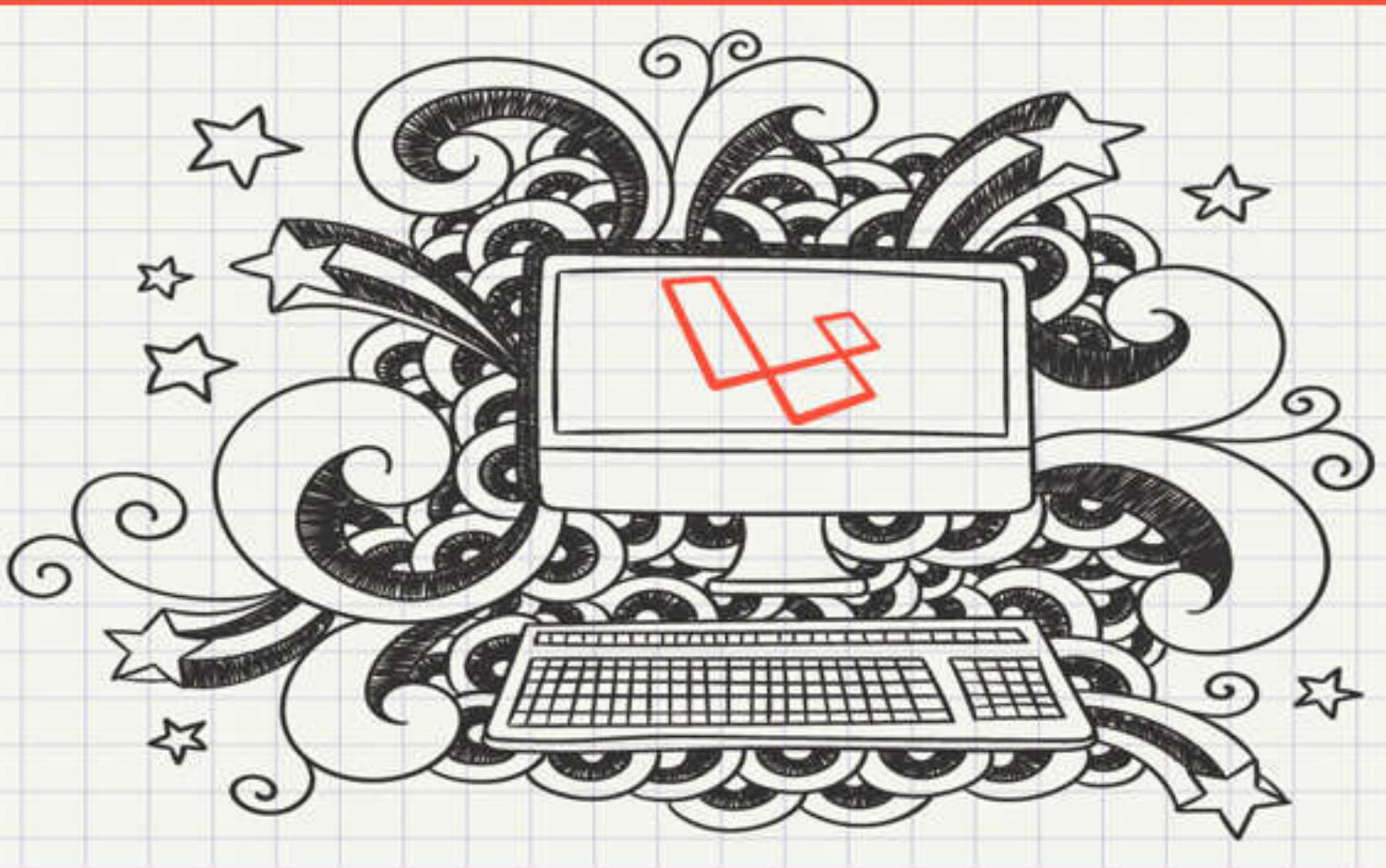
e-Book

Laravel 5

Summarize for Web Developer



โค้ชเอก

Laravel 5.2: Summarize for Web Developer

พัฒนา Web Application ด้วย Laravel 5.2

โค้ชเอก

หนังสือเล่มนี้จำหน่ายที่ http://www.codingthailand.com/laravel5ebook

เวอร์ชัน 1 ออกจำหน่ายวันที่ 1 เมษายน 2559

หนังสือเล่มนี้ผู้เขียนตั้งใจจัดทำขึ้นเพื่ออยากให้มีหนังสือภาษาไทยซักเล่มเกี่ยวกับ Laravel 5 ที่เน้นเนื้อหาตั้งแต่พื้นฐาน การทำงานกับฐานข้อมูล และการทำรายงานต่างๆโดยเน้นสรุปประเด็นที่สำคัญๆ เพื่อให้ผู้อ่านสามารถนำไปต่อยอดพัฒนาWeb Application ที่ต้องการได้ หวังว่าหนังสือเล่มนี้จะเป็นประโยชน์ ประหยัดเวลาการเรียนรู้ขอให้สนุกกับการเรียนรู้ครับ

“จงเอาชนะความไม่รู้ ด้วยการพัฒนาตัวเอง และลงมือทำอย่างสม่ำเสมอ”

สารบัญ

บทที่ 1 การเตรียมตัวและการติดตั้ง Laravel 5 ด้วย Composer

การติดตั้งต้องเตรียมอะไรบ้าง

เนื่องจากหนังสือเล่มนี้ไม่ใช่หนังสือพื้นฐาน การเตรียมตัวอย่างแรกในการอ่านหนังสือเล่มนี้คือ เราจะต้องมีพื้นฐานภาษา PHP และ มีความรู้เกี่ยวกับการเขียนโปรแกรมในรูปแบบของ Object Oriented Programming หรือ OOP มาก่อน เพื่อให้เกิดประโยชน์สูงสุดในการเรียนรู้ สำหรับคนที่ยังไม่มีพื้นฐานความรู้ดังที่กล่าวมา ผมว่าแนะนำควรให้ศึกษาก่อนครับ

ในการติดตั้ง Laravel 5 นั้น ก่อนเรียนต้องเตรียมตัวและติดตั้งโปรแกรมต่างๆ ประกอบด้วย

1. XAMPP สำหรับจำลองเครื่องเราให้เป็น Web Server ประกอบด้วย Apache, PHP, MySQL/MariaDB และ phpMyAdmin หรือโปรแกรมจำลอง Web Server อื่นๆ
2. Netbeans สำหรับใช้เขียนโค้ด หรือจะใช้ IDE ที่ถนัดก็ได้
3. Composer สำหรับการจัดการกับ PHP Packages และ Library ต่างๆ

ขั้นตอนการติดตั้งทั้งหมด สามารถเปิดดูวิดีโอได้ที่ https://www.youtube.com/watch?v=6DH_aEaJ3X4

Extensions ของ PHP ที่ควรเปิดไว้

บางครั้งระหว่างติดตั้ง Composer หรือ พัฒนา Web Application อาจมี errors สำหรับบางคำสั่ง ก่อนติดตั้ง Laravel ควรไปตรวจสอบหรือเปิด extension ในไฟล์ php.ini ให้เรียบร้อย คือ ให้**เปิดไฟล์** php.ini (C:\xampp\php\php.ini) ค้นหา extensions แล้วเอาเครื่องหมาย; (เซมิโคล่อน) ข้างหน้าออก เสร็จแล้วบันทึกไฟล์แล้ว **Restart Apache** ส่วนรายการ extensions ที่ควรเปิด มีดังต่อไปนี้

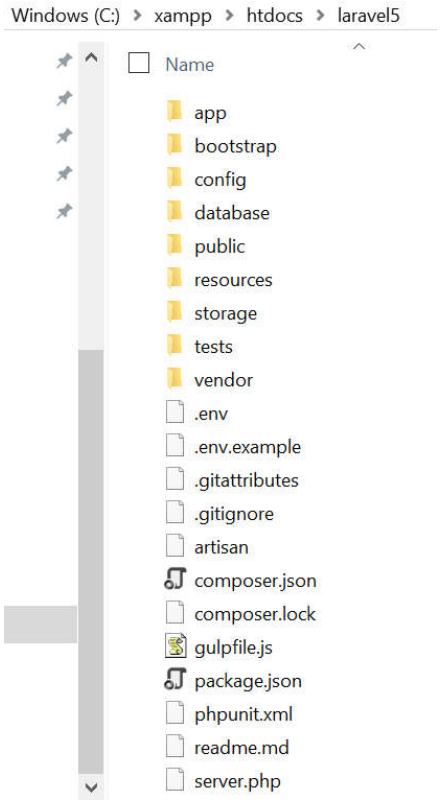
extension=php_bz2.dll extension=php_curl.dll extension=php_mbstring.dll extension=php_fileinfo.dll extension=php_gd2.dll extension=php_openssl.dll extension=php_intl.dll extension=php_pdo_mysql.dll extension=php_mbstring.dll

```
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
```

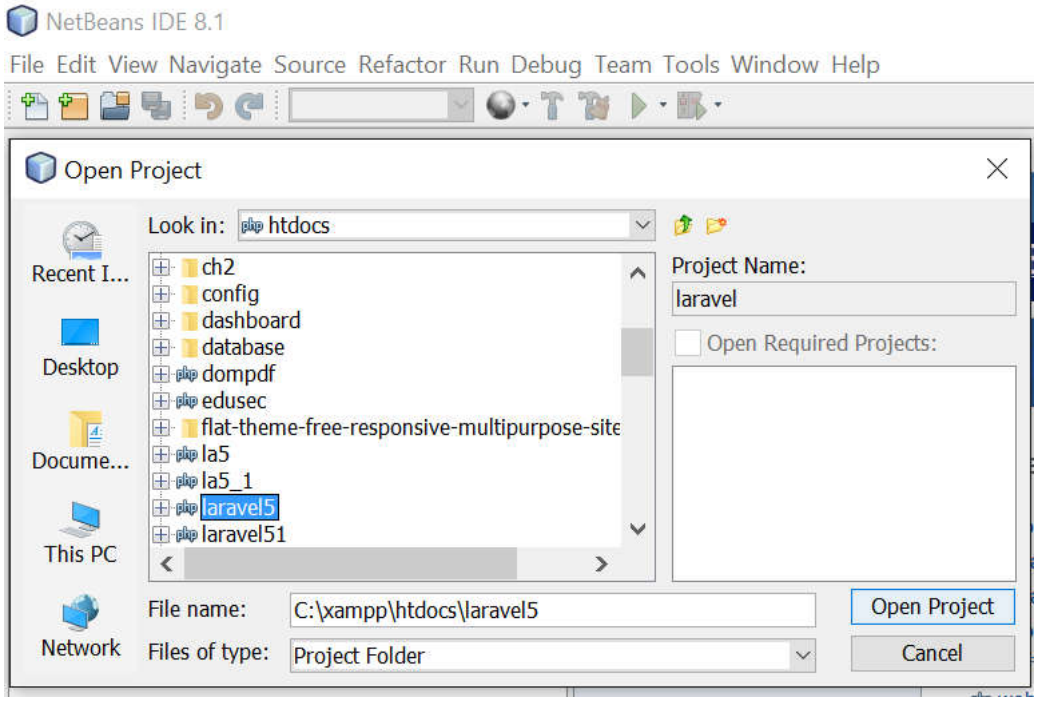
การติดตั้ง Laravel 5 ด้วย Composer

การติดตั้ง Laravel 5 นั้น วิธีที่ง่ายและสะดวก แนะนำติดตั้งผ่าน Composer โดยมีขั้นตอนการติดตั้ง ดังนี้

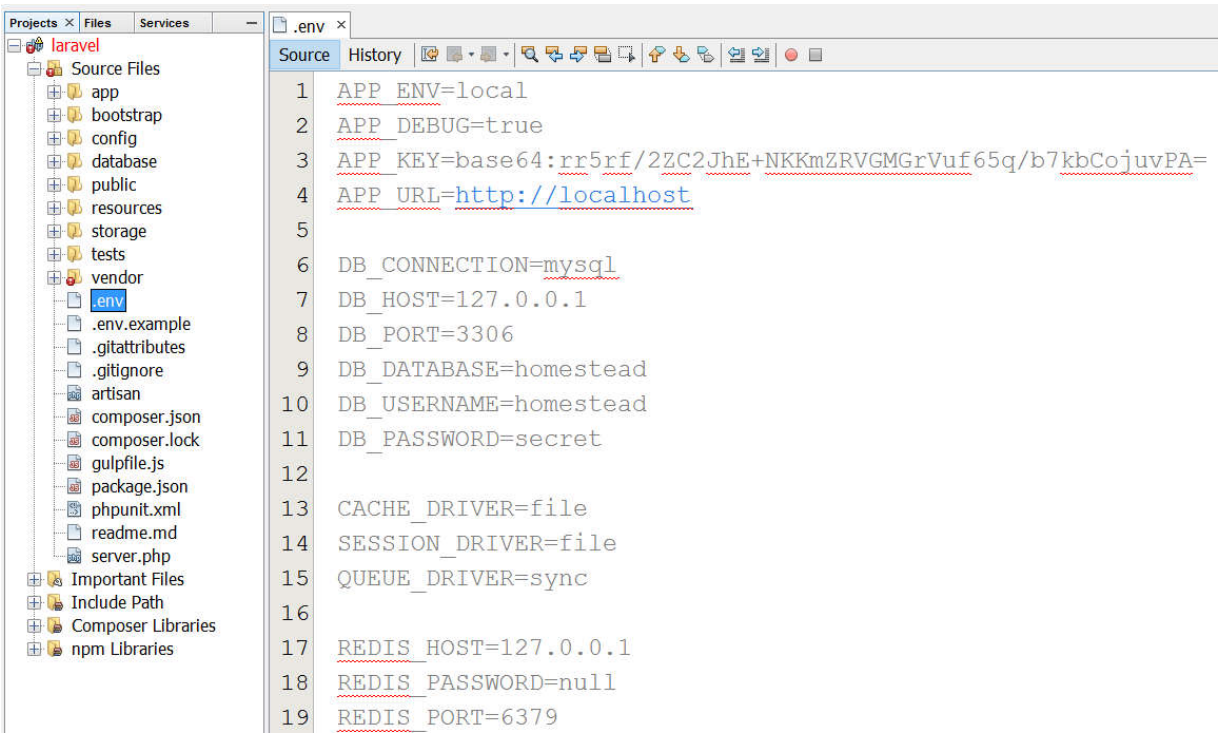
1. เข้าไปที่โฟลเดอร์ htdocs ของ XAMPP ที่ C:\xampp\htdocs คลิกขวา เลือก Use Composer here (จะเปิด Command Prompt แล้วค่อยพิมพ์ cd C:\xampp\htdocs ก็ได้เช่นเดียวกัน)
2. พิมพ์คำสั่ง **composer create-project --prefer-dist laravel/laravel laravel5 "5.2.*"** แล้วกด enter
   (laravel5 คือ ชื่อโฟลเดอร์ที่เก็บโปรเจคของเรา สามารถตั้งชื่ออื่นได้)
3. รอสักครู่จนการติดตั้งเสร็จเรียบร้อย (โครงสร้างโฟลเดอร์ของ Laravel 5 หลังติดตั้ง)



4. เปิดโปรแกรม Netbeans คลิกที่เมนู File -> Open Project… จากนั้นเลือกโฟลเดอร์ที่เราได้ติดตั้งไว้ (ในที่นี้คือ laravel5)



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



Note: ไฟล์ .env หากใครเคยใช้ MySQL/MariaDB แล้ว สามารถกรอกรายละเอียดการติดต่อกับข้อมูลฐานข้อมูล ได้ตั้งแต่บรรทัดที่ 6 ถึงบรรทัดที่ 11

Note: สามารถดูวิดีโอการติดตั้งเพิ่มเติมได้ที่ https://www.youtube.com/watch?v=XCbnaTH54xM

6. ทดสอบ Laravel ผ่าน Browser โดยพิมพ์ URL ดังนี้ http://localhost/laravel5/public/ แค่นี้ก็ติดตั้ง Laravel เรียบร้อยครับ

Laravel 5

การตั้งค่าระบบของ Laravel

หลังจากการติดตั้งแล้ว เราควรทำความรู้จักกับการตั้งค่าต่างๆของ Laravel กันก่อน โดยให้เปิดโฟลเดอร์ config การตั้งค่าสำคัญๆ ได้แก่

- **app.php**: เป็นรายละเอียดการตั้งค่าภาพรวมของระบบเราทั้งหมด เช่น กำหนดการเปิด-ปิด ของ Debug Mode, การกำหนด timezone ให้กับ Web Application เป็นต้น แน่นอนเราอยู่ในประเทศไทย ก็ควรกำหนดเป็น **'timezone' => 'Asia/Bangkok'**
- **auth.php**: เป็นรายละเอียดการตั้งค่าเกี่ยวกับการล็อกอิน การรับรอง หรือตรวจสอบผู้ใช้ เช่น การกำหนดตารางผู้ใช้ในการฐานข้อมูล, การตั้งค่าเกี่ยวกับการ reset รหัสผ่าน เป็นต้น
- **cache.php**: รายละเอียดการตั้งค่าของ cache โดย Laravel รองรับประเภท cache ได้หลายตัว ได้แก่ filesystem, database, mem-cached, 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
- **filesystems.php**: รายละเอียดการตั้งค่าและกำหนดปลายทางของระบบไฟล์ในโปรเจคของเรา เช่น การจัดการกับไฟล์เมื่อเราอัพโหลดไฟล์ต่างๆ เป็นต้น โดยรองรับทั้งแบบ local disk และ Amazon S3
- **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 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 เราควรกำหนดค่าพวกนี้ที่ไฟล์ .env ครับ

การตั้งค่า timezone

สิ่งแรกที่ควรทำหลังการติดตั้งอันต่อมาคือ การตั้งค่าวันที่และเวลาให้ถูกต้องกับ timezone ของประเทศไทย มีขั้นตอนดังนี้

1. เปิดไฟล์ config/app.php แล้วแก้ไขค่า timezone จาก UTC เป็น Asia/Bangkok

app.php

```
44      /*
45      |--------------------------------------------------------------------------
46      | Application Timezone
47      |--------------------------------------------------------------------------
48      |
49      | Here you may specify the default timezone for your application, which
50      | will be used by the PHP date and date-time functions. We have gone
51      | ahead and set this to a sensible default for you out of the box.
52      |
53      */
54
55      'timezone' => 'Asia/Bangkok',
56
```

2. จากนั้นให้ทดสอบเขียนโค้ดเพื่อแสดงวันที่และเวลาว่าถูกต้องหรือไม่ โดยให้เปิดไฟล์ resources/views/welcome.blade.php แล้วเขียนโค้ดเพื่อแสดงวันที่และเวลาปัจจุบัน เสร็จแล้วบันทึกไฟล์ แล้วลองรันดูว่าวันที่และเวลาถูกต้องหรือไม่

welcome.blade.php

```
34                  font-size: 96px;
35              }
36          </style>
37      </head>
38      <body>
39          <div class="container">
40              <div class="content">
41                  <div class="title">Laravel 5</div>
42
43                  <p>{{ date("d/m/Y H:i:s")  }}</p>
44
45              </div>
46          </div>
47      </body>
48  </html>
```

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');
}
```

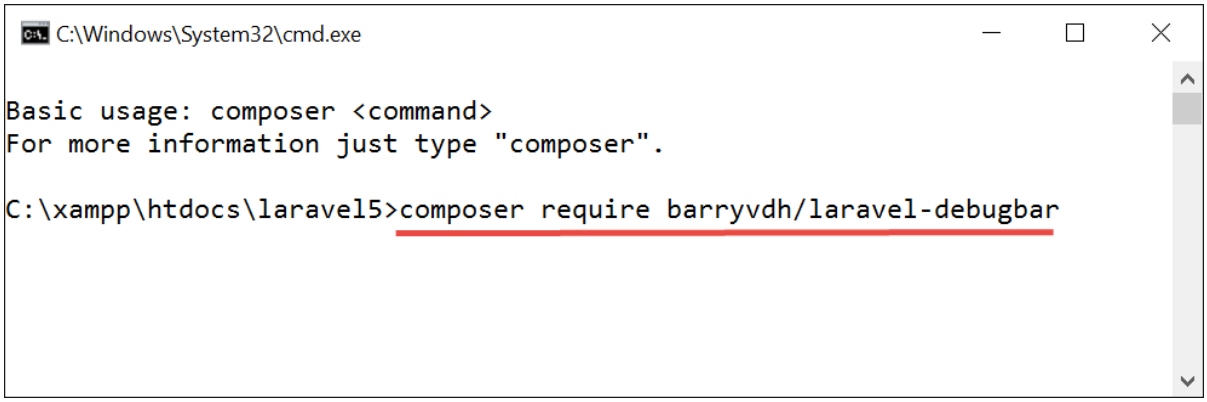
- การใช้ Laravel Logger

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

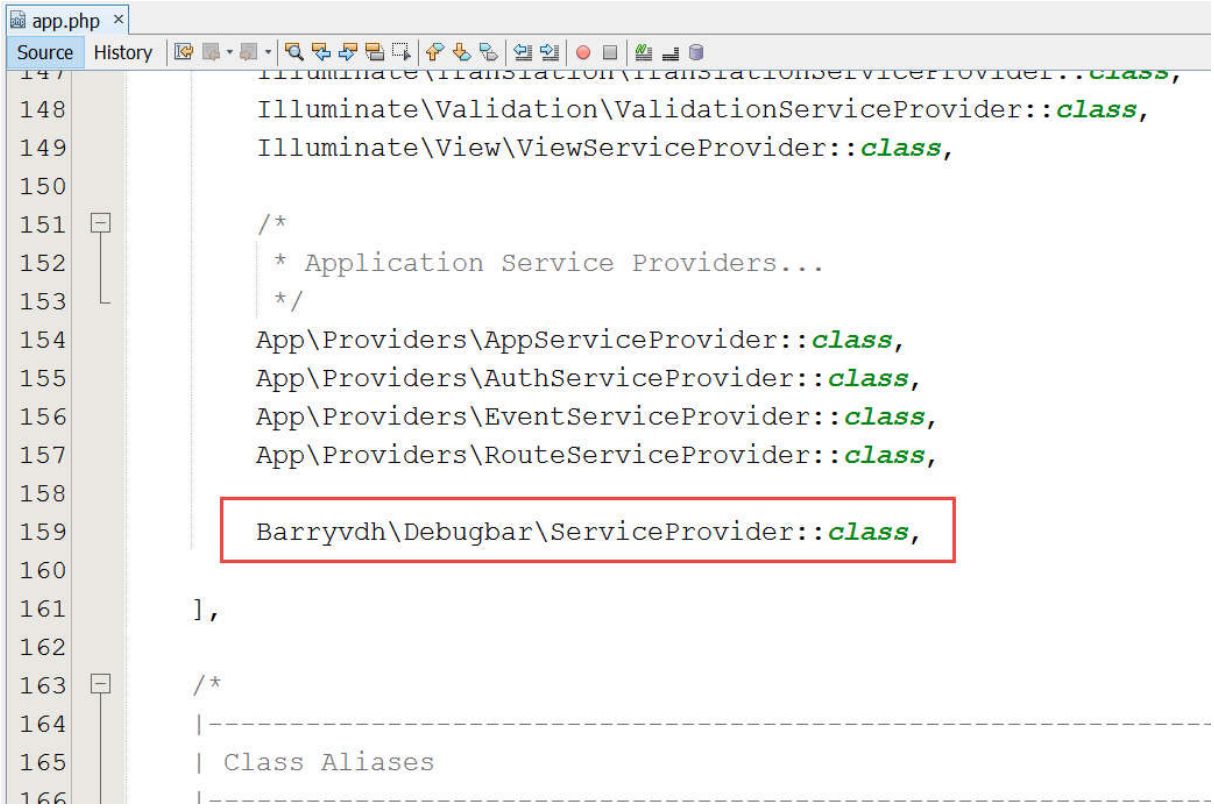
```
\Log::info('ข้อความเกี่ยวกับ information');
\Log::warning('มีบางอย่างผิดปกติ');
\Log::error('เกิด errors ในส่วนนี้');
\Log::critical('อันตราย!');
```

- การใช้ Laravel Debugbar (แนะนำตัวนี้เพราะสามารถดูผ่าน Browser ได้เลย) การติดตั้งมีขั้นตอนดังนี้

  1. เข้าเว็บ https://github.com/barryvdh/laravel-debugbar
  2. เปิดหรือเข้าไปในโฟลเดอร์โปรเจคของเราที่ C:\xampp\htdocs\laravel5 เปิด Composer ขึ้นมา แล้วพิมพ์คำสั่ง **composer require barryvdh/laravel-debugbar** จากนั้นกด enter เพื่อติดตั้ง



3. เปิดไฟล์ config/app.php เพื่อกำหนดค่าส่วนของ Autoloaded Service Providers โดยให้ copy โค้ดนี้ Barryvdh\Debugbar\ServiceProvider::class, ไปวางไว้บรรทัดสุดท้ายในส่วนของอาร์เรย์ providers



4. จากนั้นให้ copy โค้ด 'Debugbar' => Barryvdh\Debugbar\Facade::class, ไปวางไว้ที่ส่วน Class Aliases (เลื่อนลงมาอีกนิดหน่อยในไฟล์นี้) โดยวางไว้บรรทัดสุดท้าย ส่วนนี้เป็นการตั้งชื่อเล่นสั้นๆให้กับ Class นั่นเอง

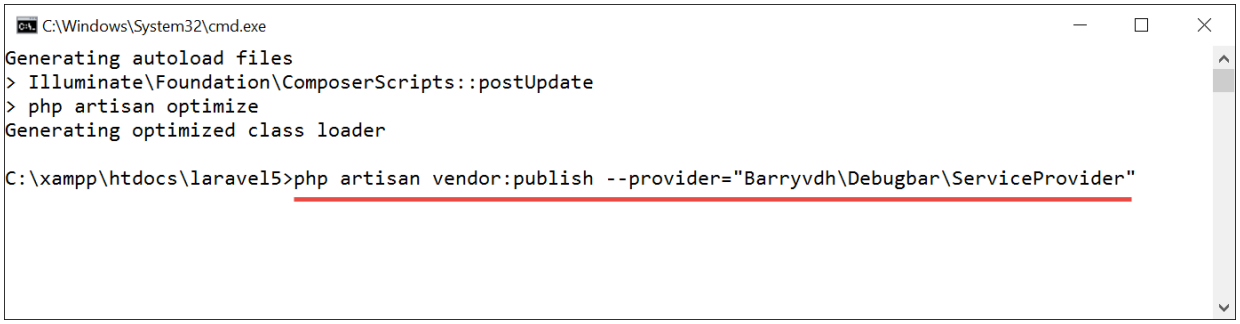
app.php

```
195            'Redirect' => Illuminate\Support\Facades\Redirect::class,
196            'Redis' => Illuminate\Support\Facades\Redis::class,
197            'Request' => Illuminate\Support\Facades\Request::class,
198            'Response' => Illuminate\Support\Facades\Response::class,
199            'Route' => Illuminate\Support\Facades\Route::class,
200            'Schema' => Illuminate\Support\Facades\Schema::class,
201            'Session' => Illuminate\Support\Facades\Session::class,
202            'Storage' => Illuminate\Support\Facades\Storage::class,
203            'URL' => Illuminate\Support\Facades\URL::class,
204            'Validator' => Illuminate\Support\Facades\Validator::class,
205            'View' => Illuminate\Support\Facades\View::class,
206
207            'Debugbar' => Barryvdh\Debugbar\Facade::class,
208
209        ],
210
211    ];
```

5. เปิด Composer ขึ้นมาอีกครั้ง แล้วพิมพ์

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

   จากนั้นกด enter เพื่อ publish และ copy ไฟล์ config ของ debugbar ไปยังโฟลเดอร์ config/ ถ้าเรียบร้อยจะสังเกตเห็นว่ามีไฟล์ใหม่ชื่อว่า debugbar.php ถูกสร้างขึ้นมาครับ (ในโฟลเดอร์ config/)



6. ตรวจสอบการติดตั้ง Laravel Debugbar โดยเปิดและรีเฟรช Browser อีกครั้ง

   ต่อไปเราก็สามารถตรวจสอบ errors ได้สะดวกแล้วครับ

รายละเอียดของ Tab ต่างใน Laravel Debugbar มีดังนี้

- Messages: เอาไว้ดู errors หรือข้อความต่างๆจากไฟล์ log
- Timeline: ใช้สำหรับดูเวลารวมในการโหลด page ต่างๆ
- Exceptions: ใช้สำหรับดูข้อผิดพลาด (exceptions) เมื่อเราโยน (thrown) ออกมาจากระบบ
- 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

บทที่ 2 ทำความรู้จักกับ Laravel, MVC และ Best Practices

ทำไมต้องใช้ PHP Framework

- มีการเขียนโค้ดที่เป็นมาตรฐาน ช่วยลดและกำจัดโค้ดที่ไม่จำเป็น
- ช่วยลดระยะเวลาในการทำงาน เช่น เรื่องความปลอดภัย การสร้างฟอร์ม เป็นต้น
- ช่วยทำให้การทำงานเป็นทีมง่ายขึ้น เพราะต้องเขียนโค้ดเป็นมาตรฐานเดียวกัน
- ช่วยในการบำรุงรักษาโค้ดได้ง่ายขึ้น
- มี community ที่เข้มแข็ง เราสามารถถาม และคอยขอคำแนะนำได้

ทำความรู้จักกับ Laravel

Laravel เป็น web application framework ที่มีคุณสมบัติที่ช่วยให้เราเขียน web application ได้ง่ายขึ้น มีคุณสมบัติครบถ้วน มีจุดเด่นตรงการเขียนโค้ดสั้น กระชับ และยังเหมาะกับการทำงานร่วมกับด้าน front-end เป็นอย่างมาก

โครงสร้างของ Laravel

โครงสร้างแต่ละโฟลเดอร์ของ Laravel มีดังนี้

```
./app/                      # Your Laravel application
 ./app/Console/              # Commands classes  ./app/Console/
  ./app/Console/Commands/   # Command-line scripts
 ./app/Events/               # Events that your application can raise
 ./app/Exceptions/
 ./app/Http/
  ./app/Http/Controllers/   # Your application's controllers
  ./app/Http/Middleware/    # Filters applied to requests
  ./app/Http/Requests/      # Classes that can modify requests
  ./app/Http/routes.php     # URLs and their corresponding handlers
 ./app/Providers             # Service provider classes
 ./app/Services              # Services used in your application

./bootstrap/                  # Application bootstrapping scripts

./config/                     # Configuration files

./database/
```

```
./database/migrations/     # Database migration classes
./database/seeds/          # Database seeder classes

./public/                 # Your application's document root
./public/.htaccess        # Sends incoming requests to index.php
./public/index.php        # Starts Laravel application

./resources/
./resources/assets/       # Hold raw assets like LESS & Sass files
./resources/lang/         # Localization and language files
./resources/views/        # Templates that are rendered as HTML

./storage/
./storage/app/            # App storage, like file uploads etc
./storage/framework/      # Framework storage (cache)
./storage/logs/           # Contains application-generated logs

./tests/                   # Test cases
./vendor/                  # Third-party code installed by Composer
./.env.example            # Example environment variable file
./artisan                  # Artisan command-line utility
./composer.json            # Project dependencies manifest
./phpunit.xml              # Configures PHPUnit for running tests
./server.php               # A lightweight local development server
```

MVC และ Best Practices

รูปแบบการเขียนแบบ MVC (Model, View, Controller) นั้น การจะเขียนให้ดี ต้องศึกษาแนวทางกันก่อนที่ดีกันก่อน สรุปให้ดังนี้

สรุปการเขียน Model ที่ดี

- ประกอบด้วย โค้ดในส่วน business data
- ประกอบด้วย โค้ดในการส่วนของการตรวจสอบความถูกต้องของข้อมูล
- ประกอบ ด้วย เมธอด การทำงานในส่วนของ business logic
- อย่าเขียนโค้ดเกี่ยวกับการ request, session หรือโค้ดเกี่ยวกับสภาพแวดล้อมของระบบ
- ระวังหรือหลีกเลี่ยงเขียนโค้ดเกี่ยวกับ HTML ในส่วนของการแสดงผลใน Model ให้ไปเขียนที่ view แทน

สรุปการเขียน View ที่ดี

- View จะต้องมีโค้ดเฉพาะ HTML และ PHP ที่เกี่ยวข้องกับการแสดงผล จัดรูปแบบข้อมูลต่างๆเท่านั้น
- จะต้องไม่โค้ดเกี่ยวกับการ query ฐานข้อมูลต่างๆ

- หลักเลี่ยงการรับค่า $_GET, $_POST เพราะเป็นหน้าที่ของ Controller
- ถ้าเรารับค่ามาจาก model จะต้องไม่ไปแก้ไขค่าที่รับมา
- ใช้คลาสในกลุ่ม Helper เพื่อช่วยในการจัดรูปแบบข้อมูล

สรุปการเขียน Controller ที่ดี

- มีไว้เขียนเกี่ยวกับ request ข้อมูล
- มีไว้เรียกเมธอดเกี่ยวกับ Models และ เรียก component ต่างๆ
- มีไว้ส่งข้อมูลต่างๆ ไปให้ views เพื่อนำไปแสดงผล
- 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 app\Http\routes.php เพื่อสร้างเส้นทาง หรือให้มองว่าเป็น URL ก็ได้ครับ เขียนโค้ดดังนี้

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

   routes.php

   Source | History

   ```
   <?php

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

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

   ```

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

2. เข้าไปที่โฟลเดอร์โปรเจค (C:\xampp\htdocs\laravel5) จากนั้นเปิด Composer เพื่อพิมพ์คำสั่งสำหรับสร้าง Controller ดังนี้

   php artisan make:controller SiteController

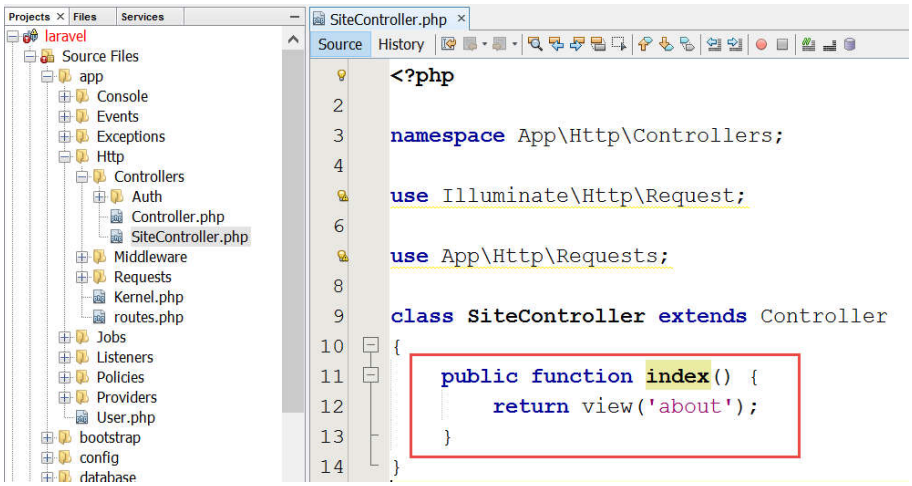
   C:\Windows\System32\cmd.exe

   ```
   Basic usage: composer <command>
   For more information just type "composer".

   C:\xampp\htdocs\laravel5>php artisan make:controller SiteController
   Controller created successfully.

   C:\xampp\htdocs\laravel5>
   ```

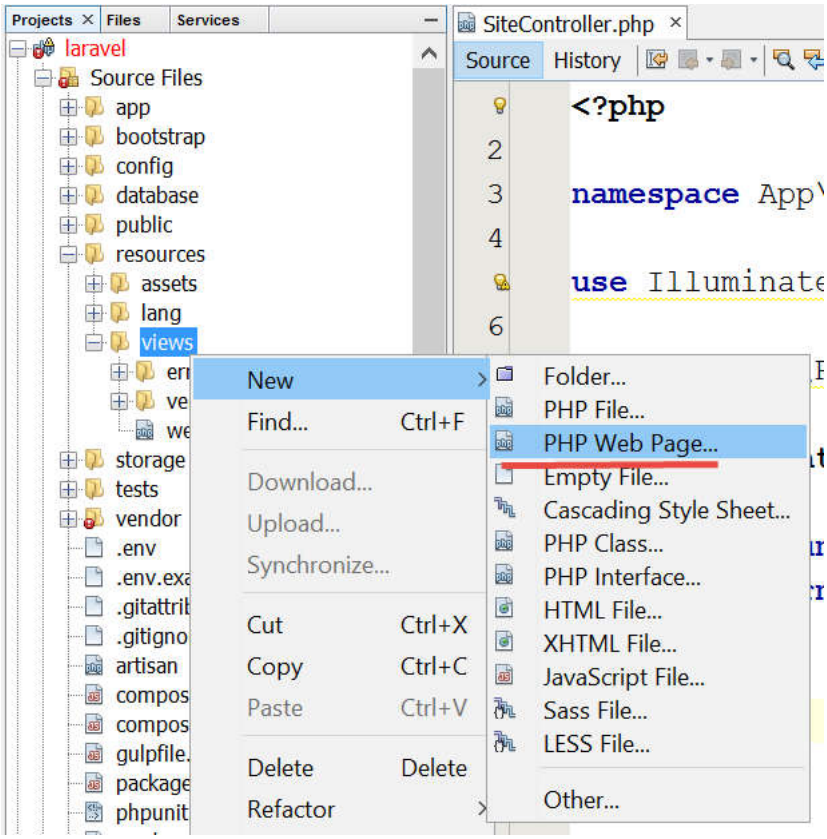
   อธิบาย การสร้าง Controller ใหม่จะใช้คำสั่ง make:controller ตามด้วยชื่อของ controller ที่ต้องการสร้าง (การตั้งชื่อแนะนำให้ขึ้นต้นด้วยตัวพิมพ์ใหญ่และตามด้วยคำว่า Controller)

Note: หากต้องการศึกษาคำสั่งทั้งหมดของ artisan ให้พิมพ์ **php artisan** แล้วกด enter และถ้าหากต้องการรู้วิธีการใช้ในแต่ละคำสั่งให้พิมพ์ –help ต่อท้าย เช่น **php artisan make:controller --help**

3. ไฟล์ SiteController.php จะถูกสร้างที่โฟลเดอร์ app\Http\Controllers ให้เปิดไฟล์ SiteController.php แล้วเขียน เมธอด ชื่อว่า index ดังนี้ (เมธอด ที่ตั้งขึ้นมาจะต้องสอดคล้องกับการเขียน route ในข้อ 1)

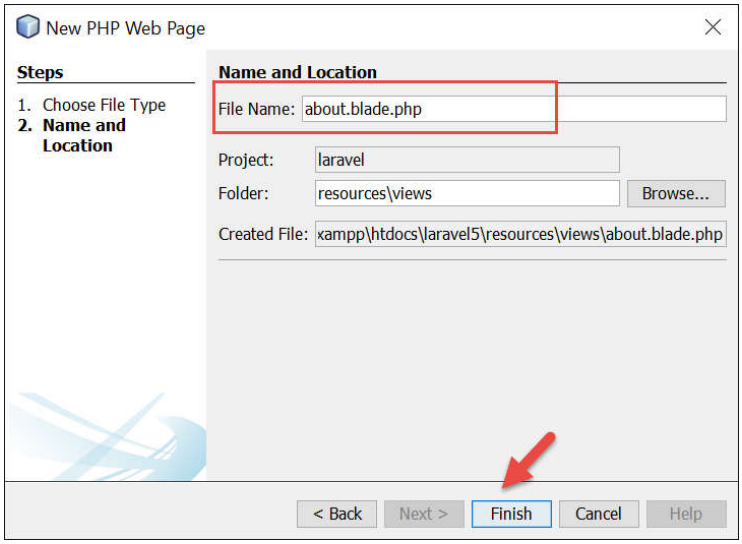


4. จะเห็นว่าตอนนี้เราได้สร้าง route กับ controller เรียบร้อยแล้ว ถ้าสังเกตใน เมธอด index() จะเห็นว่าเราได้เขียนโค้ดเพื่อสั่งให้ render ไปที่ view ชื่อว่า about นั่นเอง การสร้างไฟล์ view นั้นเราสามารถสร้างไฟล์ได้ที่โฟลเดอร์ resources\views ดังนี้ (หากใช้ Netbeans สามารถคลิกขวาที่โฟลเดอร์แล้วเลือก PHP File หรือ PHP Web Page ได้เลย)

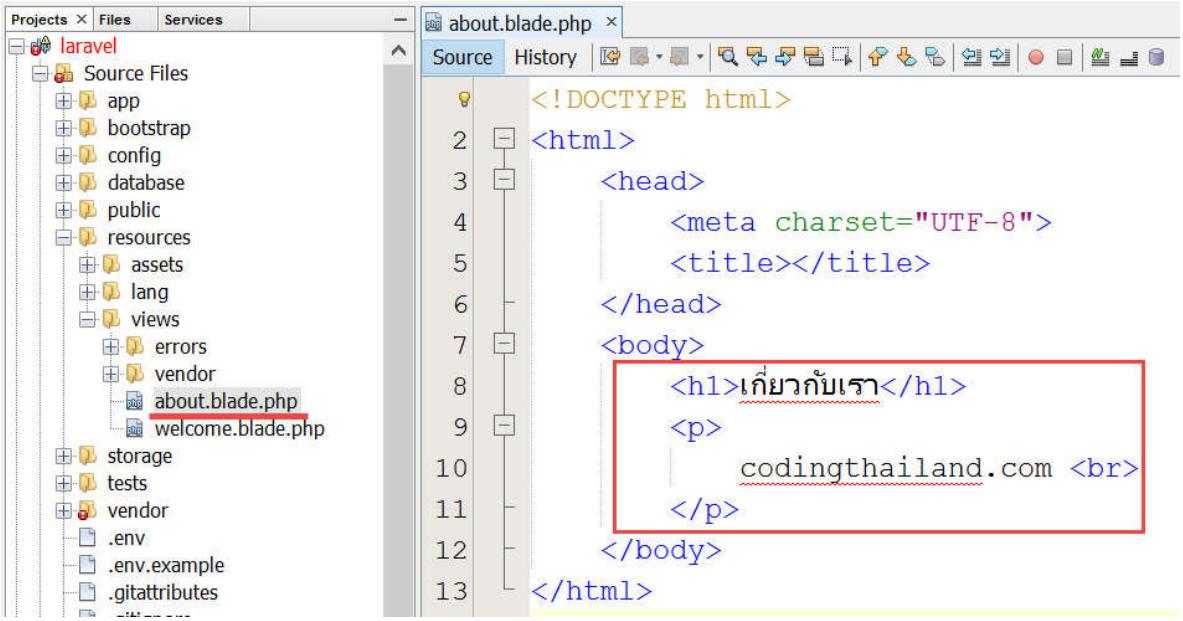


Note: ในโฟลเดอร์ views นี้เราสามารถสร้างโฟลเดอร์ซ้อนกันได้ครับ หากมีโฟลเดอร์ซ้อนกันให้แก้โค้ดใน Controller เช่น **return view('site.about')** หมายถึง อ้างไฟล์ about.blade.php ซึ่งอยู่ในโฟลเดอร์ site

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

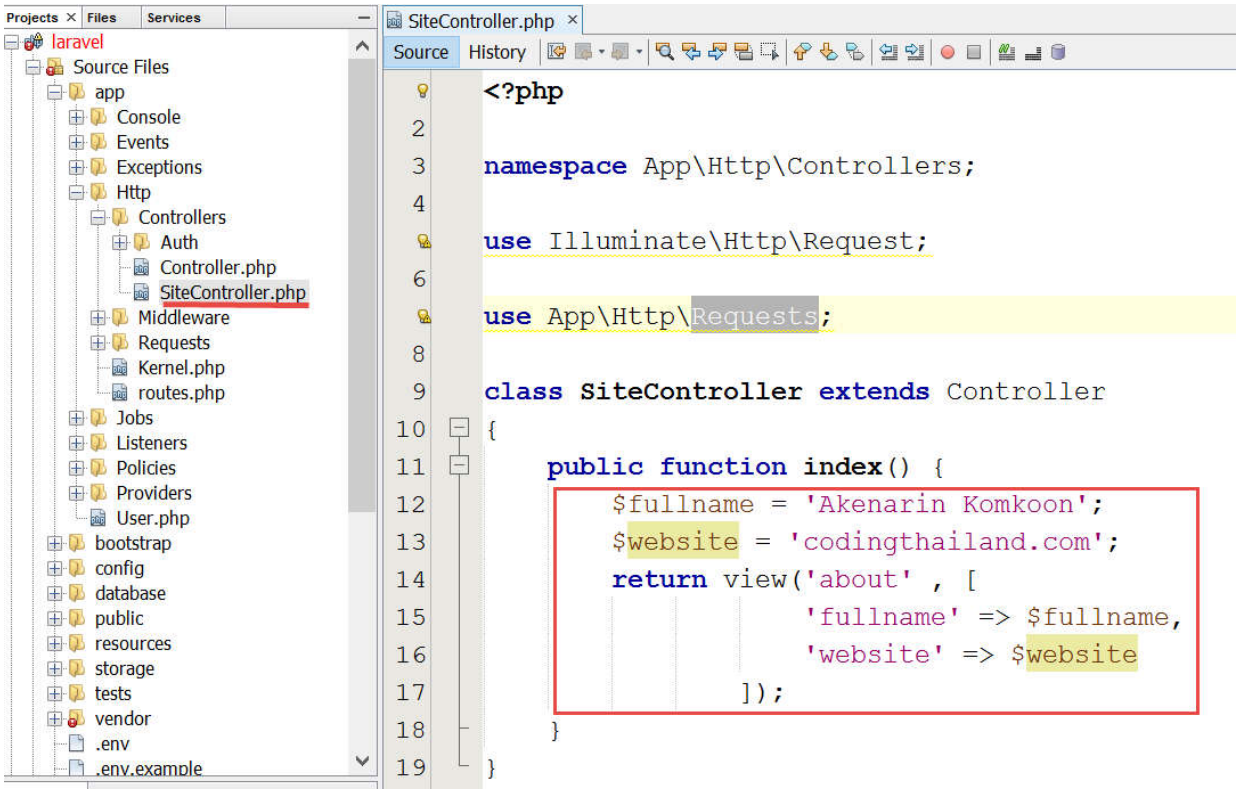


6. ทดลองแก้ไขไฟล์ about.blade.php



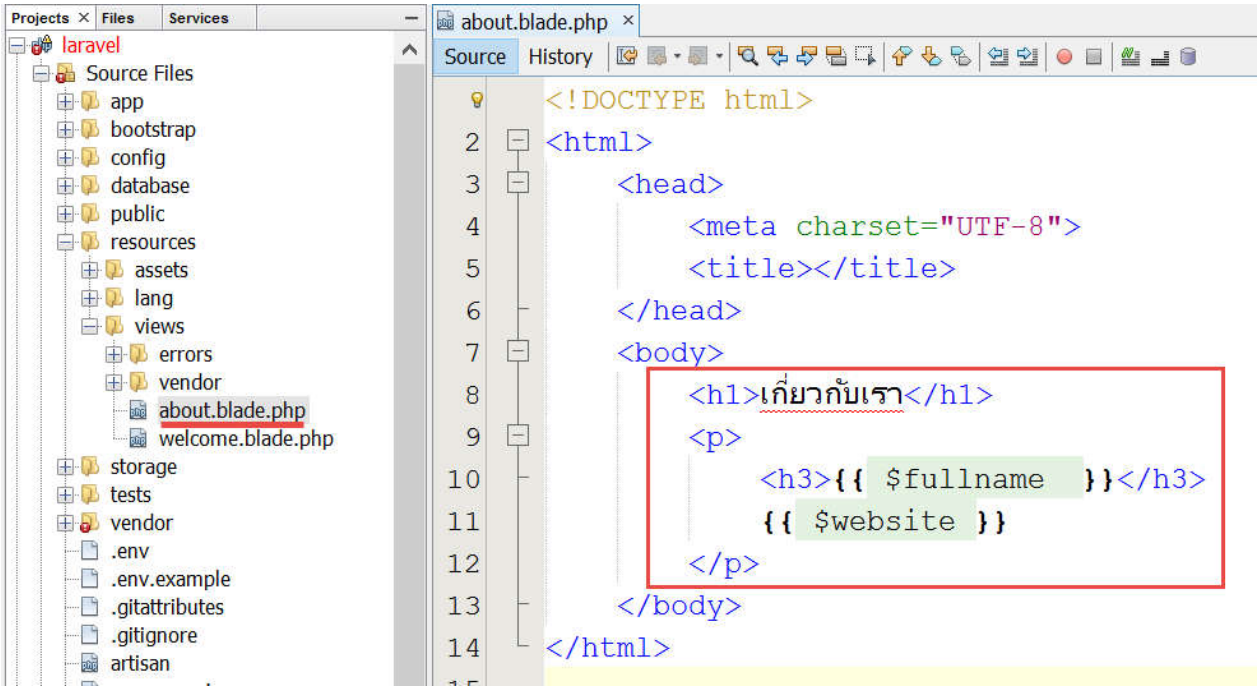
7. บันทึกไฟล์แล้วพิมพ์ URL เพื่อทดสอบการทำงาน ดังนี้ http://localhost/laravel5/public/about

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

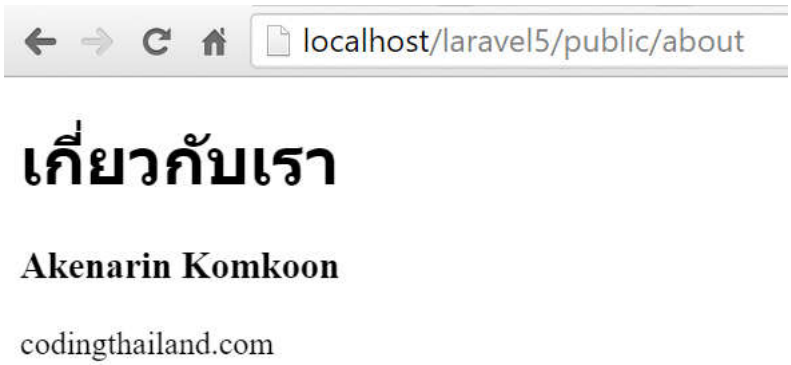


อธิบายโค้ด เราสามารถส่งตัวแปร และข้อมูลที่ต้องการเพื่อไปแสดงผลในหน้า View ได้ หากมีตัวแปรที่ต้องการส่งหลายตัวก็ให้คั่นด้วยเครื่องหมายคอมม่า โดยในตัวอย่างจะส่งตัวแปร $fullname และ $website เพื่อไปแสดงผลที่หน้า View (about.blade.php)

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



10. บันทึกไฟล์ทั้งหมดแล้วรันอีกครั้ง http://localhost/laravel5/public/about



เพียงเท่านี้เราก็สามารถสร้างหน้าเพจ และส่งข้อมูลจาก Controller ไปให้ View ได้เรียบร้อย ☺

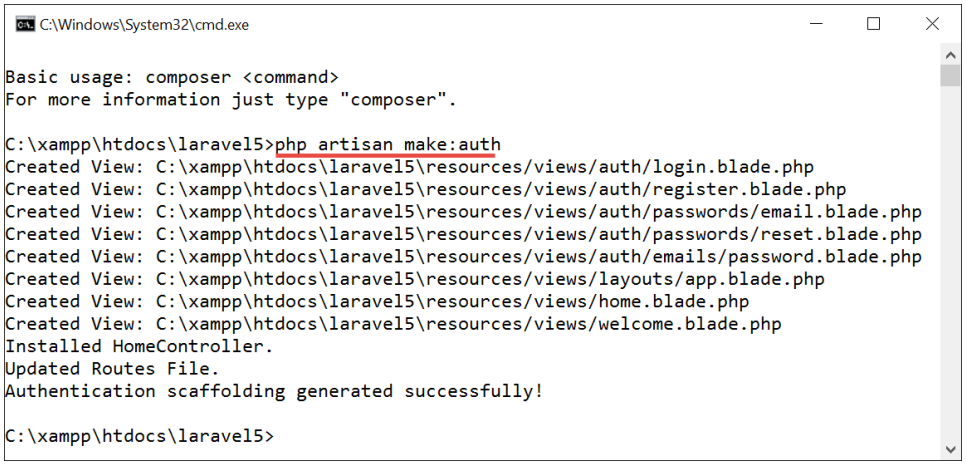
> Note: ขั้นตอนการเขียน route การสร้าง Controller การสร้าง View และการส่งตัวแปรให้แสดงผลที่ Views ควรฝึกและทำความเข้าใจส่วนนี้เยอะๆ เพราะเป็นพื้นฐานสำคัญและได้ใช้บ่อยมาก อาจทดลองโดยการสร้างหน้าเพจอื่นโดยไม่ต้องอ่านหนังสือดูครับ

การสร้างไฟล์ และการจัดการ Layout

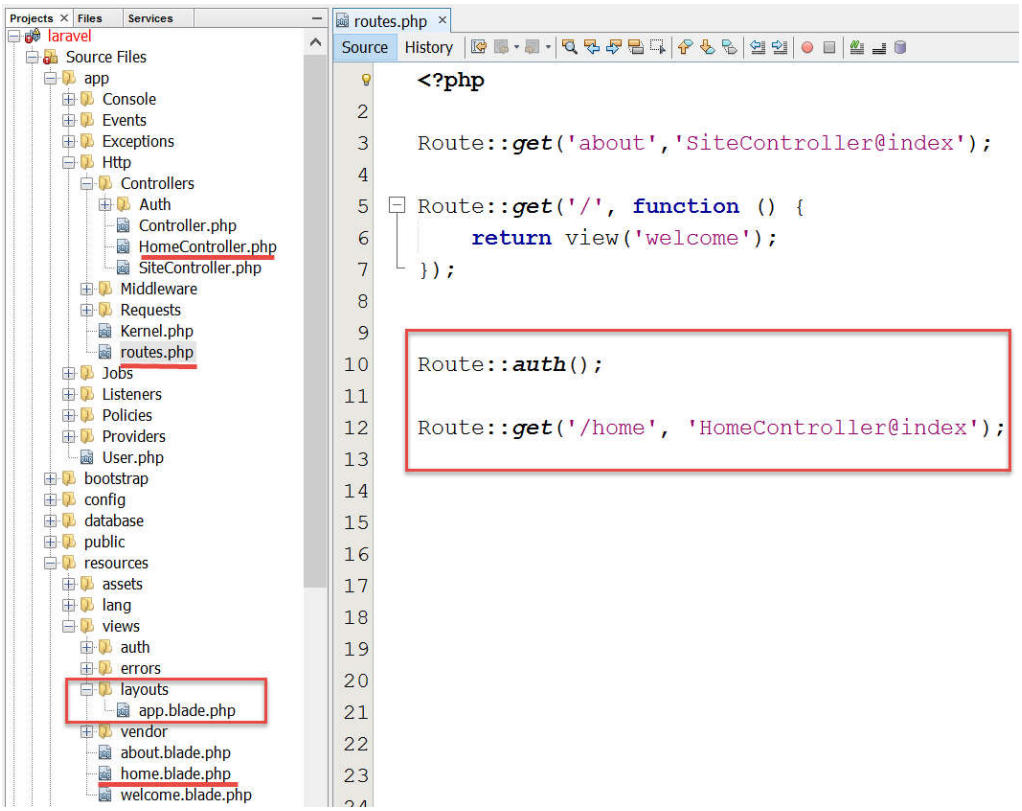
ในกรณีที่เราสร้างหน้าเพจ (View) แล้วมีโค้ด HTML ซ้ำๆกันในแต่ละหน้า แนะนำให้แยกออกมาเป็นไฟล์ layout ต่างหากดีกว่า เพราะเวลาแก้ไขโค้ดจะได้ไม่ต้องตามเปิดแก้ทีละไฟล์ หากเราใช้ layout เราก็สามารถแก้ไขโค้ดได้เพียงจุดเดียว ทำให้ทุกหน้าที่เรียก layout นั้นๆ เปลี่ยนตามที่แก้ในทันที หากเทียบกับการเขียน PHP ปกติ ก็เทียบได้กับคำสั่ง include หรือ require นั่นเอง

ในหนังสือเล่มนี้ เราจะใช้ Bootstrap ซึ่งเป็น CSS Framework ที่ได้รับความนิยม และทำให้โปรเจคของเราสามารถแสดงผลแบบ Responsive ได้เลย ประเด็นคือ ใน Laravel 5.2 ขึ้นไป จะมีการสร้างโค้ดอัตโนมัติให้ในส่วนของการล็อกอินเข้าใช้งานระบบ และตรงนี้เองเราไม่ต้องสร้าง layout เองเลย Laravel จะจัดการให้ มาดูขั้นตอน **การสร้างระบบล็อกอิน** กัน ซึ่งแน่นอนเราจะได้ไฟล์ layout เบื้องต้นมาด้วย (รายละเอียดเกี่ยวกับการจัดการสิทธิ์ผู้ใช้จะอธิบายในบทต่อๆไป)

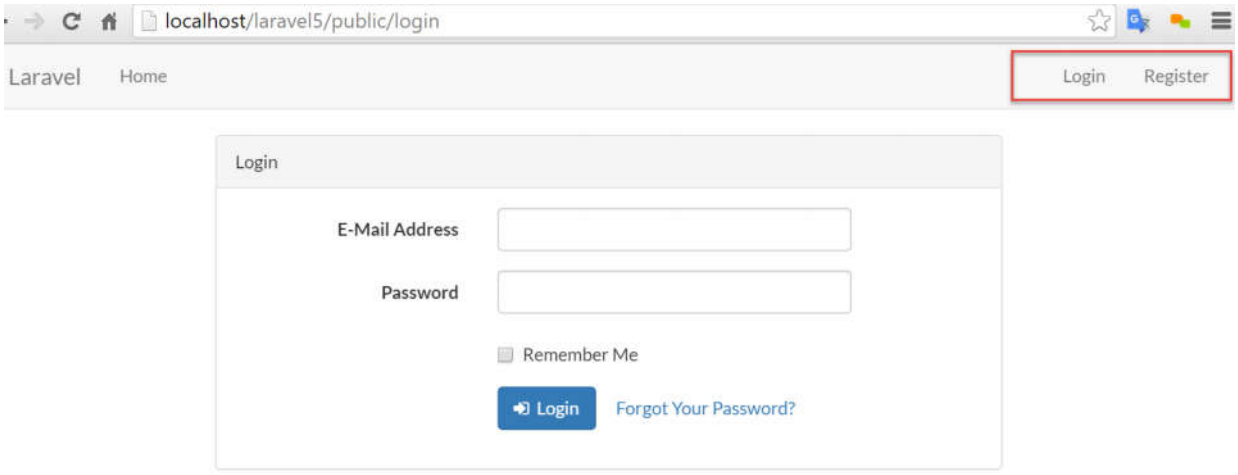
1. เข้าโปรเจคของเราแล้วเปิด Composer พิมพ์คำสั่ง **php artisan make:auth** แล้วกด Enter



2. จากนั้น Laravel จะสร้างโค้ดอัตโนมัติให้เราทั้งในส่วนของ views , HomeController.php และเพิ่มโค้ดในไฟล์ routes.php ให้ด้วย และแน่นอนจะมีการสร้างโฟลเดอร์และ ไฟล์ layout ที่เป็น Bootstrap Framework มาให้เลย (app\resources\views\layouts\app.blade.php)

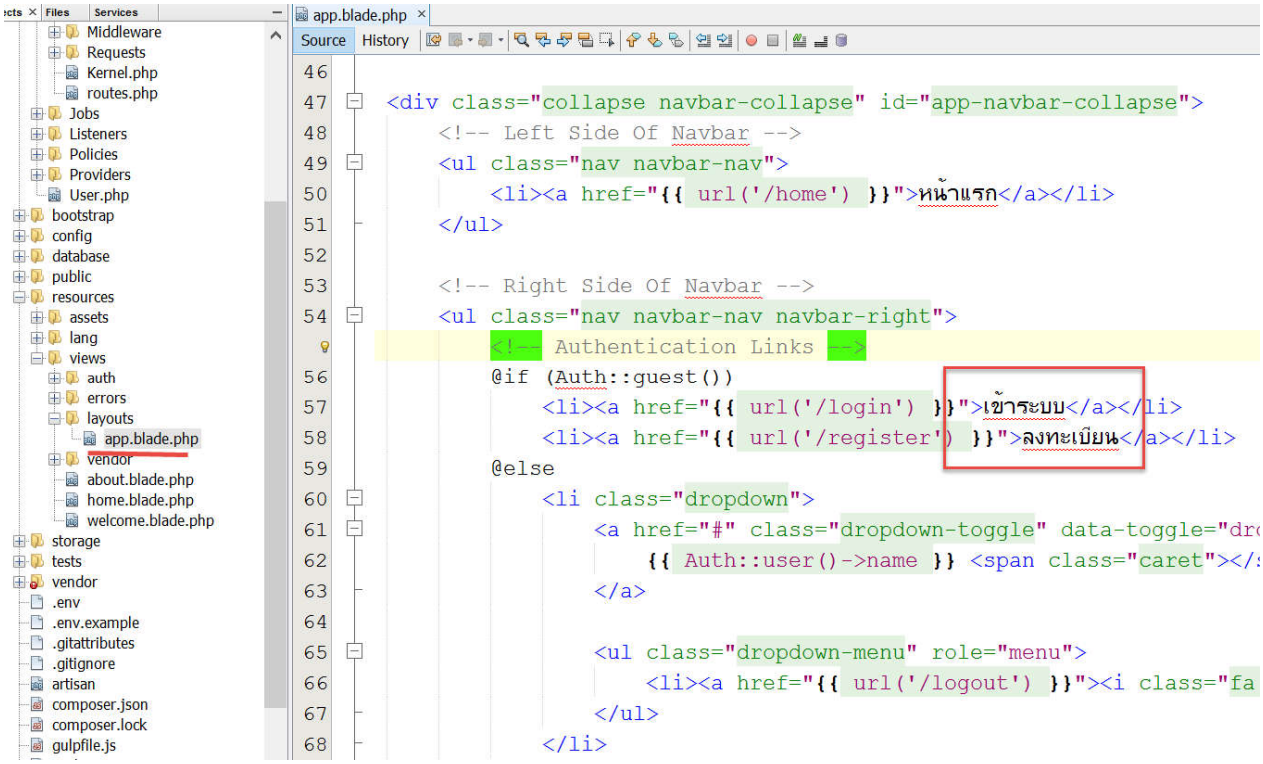


3. ลองทดสอบและเปิดใน Browser http://localhost/laravel5/public/ สังเกตว่าจะมีเมนู Login และเมนู Register มาให้แล้ว!



Note: ตอนนี้ยังไม่สามารถล็อกอินหรือลงทะเบียนได้ เพราะเรายังไม่ได้สร้างตารางในฐานข้อมูลซึ่งจะกล่าวถึงในบทต่อๆ ไป

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



Note: ลองเปิดไฟล์ views ที่เกี่ยวกับระบบล็อกอินได้ที่ในโฟลเดอร์ app\resources\views\auth\ จากนั้นให้ลองเปิดไฟล์แต่ละไฟล์แล้วแก้ข้อความเป็นภาษาไทยได้ครับ

การเรียกใช้ Layout ใน Laravel

การเรียกใช้ Layout ไฟล์ที่จะเรียกใช้ให้เขียนคำสั่ง **@extends**('ชื่อไฟล์ layout ที่ต้องการ') วางไว้ตำแหน่งบนสุดของไฟล์ ส่วนเนื้อหาจะใช้คำสั่ง **@section**('ชื่อที่ตั้งขึ้นจาก @yield ในไฟล์ layout') และปิดท้ายด้วย @endsection เรามาลองปรับหน้าเกี่ยวกับเราที่เคยสร้างไว้แล้วให้เรียกใช้ layout ดูครับ

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

about.blade.php

Source | History

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-12">
            <div class="panel panel-default">
                <div class="panel-heading">เกี่ยวกับเรา</div>
                <div class="panel-body">
                    {{ $fullname }}
                    <br>
                    {{ $website }}
                </div>
            </div>
        </div>
    </div>
</div>
@endsection

```

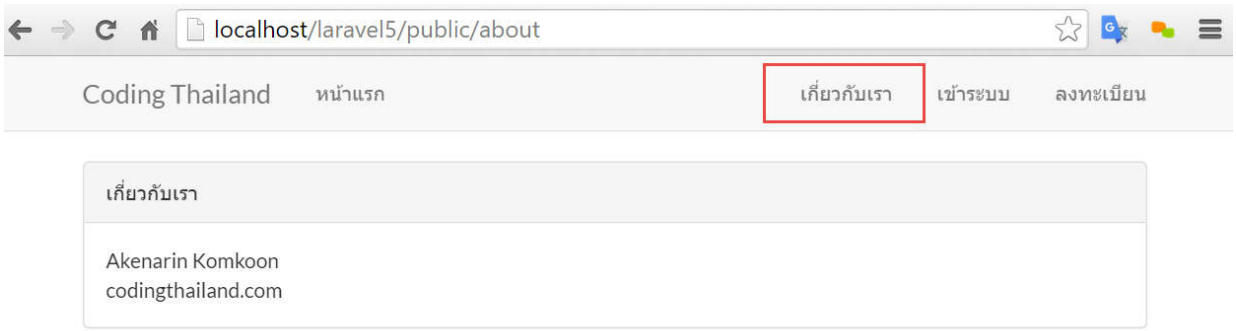
2. ต่อมา เพื่อความสะดวกให้เราสร้างเมนูเพื่อลิงก์มาที่ about ด้วย ให้เปิดไฟล์ app\resources\views\layouts**app.blade.php** แล้วเพิ่มโค้ด ดังนี้

app.blade.php

Source | History

```
49      <ul class="nav navbar-nav">
50          <li><a href="{{ url('/home') }}">หน้าแรก</a></li>
51      </ul>
52
53      <!-- Right Side Of Navbar -->
54      <ul class="nav navbar-nav navbar-right">
55          <!-- Authentication Links -->
56          @if (Auth::guest())
57              <li><a href="{{ url('/about') }}">เกี่ยวกับเรา</a></li>
58              <li><a href="{{ url('/login') }}">เข้าระบบ</a></li>
59              <li><a href="{{ url('/register') }}">ลงทะเบียน</a></li>
60          @else
61              <li class="dropdown">
62                  <a href="#" class="dropdown-toggle" data-toggle="dropdown"
63                      {{ Auth::user()->name }} <span class="caret"></span>
64                  </a>
65
66                  <ul class="dropdown-menu" role="menu">
67                      <li><a href="{{ url('/logout') }}"><i class="fa fa-btn
68                  </ul>
69              </li>
70          @endif
71      </ul>
```

3. เปิด Browser แล้วลองคลิกเมนู **เกี่ยวกับเรา** ก็เป็นอันเสร็จเรียบร้อยครับ สำหรับการนำ layout เข้ามาใช้งาน



Note: การ extend layout มาใช้ด้วยคำสั่ง @extends('layouts.app') โค้ด layouts.app หมายถึง การอ้างอิงไฟล์ app.blade.php ซึ่งอยู่ในโฟลเดอร์ layouts เพราะฉะนั้นหากเราสร้างไฟล์ layout เองก็อย่าลืมอ้างอิงให้ถูกต้องด้วยครับ

การสร้าง Section ใหม่โดยใช้ @yield

หากเราต้องการสร้าง Section ใหม่ให้กับไฟล์ view ใดๆ ที่มาเรียกใช้ layout สามารถกำหนดได้โดยใช้คำสั่ง **@yield('ชื่อที่ตั้งขึ้นมา')** เช่น เราอาจสร้าง **@yield('footer')** ในไฟล์ layout หากหน้า view ใดมีการแทรก JavaScript ก็สามารถเรียกใช้ตรงนี้ได้ การเรียกใช้ก็แค่ใช้คำสั่ง @section('footer') แล้วปิดท้ายด้วย **@endsection** มาลองสร้างกันดูครับ

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

```
app.blade.php
Source | History
        </div>
    </nav>

    @yield('content')

    <!-- JavaScripts -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>
    {{-- <script src="{{ elixir('js/app.js') }}"></script> --}}

    @yield('footer')

</body>
</html>
```

2. เปิดไฟล์ views ใดๆที่ต้องการเรียกใช้ ในที่นี้จะยกตัวอย่างไฟล์ about.blade.php การเรียกใช้ ก็ให้เพิ่มคำสั่ง **@section('content')** แล้วปิดท้ายด้วย **@endsection** หากเราต้องการเขียนโค้ด JavaScript ก็สามารถเขียนตรงได้เลยครับ

```
about.blade.php
Source | History
                {{ $website }}
            </div>
        </div>
    </div>
</div>
@endsection

@section('footer')
    <script>
        alert("Hello, About Page");
    </script>
@endsection
```

3. ลองทดสอบรันดูจะพบว่าโค้ด JavaScript บรรทัดนี้ จะมีการทำงานเฉพาะหน้าที่เรียกใช้เท่านั้น ไม่กระทบกับหน้าอื่นๆเลย

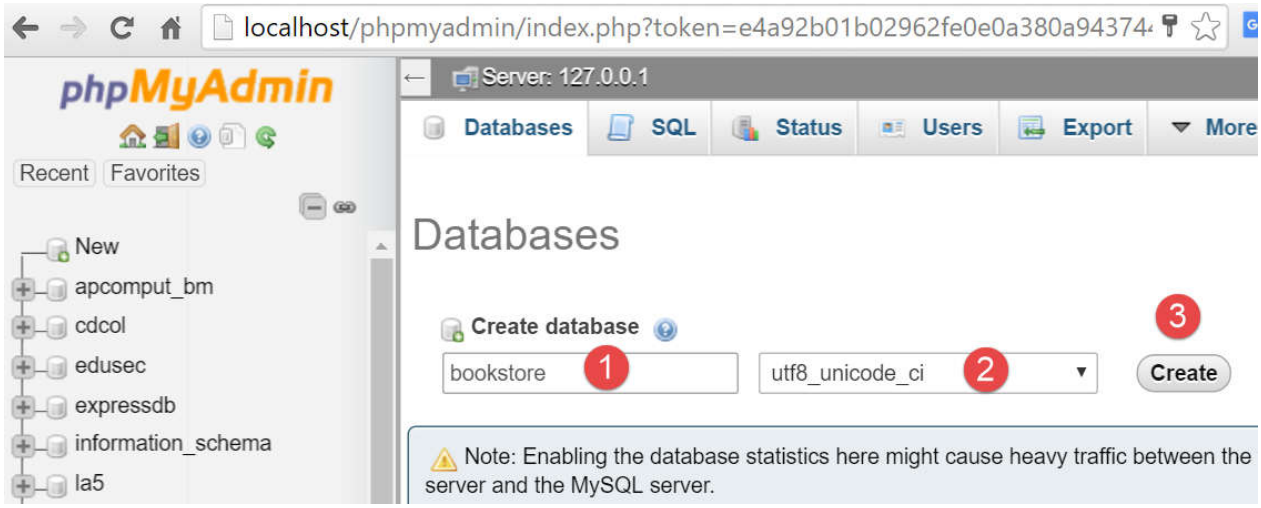
บทที่ 4 ออกแบบฐานข้อมูลและตารางด้วย Artisan, Database Migrations และการทำ Seeding

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

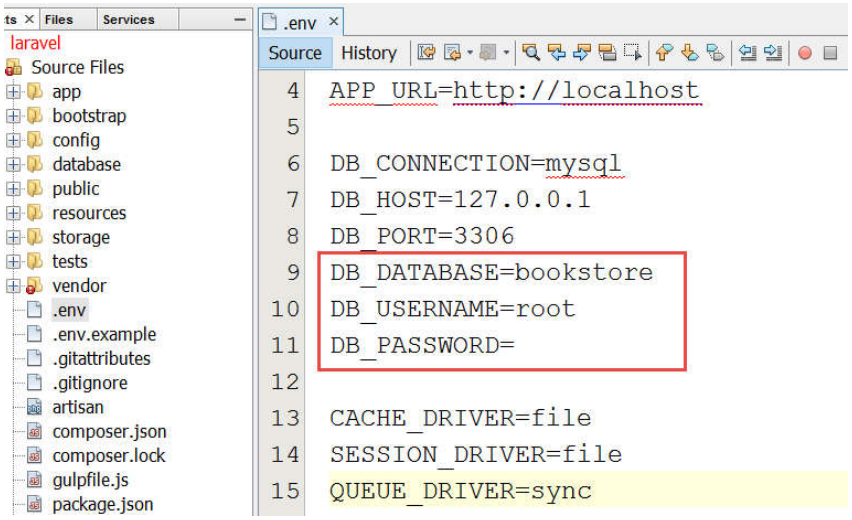
การตั้งค่าฐานข้อมูล

การตั้งค่าฐานข้อมูลเป็นสิ่งที่ควรกำหนดเลย หากเรามีการใช้งานฐานข้อมูลในระบบ เพราะถ้าไม่ตั้งค่า Laravel จะไม่สามารถติดต่อฐานข้อมูลได้ หากเราใช้ MySQL/MariaDB สามารถกำหนดผู้ใช้, รหัสผ่านผู้ใช้, และฐานข้อมูลได้ ในไฟล์ที่ชื่อว่า .env

1. เปิดโปรแกรม phpMyAdmin เปิด Browser แล้วพิมพ์ http://localhost/phpmyadmin เพื่อสร้างฐานข้อมูลใหม่ ในหนังสือเล่มนี้เราจะใช้ฐานข้อมูลชื่อว่า **bookstore** พิมพ์ชื่อฐานข้อมูล แล้วกด Create



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



Note: ในส่วนของ DB_PASSWORD โปรแกรม XAMPP จะไม่ได้กำหนดรหัสผ่านมาให้จึงได้ว่างไว้ แต่หากระบบเรามีชื่อผู้ใช้ หรือรหัสผ่านก็อย่าลืมกรอกข้อมูลให้ตรงด้วย

การสร้างตารางฐานข้อมูลด้วย Migration

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

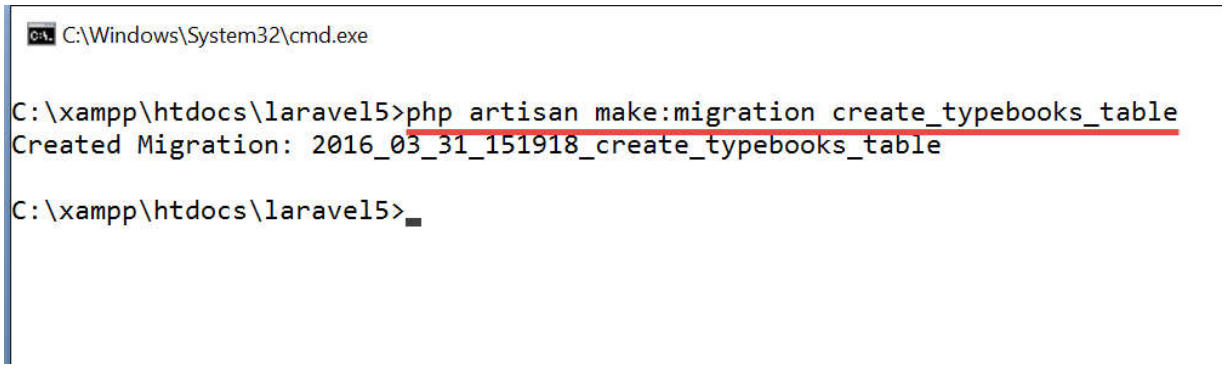
- php artisan make:migration <ชื่อคลาส> เป็นคำสั่งสำหรับสร้างไฟล์ Migration ซึ่งต้องระบุชื่อคลาสด้วย
- php artisan migrate:install เป็นคำสั่งสำหรับสร้างตาราง migrations ในฐานข้อมูล
- php artisan migrate เป็นคำสั่งสำหรับรัน migration
- php artisan migrate:refresh เป็นคำสั่งให้ reset และสั่งรัน migration ใหม่ทั้งหมด
- php artisan migrate:rollback เป็นคำสั่งสำหรับใช้ undo การทำงานก่อนหน้านั้น

Note: การใช้งาน Migration ควรออกแบบฐานข้อมูลให้เสร็จเสียก่อนจะได้ไม่เสียเวลา ถามว่าไม่ต้องใช้ migration ได้หรือเปล่า คำตอบคือ ได้ ขึ้นกับเรา อาจออกแบบด้วย phpMyAdmin แบบปกติก็ได้เช่นเดียวกัน

หลังจากเรียนรู้คำสั่งเกี่ยวกับ Migration แล้ว มาลองสร้างตารางกันได้เลยครับ โดยเราจะสร้าง 2 ตาราง ได้แก่ typebooks (ประเภทหนังสือ) และ ตาราง books (หนังสือ) ส่วนตาราง users และ password_resets นั้น Laravel สร้างมาให้เราเรียบร้อยแล้ว

1. สร้างไฟล์ migration ใหม่ (ตาราง typebooks) เข้าไปที่โฟลเดอร์โปรเจคของเรา เปิด Composer พิมพ์

   php artisan make:migration create_typebooks_table แล้วกด enter

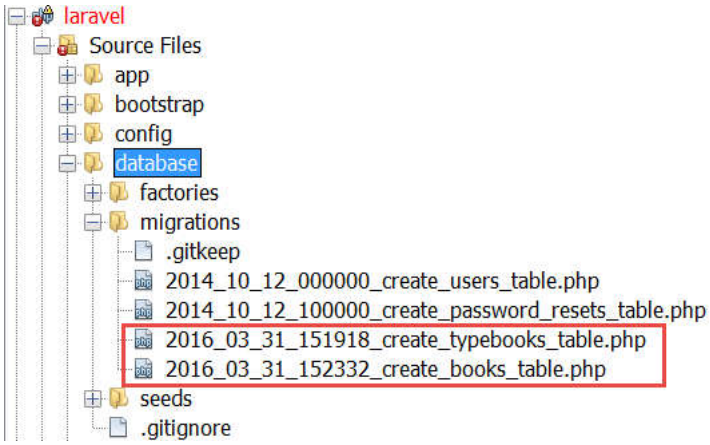


   Note: หากมีการใช้งาน foreign key (FK) ควรสร้างตาราง parent หรือ master ก่อน

2. สร้างไฟล์ migration ใหม่ (ตาราง books) พิมพ์ **php artisan make:migration create_books_table** แล้ว enter อีกครั้ง

```
C:\Windows\System32\cmd.exe

C:\xampp\htdocs\laravel5>php artisan make:migration create_typebooks_table
Created Migration: 2016_03_31_151918_create_typebooks_table

C:\xampp\htdocs\laravel5>php artisan make:migration create_books_table
Created Migration: 2016_03_31_152332_create_books_table

C:\xampp\htdocs\laravel5>_
```

3. เมื่อเราสร้างไฟล์ migration ไฟล์ทั้งหมดสามารถตรวจสอบได้ที่โฟลเดอร์ database\migrations



Note: ตัวเลขด้านหน้าคือวันที่และเวลาที่สร้างขณะนั้น ไม่มีผลกระทบต่อโค้ดอะไร ซึ่งแต่ละคนจะไม่เหมือนกัน

4. เปิดไฟล์ xxx_create_typebooks_table.php เพื่อเขียนโค้ดในการสร้างโครงสร้างของตาราง โดยโค้ดสร้าง table ที่ เมธอด ชื่อว่า up() และเขียนเพื่อลบ table ใน เมธอด ชื่อว่า down() ดังนี้

```php
<?php

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

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

```
            $table->timestamps();
        });
    }

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

5. เปิดไฟล์ xxx_create_books_table.php เพื่อเขียนโค้ดในการสร้างโครงสร้างของตาราง เช่นเดียวกัน โดยตาราง books นี้จะมี FK เพื่อ relation ไปที่ตาราง typebooks ด้วยว่าหนังสือเล่มนี้มีประเภทหนังสืออะไร

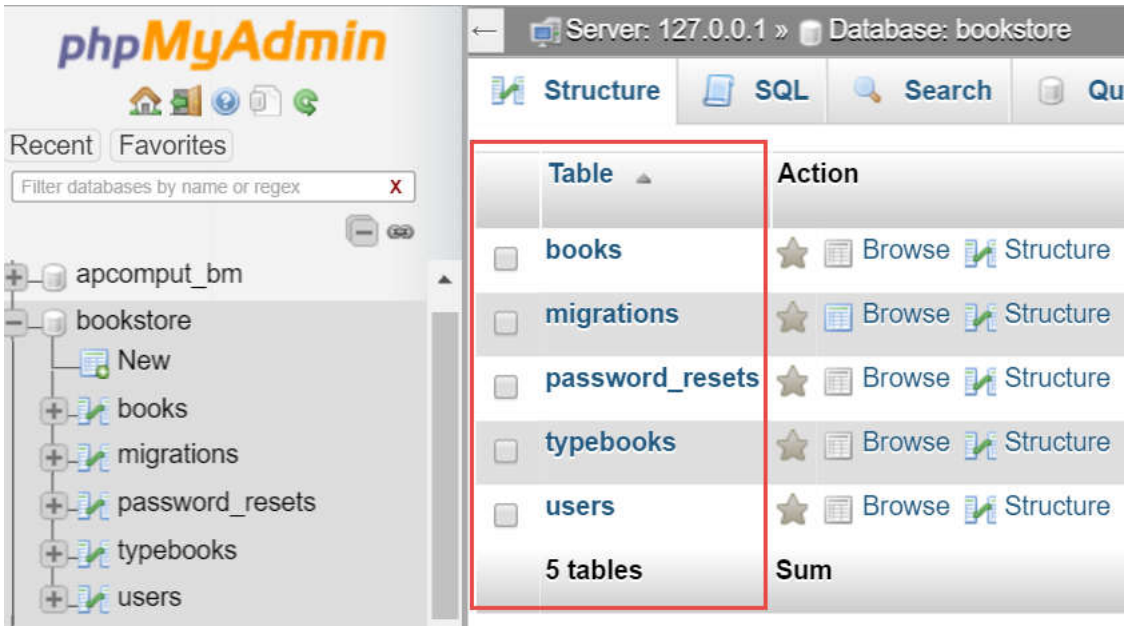
```
<?php

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

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

    /**
     * Reverse the migrations.
```

```
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('books');
    }
}
```

6. เปิด Composer ขึ้นมา พิมพ์ **php artisan migrate:install** แล้ว enter เพื่อสร้างตาราง migrations ในฐานข้อมูล จากนั้นให้พิมพ์คำสั่ง **php artisan migrate** เพื่อสั่งรันไฟล์ migration ทั้งหมด แค่นี้เราก็จะได้ตารางสำหรับฐานข้อมูลมาใช้แล้วครับ โดยสามารถตรวจสอบตารางที่สร้างได้ที่ phpMyAdmin

```
C:\Windows\System32\cmd.exe

C:\xampp\htdocs\laravel5>php artisan migrate:install
Migration table created successfully.

C:\xampp\htdocs\laravel5>php artisan migrate
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrated: 2016_03_31_151918_create_typebooks_table
Migrated: 2016_03_31_152332_create_books_table

C:\xampp\htdocs\laravel5>
```

รายการตารางที่สร้างใน phpMyAdmin



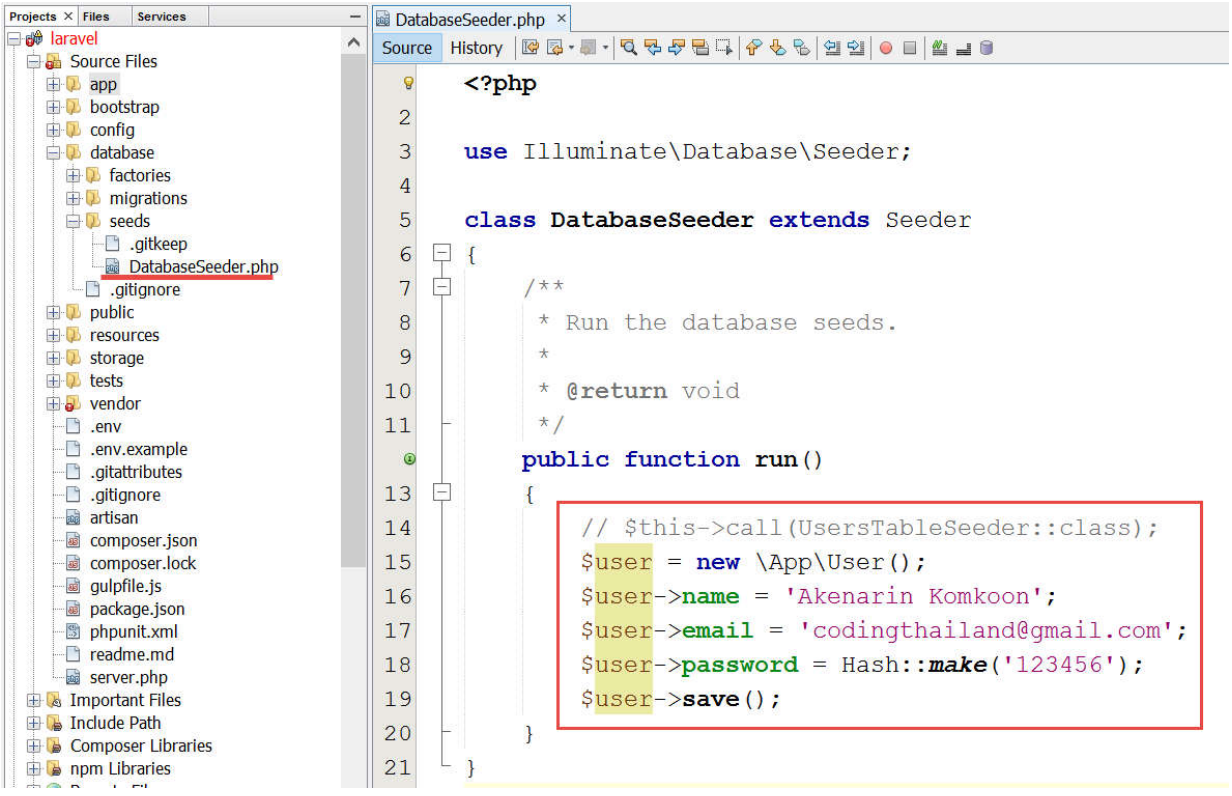
ตัวอย่างอื่น ๆ สำหรับการเขียนโค้ดเพื่อกำหนดโครงสร้างของตาราง สำหรับทำ Migration

- $table->string('comments')->nullable(); กำหนดคอลัมน์และอนุญาตค่า null ได้
- $table->tinyInteger('age')->unsigned(); กำหนดคอลัมน์ให้ไม่ติดเครื่องหมาย
- $table->tiyInteger('age')->unsigned()->default(0); กำหนดค่าปริยาย (default) เป็น 0

การเพิ่มข้อมูลเริ่มต้นลงในตารางด้วย Seeding

เราสามารถเพิ่มข้อมูลเริ่มต้นให้กับแถวในตารางได้ เช่น การตั้งค่าระบบ หรือแม้แต่ชื่อผู้ใช้ หรือรหัสผ่านที่เราต้องการเพิ่มตอนทำ Migration เลย ไฟล์สำหรับการเขียน seeding นั้นจะอยู่ที่โฟลเดอร์ database\seeds ในตัวอย่างนี้จะลองทดสอบสร้างผู้ใช้ในระบบเราขึ้นมา 1 คน มีขั้นตอน ดังนี้

1. เปิดไฟล์ database\seeds\DatabaseSeeder.php แล้วเขียนโค้ดสำหรับเพิ่มข้อมูลในตาราง ดังนี้

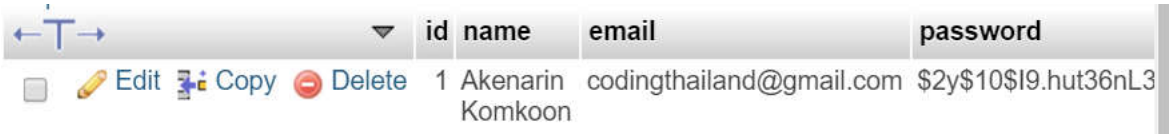


Note: Hash::make() เป็นคำสั่งสำหรับเข้ารหัสของ password

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

php artisan migrate:refresh --seed

Laravel ก็จะสร้างลบ table เก่า แล้วสร้าง table ใหม่พร้อมกับ seeding ให้เลย เมื่อรันคำสั่งแล้วใส phpMyAdmin จะสังเกตว่ามีแถวในตาราง users เพิ่มให้เรียบร้อยแล้ว



	id	name	email	password
Edit Copy 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 เพื่ออ้างถึง table ในฐานข้อมูล และการใช้งาน Eloquent ORM สำหรับการจัดการกับฐานข้อมูลที่ง่ายขึ้น และเขียนโค้ดสั้นลง โดยไม่ต้องใช้ภาษา SQL ครับ

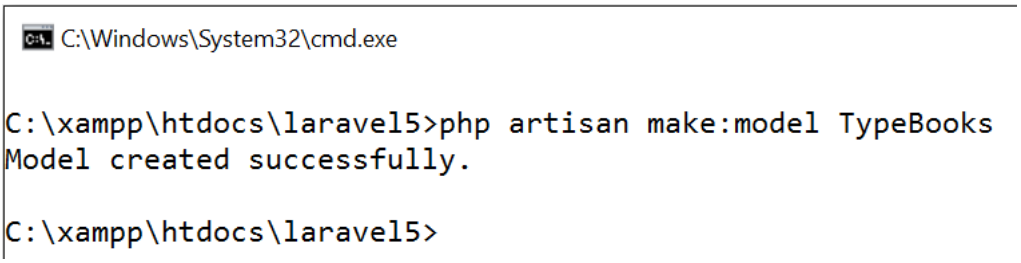
การสร้าง Models

เมื่อสร้างตารางในฐานข้อมูลแล้วลำดับต่อมา คือเราควรสร้าง Model ให้กับตารางแต่ละตาราง การสร้าง Model สามารถใช้ artisan ช่วยในการสร้าง มีรูปแบบดังนี้

php artisan make:model <ชื่อคลาสของโมเดล>

1. สร้าง Model ตาราง typebooks เข้าไปที่โฟลเดอร์โปรเจค เปิด Composer แล้วพิมพ์

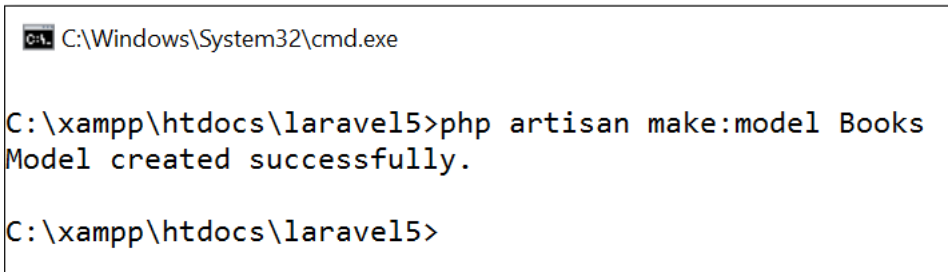
   php artisan make:model TypeBooks



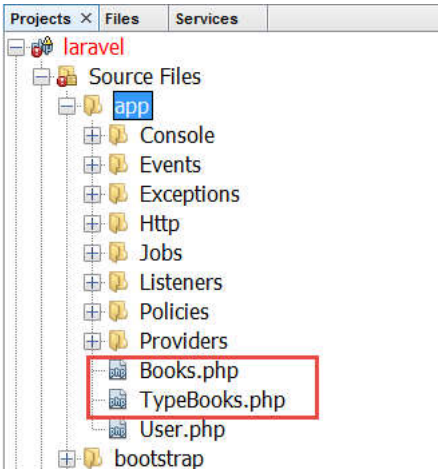
   Note: ตอนที่สร้าง Models หากต้องการทำ migration ด้วยสามารถใส่ -m ต่อท้ายคำสั่งได้ เช่น

   php artisan make:model TypeBooks -m

2. สร้าง Model ตาราง books พิมพ์คำสั่ง **php artisan make:model Books**



3. เมื่อใช้คำสั่งสร้าง Model แล้วไฟล์ Model ค่าปริยายจะอยู่ในโฟลเดอร์ app



4. เปิดไฟล์ app\TypeBooks.php เพื่อเขียนโค้ดกำหนดชื่อ table ดังนี้

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class TypeBooks extends Model
{
    protected $table = 'typebooks'; //กำหนดชื่อตารางให้ตรงกับฐานข้อมูล
}
```

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

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

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

การใช้งาน Eloquent ORM

Eloquent เป็นตัวช่วยเราให้สามารถเขียนโค้ดเพื่อจัดการกับฐานข้อมูลได้ง่ายขึ้น 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 ที่ไม่ใช่ตัวเลข (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','ก%')->get();

คำสั่งสำหรับการเพิ่มข้อมูล และแก้ไขข้อมูล

- ใช้ save() สำหรับเพิ่มหรือแก้ไขข้อมูล
- ใช้ create() สำหรับเพิ่มข้อมูลแบบบรรทัดเดียวหรือเรียกว่า Mass Assignment แต่ก่อนจะใช้ ต้องไปกำหนดฟิลด์ที่ต้องการเพิ่มให้กับตัวแปร $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

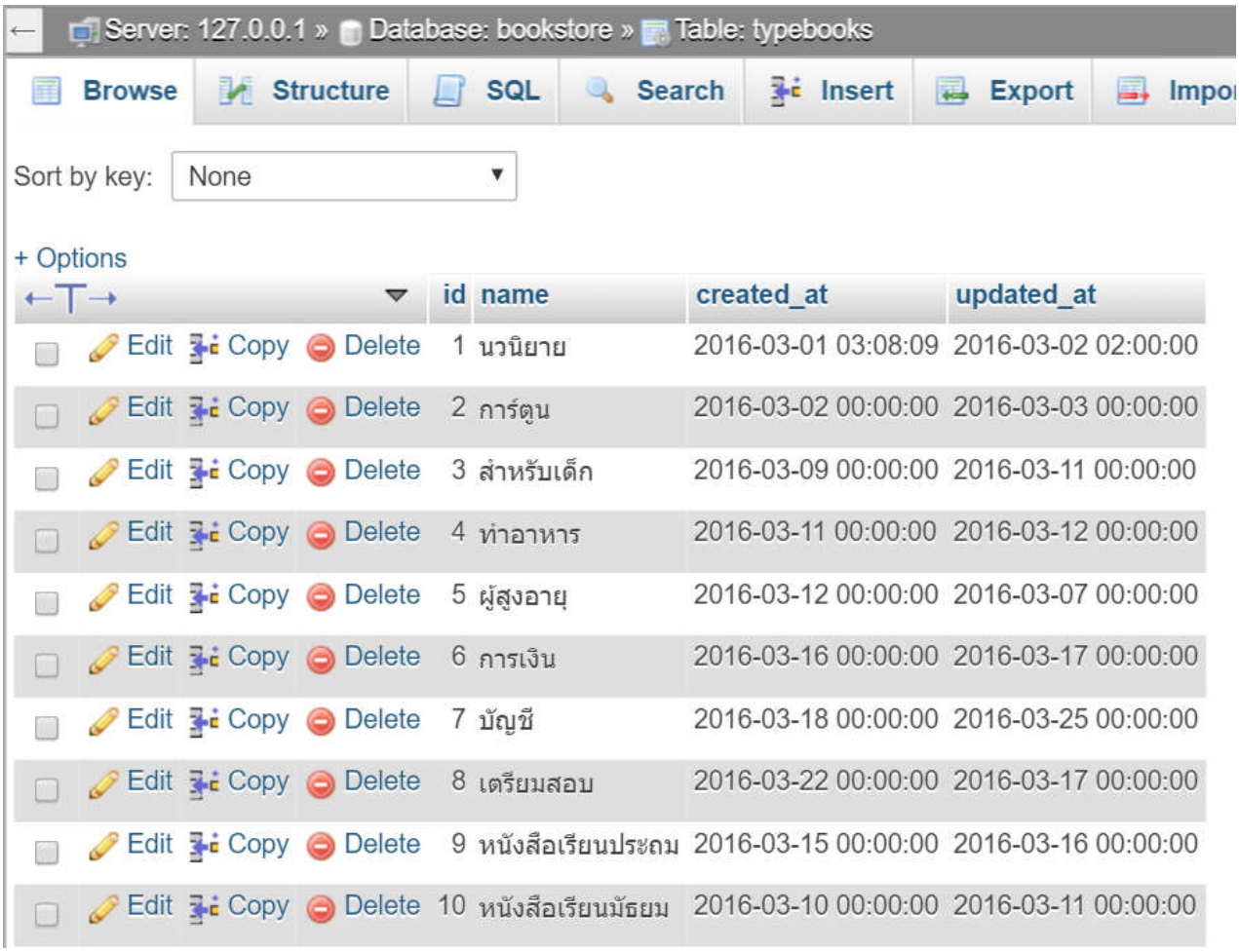
  Cat::destroy(1);

  หรือ

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

แสดงข้อมูลตาราง ประเภทหนังสือ (typebooks)

หลังจากเรียนรู้คำสั่งของ Eloquent แล้ว เราจะมาทดลองสร้างหน้าเพจสำหรับแสดงข้อมูลประเภทหนังสือ แต่ก่อนอื่นแนะนำใช้ phpMyAdmin เพื่อเพิ่มข้อมูลซัก 10 แถวในตาราง typebooks ก่อนครับ เพราะจะได้เห็นข้อมูลเมื่อแสดงผลในหน้าเพจ



	id	name	created_at	updated_at
Edit Copy Delete	1	นวนิยาย	2016-03-01 03:08:09	2016-03-02 02:00:00
Edit Copy Delete	2	การ์ตูน	2016-03-02 00:00:00	2016-03-03 00:00:00
Edit Copy Delete	3	สำหรับเด็ก	2016-03-09 00:00:00	2016-03-11 00:00:00
Edit Copy Delete	4	ทำอาหาร	2016-03-11 00:00:00	2016-03-12 00:00:00
Edit Copy Delete	5	ผู้สูงอายุ	2016-03-12 00:00:00	2016-03-07 00:00:00
Edit Copy Delete	6	การเงิน	2016-03-16 00:00:00	2016-03-17 00:00:00
Edit Copy Delete	7	บัญชี	2016-03-18 00:00:00	2016-03-25 00:00:00
Edit Copy Delete	8	เตรียมสอบ	2016-03-22 00:00:00	2016-03-17 00:00:00
Edit Copy Delete	9	หนังสือเรียนประถม	2016-03-15 00:00:00	2016-03-16 00:00:00
Edit Copy Delete	10	หนังสือเรียนมัธยม	2016-03-10 00:00:00	2016-03-11 00:00:00

ขั้นตอนการแสดงข้อมูลมีดังนี้

1. เปิดไฟล์ app\Http\routes.php เพื่อสร้าง route โดยเราจะสร้าง 2 ตัวเพื่อการแสดงข้อมูล และการลบข้อมูล ดังนี้

routes.php

Source History

```php
<?php

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

//สำหรับแสดงข้อมูลทั้งหมด
Route::get('typebooks','TypeBooksController@index');
//สำหรับลบข้อมูล ตาม id ที่รับมาจาก url (รับแบบ get)
Route::get('typebooks/destroy/{id}','TypeBooksController@destroy');

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


Route::auth();

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

2. เปิด Composer เพื่อพิมพ์คำสั่งสำหรับสร้าง Controller ดังนี้

php artisan make:controller TypeBooksController

```
C:\Windows\System32\cmd.exe

C:\xampp\htdocs\laravel5>php artisan make:controller TypeBooksController
Controller created successfully.

C:\xampp\htdocs\laravel5>
```

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

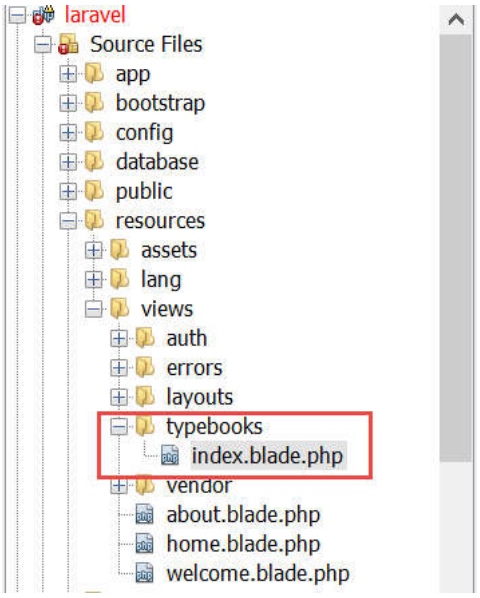
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\TypeBooks;//นำเอาโมเดล TypeBooks เข้ามาใช้งาน

class TypeBooksController extends Controller {

    public function index() {
```

```
        $typebooks = TypeBooks::all();
        //$typebooks = TypeBooks::orderBy('id','desc')->get();
        $count = TypeBooks::count(); //นับจำนวนแถวทั้งหมด
        return view('typebooks.index', [
            'typebooks' => $typebooks,
            'count' => $count
        ]); // ส่งไปที่ views โฟลเดอร์ 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 ส่งไปแสดงผลที่ view เช่นเดียวกัน

ส่วน เมธอด destroy($id) เราจะใช้เพื่อรับค่า primary key จาก URL แล้วทำการลบแถวออกจากตารางครับ

4. สร้างโฟลเดอร์ typebooks และสร้างไฟล์ view ชื่อว่า index.blade.php เพื่อรอรับตัวแปรต่างๆ จาก TypeBooksController เพื่อแสดงผล ดังรูป

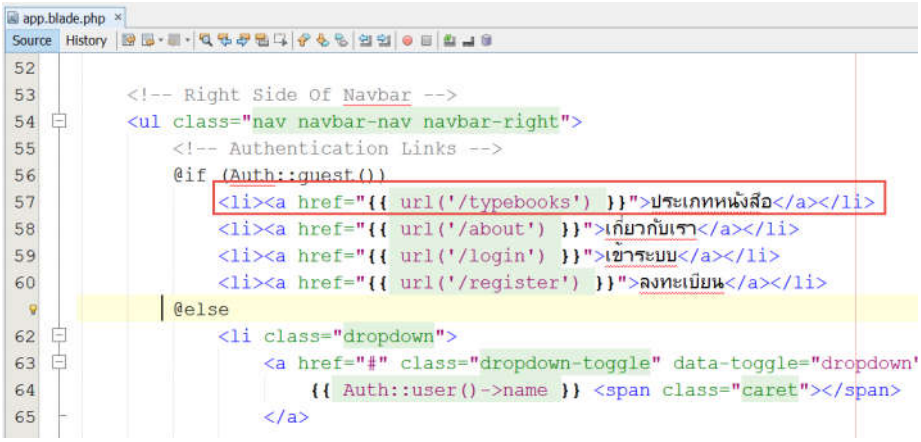


5. จากข้อ 4 เปิดไฟล์ resources\views\typebooks\index.blade.php เขียนโค้ดเพื่อแสดงผล ดังนี้

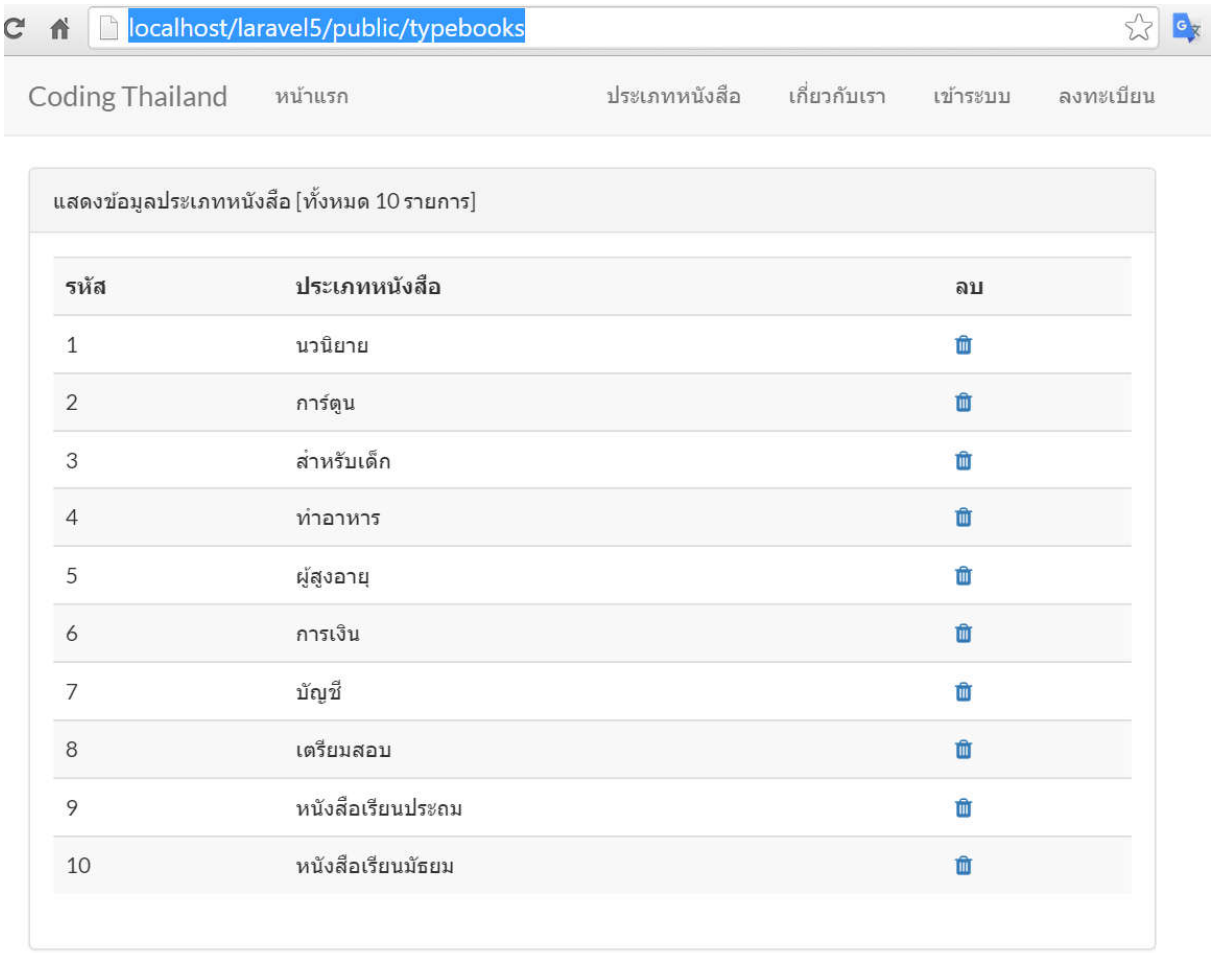
```
@extends('layouts.app')

@section('content')
```

```
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลประเภทหนังสือ [ทั้งหมด {{ $count }}  รายการ]
</div>

                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
                            <th>รหัส</th>
                            <th>ประเภทหนังสือ</th>
                            <th>ลบ</th>
                        </tr>
                        @foreach ($typebooks as $typebook)
                        <tr>
                            <td>{{ $typebook->id }}</td>
                            <td>{{ $typebook->name }}</td>
                            <td><a href="{{
url('/typebooks/destroy/'.$typebook->id)  }}"><i class="fa fa-
trash"></i></a></td>
                        </tr>
                        @endforeach
                    </table>

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

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



7. บันทึกไฟล์ทั้งหมดแล้วลองรันดูครับ http://localhost/laravel5/public/typebooks



การลบข้อมูล ประเภทหนังสือ (typebooks)

จากหัวข้อที่แล้วในส่วนของไฟล์ resources\views\typebooks**index.blade.php** เราได้แทรกไอคอนรูปถังขยะไว้สำหรับให้ผู้ใช้คลิกแล้วลบข้อมูลออกไป ตามโค้ดนี้ **id) }}">** เมื่อผู้ใช้คลิกลบ เราจะส่งค่า id คือ primary key ไปกับ URL เพื่อส่งไปลบยัง เมธอด **destroy**($id) ของ TypeBooksController ครับ

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

```
21      public function destroy($id) {
22          //TypeBooks::find($id)->delete();
23          TypeBooks::destroy($id);
24          return back();
25      }
```

จากนั้นให้ทดสอบลบได้เลยครับ (เมื่อลบแล้วเราใช้ back() เมื่อย้อนกลับ URL ก่อนหน้านี้)

การแบ่งหน้าข้อมูล (Pagination)

หากข้อมูลมีปริมาณมาก การแสดงข้อมูลทั้งหมดในหน้าเดียวอาจทำให้ข้อมูลโหลดได้ช้า เราควรทำการแบ่งหน้าข้อมูล และ Laravel ได้เตรียม เมธอด ให้เราเรียกใช้ไว้แล้วครับ โดยเราจะเขียนโค้ดแบ่งหน้านี้ที่ Controller และอีกส่วนจะเขียนที่ views ได้แก่

- การแบ่งหน้าแบบปกติ จะใช้ **paginate**(จำนวนแถวต่อหน้า) ตัวอย่างเช่น

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

- การแบ่งหน้าอย่างง่าย จะใช้ **simplePaginate**(จำนวนแถวต่อหน้า) ตัวอย่างเช่น

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

- และในส่วนของ view ให้เขียนโค้ดเพื่อ render ดังนี้

  **{!! $persons->render() !!}** //$persons คือ ตัวแปรที่ส่งมาจาก Controller

  และหากต้องการแสดงจำนวนข้อมูลทั้งหมดให้เขียนแบบนี้

  **{{ $persons->total() }}** //$persons คือ ตัวแปรที่ส่งมาจาก Controller

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

มาลองแบ่งหน้าข้อมูลประเภทหนังสือกัน

1. เปิดไฟล์ app\Http\Controllers**TypeBooksController.php** โดยเพิ่มโค้ดแบ่งหน้าที่เมธอด index() ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\TypeBooks;

class TypeBooksController extends Controller {

    public function index() {
        //$typebooks = TypeBooks::all();
        //$typebooks = TypeBooks::orderBy('id','desc')->get();
        $count = TypeBooks::count(); //นับจำนวนแถวทั้งหมด

        //แบ่งหน้า
        //$typebooks = TypeBooks::simplePaginate(5);
```

```
        $typebooks = TypeBooks::paginate(5);

        return view('typebooks.index', [
            'typebooks' => $typebooks,
            'count' => $count
        ]); // ส่งไปที่ views โฟลเดอร์ typebooks ไฟล์ index.blade.php
    }

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

}
```

ให้ลองเปิด-ปิด comment และดูข้อแตกต่างได้ และถ้าเราเขียนโค้ดการแบ่งหน้าก็ไม่ต้องเรียก all() อีกครับ

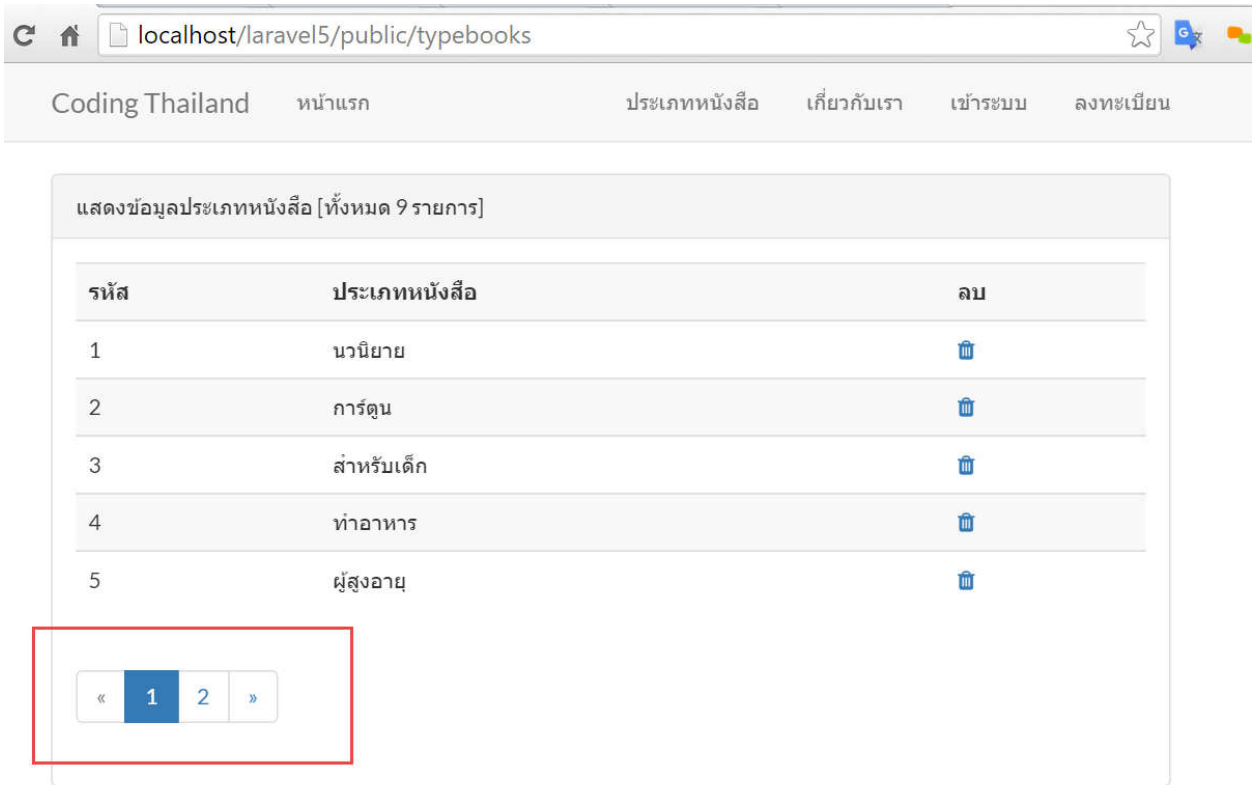
```
        //แบ่งหน้า
        //$typebooks = TypeBooks::simplePaginate(5);
        $typebooks = TypeBooks::paginate(5);
```

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

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลประเภทหนังสือ [ทั้งหมด {{ $count }} รายการ]
</div>

                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
                            <th>รหัส</th>
                            <th>ประเภทหนังสือ</th>
                            <th>ลบ</th>
                        </tr>
                        @foreach ($typebooks as $typebook)
                        <tr>
```

```
                            <td>{{ $typebook->id }}</td>
                            <td>{{ $typebook->name }}</td>
                            <td><a href="{{
url('/typebooks/destroy/'.$typebook->id)  }}"><i class="fa fa-
trash"></i></a></td>
                        </tr>
                        @endforeach
                    </table>

                     {!! $typebooks->render() !!}

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

3. บันทึกไฟล์แล้วทดลองรันดูครับ http://localhost/laravel5/public/typebooks



Query scopes

Query scopes เป็นเทคนิคการรวมเอา query ที่มีความซับซ้อนมาเขียนไว้ที่ Models แทนที่จะเขียนที่ Controllers ประโยชน์คือ ทำให้โค้ด Controller อ่านง่าย สะอาด และยืดหยุ่นขึ้นครับ โดยข้างหน้าชื่อเมธอดจะต้องขึ้นต้นด้วยคำว่า scope เสมอ ตัวอย่างเช่น เราต้องการหาผู้ใช้ที่อายุมากกว่า 18 ปี แทนที่เราจะเขียนโค้ดเยอะๆ 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) ลองนำไปใช้ดูได้

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

การสร้าง Accessors

Accessors หากเรียกง่ายๆอีกชื่อหนึ่งก็คือ getter นั่นเอง เป็นเมธอดที่มีประโยชน์ คือ เราสามารถสร้าง attribute ที่ไม่ใช่ attribute ในฐานข้อมูลได้ โดยให้กำหนดที่ Models นั้นๆ อาจทำการประมวลผล หรือคำนวณค่าข้อมูลจากตาราง เช่น การนำชื่อและนามสกุลมาเชื่อมกัน, การคำนวณราคารวมสินค้า หากเราไม่กำหนดตารางฐานข้อมูล 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 โดยเมธอดที่สร้างขึ้นนั้นจะเป็นการรับค่าพารามิเตอร์เข้ามาเพื่อ set ค่าให้กับ attribute ของ Models

ข้อกำหนดของ Mutators คือ ชื่อเมธอดจะต้องขึ้นต้นด้วยคำว่า 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 นั้นเป็นการกำหนดว่ามีตารางใดบ้างในฐานข้อมูลที่มีความสัมพันธ์กันอยู่ รูปแบบของความสัมพันธ์หรือ relations ที่ใช้บ่อยๆ มีดังต่อไปนี้

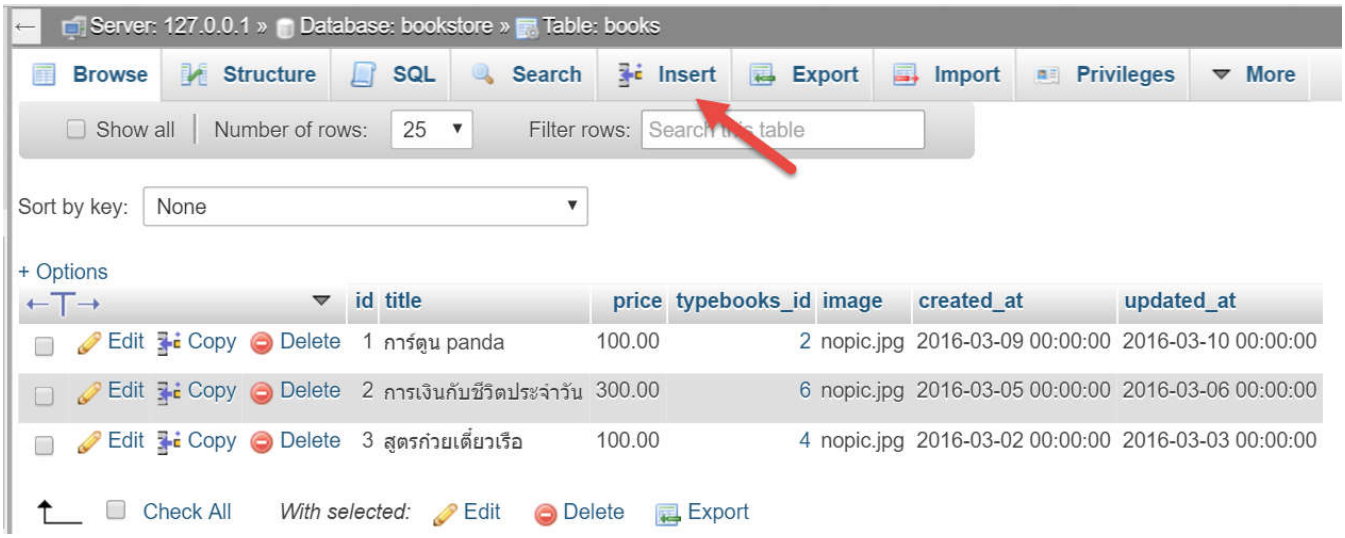
- One To One ความสัมพันธ์แบบหนึ่งต่อหนึ่ง
- One To Many ความสัมพันธ์แบบหนึ่งต่อกลุ่ม หรือเรียกว่า Belongs To Relation ก็ได้
- Many To Many ความสัมพันธ์แบบกลุ่มต่อกลุ่ม

การกำหนด Relation เราจะสร้างเมธอดเพิ่มที่ Models ที่มีความสัมพันธ์กัน เช่น หากตารางใดมีความสัมพันธ์แบบหนึ่งต่อหนึ่งก็ให้เรียกใช้เมธอด hasOne() และอีกตารางที่เชื่อมไปก็ให้กำหนดเป็น belongsTo() พร้อมทั้งระบุ FK ที่ใช้ด้วย เช่นเดียวกันหากมีความสัมพันธ์เป็นแบบ One To Many จะกำหนดเป็น hasMany() และตารางที่เชื่อมกันก็จะใช้ belongsTo() หรือเราเรียก belongsTo() อีกอย่างหนึ่งว่า การ inverse ความสัมพันธ์ก็ได้ ส่วนความสัมพันธ์แบบ Many To Many ให้กำหนดเป็น belongsToMany() ทั้งสองฝั่งครับ

ในหนังสือเล่มนี้จะยกตัวอย่างความสัมพันธ์ที่ใช้บ่อยที่สุดได้แก่ One To Many นั่นเอง (ความสัมพันธ์แบบ One To One ดูได้จากวิดีโอในบทที่ 7 เรื่องการทำ User Profiles)

แสดงข้อมูลตารางหนังสือ (books) ด้วยการทำ relations (join table)

ในหัวข้อนี้เราจะสร้างหน้าเพจเพื่อแสดงข้อมูลจากตารางหนังสือ (books) ซึ่งมีความสัมพันธ์กันกับตารางประเภทหนังสืออยู่ (One To Many) ก่อนอื่นให้เราเปิด phpMyAdmin เพื่อเพิ่มข้อมูลหนังสือ เพื่อเป็นตัวอย่างก่อนนะครับ ตัวอย่างการกรองข้อมูล ดังนี้



สังเกตว่า**คอลัมน์** image **ให้เรากรอกเป็น** nopic.jpg ไว้ก่อน

ขั้นตอนการแสดงข้อมูลหนังสือ มีดังนี้

1. เปิดไฟล์ routes.php เพื่อสร้าง route แต่ครั้งนี้จะเราจะสร้าง route ในรูปแบบของ resource สังเกตว่าจะไม่มีการเติม @ ต่อท้ายชื่อ Controller เราจะให้ Laravel จัดการให้ ดังนี้

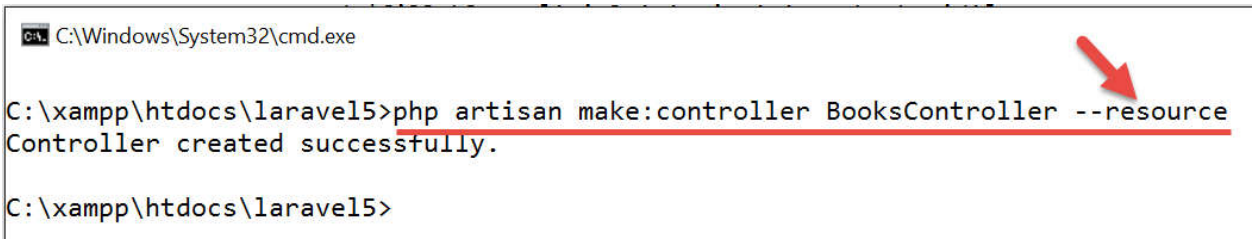
2. กำหนดความสัมพันธ์ระหว่างตาราง typebooks และ books ในรูปแบบของ One to Many เปิดไฟล์ app\TypeBooks.php เพิ่มเมธอดสำหรับกำหนด relations ดังนี้

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class TypeBooks extends Model
{
    protected $table = 'typebooks';

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

เปิดไฟล์ app\Books.php เพื่อทำการ inverse relation โดยใช้

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Books extends Model
{
   protected $table = 'books';
   protected $fillable = ['title','price','typebooks_id'];

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

3. สร้างไฟล์ BooksController.php ในรูปแบบของ resource หรือเรียกว่า RESTful Controller ก็ได้ ให้เข้าไปโฟลเดอร์โปรเจค แล้วเปิด Composer ขึ้นมา พิมพ์คำสั่ง php artisan make:controller BooksController –resource แล้วกด enter

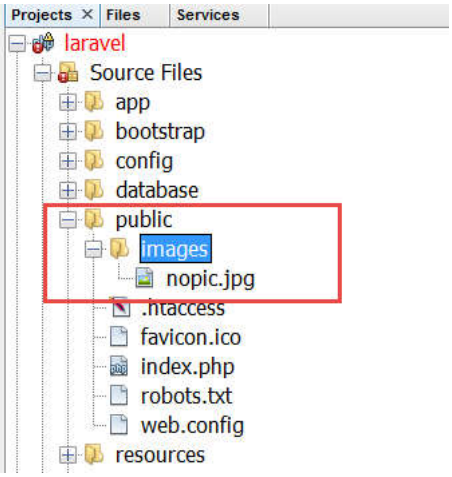


4. จากนั้นลองเปิดไฟล์ BooksController.php จะเห็นว่า Laravel ได้สร้างเมธอดต่างๆ ในรูปแบบของ RESTful มาให้เรียบร้อยโดยที่เราไม่ต้องสร้างเอง (แนะนำวิธีนี้)
5. จากนั้นลองเปิดไฟล์ BooksController.php เขียนคำสั่งที่เมธอด index() เพื่อดึงข้อมูลหนังสือโดยใช้เมธอด with() เพื่อเชื่อม relation กับ typebooks แล้วส่งรายการหนังสือทั้งหมดไปที่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

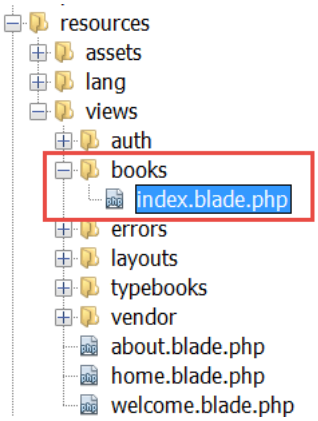
```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Books; //อย่าลืม use โมเดลเข้ามาใช้งาน

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()
    {
        //
    }

    /**
     * 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\image (อย่าลืมสร้างโฟลเดอร์ image ก่อน) ดังนี้



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



8. เปิดไฟล์ index.blade.php จากข้อ 6 แล้วเขียนคำสั่งเพื่อแสดงผลในรูปแบบตารางดังนี้

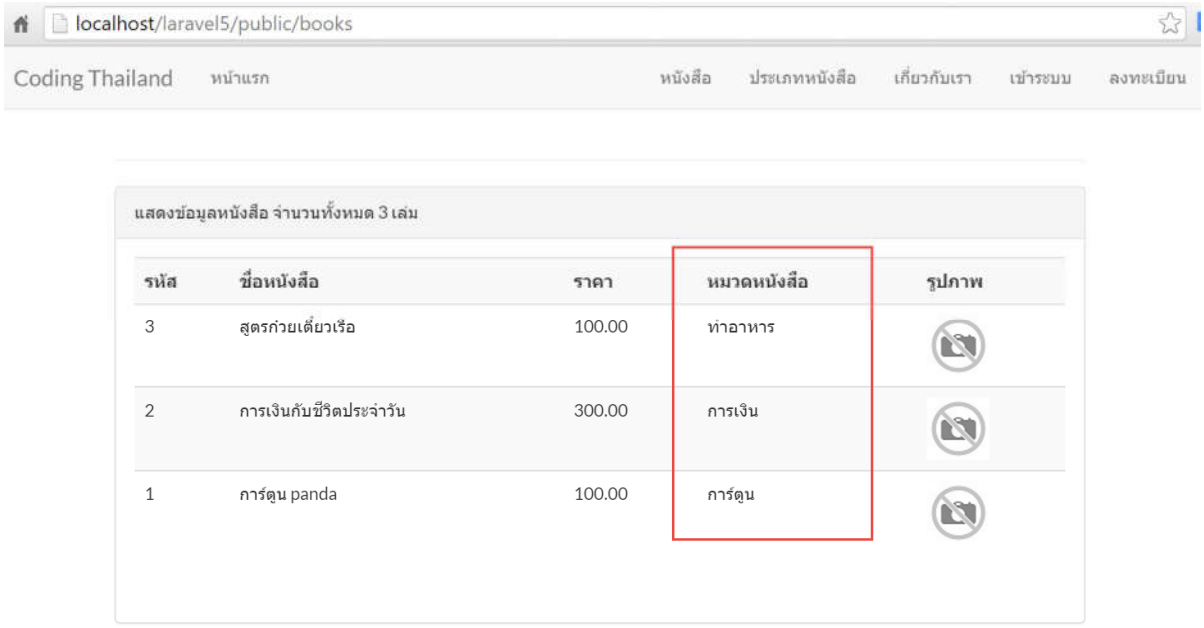
```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

            <hr>

            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{    $books-
>total()   }} เล่ม</div>
                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
```

```
                        <th>รหัส</th>
                        <th>ชื่อหนังสือ</th>
                        <th>ราคา</th>
                        <th>หมวดหนังสือ</th>
                        <th>รูปภาพ</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)
}}"><img src="{{ asset('images/'.$book->image) }}"></a>
                        </td>
                    </tr>
                    @endforeach
                </table>
                <br>
                {!!  $books->render()   !!}



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

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

```
53  -- Right Side Of Navbar -->
54  . class="nav navbar-nav navbar-right">
55    <!-- Authentication Links -->
56    @if (Auth::guest())
57        <li><a href="{{ url('/books') }}">หนังสือ</a></li>
58        <li><a href="{{ url('/typebooks') }}">ประเภทหนังสือ</a></li>
59        <li><a href="{{ url('/about') }}">เกี่ยวกับเรา</a></li>
60        <li><a href="{{ url('/login') }}">เข้าระบบ</a></li>
61        <li><a href="{{ url('/register') }}">ลงทะเบียน</a></li>
```

10. ลองรันทดสอบ จะเห็นว่า ข้อมูลประเภทหนังสือที่มีความสัมพันธ์กันกับหนังสือ ได้แสดงขึ้นมาเรียบร้อย ☺



Note: รูปภาพ nopic.jpg ควรย่อขนาดให้เล็กลงก่อนเพื่อความสวยงาม และเหมาะสม

บทที่ 6 การสร้าง Web Forms การตรวจสอบความถูกต้องของข้อมูลและการอัพโหลดไฟล์

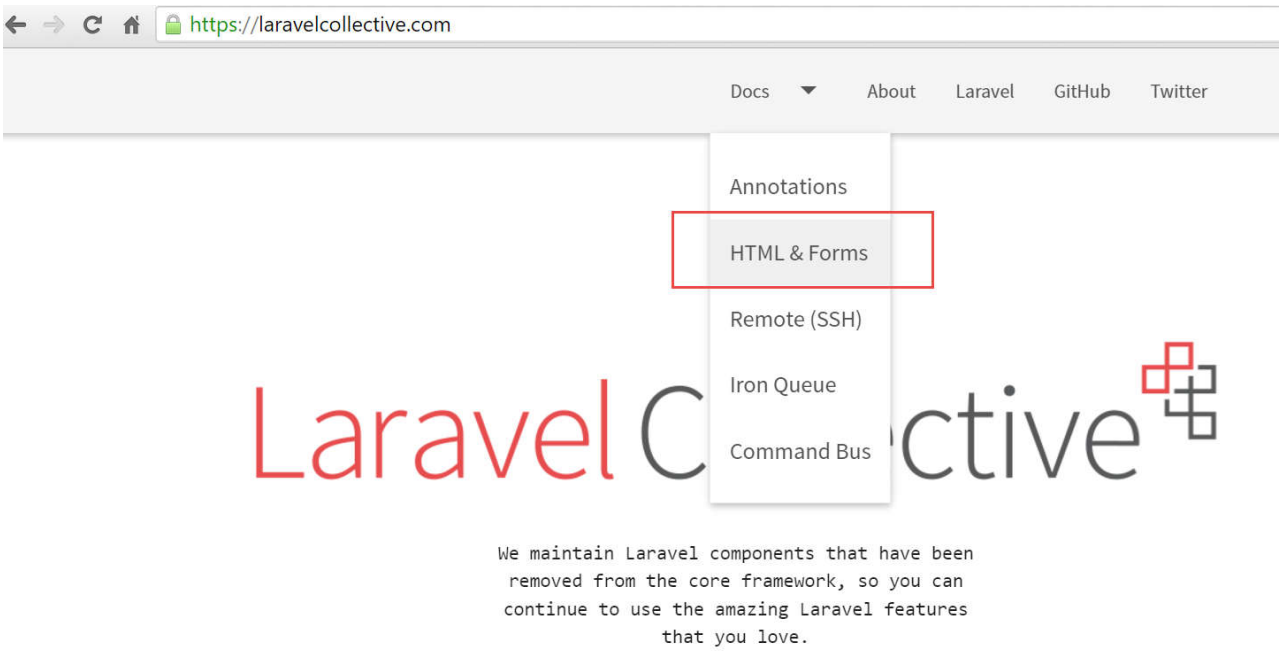
การสร้างฟอร์มใน Laravel

การสร้างฟอร์มใน Laravel มี 2 วิธี ได้แก่

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

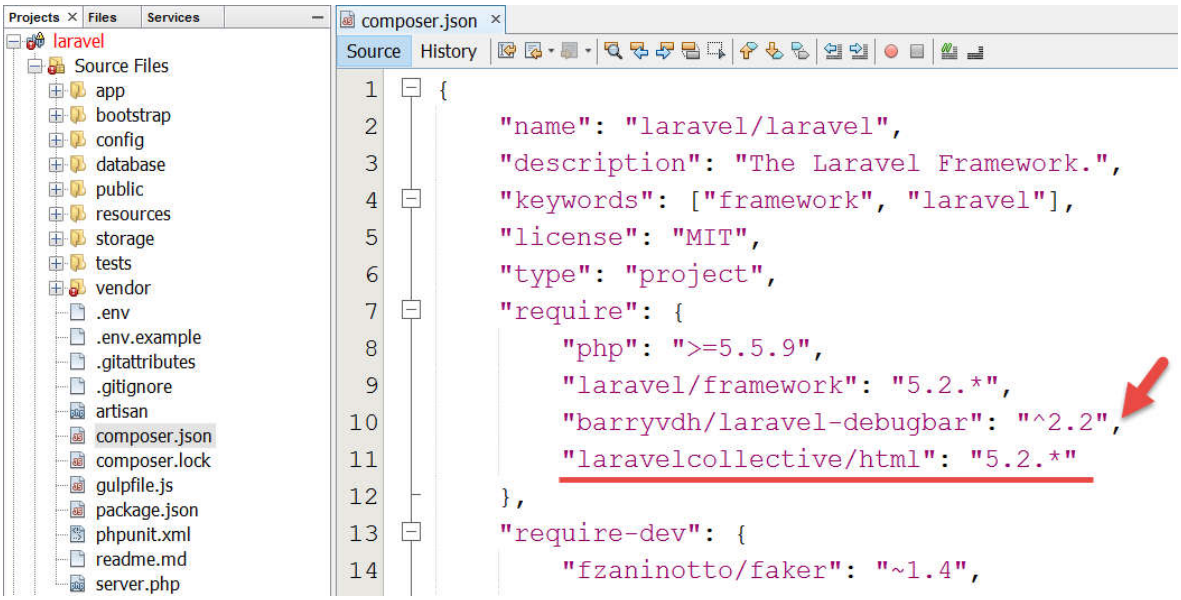
การติดตั้ง และใช้งาน Laravel Collective

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

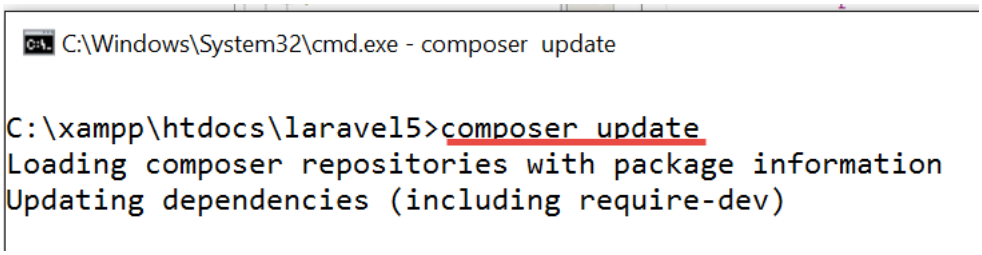


ขั้นตอนการติดตั้ง

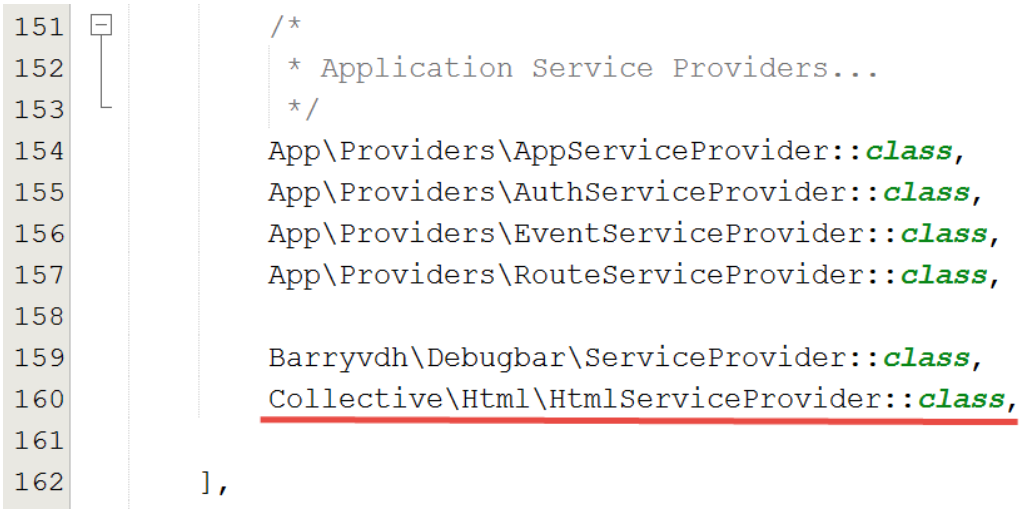
1. เปิดไฟล์ composer.json แล้วพิมพ์คำสั่ง "laravelcollective/html": "5.2.*" ดังรูป (อย่าลืมใส่คอมม่าด้วย)



2. เข้าไปในโฟลเดอร์โปรเจคของเรา เปิด Composer ขึ้นมา พิมพ์คำสั่ง composer update กด enter เพื่อติดตั้ง



3. เปิดไฟล์ config/app.php เพิ่มโค้ดในส่วนของ Application Service Providers ดังนี้ (สามารถ copy ได้ในหน้าคู่มือ)



และในส่วนของ Class Aliases ดังนี้

```
203             'Storage' => Illuminate\Support\Facades\Storage::class,
204             'URL' => Illuminate\Support\Facades\URL::class,
205             'Validator' => Illuminate\Support\Facades\Validator::class,
206             'View' => Illuminate\Support\Facades\View::class,
207
208             'Debugbar' => Barryvdh\Debugbar\Facade::class,
209             'Form' => Collective\Html\FormFacade::class,
210             'Html' => Collective\Html\HtmlFacade::class,
211
212         ],
213
214     ];
```

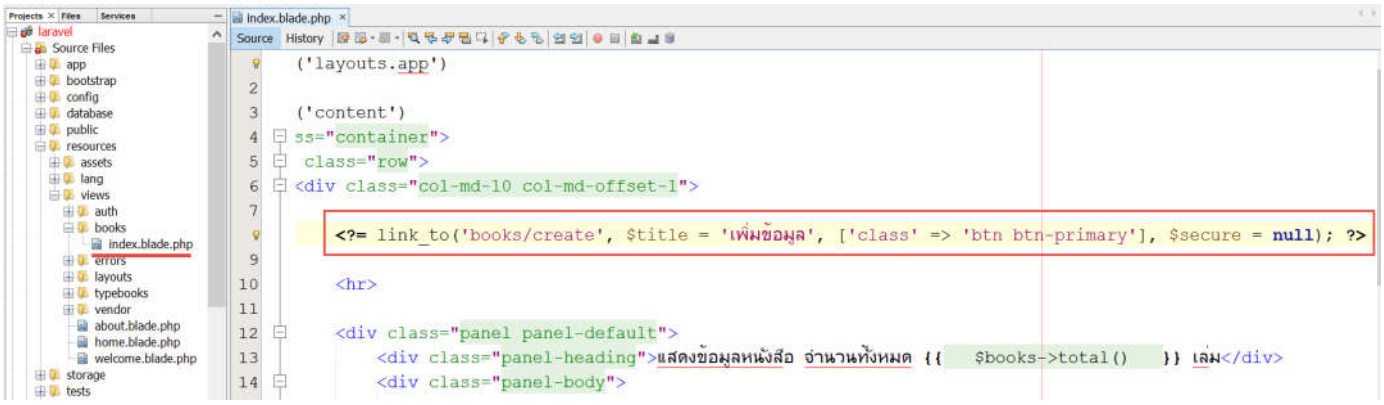
เพียงเท่านี้เราก็สามารถเรียกใช้คลาส Form และ คลาส Html ได้แล้วครับ

สร้างฟอร์มเพิ่มข้อมูลหนังสือ (books)

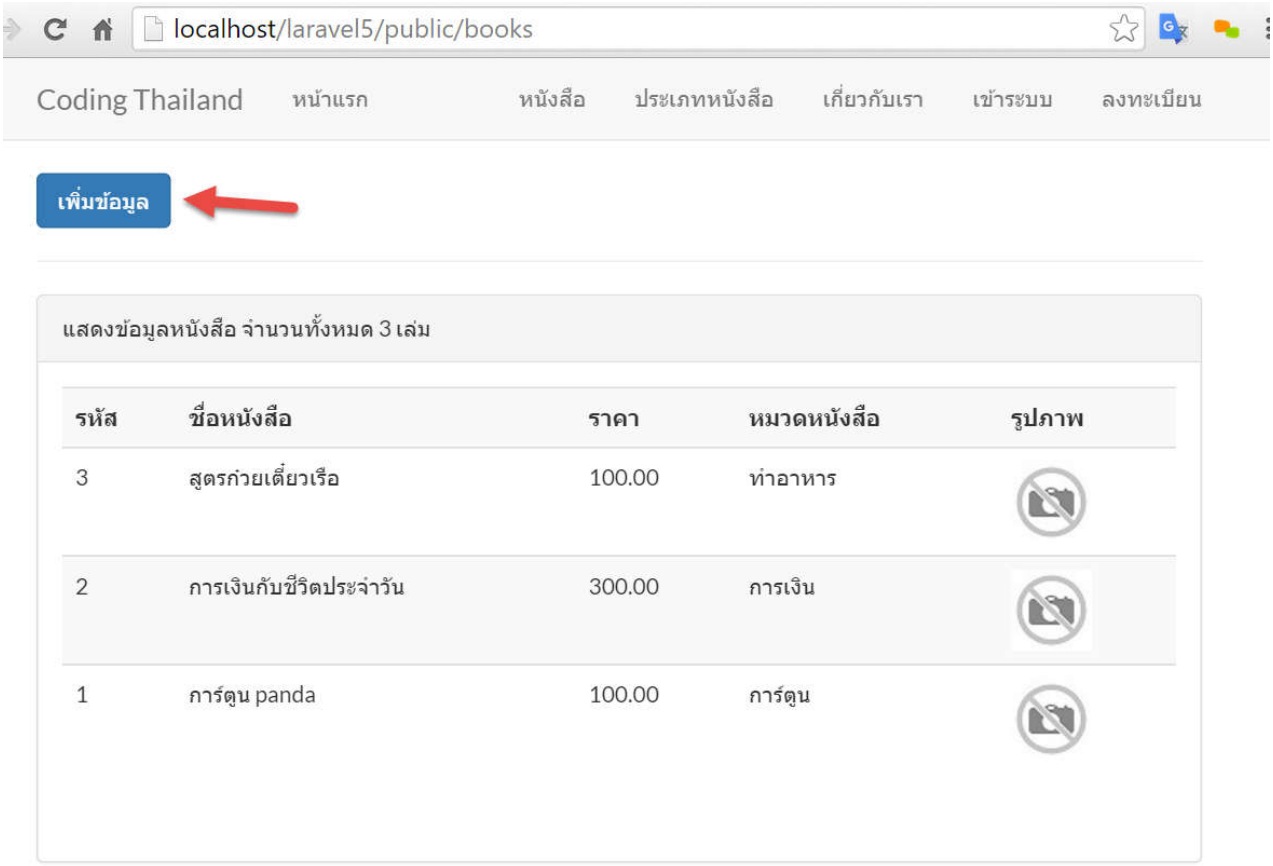
หลังจากที่เราติดตั้ง Laravel Collective เรียบร้อย เราก็สามารถสร้างฟอร์ม ปุ่ม หรือลิงก์ต่างๆได้ เพื่อเป็นการทดสอบว่าเราติดตั้ง Laravel Collective สมบูรณ์หรือไม่ ลองสร้างลิงก์ที่อยู่ในรูปแบบปุ่ม ดังนี้

เปิดไฟล์ resources\views\books\index.blade.php แล้วเพิ่มคำสั่ง เมธอด link_to() เพิ่มสร้างลิงก์เพิ่มข้อมูล ดังนี้

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



บันทึกไฟล์แล้วลองรันดู หากมีปุ่มลิงก์เพิ่มเข้ามาแสดงว่าการติดตั้งเรียบร้อยดี ไม่มีปัญหา



เมื่อกดปุ่มเพิ่มข้อมูล ต่อไปเราจะมาสร้างฟอร์มเพิ่มข้อมูลหนังสือ โดยเราต้องสร้าง views รองรับ และเขียนเมธอดที่ Controller ให้ตรงกับเมธอดที่ลิงก์ไปด้วย

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

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

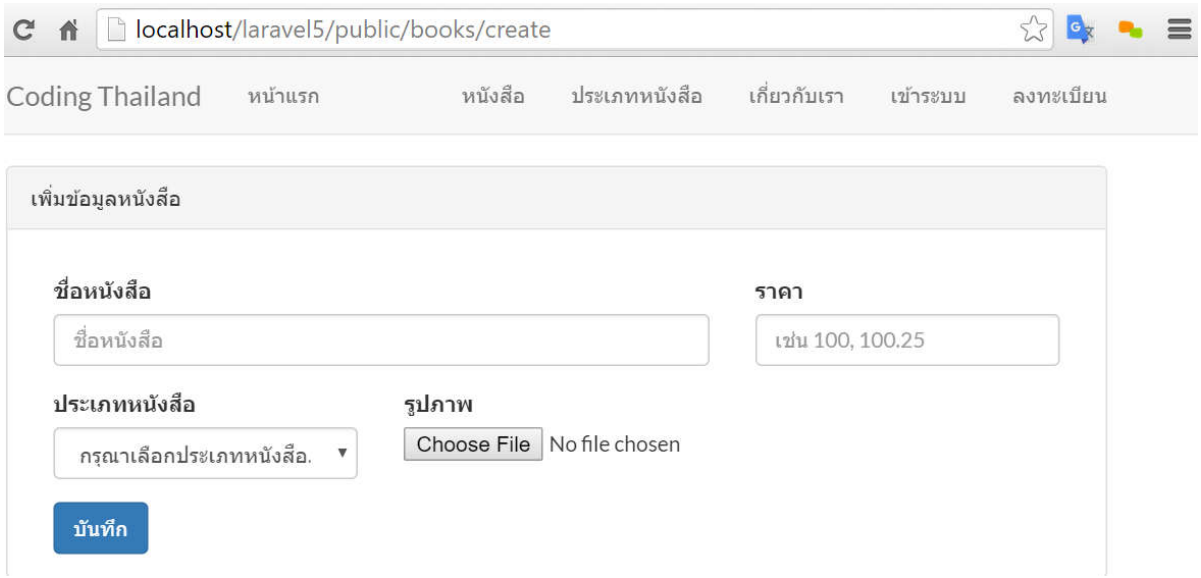
2. มาที่ views ให้สร้างไฟล์ create.blade.php ในโฟลเดอร์ books ดังนี้

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

            <div class="panel panel-default">
                <div class="panel-heading">เพิ่มข้อมูลหนังสือ</div>
```

```
                <div class="panel-body">

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

                    <div class="col-xs-8">
                        <div class="form-group">
                            <?= Form::label('title', 'ชื่อหนังสือ'); ?>
                            <?= Form::text('title', null, ['class' => 'form-
control', 'placeholder' => 'ชื่อหนังสือ']); ?>
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('price', 'ราคา');  !!}
                            {!! Form::text('price',null,['class' => 'form-
control','placeholder' => 'เช่น 100, 100.25']); !!}
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('typebooks_id', 'ประเภทหนังสือ');  !!}
                            <?= Form::select('typebooks_id',
App\TypeBooks::lists('name', 'id'), null, ['class' => 'form-control',
'placeholder' => 'กรุณาเลือกประเภทหนังสือ...']); ?>
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('image', 'รูปภาพ');  !!}
                            <?= Form::file('image', null, ['class' => 'form-
control']) ?>
                        </div>
                    </div>

                    <div class="form-group">
                        <div class="col-sm-10">
                            <?= Form::submit('บันทึก', ['class' => 'btn btn-
primary']); ?>
                        </div>
                    </div>

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

                </div>
```

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

3. ทดสอบโดยการคลิกปุ่ม เพิ่มข้อมูล เราจะได้หน้าเพจสำหรับเพิ่มข้อมูลเรียบร้อย พร้อมทั้งเลือกประเภทหนังสือได้ด้วย



อธิบายเพิ่มเติม การใช้ฟอร์มนั้นจะมีต้องการเปิด และปิดฟอร์ม เสมอ การเปิดฟอร์มจะใช้คำสั่ง

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

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

หากฟอร์มของเรามีการอัพโหลดไฟล์ด้วย ให้ระบุ 'files' => true ตอนเปิดฟอร์มนั่นเอง

การดึงข้อมูลใส่ใน dropdown list เราสามารถเรียกใช้ method lists() ได้เลย ตัวอย่างเช่น

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

หรือหากมีเงื่อนไขก็ใช้ where ตัวอย่างเช่นกัน

```
$items = Items::where('active', true)->orderBy('name')->lists('name', 'id');
```

การตรวจสอบความถูกต้องของข้อมูล (Validation)

เมื่อสร้างฟอร์มเสร็จเรียบร้อยแล้ว ก่อนกดปุ่มบันทึกควรมีความตรวจสอบความถูกต้องของข้อมูลในฟอร์มก่อน เช่น ตรวจเช็คว่าผู้ใช้กรอกข้อมูลมาหรือไม่ กรอกข้อมูลมาถูกต้องตามรูปแบบหรือเปล่า เป็นต้น ใน Laravel จะมีกฎในการตรวจสอบความถูกต้องของข้อมูลสำเร็จรูปมาให้แล้ว สามารถกำหนดได้ตามสะดวก

Note: สามารถดูกฎ (rules) ทั้งหมดได้ที่นี่ https://laravel.com/docs/5.2/validation#available-validation-rules

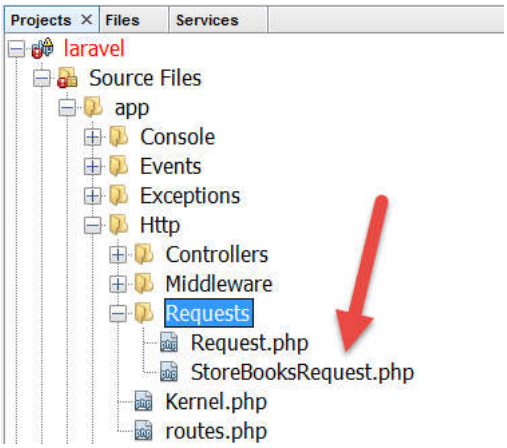
ขั้นตอนการสร้าง และตรวจสอบความถูกต้องของข้อมูลจากฟอร์ม

1. สร้าง request สำหรับตรวจสอบความถูกต้องของข้อมูล โดยให้เข้าไปในโปรเจคของเรา แล้วเปิด Composer จากนั้นพิมพ์คำสั่ง

   php artisan make:request StoreBooksRequest แล้วกด enter

   ```
   C:\Windows\System32\cmd.exe

   C:\xampp\htdocs\laravel5>php artisan make:request StoreBooksRequest
   Request created successfully.

   C:\xampp\htdocs\laravel5>
   ```

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

3. เปิดไฟล์ StoreBooksRequest.php เพื่อเขียนโค้ดกฎการตรวจสอบ และข้อความโต้ตอบที่จะแสดงให้กับผู้ใช้ทราบ ดังนี้

```php
<?php

namespace App\Http\Requests;

use App\Http\Requests\Request;

class StoreBooksRequest extends Request
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        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 ที่เมธอด store เปลี่ยนเป็น StoreBooksRequest แทน ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Books;
use App\Http\Requests\StoreBooksRequest;

class BooksController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
```

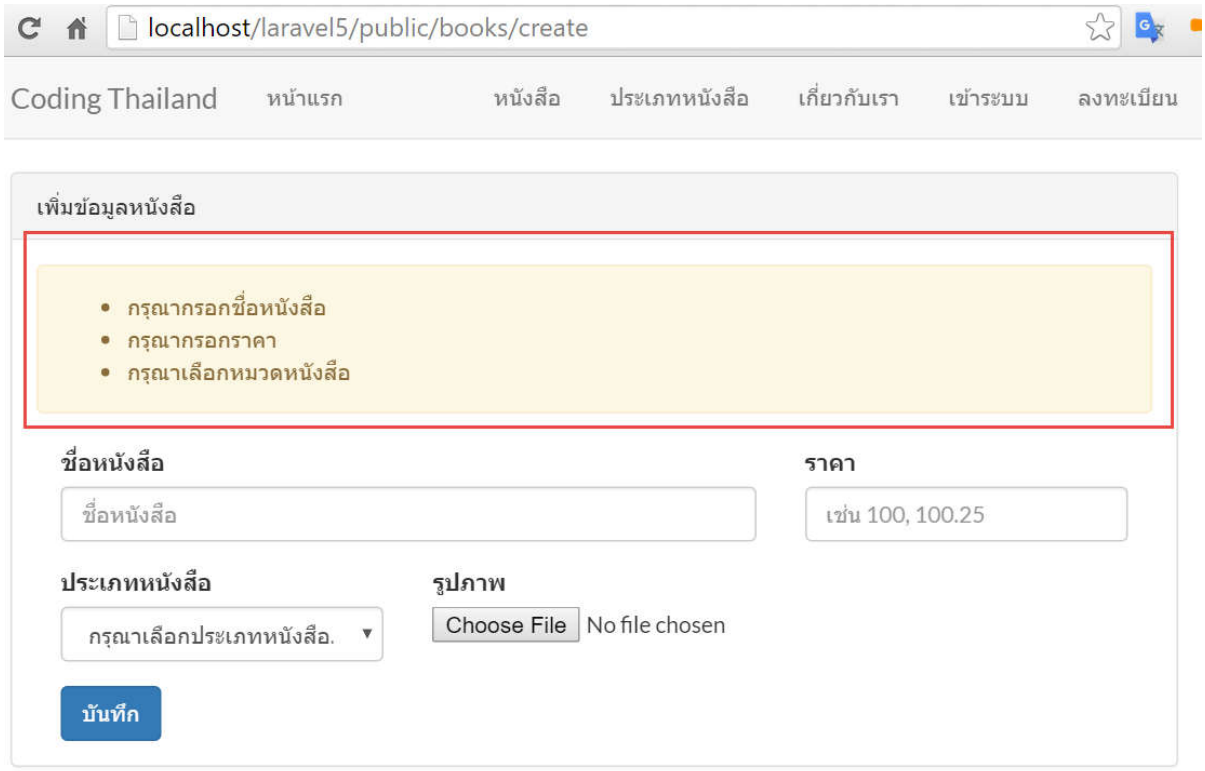
```
 */
 public function index() {
    $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
    return view('books/index',['books' => $books]); //books/index.blade.php
    //return View::make('books/index', array('books' => $books));
}

/**
 * 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 เราควรแสดงข้อความ 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

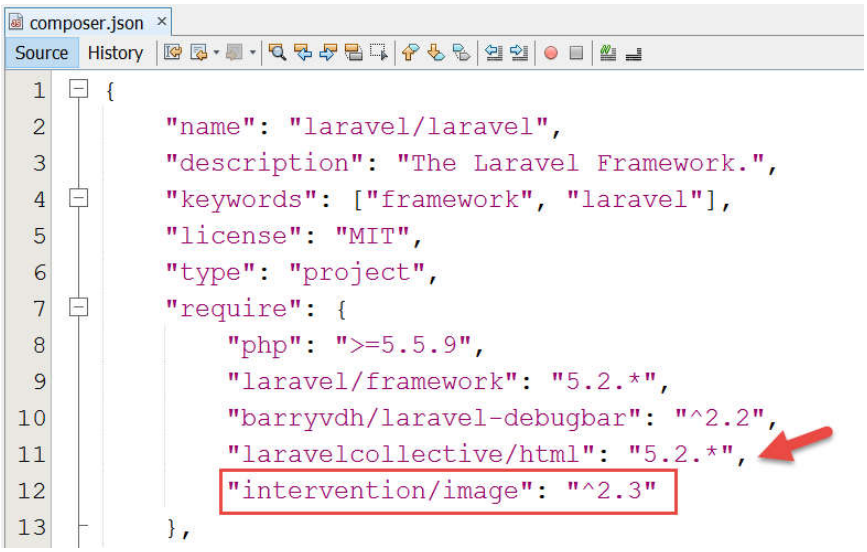


การติดตั้ง Image Library เพื่อเตรียมพร้อมก่อนอัพโหลดไฟล์

เมื่อมีการอัพโหลดไฟล์จากฟอร์มของผู้ใช้ บางครั้งรูปภาพที่ถูกอัพโหลดเข้ามาอาจมีขนาดใหญ่ หรือมีขนาดไม่พอดี ดังนั้นเราจะติดตั้ง Library สำหรับจัดการรูปภาพต่างๆ เช่น การย่อขนาดรูป เป็นต้น จากเว็บนี้ http://image.intervention.io/

ขั้นตอนการติดตั้ง Intervention Image Library

1. เปิดไฟล์ composer.json ขึ้นมาแล้วพิมพ์โค้ด ดังนี้



2. เข้าไปในโปรเจคของเรา เปิด Composer ขึ้นมาแล้วพิมพ์ composer update เพื่อติดตั้ง จากนั้นกด enter

```
C:\Windows\System32\cmd.exe - composer  update

C:\xampp\htdocs\laravel5>composer update
Loading composer repositories with package information
Updating dependencies (including require-dev)
```

Note: วิธีการติดตั้งเพิ่มเติม ดูได้จากที่นี่ http://image.intervention.io/getting_started/installation#laravel

3. เสร็จแล้วเปิดไฟล์ config\app.php เพิ่มโค้ดที่ Service Providers ดังนี้

Intervention\Image\ImageServiceProvider::class,

```
151         /*
152          * Application Service Providers...
153          */
154         App\Providers\AppServiceProvider::class,
155         App\Providers\AuthServiceProvider::class,
156         App\Providers\EventServiceProvider::class,
157         App\Providers\RouteServiceProvider::class,
158
159         Barryvdh\Debugbar\ServiceProvider::class,
160         Collective\Html\HtmlServiceProvider::class,
161         Intervention\Image\ImageServiceProvider::class,
162
163     ],
```

จากนั้นให้เพิ่มโค้ด ในส่วนของ Class Aliases ด้วย ดังนี้

'Image' => Intervention\Image\Facades\Image::class,

app.php

Source History

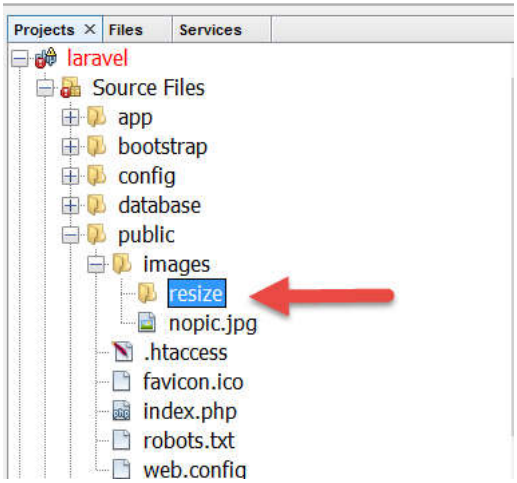
```
201         'Route' => Illuminate\Support\Facades\Route::class,
202         'Schema' => Illuminate\Support\Facades\Schema::class,
203         'Session' => Illuminate\Support\Facades\Session::class,
204         'Storage' => Illuminate\Support\Facades\Storage::class,
205         'URL' => Illuminate\Support\Facades\URL::class,
206         'Validator' => Illuminate\Support\Facades\Validator::class,
207         'View' => Illuminate\Support\Facades\View::class,
208
209         'Debugbar' => Barryvdh\Debugbar\Facade::class,
210         'Form' => Collective\Html\FormFacade::class,
211         'Html' => Collective\Html\HtmlFacade::class,
212         'Image' => Intervention\Image\Facades\Image::class,
213     ],
214
215 ];
```

เพียงเท่านี้เราก็สามารถจัดการรูปภาพต่างๆ ได้เรียบร้อยแล้ว

สร้างเพิ่มข้อมูลหนังสือ (books)

หลังจากติดตั้ง Library สำหรับจัดการรูปภาพเรียบร้อย ต่อไปให้เราเขียนโค้ดเพื่อเพิ่มข้อมูล และอัพโหลดรูปภาพ พร้อมทั้งย่อภาพด้วย การเขียนโค้ดสำหรับเพิ่มข้อมูล มีขั้นตอน ดังนี้

1. ให้สร้างโฟลเดอร์ resize เพื่อเก็บภาพที่ได้ทำการย่อไว้ในโฟลเดอร์ public\images ดังภาพ



2. เปิดไฟล์ BooksController.php ขึ้นมาแล้วเขียนโค้ดที่เมธอด store() เพื่อบันทึกข้อมูล ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Books;
use App\Http\Requests\StoreBooksRequest;
use Image; //เรียกใช้ library จัดการรูปภาพเข้ามาใช้งาน

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]); //books/index.blade.php
        //return View::make('books/index', array('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
```

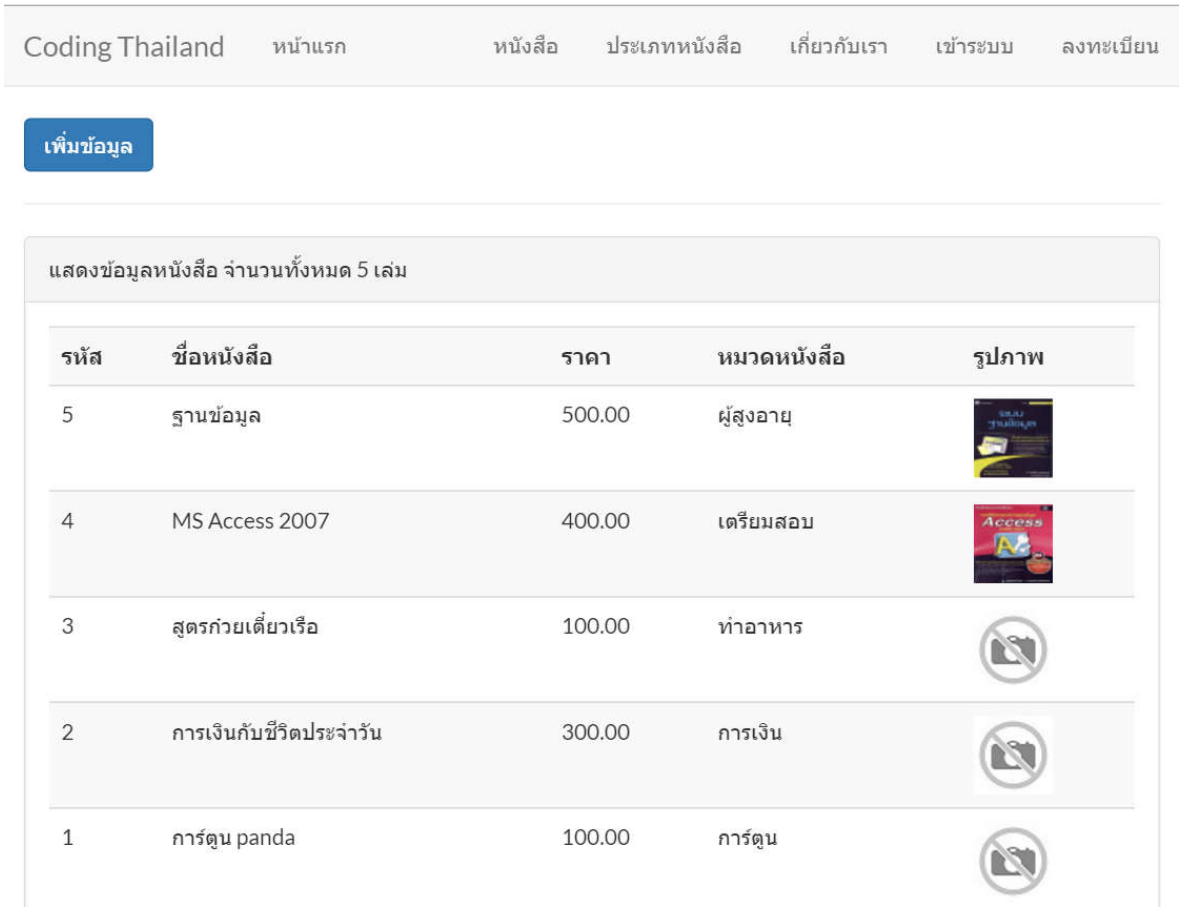
```
     */
    public function edit($id) {
        //
    }

    /**
     * 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 เมธอด store() หากการตรวจสอบข้อมูลถูกต้อง เราจะรับ request และค่าจากฟอร์มมาทั้งหมด โดยมีเราสามารถตรวจสอบได้ว่าผู้ใช้ได้เลือกอัพโหลดไฟล์มาได้หรือไม่ สามารถตรวจสอบได้โดยใช้ hasFile() หากอัพโหลดมาเราจะสุ่มชื่อไฟล์ใหม่เพื่อไม่ให้ชื่อซ้ำกัน พร้อมกับอัพโหลดไฟล์เก็บไว้ที่โฟลเดอร์ images หลังจากนั้นก็ย่อขนาดไฟล์ให้เหลือขนาด 50x50 แล้วเก็บไว้ที่โฟลเดอร์ images/resize หากผู้ใช้ไม่ได้อัพโหลดภาพเข้ามาก็ให้กำหนดชื่อว่าเป็น nopic.jpg แล้วก็สั่ง save() เพื่อบันทึกลงในตาราง

3. เปิดไฟล์ resources\views\books**index.blade.php** เพื่อแก้ไข path รูปภาพให้ถูกต้องในที่นี้เราเก็บรูปที่ย่อแล้วไว้ในโฟลเดอร์ images/resize แก้ไขใหม่เป็นดังนี้

```
<a href="{{ asset('images/'.$book->image) }}"><img src="{{
asset('images/resize/'.$book->image) }}"></a>
```

4. จากนั้นให้ copy รูปภาพ nopic.jpg ไปวางไว้ในโฟลเดอร์ /images/**resize/** ด้วยเพื่อการแสดงผลที่ถูกต้อง



สร้างฟอร์มแก้ไขข้อมูลหนังสือ (books)

การสร้างฟอร์มแก้ไขเราจะต้องสร้างลิงก์เพื่อให้ผู้ใช้คลิกแล้วเปิดฟอร์มแก้ไขขึ้นมา เปิดไฟล์ resources\views\books**index.blade.php** อีกครั้งเพื่อแทรกคอลัมน์ให้กับตาราง สำหรับการแก้ไขมีขั้นตอน ดังนี้

1. ให้เพิ่มคอลัมน์การแก้ไขข้อมูลให้กับตาราง

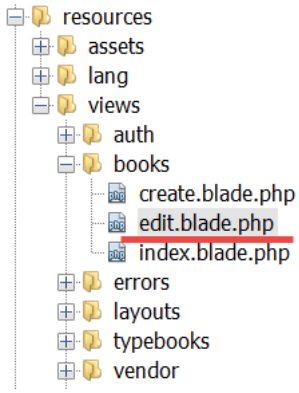
```
@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

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

            <hr>
```

```
            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{    $books->total()   }} เล่ม</div>
                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
                            <th>รหัส</th>
                            <th>ชื่อหนังสือ</th>
                            <th>ราคา</th>
                            <th>หมวดหนังสือ</th>
                            <th>รูปภาพ</th>
                            <th>แก้ไข</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) }}"><img src="{{ asset('images/resize/'.$book->image) }}"></a>
                            </td>
                            <td><a href="{{ url('/books/'.$book->id.'/edit') }}"><i class="fa fa-pencil"></i></a></td>
                        </tr>
                        @endforeach
                    </table>
                    <br>
                    {!!  $books->render()   !!}



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

2. เปิดไฟล์ BooksController.php ที่เมธอด edit($id) ให้เขียนโค้ดเพื่อแสดงเฉพาะแถวที่ส่งมาพร้อมทั้ง render view ด้วย ดังนี้

```
public function edit($id) {
        $book = Books::findOrFail($id);
        return view('books.edit', ['book' => $book]);
}
```

3. มาที่โฟลเดอร์ของ views ให้สร้างไฟล์ edit.blade.php ในโฟลเดอร์ books เพื่อรองรับการ render จาก Controller ดังนี้



4. ในการแก้ไขข้อมูลเราจะใช้วิธีที่เรียกว่า Model Binding หรือการผูกค่าโมเดลเข้ากับ input ต่างๆในฟอร์ม ดังนี้

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

            <div class="panel panel-default">
                <div class="panel-heading">แก้ไขข้อมูลหนังสือ {{ $book->title }}</div>
                <div class="panel-body">

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

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

                    <div class="col-xs-8">
```

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

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('price', 'ราคา');  !!}
                            {!! Form::text('price',null,['class' => 'form-
control','placeholder' => 'เช่น 100, 100.25']); !!}
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('typebooks_id', 'ประเภทหนังสือ');  !!}
                            <?= Form::select('typebooks_id',
App\TypeBooks::lists('name', 'id'), null, ['class' => 'form-control',
'placeholder' => 'กรุณาเลือกประเภทหนังสือ...']); ?>
                        </div>
                    </div>

                    <div class="form-group">
                        <div class="col-sm-10">
                            <?= Form::submit('บันทึก', ['class' => 'btn btn-
primary']); ?>
                        </div>
                    </div>

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


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

5. เปิดไฟล์ BooksController.php เพื่อเขียนโค้ดที่เมธอด update() เพื่อแก้ไขข้อมูล ดังนี้

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Books;
use App\Http\Requests\StoreBooksRequest;
use File;
use Image;

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]);
//books/index.blade.php
        //return View::make('books/index', array('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(StoreBooksRequest $request, $id) {
       $book = Books::find($id);
        /* $book->title = $request->title;
          $book->price = $request->price;
          $book->typebooks_id = $request->typebooks_id;
```

```
            $book->save(); */
            $book->update($request->all()); //mass asignment , define $fillable
    (model)

            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');
        }

    }
```

6. เพียงเท่านี้เราก็สามารถแก้ไขข้อมูลได้เรียบร้อย

การลบข้อมูลหนังสือ (books)

การลบข้อมูลเช่นเดียวกันให้เราเพิ่มคอลัมน์อีก 1 คอลัมน์ เปิดไฟล์ resources\views\books\index.blade.php อีกครั้งเพื่อแทรกคอลัมน์ให้กับตาราง สำหรับใช้ในการลบมีขั้นตอน ดังนี้

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

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

            <hr>
```

```
            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{   $books->total()   }}
เล่ม</div>
                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
                            <th>รหัส</th>
                            <th>ชื่อหนังสือ</th>
                            <th>ราคา</th>
                            <th>หมวดหนังสือ</th>
                            <th>รูปภาพ</th>
                            <th>แก้ไข</th>
                            <th>ลบ</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) }}"><img
src="{{ asset('images/resize/'.$book->image) }}"></a>
                            </td>
                            <td><a href="{{ url('/books/'.$book->id.'/edit') }}"><i
class="fa fa-pencil"></i></a></td>
                            <td>
                                <?= Form::open(array('url' => 'books/' . $book->id,
'method' => 'delete')) ?>
                                <button type="submit" class="btn btn-danger"><i
class="fa fa-trash"></i></button>
                                {!! Form::close() !!}
                            </td>
                        </tr>
                        @endforeach
                    </table>
                    <br>
                    {!!  $books->render()   !!}
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

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

จากนั้นให้เราเขียนโค้ดสำหรับการลบหนังสือได้ที่เมธอด destroy($id) ดังนี้

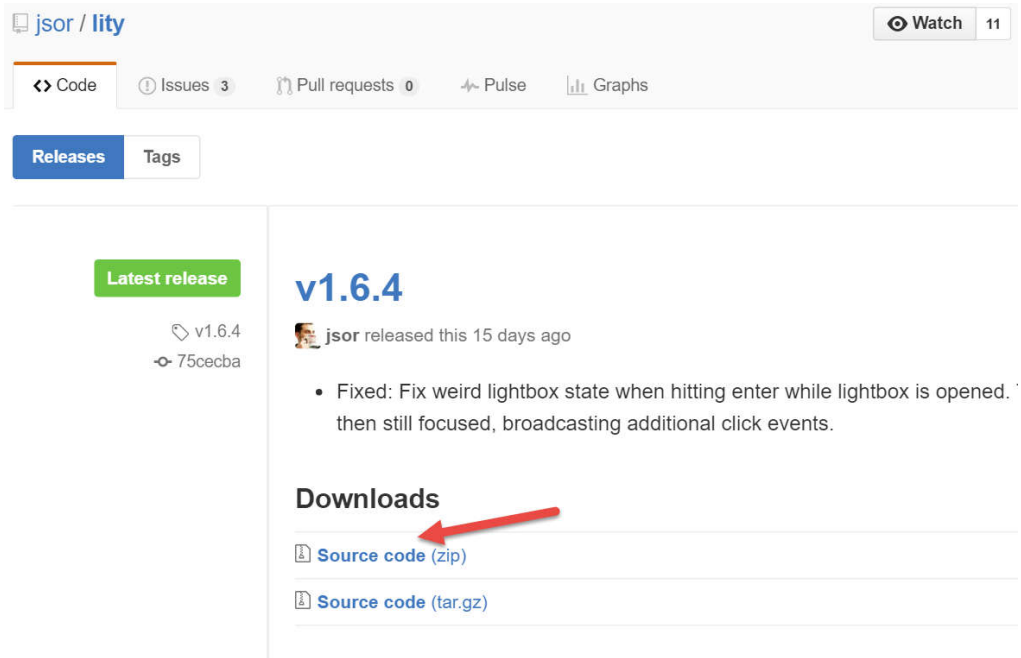
```
public function destroy($id) {
       $book = Books::find($id);
        if ($book->image != 'nopic.jpg') {
            File::delete(public_path() . '\\images\\' . $book->image);
            File::delete(public_path() . '\\images\\resize\\' . $book->image);
        }
        $book->delete();
        return redirect()->action('BooksController@index');
}
```

การลบข้อมูลที่ดีควรลบไฟล์ออกไปด้วย ในกรณีนี้เราเช็ค if ว่าถ้าชื่อไฟล์ไม่เท่ากับ nopic.jpg ก็ให้ลบไฟล์ได้เลย

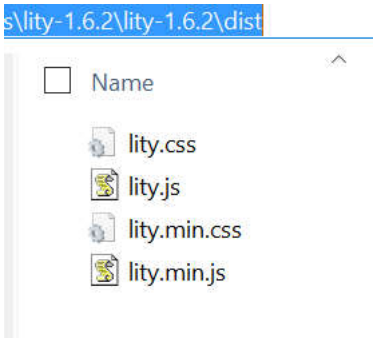
การทำ responsive lightbox โดยใช้ Lity Library

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

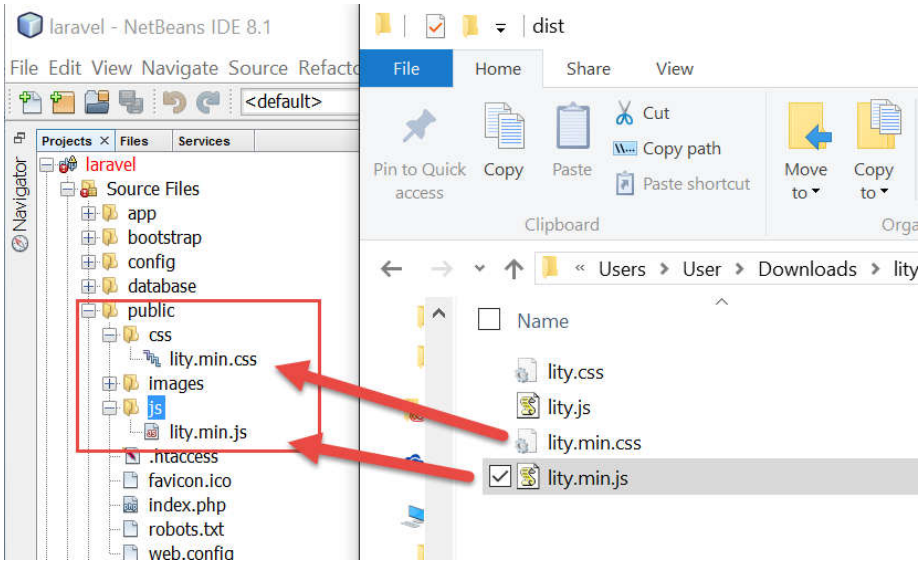
1. ดาวน์โหลด lity ได้ที่ลิงก์ https://github.com/jsor/lity/releases/latest คลิกดาวน์โหลดที่ Source code (zip)



2. ดาวน์โหลดเสร็จแล้วให้แตกไฟล์ (extract) zip ที่ได้มา ไฟล์ของ library จะอยู่ที่โฟลเดอร์ dist/



3. จากนั้นให้ copy ไฟล์ lity.min.css ไปวางไว้ที่ public/css และ copy ไฟล์ lity.min.js ไปวางไว้ที่ public/js (หากยังไม่ได้สร้างโฟลเดอร์ css และ js ใน public ให้สร้างได้เลยครับ) หรือใช้วิธี drag&drop เข้ามาในโปรแกรม Netbeans ก็ได้เช่นเดียวกัน



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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Laravel</title>

    <!-- Fonts -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.4.0/css/font-awesome.min.css" rel='stylesheet' type='text/css'>
    <link href="https://fonts.googleapis.com/css?family=Lato:100,300,400,700"
rel='stylesheet' type='text/css'>

    <!-- Styles -->
```

```
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
rel="stylesheet">
    {{-- <link href="{{ elixir('css/app.css') }}" rel="stylesheet"> --}}

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

    <style>
        body {
            font-family: 'Lato';
        }

        .fa-btn {
            margin-right: 6px;
        }
    </style>
</head>
<body id="app-layout">
    <nav class="navbar navbar-default navbar-static-top">
        <div class="container">
            <div class="navbar-header">

                <!-- Collapsed Hamburger -->
                <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#app-navbar-collapse">
                    <span class="sr-only">Toggle Navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>

                <!-- Branding Image -->
                <a class="navbar-brand" href="{{ url('/') }}">
                    Coding Thailand
                </a>
            </div>

            <div class="collapse navbar-collapse" id="app-navbar-collapse">
                <!-- Left Side Of Navbar -->
                <ul class="nav navbar-nav">
                    <li><a href="{{ url('/home') }}">หน้าแรก</a></li>
                </ul>

                <!-- Right Side Of Navbar -->
                <ul class="nav navbar-nav navbar-right">
                    <!-- Authentication Links -->
                    @if (Auth::guest())
                        <li><a href="{{ url('/books') }}">หนังสือ</a></li>
```

```
                        <li><a href="{{ url('/typebooks') }}">ประเภทหนังสือ</a></li>
                        <li><a href="{{ url('/about') }}">เกี่ยวกับเรา</a></li>
                        <li><a href="{{ url('/login') }}">เข้าระบบ</a></li>
                        <li><a href="{{ url('/register') }}">ลงทะเบียน</a></li>
                    @else
                        <li class="dropdown">
                            <a href="#" class="dropdown-toggle" data-
toggle="dropdown" role="button" aria-expanded="false">
                                {{ Auth::user()->name }} <span
class="caret"></span>
                            </a>

                            <ul class="dropdown-menu" role="menu">
                                <li><a href="{{ url('/logout') }}"><i class="fa
fa-btn fa-sign-out"></i>Logout</a></li>
                            </ul>
                        </li>
                    @endif
                </ul>
            </div>
        </div>
    </nav>

    @yield('content')

    <!-- JavaScripts -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.min.js"></scrip
t>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></scr
ipt>
    {{-- <script src="{{ elixir('js/app.js') }}"></script> --}}

    <script src="{{ url('js/lity.min.js') }}" type="text/javascript"></script>

    @yield('footer')

</body>
</html>
```

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

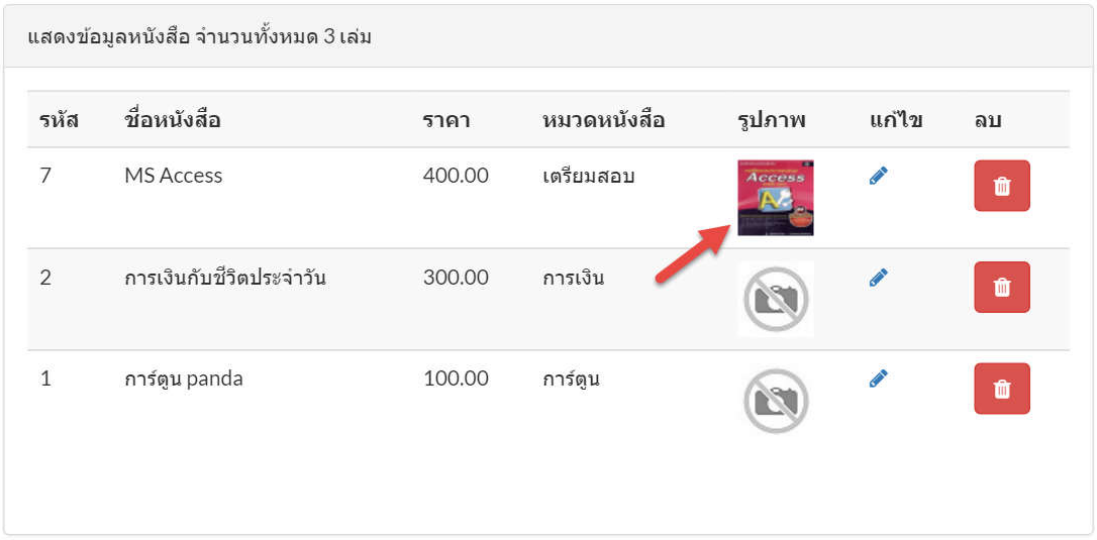
image) }}" data-lity><img src="{{ asset('images/resize/'.$book->image) }}">

โค้ดทั้งหมด ในไฟล์ resources\views\books\index.blade.php

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

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

            <hr>

            <div class="panel panel-default">
                <div class="panel-heading">แสดงข้อมูลหนังสือ จำนวนทั้งหมด {{   $books-
>total()   }} เล่ม</div>
                <div class="panel-body">

                    <table class="table table-striped">
                        <tr>
                            <th>รหัส</th>
                            <th>ชื่อหนังสือ</th>
                            <th>ราคา</th>
                            <th>หมวดหนังสือ</th>
                            <th>รูปภาพ</th>
                            <th>แก้ไข</th>
                            <th>ลบ</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><img src="{{ asset('images/resize/'.$book->image) }}"></a>
                            </td>
                            <td><a href="{{ url('/books/'.$book->id.'/edit')
}}"><i class="fa fa-pencil"></i></a></td>
                            <td>
                                <?= Form::open(array('url' => 'books/' . $book-
>id, 'method' => 'delete')) ?>
                                <button type="submit" class="btn btn-danger"><i
class="fa fa-trash"></i></button>
                                {!! Form::close() !!}
                            </td>
```

```
                        </tr>
                        @endforeach
                    </table>
                    <br>
                    {!!  $books->render()   !!}



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

6. ทดสอบโดยการคลิกที่ภาพเล็กในตาราง เมื่อคลิกแล้วรูปภาพจะขยายใหญ่ขึ้น



บทที่ 7 การใช้งาน Sessions และการกำหนดสิทธิ์ผู้ใช้

การใช้งาน Session

Session เป็นตัวแปรที่เราสามารถใช้งานข้ามหน้าเพจต่างๆได้ 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');

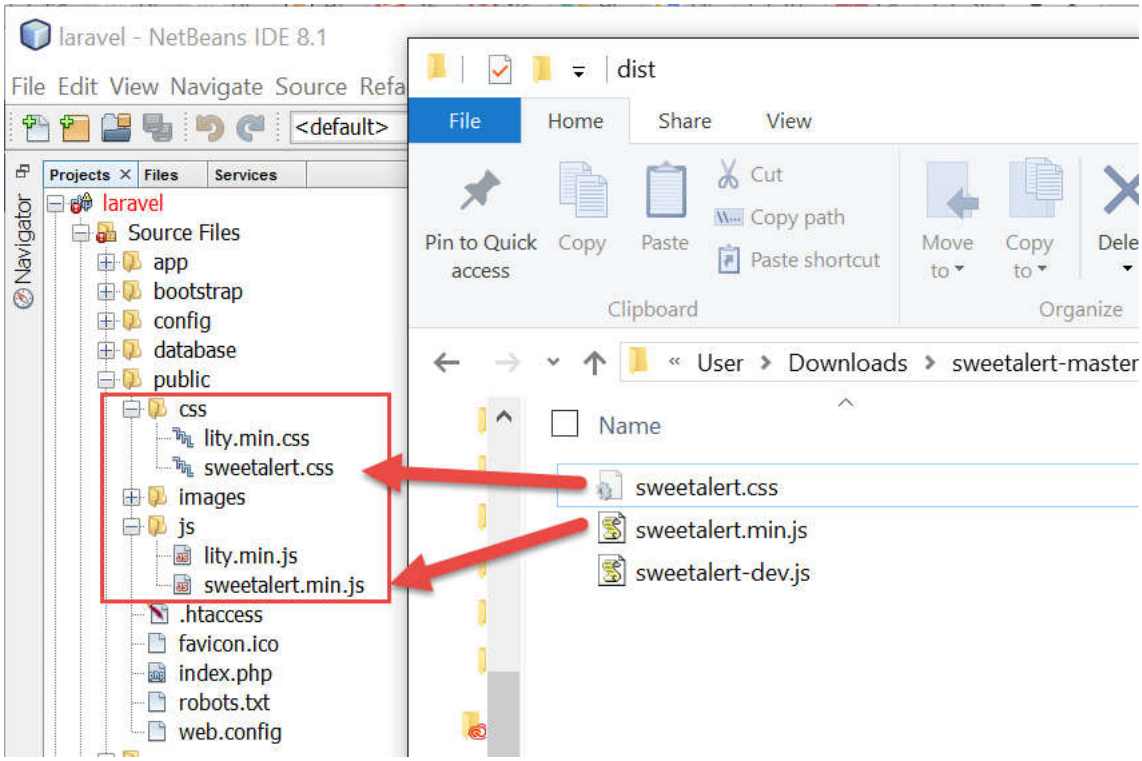
  $request->session()->flush();

การใช้งาน Flash Data

Flash Data เป็น session ที่มีอายุใช้งานชั่วคราว ใช้ได้ใน request หนึ่งๆ และจะหายไปเมื่อมี request ใหม่เกิดขึ้น เหมาะสำหรับการทำการโต้ตอบกับผู้ใช้ ณ ขณะนั้น เช่น โต้ตอบการเพิ่มข้อมูล หรือลบข้อมูลเรียบร้อยแล้ว เป็นต้น

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

1. เข้าไปดาวน์โหลด Sweet Alert ได้ที่ https://github.com/t4t5/sweetalert/archive/master.zip
2. การติดตั้งคล้ายๆ กับ lity library ในหัวข้อที่ผ่านมา คือ ให้เรา copy sweetalert.css และ sweetalert.min.js ไปวางไว้ในโฟลเดอร์ public/ หรือจะ drag&drop ก็ได้เช่นเดียวกัน ดังรูป



3. เปิดไฟล์ layouts ที่ resources\views\layouts\app.blade.php เพิ่มแทรกโค้ด css และ js ของ Sweet Alert ดังนี้

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Laravel</title>

    <!-- Fonts -->
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.4.0/css/font-awesome.min.css" rel='stylesheet' type='text/css'>
    <link href="https://fonts.googleapis.com/css?family=Lato:100,300,400,700"
rel='stylesheet' type='text/css'>

    <!-- Styles -->
    <link
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
rel="stylesheet">
    {{-- <link href="{{ elixir('css/app.css') }}" rel="stylesheet"> --}}

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

    <link href="{{  url('css/sweetalert.css')  }}" rel="stylesheet">

    <style>
```

```
        body {
            font-family: 'Lato';
        }

        .fa-btn {
            margin-right: 6px;
        }
    </style>
</head>
<body id="app-layout">
    <nav class="navbar navbar-default navbar-static-top">
        <div class="container">
            <div class="navbar-header">

                <!-- Collapsed Hamburger -->
                <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#app-navbar-collapse">
                    <span class="sr-only">Toggle Navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>

                <!-- Branding Image -->
                <a class="navbar-brand" href="{{ url('/') }}">
                    Coding Thailand
                </a>
            </div>

            <div class="collapse navbar-collapse" id="app-navbar-collapse">
                <!-- Left Side Of Navbar -->
                <ul class="nav navbar-nav">
                    <li><a href="{{ url('/home') }}">หน้าแรก</a></li>
                </ul>

                <!-- Right Side Of Navbar -->
                <ul class="nav navbar-nav navbar-right">
                    <!-- Authentication Links -->
                    @if (Auth::guest())
                        <li><a href="{{ url('/books') }}">หนังสือ</a></li>
                        <li><a href="{{ url('/typebooks') }}">ประเภทหนังสือ</a></li>
                        <li><a href="{{ url('/about') }}">เกี่ยวกับเรา</a></li>
                        <li><a href="{{ url('/login') }}">เข้าระบบ</a></li>
                        <li><a href="{{ url('/register') }}">ลงทะเบียน</a></li>
                    @else
                        <li class="dropdown">
```

```
                            <a href="#" class="dropdown-toggle" data-
toggle="dropdown" role="button" aria-expanded="false">
                                {{ Auth::user()->name }} <span
class="caret"></span>
                            </a>

                            <ul class="dropdown-menu" role="menu">
                                <li><a href="{{ url('/logout') }}"><i class="fa
fa-btn fa-sign-out"></i>Logout</a></li>
                            </ul>
                        </li>
                    @endif
                </ul>
            </div>
        </div>
    </nav>

    @yield('content')

    <!-- JavaScripts -->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.4/jquery.min.js"></scrip
t>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></scr
ipt>
    {{-- <script src="{{ elixir('js/app.js') }}"></script> --}}

    <script src="{{ url('js/lity.min.js') }}" type="text/javascript"></script>

    <script src="{{ url('js/sweetalert.min.js') }}"
type="text/javascript"></script>

    @yield('footer')

</body>
</html>
```

4. เปิดไฟล์ BooksController.php ให้เขียนโค้ดเพิ่มที่เมธอด 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();

        $request->session()->flash('status', 'บันทึกข้อมูลเรียบร้อยแล้ว');
       //กำหนด key ของ flash data ชื่อว่า status โดยใส่ค่าข้อมูลคำว่า บันทึกข้อมูลเรียบร้อยแล้ว

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

5. มาที่ส่วนของ views ให้เปิดไฟล์ resources\views\books\create.blade.php เพื่อเขียนโค้ด flash data สำหรับแสดงผล ดังนี้

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

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">

            <div class="panel panel-default">
                <div class="panel-heading">เพิ่มข้อมูลหนังสือ</div>
                <div class="panel-body">

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

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

                    <div class="col-xs-8">
                        <div class="form-group">
```

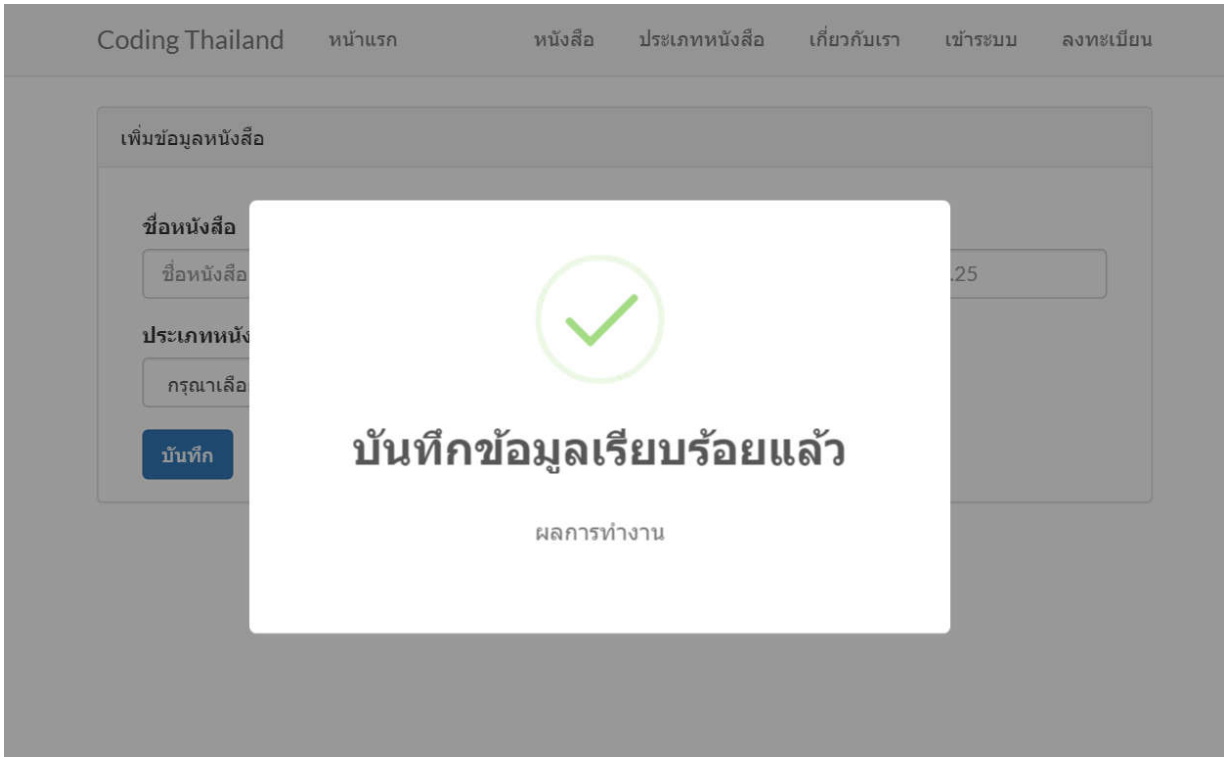
```
                            <?= Form::label('title', 'ชื่อหนังสือ'); ?>
                            <?= Form::text('title', null, ['class' => 'form-
control', 'placeholder' => 'ชื่อหนังสือ']); ?>
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('price', 'ราคา');  !!}
                            {!! Form::text('price',null,['class' => 'form-
control','placeholder' => 'เช่น 100, 100.25']); !!}
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('typebooks_id', 'ประเภทหนังสือ');  !!}
                            <?= Form::select('typebooks_id',
App\TypeBooks::lists('name', 'id'), null, ['class' => 'form-control',
'placeholder' => 'กรุณาเลือกประเภทหนังสือ...']); ?>
                        </div>
                    </div>

                    <div class="col-xs-4">
                        <div class="form-group">
                            {!! Form::label('image', 'รูปภาพ');  !!}
                            <?= Form::file('image', null, ['class' => 'form-
control']) ?>
                        </div>
                    </div>

                    <div class="form-group">
                        <div class="col-sm-10">
                            <?= Form::submit('บันทึก', ['class' => 'btn btn-
primary']); ?>
                        </div>
                    </div>

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

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

```
@section('footer')

@if (session()->has('status'))
<script>
    swal({
        title: "<?php echo session()->get('status'); ?>",
        text: "ผลการทำงาน",
        timer: 2000,
        type: 'success',
        showConfirmButton: false
    });
</script>
@endif

@endsection
```

ในการแสดงผลเราจะเช็ค if ก่อนเพื่อตรวจสอบว่ามี key ชื่อว่า status ที่สร้างไว้ BooksController.php หรือไม่ ถ้ามีจริงก็ให้แสดงค่าข้อมูลออกมา ผ่านเมธอด get() นั่นเอง

ส่วนของโค้ด JavaScript ของ Sweet Alert เรากำหนดเวลา (timer) ปิด popup หลังจากแสดง 2 วินาที

6. ทดสอบเพิ่มข้อมูลหนังสือใหม่ จะได้ผลลัพธ์การทำงานดังนี้



การกำหนดสิทธิ์ผู้ใช้

การกำหนดสิทธิ์ผู้ใช้ คือ เราสามารถอนุญาต หรือไม่อนุญาตให้เข้าถึงในส่วนต่างๆของระบบเรา สามารถเขียนกำหนดได้ที่ส่วนของ Controller

ตัวอย่าง การไม่อนุญาตให้ผู้ใช้ใช้งาน BooksController และการอนุญาตบางเมธอด

1. ลำดับแรกเราจะต้องย้ายโค้ด route ที่เราต้องการจำกัดสิทธิ์ มาวางไว้ด้านล่างในส่วนโค้ด Route::auth(); เปิดไฟล์ app\Http\routes.php แก้ไขโค้ดดังนี้

```
<?php

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

//สำหรับแสดงข้อมูลทั้งหมด
Route::get('typebooks','TypeBooksController@index');
//สำหรับลบข้อมูล ตาม id ที่รับมาจาก url (รับแบบ get)
Route::get('typebooks/destroy/{id}','TypeBooksController@destroy');

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

Route::auth();
Route::resource('books','BooksController'); //ย้ายมาวางไว้ด้านล่างของ auth()
Route::get('/home', 'HomeController@index');
```

2. ลำดับต่อมาเมื่อย้ายโค้ดแล้ว ให้เปิด BooksController.php เพื่อเขียน constructor สำหรับกำหนดสิทธิ์ 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']]);
}
```

จากโค้ดด้านบน ผู้ใช้จะไม่สามารถเข้าถึงเมธอดอื่นๆใน BooksController ได้ยกเว้นเมธอด index

โค้ดในหน้า BookController ทั้งหมด

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests;
use App\Books;
use App\Http\Requests\StoreBooksRequest;
use File;
use Image;

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
     * //$this->middleware('auth', ['except' => ['index', 'create', 'store']]);
     */
    public function index() {
        $books = Books::with('typebooks')->orderBy('id', 'desc')->paginate(5);
        return view('books/index', ['books' => $books]);
//books/index.blade.php
        //return View::make('books/index', array('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();

        $request->session()->flash('status', 'บันทึกข้อมูลเรียบร้อยแล้ว');

        return back();
        //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(StoreBooksRequest $request, $id) {
        $book = Books::find($id);
        /* $book->title = $request->title;
          $book->price = $request->price;
          $book->typebooks_id = $request->typebooks_id;
          $book->save(); */
        $book->update($request->all()); //mass asignment , define $fillable
(model)

        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');
    }

}
```

การทำ User Profiles

เนื่องจากเนื้อหาการทำ User Profiles ค่อนข้างเยอะซับซ้อน ผมจึงทำเป็นวิดีโอไว้ให้แล้ว สามารถเข้าไปดาวน์โหลดได้ที่

https://goo.gl/Tfa5zi

บทที่ 8 การสร้างรายงานในรูปแบบ PDF และ Charts

การสร้างรายงานรูปแบบ PDF

การสร้างรายงานรูปแบบ Charts

เนื่องจากเนื้อหาบทนี้ค่อนข้างเยอะและซับซ้อน ผมจึงทำเป็นวิดีโอไว้ให้แล้ว สามารถเข้าไปดาวน์โหลดได้ที่

https://goo.gl/EIlDpt

บทที่ 9 โบนัสพิเศษ

- การตั้งค่าและการส่งเมล ด้วย SMTP
- One Click Facebook Login
- การติดตั้ง Laravel 5 บน Server

เนื่องจากเนื้อหาบทนี้ค่อนข้างเยอะและซับซ้อน ผมจึงทำเป็นวิดีโอไว้ให้แล้ว สามารถเข้าไปดาวน์โหลดได้ที่

https://goo.gl/bj5Uqe

มาถึงตรงนี้ ก็ขอขอบคุณ คนที่รักการพัฒนาตัวเองทุกคนครับ

หวังว่าความรู้ในหนังสือเล่มนี้จะช่วยให้ชีวิตของทุกคนดีขึ้น

สามารถต่อยอดความรู้ เพื่อสร้างสิ่งดี ๆให้กับตัวเอง ครอบครัว และโลกนี้ต่อไป

ขอบคุณครับ

โค้ชเอก

Codingthailand.com