Http\Response  
     */  
    public function store(StoreBooksRequest $request)  
    {  
  
    }  
  
    /**  
     * Display the specified resource.  
     *  
     * @param int $id  
     * @return \Illuminate\Http\Response  
     */  
    public function show($id)
```

```
{  
    //  
}  
/**  
 * Show the form for editing the specified resource.  
 *  
 * @param int $id  
 * @return \Illuminate\Http\Response  
 */  
  
public function edit($id)  
{  
    //  
}  
  
/**  
 * Update the specified resource in storage.  
 *  
 * @param \Illuminate\Http\Request $request  
 * @param int $id  
 * @return \Illuminate\Http\Response  
 */  
  
public function update(Request $request, $id)  
{  
    //  
}  
  
/**  
 * Remove the specified resource from storage.  
 *  
 * @param int $id  
 * @return \Illuminate\Http\Response  
 */
```

```

*/
public function destroy($id)
{
    //
}

}

```

5. ต่อมาหากผู้ใช้กดบันทึก เจ้าตรวจสอบข้อความ errors บอกด้วย โดยแทรกโค้ดเข้าไปที่ไฟล์ (resources\views\books\create.blade.php) ในส่วนที่ต้องการแสดงข้อความ ดังนี้

```

@if (count($errors) > 0)
    <div class="alert alert-warning">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

```

6. ทดสอบโดยการกดปุ่มบันทึกได้เลยครับ

Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับ

เพิ่มข้อมูลหนังสือ

- กรุณากรอกชื่อหนังสือ
- กรุณากรอกราคา
- กรุณาเลือกหมวดหมู่หนังสือ

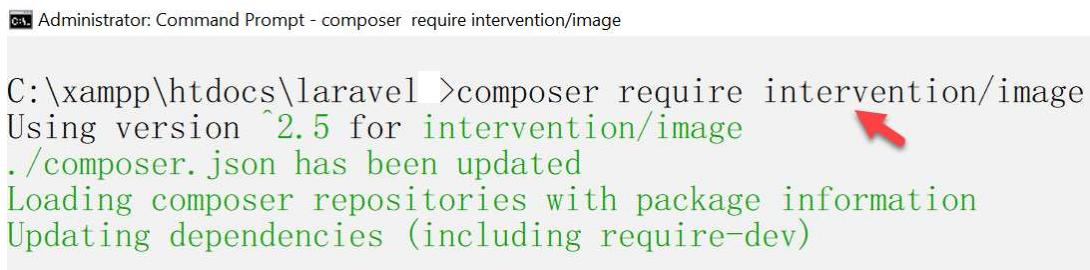
ชื่อหนังสือ

## การติดตั้ง Image Library เพื่อเตรียมพร้อมก่อนอัปโหลดไฟล์

เมื่อมีการอัปโหลดไฟล์จากฟอร์มของผู้ใช้ บางครั้งรูปภาพที่ถูกอัปโหลดเข้ามาอาจมีขนาดใหญ่ หรือมีขนาดไม่พอดี ดังนั้นเราจะติดตั้ง Library สำหรับจัดการรูปภาพต่างๆ เช่น การย่อขนาดรูป เป็นต้น จากเว็บนี้ <http://image.intervention.io/>

### ขั้นตอนการติดตั้ง Intervention Image Library

เปิด Command Prompt ขึ้นมาแล้วพิมพ์ `composer require intervention/image` เพื่อติดตั้ง จากนั้นกด enter



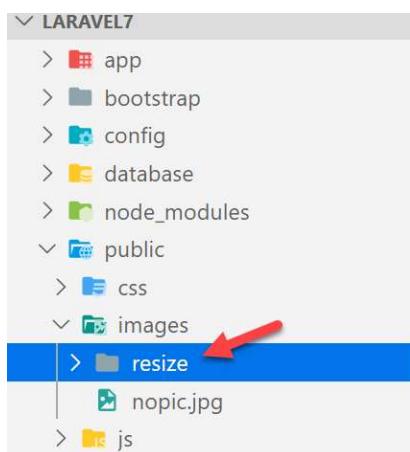
```
Administrator: Command Prompt - composer require intervention/image
C:\xampp\htdocs\laravel >composer require intervention/image
Using version ^2.5 for intervention/image
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

Note: วิธีการติดตั้งเพิ่มเติม ดูได้จากที่นี่ [http://image.intervention.io/getting\\_started/installation](http://image.intervention.io/getting_started/installation)

## การเพิ่มข้อมูลหนังสือ (books) และอัปโหลดไฟล์ภาพ

หลังจากติดตั้ง Library สำหรับจัดการรูปภาพเรียบร้อย ต่อไปให้เราเขียนโค้ดเพื่อเพิ่มข้อมูล และอัปโหลดรูปภาพ พร้อมทั้งย่อภาพด้วย การเขียนโค้ดสำหรับเพิ่มข้อมูล มีขั้นตอน ดังนี้

- ให้สร้างโฟลเดอร์ `resize` เพื่อเก็บภาพที่ได้ทำการย่อไว้ในโฟลเดอร์ `public\images` ดังภาพ



2. เปิดไฟล์ BooksController.php ขึ้นมาให้ use Image และ String Helpers เข้ามาใช้งาน พิริ่งทั้งตัวช่วยนี้คือตัวช่วยที่เมื่อคุณ调用 store() เพื่อบันทึกข้อมูล ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Books; // อย่าลืม use ไม่เดลเข้ามาใช้งาน
use App\Http\Requests\StoreBooksRequest;
use Image; // เรียกใช้ library จัดการรูปภาพเข้ามาใช้งาน
use Illuminate\Support\Str; // นำ Helpers String เข้ามาใช้งาน

class BooksController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $books = Books::with('typebooks')->orderBy('id', 'desc')-
>paginate(5);
        return view('books/index',[ 'books' => $books]);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('books.create');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(StoreBooksRequest $request)
    {
        $book = new Books();
        $book->title = $request->title;
        $book->price = $request->price;
    }
}
```

```

$book->typebooks_id = $request->typebooks_id;
if ($request->hasFile('image')) {
    $filename = Str::random(10) . '.' . $request-
>file('image')->getClientOriginalExtension();
    $request->file('image')-
>move(public_path() . '/images/', $filename);
    Image::make(public_path() . '/images/' . $filename)-
>resize(50, 50)->save(public_path() . '/images/resize/' . $filename);
    $book->image = $filename;
} else {
    $book->image = 'nopic.jpg';
}
$book->save();
return redirect()->action('BooksController@index');
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    //
}

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    //
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //
}

```

```

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    //
}

```

อธิบายโค้ด เมื่อคุณ调用 store() หากการตรวจสอบข้อมูลถูกต้อง เราจะรับ request และค่าจากฟอร์มมาทั้งหมด โดยมีความสามารถตรวจสอบได้ว่าผู้ใช้ได้เลือกอัปโหลดไฟล์มาได้หรือไม่ สามารถตรวจสอบได้โดยใช้ hasFile() หากอัปโหลดมาเราจะสุมชื่อไฟล์ใหม่เพื่อไม่ให้ชื่อซ้ำกัน พร้อมกับอัปโหลดไฟล์เก็บไว้ที่โฟลเดอร์ images หลังจากนั้นก็ย่อขนาดไฟล์ให้เหลือขนาด 50x50 แล้วเก็บไว้ที่โฟลเดอร์ images/resize หากผู้ใช้ไม่ได้อัปโหลดภาพเข้ามาก็ให้กำหนดชื่อว่าเป็น nopic.jpg และแก้ไขสั่ง save() เพื่อบันทึกลงในตาราง

- เปิดไฟล์ resources\views\books\index.blade.php เพื่อแก้ไข path รูปภาพให้ถูกต้องในที่นี่เราเก็บรูปที่ย่อขนาดแล้วไว้ในโฟลเดอร์ images/resize แก้ไขดังนี้

```

<td>
<a href="{{ asset('images/'.$book->image) }}>
    
</a>
</td>

```

- จากนั้นให้ copy รูปภาพ nopic.jpg ไปวางไว้ในโฟลเดอร์ /images/resize/ และย่อภาพด้วยเพื่อการแสดงผลที่ถูกต้อง

5. ทดลองเพิ่มข้อมูลหนังสือ 1 รายการ ก็เป็น servo จะเรียบง่าย

Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับเรา เข้าระบบ ลงทะเบียน

แสดงข้อมูลหนังสือ จำนวนทั้งหมด 4 เล่ม				
รหัส	ชื่อหนังสือ	ราคา	หมวดหนังสือ	รูปภาพ
4	Learning React	1,500.00	คอมพิวเตอร์	
3	นักเขียนองค์	500.00	นักเขียน	 NOPIC
2	สามก๊ก	1,500.00	นวนิยาย	 NOPIC
1	การ์ตูน panda	100.00	นวนิยาย	 NOPIC

### สร้างฟอร์มแก้ไขข้อมูลหนังสือ (books) และแก้ไขภาพที่ต้องการอัปโหลด

การสร้างฟอร์มแก้ไข เราจะต้องสร้างลิงก์เพื่อให้ผู้ใช้คลิกแล้วเปิดฟอร์มแก้ไขขึ้นมา เปิดไฟล์ resources\views\books\index.blade.php ออกครึ่งเพื่อแทรกโคดมันให้กับตาราง สำหรับการแก้ไขมีขั้นตอน ดังนี้

- ให้เพิ่มคอลัมน์การแก้ไขข้อมูลให้กับตาราง

```
@extends('layouts.app')

@section('content')


<?= link_to('books/create', $title = 'เพิ่มข้อมูล', ['class' => 'btn btn-primary'], $secure = null); ?>

<div class="card mt-3">


```

```

<div class="card-header h3">
    ແສດງຂໍ້ມູນທັງສອງ ຈຳນວນທັງໝາດ {{ $books->total() }} ເລີ່ມ
</div>

<div class="card-body">

    <table class="table table-striped">
        <tr>
            <th>ຮັດສິນ</th>
            <th>ຊື່ອໜັງສືອ</th>
            <th>ຈາກາ</th>
            <th>ໜ່າຍດ້ານັ້ນສືອ</th>
            <th>ຮູບພາບ</th>
            <th>ແກ່ໄໝ</th>
        </tr>
        @foreach ($books as $book)
        <tr>
            <td>{{ $book->id }}</td>
            <td>{{ $book->title }}</td>
            <td>{{ number_format($book->price,2) }}</td>
            <td>{{ $book->typebooks->name }}</td>
            <td>
                <a href="{{ asset('images/resize/'.$book->image) }}></a>
            </td>
            <td>
                <a href="{{ url('/books/'.$book->id.'/edit') }}>ແກ່ໄໝ</a>
            </td>
        </tr>
        @endforeach
    </table>

```

```

<br>
{{ $books->links() }}

</div>
</div>
</div>
</div>
</div>
</div>
@endsection

```

2. เปิดไฟล์ BooksController.php ที่เมธอด edit(\$id) ให้เขียนโค้ดเพื่อแสดงเฉพาะແղວທີ່ສົ່ງມາພ້ອມທີ່ render view ດ້ວຍ ດັນນີ້

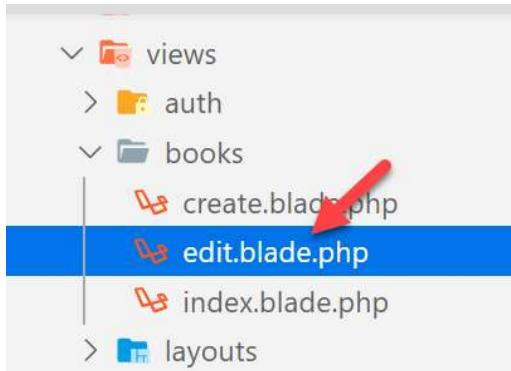
```

public function edit($id)
{
    $book = Books::findOrFail($id);

    return view('books.edit', ['book' => $book]);
}

```

3. มาที่โฟลเดอร์ของ views ให้สร้างไฟล์ edit.blade.php ในโฟลเดอร์ books เพื่อรับการ render จาก Controller ດັນນີ້



4. ในการเก็บข้อมูลเวลาจะใช้คิวชีทเรียกว่า Model Binding หรือการผูกค่าไม่เดลเข้ากับ input ต่างๆในฟอร์ม ดังนี้

```

@extends('layouts.app')

@section('content')


แก้ไขข้อมูลหนังสือ {{ $book->title }}



@if (count($errors) > 0)


@foreach ($errors->all() as $error)
- {{ $error }}

@endforeach


@endif

<?= Form::model($book, array('url' => 'books/' . $book->id,
'method' => 'put','files' => true)) ?>



<?= Form::label('title', 'ชื่อหนังสือ'); ?>
<?= Form::text('title', null, ['class' => 'form-control',
'placeholder' => 'ชื่อหนังสือ']); ?>



{!! Form::label('price', 'ราคา'); !!}
{!! Form::text('price', null, ['class' => 'form-control',
'placeholder' => 'ตั้ง 100, 100.25']); !!}


```

```

        {!! Form::label('typebooks_id', 'ประเภทหนังสือ'); !!}
<?= Form::select('typebooks_id', App\Models\TypeBooks::all()-
>pluck('name', 'id'), null, ['class' => 'form-control', 'placeholder' =>
'กรุณาเลือกประเภทหนังสือ...']); ?>
</div>

<div>
<a href="{{ asset('images/'.$book->image) }}></a>
</div>

<br>

<div class="form-group">
{!! Form::label('image', 'แก้ไขรูปภาพ'); !!}
<?= Form::file('image', null, ['class' => 'form-control']) ?>
</div>

<div class="form-group">

<?= Form::submit('บันทึก', ['class' => 'btn btn-primary']); ?>

</div>

{!! Form::close() !!}

</div>
</div>
</div>
</div>
</div>
@endsection

```

5. ทดลองเลือกรายการเพื่อแก้ไขด้วยฟอร์ม ดังรูป

6. เปิดไฟล์ BooksController.php เพื่อเขียนโค้ดที่เมธอด update() เพื่อแก้ไขข้อมูล และ นำ File เข้ามาด้านบน เพื่อเรียกใช้การ upload ด้วยในกรณีผู้ใช้อัปโหลดไฟล์มาใหม่ ดังนี้

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use App\Models\Books; //อย่าลืม use ไมเดลเข้ามาใช้งาน
use App\Http\Requests\StoreBooksRequest;
use Image; //เรียกใช้ library จัดการรูปภาพเข้ามาใช้งาน
use Illuminate\Support\Str; //นำ Helpers String เข้ามาใช้งาน
use File;
```

```
class BooksController extends Controller
```

```
{
```

```

/**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
    return view('books/index',['books' => $books]);
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('books.create');
}

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(StoreBooksRequest $request)
{
    $book = new Books();
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;
    if ($request->hasFile('image')) {
        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/', $filename);
    }
}

```

```

Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .
$filename);

$book->image = $filename;

} else {

    $book->image = 'nopic.jpg';

}

$book->save();

return redirect()->action('BooksController@index');

}

/**

 * Display the specified resource.

 *

 * @param int $id

 * @return \Illuminate\Http\Response

 */

public function show($id)

{
    //

}

/**

 * Show the form for editing the specified resource.

 *

 * @param int $id

 * @return \Illuminate\Http\Response

 */

public function edit($id)

{
    $book = Books::findOrFail($id);

    return view('books.edit', ['book' => $book]);

}

/**

 * Update the specified resource in storage.

 *

```

```

 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $book = Books::find($id);
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;

    if ($request->hasFile('image')) {

        // delete old file before update
        if ($book->image != 'nopic.jpg') {
            File::delete(public_path() . '\\images\\' . $book->image);
            File::delete(public_path() . '\\images\\resize\\' . $book->image);
        }

        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/', $filename);
        Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .
$filename);
        $book->image = $filename;
    }

    $book->save();

    return redirect()->action('BooksController@index');
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)

```

```
{
}
}
```

7. ทดสอบแก้ไขข้อมูล ทั้งในส่วนของข้อมูลปกติ และรูปภาพ เป็นอันเสร็จเรียบร้อย

### **สร้างฟอร์มการลงทะเบียนข้อมูลหนังสือ (books)**

การลงทะเบียนข้อมูลเขียนเดียวกันให้เราเพิ่มคอลัมน์อีก 1 คอลัมน์ เปิดไฟล์ resources\views\books\index.blade.php อีกครั้งเพื่อแทรกคอลัมน์ใหม่กับตาราง สำหรับใช้ในการลงทะเบียนข้อมูล ดังนี้

```
@extends('layouts.app')

@section('content')


<?= link_to('books/create', $title = 'เพิ่มข้อมูล', ['class' => 'btn btn-primary'], $secure = null); ?>

<div class="card mt-3">
    <div class="card-header h3">
        แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{ $books->total() }} เล่ม
    </div>
    <div class="card-body">
        <table class="table table-striped">
            <tr>
                <th>รหัส</th>


```

```

<th>ឈ្មោះសីអ៊ូ</th>
<th>រាជការ</th>
<th>អំពើដែនដងសីអ៊ូ</th>
<th>ខ្លួនភាព</th>
<th>កំណើន</th>
<th>លប់</th>
</tr>

@foreach ($books as $book)

<tr>

<td>{{ $book->id }}</td>
<td>{{ $book->title }}</td>
<td>{{ number_format($book->price,2) }}</td>
<td>{{ $book->typebooks->name }}</td>
<td>
<a href="{{ asset('images/'.$book->image) }}"></a>
</td>
<td>
<a href="{{ url('/books/'.$book->id.'/edit') }}"><span>ແກ້ໄຂ</span></a>
</td>
<td>

<?= Form::open(array('url' => 'books/' . $book->id, 'method' => 'delete','onsubmit' => 'return confirm("នៅឲ្យវាតែងការលប់ខ្លួនមូល?");')) ?>
<button type="submit" class="btn btn-danger">លប់</button>
{!! Form::close() !!}
</td>

</tr>

@endforeach

</table>

```

```

<br>
{{ $books->links()  }}

</div>
</div>
</div>
</div>
</div>
@endsection

```

อธิบายโค้ดเพิ่มเติม ในการลบข้อมูลเราต้องเพิ่มในส่วนของ 'method' => 'delete' และเปิด-ปิดฟอร์มด้วย

Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับเรา

รหัส	ชื่อหนังสือ	ราคา	หมวดหนังสือ	รูปภาพ	แก้ไข	ลบ
4	React 16	1,500.00	คอมพิวเตอร์		แก้ไข	ลบ
3	บัญชีเบื้องต้น	500.00	บัญชี	 NOPIC	แก้ไข	ลบ
2	สามก๊ก	1,500.00	นวนิยาย	 NOPIC	แก้ไข	ลบ
1	การ์ตูน panda	100.00	นวนิยาย	 NOPIC	แก้ไข	ลบ

จากนั้นให้เราเขียนโค้ดสำหรับการลบหนังสือได้ที่เมธอด destroy(\$id) ที่ไฟล์ BooksController.php ดังนี้

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
use App\Models\Books; //อย่าลืม use ไมเดลเข้ามาใช้งาน
```

```

use App\Http\Requests\StoreBooksRequest;
use Image; //ເຮັດໃຫ້ library ຈົດກາຣຸປັກພເຂົ້າມາໃຫ້ງານ
use Illuminate\Support\Str; //ນໍາ Helpers String ເຂົ້າມາໃຫ້ງານ
use File;

class BooksController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
        return view('books/index',['books' => $books]);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('books.create');
    }

    /**
     * Store a newly created resource in storage.
     *

```

```

* @param \Illuminate\Http\Request $request
* @return \Illuminate\Http\Response
*/
public function store(StoreBooksRequest $request)
{
    $book = new Books();
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;
    if ($request->hasFile('image')) {
        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/', $filename);
        Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .
$filename);
        $book->image = $filename;
    } else {
        $book->image = 'nopic.jpg';
    }
    $book->save();
    return redirect()->action('BooksController@index');
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
*/
public function show($id)
{
    //
}

```

```

}

/***
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $book = Books::findOrFail($id);
    return view('books.edit', ['book' => $book]);
}

/***
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $book = Books::find($id);
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;

    if ($request->hasFile('image')) {
        // delete old file before update
    }
}

```

```

if ($book->image != 'nopic.jpg') {

    File::delete(public_path() . '\\images\\' . $book->image);

    File::delete(public_path() . '\\images\\resize\\' . $book->image);

}

$filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();

$request->file('image')->move(public_path() . '/images/', $filename);

Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .

$filename);

$book->image = $filename;

}

$book->save();

return redirect()->action('BooksController@index');

}

/**

 * Remove the specified resource from storage.

 *

 * @param int $id

 * @return \Illuminate\Http\Response

 */

public function destroy($id)

{

    $book = Books::find($id);

    if ($book->image != 'nopic.jpg') {

        File::delete(public_path() . '\\images\\' . $book->image);

        File::delete(public_path() . '\\images\\resize\\' . $book->image);

    }

    $book->delete();
}

```

```

        return redirect()->action('BooksController@index');

    }

}

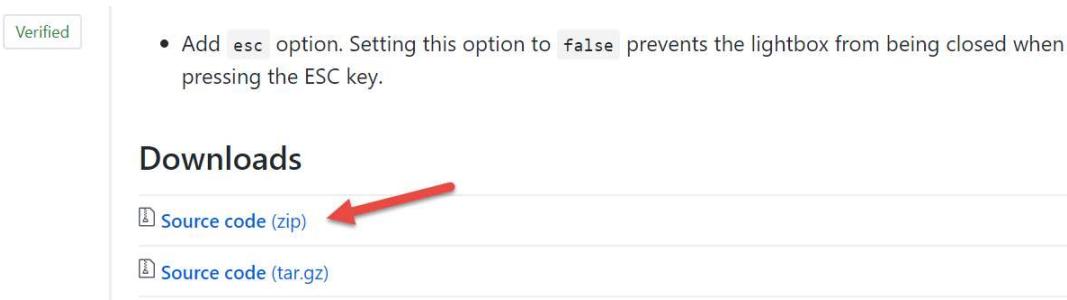
```

การลบข้อมูลที่ดีควรลบไฟล์ภาพออกไปด้วย ในกรณีนี้เราเช็ค if ว่าถ้าชื่อไฟล์ไม่เท่ากับ nopic.jpg ก็ให้ลบไฟล์ได้เลย

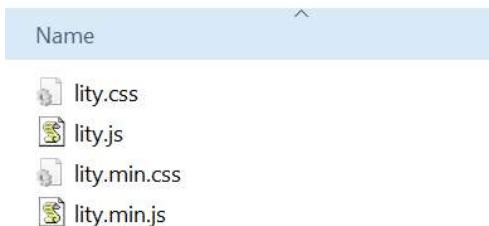
### การทำ responsive lightbox โดยใช้ Lity Library

Lity เป็น lightbox ที่ช่วยให้การแสดงรูปภาพน่าสนใจ และสวยงามมากขึ้น เราสามารถเข้าไปดูการใช้งาน ได้ที่ <http://sorgalla.com/lity/> ตัวอย่างนี้ เราจะเพิ่ม lity เข้าไปใช้งานในหน้าของหนังสือ เมื่อผู้ใช้คลิกภาพเล็ก (ภาพที่ resize) ให้แสดงภาพใหญ่ในไฟลเดอร์ images/ นั่นเอง มีขั้นตอนดังนี้

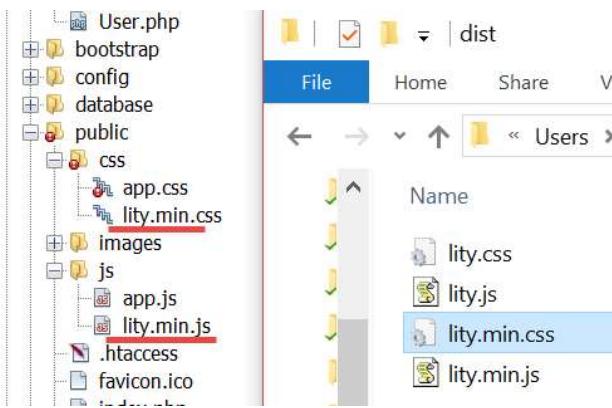
- ดาวน์โหลด lity ได้ที่ลิงก์ <https://github.com/jsor/lity/releases/latest> คลิกดาวน์โหลดที่ Source code (zip)



- ดาวน์โหลดเสร็จแล้วให้แตกไฟล์ (extract) zip ที่ได้มา ไฟล์ของ library จะอยู่ที่ไฟลเดอร์ `dist/`



- จากนั้นให้ copy ไฟล์ lity.min.css ไปวางไว้ที่ public/css และ copy ไฟล์ lity.min.js ไปวางไว้ที่ public/js (หากยังไม่ได้สร้างไฟลเดอร์ css และ js ใน public ให้สร้างได้เลยครับ) หรือใช้วิธี drag&drop เข้ามาใน Editor ก็ได้เช่นเดียวกัน



หมายเหตุ สามารถเลือกใช้ lity cdn ได้หากไม่อยากดาวน์โหลดตามลิงก์ <https://cdnjs.com/libraries/lity>

4. เปิดไฟล์ layouts ที่ resources\views\layouts\app.blade.php เพิ่มแทรกโค้ด css และ js ของ lity ดังนี้

```
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token() }}>

    <title>{{ config('app.name', 'Laravel') }}</title>

    <!-- Scripts -->
    <script src="{{ asset('js/app.js') }}" defer></script>
    <script src="{{ asset('js/lity.min.js') }}" defer></script>

    <!-- Fonts -->
    <link rel="dns-prefetch" href="//fonts.gstatic.com">
    <link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">
```

```

<!-- Styles -->
<link href="{{ asset('css/app.css') }}" rel="stylesheet">

<link href="{{ asset('css/lity.min.css') }}" rel="stylesheet">

</head>
<body>
<div id="app">
  <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
    <div class="container">
      <a class="navbar-brand" href="{{ url('/') }}"
        >{{ config('app.name', 'Laravel') }}

```

```

<li class="nav-item">
    <a class="nav-link" href="{{ route('books') }}">หนังสือ</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="{{ route('typebooks') }}">ประเภทหนังสือ</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="{{ route('about') }}">เกี่ยวกับเรา</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="{{ route('login') }}">เข้าระบบ</a>
</li>

@if (Route::has('register'))
    <li class="nav-item">
        <a class="nav-link" href="{{ route('register') }}">ลงทะเบียน</a>
    </li>
@endif

@else
    <li class="nav-item dropdown">
        <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-
        toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
            {{ Auth::user()->name }} <span class="caret"></span>
        </a>

        <div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
            <a class="dropdown-item" href="{{ route('logout') }}"
            onclick="event.preventDefault();
                    document.getElementById('logout-form').submit();">

```

```

{{ __('Logout') }}

</a>

<form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
    @csrf
</form>

</div>

</li>

@endguest

</ul>

</div>

</div>

</nav>

<main class="py-4">
    @yield('content')
</main>

</div>

@yield('footer')

</body>

</html>

```

5. เปิดไฟล์ views ที่ resources\views\books\index.blade.php เพื่อกำหนด attribute data-lity ใน tag html ที่ต้องการ ดังนี้
- ```

<a href="{{ asset('images/'.$book->image) }}" data-lity></a>

```
- ให้คลิกที่ไฟล์ resources\views\books\index.blade.php

```
@extends('layouts.app')
```

```

@section('content')



<?= link_to('books/create', $title = 'ເພີ່ມຂໍ້ອມຸດ', ['class' => 'btn btn-primary'], $secure = null); ?>



<div class="card-header h3">
    ແສດງຂໍ້ອມຸດທັງສືອ ຈຳນວນທັງໝາດ {{ $books->total() }} ເລີມ
</div>



<table class="table table-striped">
    <tr>
        <th>ຮັດສ</th>
        <th>ຊື່ອໜັງສືອ</th>
        <th>ຮາຄາ</th>
        <th>ໜ່ວດທັງສືອ</th>
        <th>ຮູ່ປະກາພ</th>
        <th>ແກ້ໄຂ</th>
        <th>ລົບ</th>
    </tr>
    @foreach ($books as $book)
        <tr>
            <td>{{ $book->id }}</td>
            <td>{{ $book->title }}</td>
            <td>{{ number_format($book->price,2) }}</td>
            <td>{{ $book->typebooks->name }}</td>


```

```

<td>
    <a href="{{ asset('images/'.$book->image) }}" data-lity></a>
</td>

<td>
    <a href="{{ url('/books/'.$book->id.'/edit') }}">ແກ້ໄຂ</a>
</td>

<td>

<?= Form::open(array('url' => 'books/' . $book->id, 'method' => 'delete','onsubmit' => 'return
confirm("ແນ່ໃຈວ່າຕ້ອງກາລບື້ອມຸດ?");')) ?>
    <button type="submit" class="btn btn-danger">ລົບ</button>
    {!! Form::close() !!}
</td>
</tr>
@endforeach
</table>
<br>
{{ $books->links() }}

```

```

</div>
</div>
</div>
</div>
</div>
@endsection

```

6. ทดสอบโดยการคลิกที่ภาพเล็กในตาราง เมื่อคลิกแล้วรูปภาพจะขยายให้ใหญ่ขึ้น

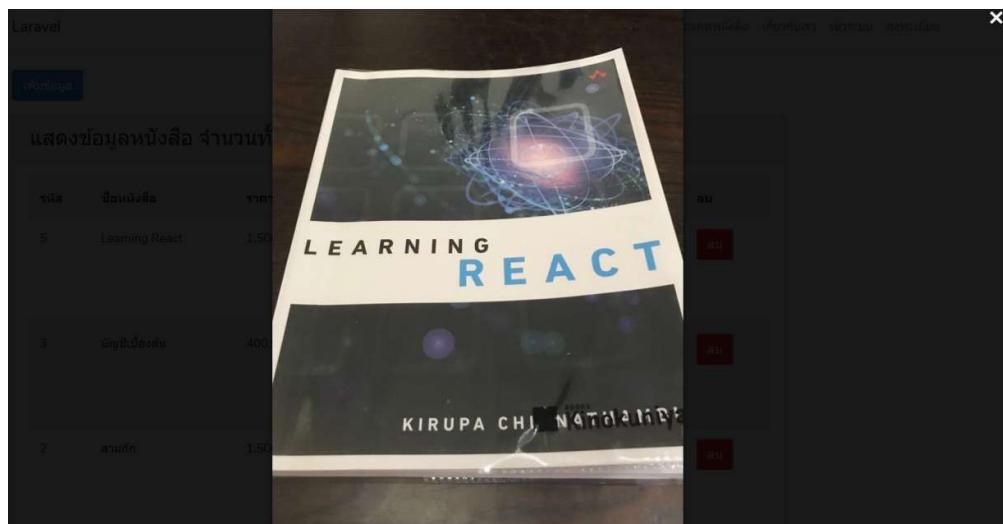
Laravel

หนังสือ ประเภทหนังสือ เกี่ยวกับเรา

เพิ่มข้อมูล

แสดงข้อมูลหนังสือ จำนวนทั้งหมด 4 เล่ม

| รหัส | ชื่อหนังสือ        | ราคา     | หมวดหนังสือ | รูปภาพ                                                                             | แก้ไข | ลบ |
|------|--------------------|----------|-------------|------------------------------------------------------------------------------------|-------|----|
| 5    | Learning React     | 1,500.00 | คอมพิวเตอร์ |  | แก้ไข | ลบ |
| 3    | มักกี้เขียนเองด้วย | 400.00   | มักกี้      |  | แก้ไข | ลบ |
| 2    | สามก๊ก             | 150.00   |             |                                                                                    |       |    |



## บทที่ 7 การใช้งาน Sessions และการกำหนดสิทธิ์ผู้ใช้

### การใช้งาน Session

Session เป็นตัวแปรที่เจ้าสามารถใช้งานข้อมูลหน้าเพจต่างๆได้ หากใครเขียน PHP ปกติมาแล้วคงคุ้นเคยกับคำสั่ง `$_SESSION` ดี ลักษณะการใช้งานก็เหมือนกันครับ

- คำสั่งการใส่ค่าข้อมูลเข้าไปใน session ใช้เมธอด `put()`  
`$request->session()->put('key', 'value');`
- การเข้าถึง key ในหน้าต่างๆ ใช้เมธอด `get()`  
`$value = $request->session()->get('key');`
- ใช้ `if` สำหรับตรวจสอบว่ามี key session หรือไม่ (ใช้เมธอด `has()`)  
`if ($request->session()->has('users')) { // }`
- คำสั่งสำหรับลบ key session ใช้เมธอด `forget()` และ `flush()` (ใช้คู่กัน)  
`$request->session()->forget('key');`  
`$request->session()->flush();`

### การใช้งาน Flash Data

Flash Data เป็น session ที่มีอยู่ใช้งานช่วงเวลา ใช้ได้ใน request หนึ่งๆ และจะหายไปเมื่อมี request ใหม่เกิดขึ้น เหมาะสำหรับการทำการโต้ตอบกับผู้ใช้ ณ ขณะนั้น เช่น โต้ตอบการเพิ่มข้อมูล หรือลบข้อมูลเว็บร้อยละ เป็นต้น

เพื่อให้เห็นการนำไปใช้จะขอเสนอการทำ flash data ร่วมกับ Sweet Alert Library คือ เมื่อผู้ใช้เพิ่มข้อมูลหนังสือ ก็ให้มี alert บอกว่า “บันทึกข้อมูลเรียบร้อยแล้ว”

Note: เว็บไซต์ของ Sweet Alert <https://sweetalert.js.org>

- เปิดไฟล์ resources\views\layouts\app.blade.php และหา javascript ให้ด้านล่าง เพื่อนำ Sweet Alert Library เข้ามาใช้งาน ดังนี้

```
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
```

การติดตั้ง sweetalert เราจะใช้วิธีการแทรก script ในรูปแบบของ CDN

ไฟล์ app.blade.php

```
<!doctype html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- CSRF Token -->
<meta name="csrf-token" content="{{ csrf_token() }}>

<title>{{ config('app.name', 'Laravel') }}</title>

<!-- Scripts -->
<script src="{{ asset('js/app.js') }}" defer></script>
<script src="{{ asset('js/lity.min.js') }}" defer></script>
<script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>

<!-- Fonts -->
<link rel="dns-prefetch" href="//fonts.gstatic.com">
<link href="https://fonts.googleapis.com/css?family=Nunito" rel="stylesheet">

<!-- Styles -->
<link href="{{ asset('css/app.css') }}" rel="stylesheet">

<link href="{{ asset('css/lity.min.css') }}" rel="stylesheet">
```

```

</head>

<body>

<div id="app">

<nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">

<div class="container">

<a class="navbar-brand" href="{{ url('/') }}">
{{ config('app.name', 'Laravel') }}
</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="{{
__('Toggle navigation') }}">
<span class="navbar-toggler-icon"></span>
</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<!-- Left Side Of Navbar -->
<ul class="navbar-nav mr-auto">

</ul>

<!-- Right Side Of Navbar -->
<ul class="navbar-nav ml-auto">

<!-- Authentication Links -->
@guest

<li class="nav-item">
<a class="nav-link" href="{{ route('books') }}">หนังสือ</a>
</li>

<li class="nav-item">
<a class="nav-link" href="{{ route('typebooks') }}">ประเภทหนังสือ</a>

```

```

</li>

<li class="nav-item">
    <a class="nav-link" href="{{ route('about') }}">ເກື່ອງກັບເຮົາ</a>
</li>

<li class="nav-item">
    <a class="nav-link" href="{{ route('login') }}">ເຂົ້າຈະບັບ</a>
</li>

@if (Route::has('register'))
    <li class="nav-item">
        <a class="nav-link" href="{{ route('register') }}">ລົງທະບຽນ</a>
    </li>
@endif

@else
    <li class="nav-item dropdown">
        <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-
        toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>
            {{ Auth::user()->name }} <span class="caret"></span>
        </a>

        <div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdown">
            <a class="dropdown-item" href="{{ route('logout') }}"
                onclick="event.preventDefault();
                          document.getElementById('logout-form').submit();">
                {{ __('Logout') }}
            </a>

            <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">
                @csrf
            </form>
        
```

```

        </div>
    </li>
@endguest
</ul>
</div>
</div>

</nav>

```

```

<main class="py-4">
    @yield('content')
</main>
</div>

```

```
@yield('footer')
```

```
</body>
</html>
```

- เปิดไฟล์ BooksController.php ให้เขียนโค้ดเพิ่มที่เมื่อ click store() ในส่วนของ flash data ดังนี้

```

public function store(StoreBooksRequest $request)
{
    $book = new Books();
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;
    if ($request->hasFile('image')) {
        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/' . $filename);
        Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .
$filename);
    }
}

```

```

$book->image = $filename;
} else {
    $book->image = 'nopic.jpg';
}
$book->save();
return redirect()->action('BooksController@index')->with('status', 'บันทึกข้อมูลเรียบร้อย');
}

```

3. มาที่ส่วนของ views ให้เปิดไฟล์ resources\views\books\index.blade.php เพื่อเขียนโค้ด flash data สำหรับแสดงผล ดังนี้

```

@extends('layouts.app')

@section('content')


<?= link_to('books/create', $title = 'เพิ่มข้อมูล', ['class' => 'btn btn-primary'], $secure = null); ?>

<div class="card mt-3">
    <div class="card-header h3">
        แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{ $books->total() }} เล่ม
    </div>
<div class="card-body">
    <table class="table table-striped">
        <tr>
            <th>รหัส</th>
            <th>ชื่อหนังสือ</th>


```

```

<th>ຮາຄາ</th>
<th>ໜົມວັດທິນໍ້ສືອ</th>
<th>ຮູບພາບ</th>
<th>ແກ້ໄຂ</th>
<th>ລົບ</th>

</tr>

@foreach ($books as $book)

<tr>

<td>{{ $book->id }}</td>
<td>{{ $book->title }}</td>
<td>{{ number_format($book->price,2) }}</td>
<td>{{ $book->typebooks->name }}</td>
<td>
<a href="{{ asset('images/'.$book->image) }}" data-lity></a>
</td>
<td>
<a href="{{ url('/books/'.$book->id.'/edit') }}">ແກ້ໄຂ</a>
</td>
<td>

<?= Form::open(array('url' => 'books/' . $book->id, 'method' => 'delete','onsubmit' => 'return
confirm("ແນ່ໃຈວ່າຕ້ອງກາລບຸ້ມຸດ?");')) ?>
<button type="submit" class="btn btn-danger">ລົບ</button>
{!! Form::close() !!}

</td>
</tr>

@endforeach

</table>
<br>

```

```

{{ $books->links() }}
```

```

</div>
```

```

@endsection
```

```

@section('footer')
@if (session()->has('status'))
<script>
    swal("<?php echo session()->get('status'); ?>", "", "success");
</script>
@endif

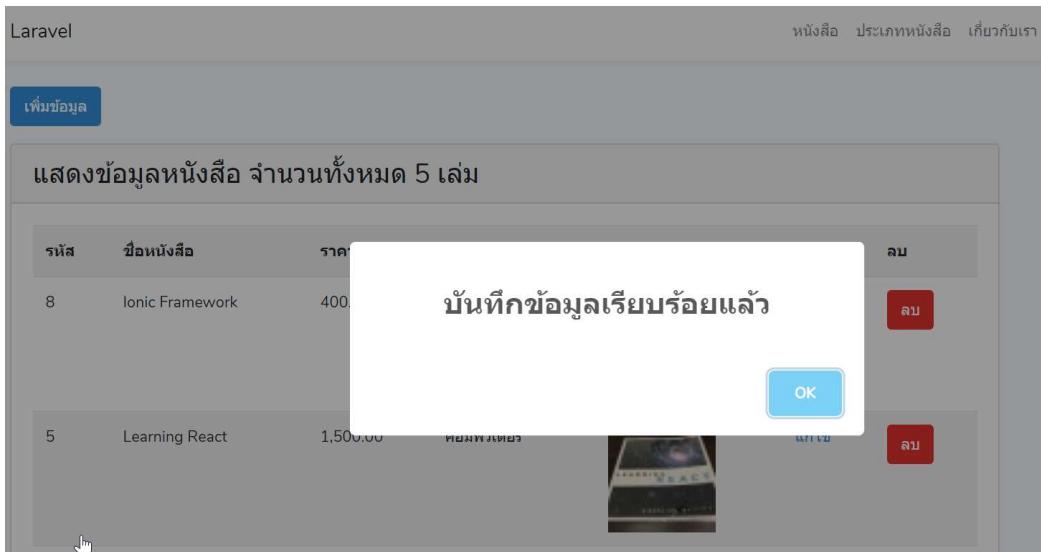
```

```

@endsection
```

ในการแสดงผลเราจะเขียน if ก่อนเพื่อตรวจสอบว่ามี key ชื่อว่า status ที่สร้างไว้ BooksController.php หรือไม่ ถ้ามีจริงก็ให้แสดงค่าข้อมูลออกมา ผ่านเมธอด get() นั้นเอง ส่วนของโค้ด JavaScript ของ Sweet Alert คือ คำสั่ง swal

4. ทดสอบเพิ่มข้อมูลหนังสือใหม่ จะได้ผลลัพธ์การทำงานดังนี้



## การกำหนดสิทธิ์ผู้ใช้

การกำหนดสิทธิ์ให้คือ เจ้าสามารถอนุญาต หรือไม่อนุญาตให้เข้าถึงในส่วนต่างๆ ของระบบ เราสามารถเปลี่ยนกำหนดได้ที่ส่วนของ Controller

ตัวอย่าง การไม่อนุญาตให้ผู้ใช้งาน BooksController และการอนุญาตบางเมธอด

1. ลำดับแรกเราจะต้องเขียนโค้ด route ที่เราต้องการจำกัดสิทธิ์ วางไว้ด้านล่างในส่วนโค้ด Auth::routes(); เปิดไฟล์ routes\web.php แก้ไขโค้ดดังนี้

<?php

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

```
/*
|-----|
| Web Routes
```

```

| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

```

```
Route::get('/typebooks', [App\Http\Controllers\TypeBooksController::class, 'index'])->name('typebooks');
```

```
Route::get('/typebooks/destroy/{id}', [App\Http\Controllers\TypeBooksController::class, 'destroy']);
```

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

```
Auth::routes();
```

//ต้องมี method index ว่า books

```
Route::resource('books', App\Http\Controllers\BooksController::class)->name('index',
'books');
```

2. ลำดับต่อมาเมื่อเขียนโค้ดแล้ว ให้เปิดไฟล์ BooksController.php เพื่อเขียน constructor สำหรับกำหนดลิฟท์ ดังนี้

```
public function __construct() {
    $this->middleware('auth');
}
```

เพียงเท่านี้ผู้ใช้ก็จะไม่สามารถเข้าถึง BooksController ได้ จะต้องล็อกอินก่อนเท่านั้น

3. หากเราต้องการอนุญาตเป็นบางเมธอดให้ผู้ใช้เข้าถึงได้ ให้เขียนโดยการใช้ except (array) เพิ่มเติม ดังนี้

```
public function __construct() {
    $this->middleware('auth', ['except' => ['index']]);
}
```

จากเดิมด้านบน ผู้ใช้จะไม่สามารถเข้าถึงเมธอดอื่นๆ ใน BooksController ได้ ยกเว้นเมธอด index

ให้ดูในหน้า BookController ทั้งหมด

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
use App\Models\Books; //อย่าลืม use โมเดลเข้ามาใช้งาน
use App\Http\Requests\StoreBooksRequest;
use Image; //เรียกใช้ library จัดการรูปภาพเข้ามาใช้งาน
use Illuminate\Support\Str; //นำ Helpers String เข้ามาใช้งาน
use File;
```

```
class BooksController extends Controller
```

```
{
```

```
public function __construct() {
    $this->middleware('auth', ['except' => ['index']]);
    //$this->middleware('auth', ['except' => ['index', 'create', 'store']]);
}
```

```
/**
```

```
* Display a listing of the resource.
```

```
*
```

```
* @return \Illuminate\Http\Response
```

```
*/
```

```

public function index()
{
    $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
    return view('books/index',['books' => $books]);
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('books.create');
}

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(StoreBooksRequest $request)
{
    $book = new Books();
    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;
    if ($request->hasFile('image')) {
        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/', $filename);
    }
}

```

```

Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' .
$filename);

$book->image = $filename;

} else {

$book->image = 'nopic.jpg';

}

$book->save();

return redirect()->action('BooksController@index')->with('status', 'ບັນທຶກຂໍ້ມູນເລີຍປົກອຍ');

}

/**

 * Display the specified resource.

 *

 * @param int $id

 * @return \Illuminate\Http\Response

 */

public function show($id)

{

    //



}

/**

 * Show the form for editing the specified resource.

 *

 * @param int $id

 * @return \Illuminate\Http\Response

 */

public function edit($id)

{

    $book = Books::findOrFail($id);

    return view('books.edit', ['book' => $book]);
}

```

```

}

/**

 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */

public function update(Request $request, $id)
{
    $book = Books::find($id);

    $book->title = $request->title;
    $book->price = $request->price;
    $book->typebooks_id = $request->typebooks_id;

    if ($request->hasFile('image')) {

        // delete old file before update
        if ($book->image != 'nopic.jpg') {
            File::delete(public_path() . '\\images\\' . $book->image);
            File::delete(public_path() . '\\images\\resize\\' . $book->image);
        }

        $filename = Str::random(10) . '.' . $request->file('image')->getClientOriginalExtension();
        $request->file('image')->move(public_path() . '/images/' . $filename);
        Image::make(public_path() . '/images/' . $filename)->resize(50, 50)->save(public_path() . '/images/resize/' . $filename);
        $book->image = $filename;
    }
}

```

```
$book->save();

return redirect()->action('BooksController@index');

}

/** 
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    $book = Books::find($id);

    if ($book->image != 'nopic.jpg') {
        File::delete(public_path() . "\\images\\' . $book->image);
        File::delete(public_path() . "\\images\\resize\\' . $book->image);
    }

    $book->delete();
    return redirect()->action('BooksController@index');
}
```

## บทที่ 8 การสร้างรายงานในรูปแบบ PDF และ Charts

### การสร้างรายงานรูปแบบ PDF

### การสร้างรายงานรูปแบบ Charts

เนื้อหาสำหรับการทำรายงานรูปแบบต่างๆ จัดทำในรูปแบบวิดีโอ สามารถเข้าไปดาวน์โหลดได้ที่ (คลิกมาเพื่อขอสิทธิ์ก่อน)

<https://goo.gl/EIDpt>

หมายเหตุ วิดีโอดังกล่าวเป็น Laravel 5.2 ครับ ซึ่ง concept ไม่ต่างกันมากสามารถศึกษาได้ หากสงสัยค่อยถามมาทางแฟนเพจได้ครับ

## บทที่ 9 ใบนัสพิเศษ

### สรุป 36 คำสั่งของ Laravel ที่ใช้งานบ่อย

#### 1. การแสดงผลตัวแปรต่างๆ ที่ไปที่ view

```
view('task.index')->with('tasks', Task::all());
```

หรือ

```
view('task.index',[ 'tasks', Task::all()]);
```

#### 2. Route cache

```
php artisan route:cache
```

#### 3. ล้าง Route cache

```
php artisan route:clear
```

#### 4. สร้าง csrf tokens field ให้กับฟอร์ม

```
{{ csrf_field(); }}
```

#### 5. คำสั่งเกี่ยวกับการ Redirects

```
return redirect()->to('login');
```

หรือ

```
return redirect('login');
```

#### 6. Route redirect เช่น

```
return redirect()->route('home.index');
```

```
return redirect()->route('home.show',[ 'id', 99 ]);
```

#### 7. Redirect back() ใช้

```
redirect()->back();
```

หรือเขียนย่อๆ คันธี

```
back();
```

#### 8. Redirect ไปที่ route ที่ชื่อว่า home

```
home();
```

#### 9. Refresh หน้า

```
refresh();
```

#### 10. redirect โดยใช้ action() เช่น

```
redirect()->action('ชื่อController@ชื่омethod');
```

#### 11. สร้าง flash data session

```
redirect()->with(['error'=>true,'message'=>'Whoops!']);
```

#### 12. aborting the request เช่น

```
abort(403,'คุณไม่มีสิทธิ์เข้า้งานส่วนนี้');
```

#### 13. return json

```
return response()->json(User::all());
```

#### 14. ส่งไฟล์เพื่อทำการดาวน์โหลด

```
return response()->download('file1.pdf','file2.pdf');
```

หรือจะแสดงที่ Browser ให้เช่น

```
return response()->file('file1.pdf');
```

#### 15. รับ input ทั้งหมดจาก request

```
$request->all();
```

#### 16. รับ input ยกเว้นบางตัวใช้ except

```
$request->except('_token');
```

17. รับ input เฉพาะที่ต้องการใช้ only

```
$request->only(['firstname','email']);
```

18. ใช้ has จะ return false ถ้ามีตัวแปร และว่า

```
if ($request->has('file')) {
```

```
}
```

19. จะ return true ถ้ามีตัวแปร และว่า

```
if ($request->exists('email')) {
```

```
}
```

20. รับ request ทีละฟิลด์

```
$request->input('email')
```

21. ถ้าเป็น JSON Input ก็ใช้เหมือนกัน อ้างจุดไปที่ object เช่น

```
$request->input('data.email')
```

22. Accessors = getting data ของ Model

23. Mutators = setting data ของ Model

24. หากอยากร่อนบางฟิลด์ ก็กำหนดที่ Model นั้นๆ (\$hidden) เช่น

```
class Contact extends Model {
```

```
public $hidden = ['password','email'];
```

หรือ เลือกแสดงบางฟิลด์ก็ได้ (\$visible) เช่น

```
public $visible = ['name','gpa'];
}
```

25. เข้าถึงข้อมูลของ user โดยใช้ request เช่น อย่างได้ฟิลด์อีเมล ก็เขียนง่ายๆ ตามนี้

```
$request->user()->email  
หรือเขียนที่ view ก็ได้ เช่น  
ยินดีต้อนรับคุณ {{ auth()->user()->name }}
```

26. ตั้งชื่อให้กับ Route เพื่อง่ายต่อการเรียกใช้งาน โดยระบุ ->name('ชื่อ route') เช่น

```
Route::get('/home', 'HomeController@index')->name('home');
```

27. config cache ใช้คำสั่ง

```
php artisan config:cache
```

28. ล้าง config cache ใช้คำสั่ง

```
php artisan config:clear
```

29. ล้าง Application cache

```
php artisan cache:clear
```

30. Compiling Assets (Laravel Mix)

ติดตั้ง Dependencies ใช้คำสั่ง npm install

รัน Laravel Mix ใช้คำสั่ง npm run dev หรือ npm run watch

หรือหากต้องการรันเพื่อ production ก็ใช้คำสั่ง npm run production

31. ดูว่า Laravel เตรียม frontend preset อะไรให้เราบ้างใช้คำสั่ง (ปกติมี bootstrap, vue, react, none)

```
php artisan preset --help
```

32. สร้างระบบ Authentication ใช้คำสั่ง (มีระบบล็อกอินมาให้เลย)

```
php artisan make:auth
```

### 33. แสดง Route ทั้งหมดของ app เรา

```
php artisan route:list
```

### 34. คำสั่งสำหรับลบตารางทั้งหมด และสร้าง migrate ตารางใหม่อีกครั้ง

```
php artisan migrate:fresh
```

### 35. คำสั่งแบ่งหน้า ใช้

```
$persons = Person::paginate(20);
```

หรือ

```
$persons = Person::simplePaginate(15);
```

### 36. ตัวอย่างการทำ Validation (เขียนที่ controller)

```
$request->validate([
```

```
'title' => 'required',
```

```
'price' => 'required|numeric',
```

```
'image' => 'mimes:jpeg,jpg,png'
```

```
],[
```

```
'title.required' => 'กรุณากรอกชื่อสินค้าด้วย',
```

```
'price.required' => 'กรุณากรอกราคา',
```

```
'price.numeric' => 'กรุณากรอกราคาเป็นตัวเลขเท่านั้น',
```

```
'image.mimes' => 'ไฟล์ที่เลือกต้องนามสกุล jpeg, jpg, png เท่านั้น'
```

```
]);
```

### 37. แสดงวันที่และเวลาปัจจุบัน ใช้คำสั่ง now() หรือ วันที่อย่างเดียวใช้ today() เช่น

```
{{ now() }}
```

```
{{ today() }}
```

### 38. ใช้ Bcrypt เพื่อ hash รหัสผ่าน เช่น

```
$password = bcrypt('1234');
```

### 39. เรียกค่า config จากไฟล์ .env ใช้ config() แต่ตอนอ้างถึงใช้เครื่องหมายจุด แทน \_ (underscore) เช่น

```
$value = config('app.timezone');
```

มาถึงตรงนี้ ก็ขอขอบคุณ คนที่รักการพัฒนาตัวเองทุกคนครับ  
หวังว่าความรู้ในหนังสือเล่มนี้จะช่วยให้ชีวิตของทุกคนดีขึ้น  
สามารถต่อยอดความรู้ เพื่อสร้างสิ่งดีๆ ให้กับตัวเอง ครอบครัว และโลกนี้ต่อไป

ขอบคุณครับ

โค้ชเอก

Codingthailand.com