ปฏิบัติการที่ 1	Basic Routing
วัตถุประสงค์	เพื่อให้เข้าใจการทำงานของ Route และความสัมพันธ์ระหว่าง Route และ Controller

1. หลังจากติดตั้ง Laravel 5 ให้เริ่มใช้งาน Development Server ผ่าน CLI ที่ directory ของ project ด้วยคำสั่ง

```
php artisan serve
```

2. เปิดไฟล์ /app/Http/routes.php เพิ่มคำสั่งต่อไปนี้

```
Route::get('articles', function(){
      return "This is article page";
});
```

ทดสอบเข้า URL http://localhost:8000/articles ผ่าน Web Browser

3. เพิ่มคำสั่ง Route::get() โดยเปลี่ยน parameter เป็น categories และ return ข้อความที่แตกต่างออกไป และทดสอบด้วย URL http://localhost:8000/categories

4. สร้าง ArticlesController โดยใช้ CLI ด้วยคำสั่ง

```
php artisan make:controller ArticlesController --plain
```

แล้วเพิ่ม function index() ใน ArticlesController (/app/Http/Controllers/ArticlesController.php)

```
public function index(){
      return "This is article page from controller";
}
```

5. เปลี่ยนคำสั่ง Route::get() ในข้อ 2 เป็น

```
Route::get('articles', 'ArticlesController@index');
```

ทดสอบเข้า URL http://localhost:8000/articles ผ่าน Web Browser

6. ในทำนองเดียวกัน สร้าง CategoriesController และแก้ไข Route::get() สำหรับ categories

7. เพิ่ม function show() ใน ArticlesController

```
public function show($id){
      return "This is article page for ID " . $id;
}
```

8. เพิ่มคำสั่งใน /app/routes.php ดังนี้

```
Route::get('articles/{id}', 'ArticlesController@show');
```

ทดสอบเข้า URL http://localhost:8000/articles

http://localhost:8000/articles/1
http://localhost:8000/articles/news

9. เปลี่ยนคำสั่ง Route::get() ในข้อ 8 เป็น

```
Route::get('articles/{id?}', 'ArticlesController@show');
```

ทดสอบเข้า URL http://localhost:8000/articles

http://localhost:8000/articles/1
http://localhost:8000/articles/news

10. สร้าง PagesController (ดูวิธีการได้ในข้อ 4) แล้วเพิ่ม function index() และ function show()

```
public function index(){
      return "This is PagesController@index";
}

public function show(){
      return "This is PagesController@show";
}
```

เพิ่มคำสั่งใน /app/routes.php ดังนี้

```
Route::controller('pages', 'PagesController');
```

11. ทดสอบเข้า URL http://localhost:8000/pages และ http://localhost:8000/pages/show

จะพบ Error

12. เปลี่ยนชื่อ function index() เป็น getIndex() และ function show() เป็น getShow()

```
public function getIndex(){
      return "This is PagesController@index";
}

public function getShow(){
      return "This is PagesController@show";
}
```

ทดสอบเข้า URL http://localhost:8000/pages และ http://localhost:8000/pages/show อีกครั้ง

13. สร้างไฟล์ /resources/views/pages/about.php และพิมพ์ HTML Code ต่อไปนี้

```
<!Doctype html>
<html>
<head>
</head>
<body>
   <h1>About Me</h1>
   <p>The short description about me</p>
</body>
</html>
```

14. เพิ่ม function getAbout() ใน PagesController (/app/Http/Controllers/pagesController.php)

```
public function getAbout(){
      return view('pages.about');
}
```

ทดสอบเข้า URL http://localhost:8000/pages/about

15. ใช้วิธีการเดียวกันในข้อ 13 และ 14 เพิ่ม function getContact() เพื่อสร้างหน้า Contact Us

ปฏิบัติการที่ 2	Passing Data to View
วัตถุประสงค์	เพื่อให้เข้าใจ Controller และวิธีการส่งข้อมูล

1. การส่งค่าด้วย method with()

   แก้ไข function getAbout() ใน PagesController (/app/Http/Controllers/PagesController.php)

```
public function getAbout(){
      $skills = ['HTML5', 'jQuery', 'Bootstrap', 'Laravel5'];
      return view('pages.about')->with(
         ['first' => 'Adam', 'last' => 'Smith', 'skills' => $skills]
      );
}
```

2. แก้ไข view pages/about (/resources/views/pages/about.php)

```
<!Doctype html>
<html>
<head>
      <title>Page: About Me</title>
</head>
<body>
   <h1>About Me: <?= $first ?> <?= $last ?></h1>
   <p>My Skills</p>
   <ul>
      <?php foreach($skills as $skill): ?>
            <li><?= $skill ?></li>
       <?php endforeach; ?>
   </ul>
</body>
</html>
```

   ทดสอบเข้า URL http://localhost:8000/pages/about

3. การส่งค่าด้วย parameter ที่ 2 ของ function view()

   แก้ไข function getAbout() ใน PagesController (/app/Http/Controllers/PagesController.php)

```
public function getAbout(){
   $first = 'Alice';
   $last = 'Abernathy';
   $skills = ['HTML5', 'CSS3', 'Laravel5'];
   return view('pages.about',
      ['first' => $first, 'last' => $last, 'skills' => $skills]
   );
}
```

   ทดสอบเข้า URL http://localhost:8000/pages/about

4. การส่งค่าด้วย method with() แบบพิเศษ

   แก้ไข function getAbout() ใน PagesController (/app/Http/Controllers/PagesController.php)

```
public function getAbout(){
   $first = 'Alexia';
   $last = 'Ashford';
   $skills = ['HTML5', 'CSS3', 'Laravel5', 'CakePHP'];
   return view('pages.about')->withFirst($first)
                             ->withLast($last)
                             ->withSkills($skills);
}
```

   ทดสอบเข้า URL http://localhost:8000/pages/about

5. ส่งค่าจาก function getContact() เพื่อไปแสดงผลใน view ของ Contact Us

ปฏิบัติการที่ 3	Blade Template
วัตถุประสงค์	เพื่อให้เข้าใจ Blade Template และการสร้าง Layout

1. แก้ไขชื่อไฟล์ view pages/about จาก about.php เป็น about.blade.php (ใน /resources/views/pages/)

```
<!Doctype html>
<html>
<head>
      <title>Page: About Me</title>
</head>
<body>
   <h1>About Me: {{ $first }} {{ $last }}</h1>
   <p>My Skills</p>
   <ul>
      @foreach($skills as $skill)
            <li>{{ $skill }}</li>
      @endforeach
   </ul>
</body>
</html>
```

ทดสอบเข้า URL http://localhost:8000/pages/about

2. สร้าง layout main ไว้ที่ /resources/views/layouts/main.blade.php

```
<!Doctype html>
<html>
<head>
      <title>Page: @yield('site_title')</title>
</head>
<body>
      <div class="container">
            @yield('content')
      </div>

      <div class="footer">
            Copyright &copy; 2015
      </div>
</body>
</html>
```

3. แก้ไข view pages/about (/resources/views/pages/about.blade.php) ให้เรียกใช้งาน layout main

```
@extends('layouts.main')
@section('site_title', 'About Me')
@section('content')
   <h1>About Me: {{ $first }} {{ $last }}</h1>
   <p>My Skills</p>
   <ul>
      @foreach($skills as $skill)
            <li>{{ $skill }}</li>
      @endforeach
   </ul>
@stop
```

ทดสอบเข้า URL http://localhost:8000/pages/about

4. การสร้าง Link ภายใน

ใช้ function action() เพื่อสร้าง URL ภายใน ตัวอย่างเช่น

```
   <a href="{{ action('PagesController@getContact') }}">
      Contact
   </a>
```

ผลลัพธ์ที่ได้คือ Contact

หากต้องการส่ง Parameter ของ URL ด้วย ให้ส่งเป็น array ใน Parameter ที่สองของ action() เช่น

```
   <a href="{{ action('ArticlesController@show', [1]) }}">
      Article 1
   </a>
```

ผลลัพธ์ที่ได้คือ Article 1

ทั้งนี้ ผลลัพธ์จะขึ้นอยู่กับการตั้งค่า Route ด้วย

นอกจากนี้สามารถใช้ function url() เพื่อสร้าง URL ภายใน ตัวอย่างเช่น

```
    <a href="{{ url('pages/contact') }}">Contact</a>
```

ผลลัพธ์ที่ได้คือ Contact

หรือส่ง Parameter ของ URL โดยส่งเป็น array ใน Parameter ที่สองของ action() เช่น

```
<a href="{{ url('articles', [1, 'edit']) }}">
   Edit Article 1
</a>
```

ผลลัพธ์ที่ได้คือ Edit Article 1

5. การแสดงรูปภาพบนเว็บไซต์

   นำไฟล์ภาพไปไว้ที่ /public/images/

   เมื่อต้องการแสดงภาพด้วย <img> tag ให้ประยุกต์ใช้ function url() ในข้อ 4 ตัวอย่างเช่น

```
<img src="{{ url('images/image_file_name.png') }}" alt="" >
```

6. การใช้ HTML Builder เพื่อนำเข้า CSS และ JavaScript อาจประยุกต์ใช้ function url() ใน Link Tag หรือ Script Tag ก็ได้ แต่ Laravel มี HTML Builder เป็นตัวช่วยสร้าง HTML Tags

   a. ดาวน์โหลด Bootstrap จาก http://getbootstrap.com/getting-started/

   b. Extract File ที่ดาวน์โหลดในข้อ a จะได้โฟลเดอร์ bootstrap/ ให้นำโฟลเดอร์ดังกล่าวไปไว้ที่ /public/ ของ Laravel

   c. ดาวน์โหลด jQuery 2.x จาก http://jquery.com/download/ แล้วนำไฟล์ jquery.min.js ไปไว้ที่ /public/js/ ของ Laravel

   d. ในไฟล์ /resources/views/layouts/main.blade.php ใช้คำสั่งต่อไปนี้เพื่อนำเข้าไฟล์ CSS และ Javascript

```
{!! HTML::style('bootstrap/css/bootstrap.min.css') !!}
{!! HTML::script('js/jquery.min.js') !!}
{!! HTML::script('bootstrap/js/bootstrap.min.js') !!}
```

   e. เมื่อเรียกหน้าเว็บผ่าน Browser จะแสดง Error ว่า ไม่พบ Illuminate\Html\ServiceProvider ให้เพิ่มบรรทัดต่อไปนี้ ในไฟล์ /config/app.php ที่ key 'providers'

```
'Illuminate\Html\HtmlServiceProvider'
```

f. ในไฟล์ /config/app.php ที่ key 'aliases' เพิ่ม Form และ HTML ดังนี้

```
'Form' => 'Illuminate\Html\FormFacade',
'HTML' => 'Illuminate\Html\HtmlFacade',
```

g. ในไฟล์ /composer.json ที่ key "require" เพิ่ม

```
"illuminate/html": "5.*"
```

h. ดาวน์โหลด package illuminate/html ผ่าน CLI ด้วยคำสั่ง

```
composer update illuminate/html –vvv --profile
```

7. นอกจาก HTML::style() สำหรับนำเข้า CSS และ HTML::script() สำหรับนำเข้า JavaScript แล้ว ยังมีคำสั่ง HTML Builder สำหรับสร้าง <a> Tag และ <img> Tag คือ HTML::link() และ HTML::image() ตามลำดับ

```
{!! HTML::link("pages/contact", "Contact", ["class" => "btn"]) !!}
{!! HTML::image("images/image.png", "alt", ["class" => "img"]) !!}
```

ปฏิบัติการที่ 4	Bootstrap
วัตถุประสงค์	เพื่อให้เข้าใจ Bootstrap และการนำมาใช้งาน

1. ในไฟล์ /resources/views/layouts/main.blade.php

```
<html>
<head>
    <meta charset="utf-8"/>
    <title>Breaking News - @yield('site_title')</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    {!! HTML::style('bootstrap/css/bootstrap.min.css') !!}
</head>
<body>
...
    {!! HTML::script('js/jquery.min.js') !!}
    {!! HTML::script('bootstrap/js/bootstrap.min.js') !!}
</body>
</html>
```

2. Navigation Bar – ไฟล์ /resources/views/layouts/main.blade.php

```
...
<body>
<div class="navbar navbar-default" role="navigation">
    <div class="container-fluid">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle"
                  data-toggle="collapse" data-target=".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>
            <a class="navbar-brand" href="{{ url('article') }}">Home</a>
        </div>
```

(มีต่อหน้าถัดไป)

```
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li class="active">
                    <a href="{{ url('pages/about') }}">About</a>
                </li>
                <li>
                    <a href="{{ url('pages/contact') }}">Contact</a>
                </li>
            </ul>

            <ul class="nav navbar-nav navbar-right>
                <li>
                    <a href="{{ url('articles/create') }}">New Article</a>
                </li>
            </ul>
        </div><!--/.nav-collapse -->
    </div>
</div>
...
</body>
...
```

3. Page Content Container – ไฟล์ /resources/views/layouts/main.blade.php

```
...
<div class="navbar navbar-default" role="navigation">...</div>
<div class="container">
      @yield('content')
</div>
<div class="footer container-fluid">
      Copyright &copy; 2015
</div>
...
```

4. Page Heading – ไฟล์ /resources/views/pages/about.blade.php

```
@extends('layouts.main')
@section('site_title', 'About Me')
@section('content')
   <div class="page-header">
      <h1>About Me: {{ $first }} {{ $last }}</h1>
   </div>
   ...
@stop
```

5. Blockqoute – ไฟล์ /resources/views/pages/about.blade.php

```
...
    <h4>Favorite Quote</h4>
    <blockquote>
        <p>Life is like riding a bicycle. To keep your balance, you must
keep moving.</p>
        <footer>Albert Einstein</footer>
    </blockquote>
...
```

6. Panel and List Group – ไฟล์ /resources/views/pages/about.blade.php

```
...
   <div class="panel panel-info">
      <div class="panel-heading">
         <h3 class="panel-title">My Skills</h3>
      </div>
      <div class="panel-body">
         <ul class="list-group">
            @foreach($skills as $skill)
               <li class="list-group-item">{{ $skill }}</li>
            @endforeach
         </ul>
      </div>
   </div><!--/.panel-info -->
...
```

7. Grid System and Glyphicon

```
...
   <div class="row">
        <div class="col-md-3 col-sm-4 col-xs-6 well well-lg">
            <i class="glyphicon glyphicon-film"></i>
            Harry Potter
        </div>
        <div class="col-md-3 col-sm-4 col-xs-6 well well-lg">
            <i class="glyphicon glyphicon-music"></i>
            Love Me Like You Do
        </div>
        <div class="col-md-3 col-sm-4 col-xs-6 well well-lg">
            <i class="glyphicon glyphicon-heart"></i>
            Laravel 5
        </div>
        <div class="col-md-3 col-sm-4 col-xs-6 well well-lg">
            <i class="glyphicon glyphicon-plane"></i>
            Paris, France
        </div>
    </div>
...
```

ปฏิบัติการที่ 5	Migrations and Tinker
วัตถุประสงค์	เพื่อให้เข้าใจ Migration, Model, CRUD และการใช้งาน Model ผ่าน Tinker

1. สร้างฐานข้อมูลชื่อ breaking_news

2. กำหนดค่า Database Connection ที่ /config/database.php และกำหนดค่าใน /.env

   ในการอบรมให้ใช้ข้อมูลเกี่ยวกับฐานข้อมูลในไฟล์ /.env ดังนี้

```
DB_HOST=localhost
DB_DATABASE=breaking_news
DB_USERNAME=root
DB_PASSWORD=
```

3. สร้าง Model Article ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:model Article
```

   จะได้ไฟล์ /app/Article.php และไฟล์ /database/migrations/[วันเวลาที่สร้าง]_create_articles_table.php

4. หากไม่พบไฟล์ Migration ของตาราง articles ใน /database/migrations/ ให้สร้างไฟล์ Migration ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:migration create_articles_table --create=articles
```

   จะได้ไฟล์ /database/migrations/[วันเวลาที่สร้าง]_create_articles_table.php

5. แก้ไข function up() ในไฟล์ Migration create_articles_table

```
public function up(){
      Schema::create('articles', function(Blueprint $table){
            $table->increments('id');
            $table->text('title');
            $table->text('body');
            $table->timestamps();
            $table->timestamp('published_at');
      });
}
```

6. Migrate ตาราง articles จากที่กำหนดไว้ในข้อ 5 ผ่าน CLI ด้วยคำสั่ง

```
php artisan migrate
```

   เมื่อตรวจสอบในฐานข้อมูล จะพบว่ามีตาราง articles ที่มี field id, title, body, created_at, updated_at และ published_at ตามที่กำหนดไว้ในไฟล์ Migration

7. เพื่อความเข้าใจใน Eloquent ORM จะใช้ tinker ผ่าน CLI ด้วยคำสั่ง

```
php artisan tinker
```

เมื่อเข้าสู่ Psy Shell (PHP cli) จะสามารถพิมพ์คำสั่ง PHP และแสดงผลลัพธ์ทันที

8. การสร้าง (Create) Article

วิธีที่ 1 กำหนดค่าต่าง ๆ ผ่าน properties ของ object Article

```
>>> $article = new App\Article;
>>> $article->title = 'Article Title';
>>> $article->body = 'Article Body';
>>> $article->published_at = Carbon\Carbon::now();
>>> $article->save();
```

วิธีที่ 2 สร้างโดยวิธี Mass Assignment ผ่าน method create()

```
>>> App\Article::create([
...   'title' => 'Article Title 2',
...   'body' => 'Article Body 2',
...   'published_at' => Carbon\Carbon::now()
... ]);
```

เนื่องจาก Laravel มีการป้องกันการเขียนข้อมูลจาก User Input ด้วยวิธี Mass Assignment ทำให้ต้องกำหนดว่าจะอนุญาต field ใดบ้างที่จะใช้วิธี Mass Assignment ได้

ในไฟล์ Model /app/Article.php ให้เพิ่ม attribute $fillable

```
protected $fillable = ['title', 'body', 'published_at'];
```

จากนั้นทำตามวิธีที่ 2 อีกครั้ง และสร้าง Article เพิ่มมาอีกจำนวนหนึ่งด้วยวิธีที่ 1 หรือวิธีที่ 2 เพื่อให้มีข้อมูลสำหรับการเรียกดู การแก้ไข และการลบ

9. การเรียกดู (Retrieve) Article

การเรียกดู Article ทั้งหมด อาจใช้ method all() หรือ get()

```
>>> $articles = App\Article::all();
```

```
>>> $articles = App\Article::get();
```

สังเกตว่า ค่าที่ return จาก method all() หรือ get() จะเป็น array

การเรียกดู Article แบบเจาะจง id อาจใช้ method find($id) หรือ findOrFail($id)

```
>>> $article = App\Article::find(1);
```

หากไม่พบ Article ที่ id = 1 จะ return null แต่หากพบจะ return object ของ Article

```
>>> $article = App\Article::findOrFail(1);
```

หากไม่พบ Article ที่ id = 1 จะ throw ModelNotFoundException แต่หากพบจะ return object ของ Article

การเรียกดู Article แบบมีเงื่อนไข ให้ระบุเงื่อนไขใน method where()

```
>>> $articles = App\Article::where('title', '=', 'Title')->get();
```

10. การแก้ไข (Update) Article

ก่อนแก้ไข ต้องระบุว่าจะแก้ไขที่ record ใด แล้วแก้ไขค่าที่ attribute ของ object หลังจากนั้นใช้ method save()

```
>>> $article = App\Article::find(1);
>>> $article->title = "New Title 1";
>>> $article->body = "New Body 1";
>>> $article->save();
```

หรือแก้ไขโดยใช้วิธี Mass Assignment ด้วย method update()

```
>>> App\Article::find(1)->update([
...     'title' => 'Updated Article 1',
...     'body' => 'Updated body 1'
... ]);
```

11. การลบ (Delete) Article

ก่อนลบ ต้องระบุว่าจะลบที่ record ใด หลังจากนั้นใช้ method delete()

```
>>> $article = App\Article::find(1);
>>> $article->delete();
```

ปฏิบัติการที่ 6	MVC on Article
วัตถุประสงค์	เพื่อให้เข้าใจการสร้างเว็บแอปพลิเคชัน และความสัมพันธ์ระหว่าง Model-View-Controller

1. สร้าง Route สำหรับ articles ในไฟล์ /app/Http/routes.php

```
Route::resource('articles', 'ArticlesController');
```

2. สร้าง ArticlesController ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:controller ArticlesController
```

จะได้ไฟล์ /app/Http/Controllers/ArticlesController.php จากนั้นแก้ไข function index() และ show()

```
public function index(){
    $articles = Article::all();
    return view('articles.index', compact('articles'));
}

public function show($id){
    $article = Article::find($id);
    if(empty($article))
        abort(404);
    return view('articles.show', compact('article'));
}
```

(ต้องระบุ use App\Article; ใน ArticlesController ด้วย)

3. สร้าง view articles/index (/resources/views/articles/index.blade.php)

```
@extends('layouts.main')
@section('content')
    <div class="page-header" id="banner">
        <div class="row">
            <div class="col-xs-12">
                <h1>Articles</h1>
                <p class="lead">All articles</p>
            </div>
        </div>
    </div>
    @foreach($articles as $article)
        <div class="panel panel-default">
            <div class="panel-heading">
                <a href="{{ url('articles', $article->id) }}">
                    {{ $article->title }}
                </a>
            </div>
            <div class="panel-body">{{ $article->body }}</div>
            <div class="panel-footer">
                {{ $article->published_at }}
            </div>
        </div>
    @endforeach
@stop
```

4. สร้าง view articles/show (/resources/views/articles/show.blade.php)

```
@extends('layouts.main')
@section('content')
    <div class="page-header" id="banner">
        <h1>{{ $article->title }}</h1>
    </div>
    <div class="panel panel-default">
        <div class="panel-body">{{ $article->body }}</div>
    </div>
@stop
```

ทดสอบเข้า URL http://localhost:8000/articles

5. แก้ไข function create() และ store() ในไฟล์ /app/Http/Controllers/ArticlesController.php

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

public function store(){
    $input = Request::all();
    Article::create($input);
    Return redirect('articles');
}
```

(ต้องระบุ use Request; ใน ArticlesController และลบ use Illuminate\Http\Request;)

6. สร้าง view articles/create (/resources/views/articles/create.blade.php)

```
@extends('layouts.main')
@section('content')
  <h1 class="page-title">Write a New Article</h1>
  {!! Form::open(['url' => 'articles']) !!}
    <div class="form-group">
      {!! Form::label('title', 'Article Title: ') !!}
      {!! Form::text('title', null, ['class'=>'form-control']) !!}
    <div>
    <div class="form-group">
      {!! Form::label('body', 'Body:') !!}
      {!! Form::textarea('body',null,['class'=>'form-control']) !!}
    </div>
    <div class="form-group">
      {!! Form::submit('Add Article',
                  ['class' => 'btn btn-primary form-control']) !!}
    </div>
  {!! Form::close() !!}
@stop
```

ทดสอบสร้าง Article ที่ URL http://localhost:8000/articles/create

ปฏิบัติการที่ 7	Date, Mutators, Scopes
วัตถุประสงค์	เพื่อให้เข้าใจการจัดการวันเวลา การจัดการข้อมูลก่อนบันทึก และการกำหนดเงื่อนไขในการเรียกดูข้อมูล

1. ในไฟล์ /app/Article.php เพิ่ม attribute $dates เพื่อให้ field published_at เป็น Carbon object

```
protected $dates = ['published_at'];
```

ในไฟล์ /resources/views/articles/index.blade.php เปลี่ยนการแสดงค่า published_at โดยเรียก method diffForHumans() ซึ่งเป็น method ของ Carbon object

```
...
            <div class="panel-footer">
                {{ $article->published_at->diffForHumans() }}
            </div>
...
```

ทดสอบเข้า URL http://localhost:8000/articles สังเกตการเปลี่ยนแปลงของค่า published_at

2. ในไฟล์ view articles/create (/resources/views/articles/create.blade.php) เพิ่ม Form input ของ published_at

```
<div class="form-group">
   {!! Form::label('published_at', 'Publish on: ') !!}
   {!! Form::input('date', 'published_at',
                \Carbon\Carbon::now()->format('Y-m-d'),
                ['class'=>'form-control']) !!}
</div>
```

ทดสอบสร้าง Article ที่ URL http://localhost:8000/articles/create

3. ในไฟล์ /app/Article.php เพิ่ม function setPublishedAtAttribute($date)

```
public function setPublishedAtAttribute($date){
    $this->attributes['published_at'] =
                                Carbon::parse($date)->subDay();
}
```

เพื่อเปลี่ยนค่าที่ส่งจาก Form ซึ่งมี format เป็น date ให้เป็น format datetime และลดวันลง 1 วันก่อนบันทึก

(ต้องระบุ use Carbon\Carbon; ใน Model Article ด้วย)

4. สร้าง Article มาจำนวนหนึ่ง โดยให้ published_at เป็นวันที่ในอดีต และในอนาคต

   โดย Article ที่มี published_at เป็นวันที่ในอนาคตก็ยังไม่ควรนำมาแสดง

   สามารถกำหนดเป็น Scope ของการเรียกดูได้ ในไฟล์ /app/Article.php เพิ่ม function scopePublished() และ scopeUnpublished()

```
public function scopePublished($query){
    $query->where('published_at', '<=', Carbon::now());
}
public function scopeUnpublished($query){
    $query->where('published_at', '>', Carbon::now());
}
```

   และนำไปใช้งานใน ArticlesController@index (ไฟล์ /app/Http/Controllers/ArticlesController.php)

```
public function index(){
    $articles = Article::latest('published_at')->published()->get();
    return view('articles.index', compact('articles'));
}
```

   ทดสอบเข้า URL http://localhost:8000/articles

ปฏิบัติการที่ 8	Form Request Validation
วัตถุประสงค์	เพื่อให้เข้าใจ Form Request Validation และวิธีการตรวจสอบความถูกต้องของข้อมูลก่อนบันทึก

1. สร้าง ArticleRequest ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:request ArticleRequest
```

จะได้ไฟล์ /app/Http/Requests/ArticleRequest.php แก้ function authorize() ให้ return true

```
public function authorize(){
    return true;
}
```

แก้ไข function rules() ซึ่งเป็น function สำหรับกำหนดกฎของการส่งค่าจาก Form input

```
public function rules(){
    return [
        'title' => 'required|min:5|max:255',
        'body' => 'required',
        'published_at' => 'required|date'
    ];
}
```

2. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function store() ให้เรียกใช้ Form Request Validation ที่สร้างขึ้นในข้อ 1. โดยกำหนดเป็น Argument ของ function และเปลี่ยน Request::all() เป็น $request->all()

```
public function store(ArticleRequest $request){
    Article::create($request->all());
    return redirect('articles');
}
```

3. การแสดง Error จาก Validation

เมื่อผู้ใช้งานส่งค่าผ่าน Form input ไม่ถูกต้อง จะมีตัวแปรพิเศษ $errors ที่ระบุข้อผิดพลาดไว้ สามารถนำมาใช้ใน view articles/create (/resources/views/articles/create.blade.php)

```
@if($errors->any())
      <ul class="alert alert-danger">
      @foreach($errors->all() as $error)
            <li>{{ $error }}</li>
      @endforeach
      </ul>
@endif
```

หรือเลือกแสดงเฉพาะ Error ของ field ที่ต้องการ

```
@if($errors->first('title'))
      <div class="alert alert-danger">
            {{ $errors->first('title') }}
      </div>
@endif
```

ทดสอบการสร้าง Article แบบที่มีข้อผิดพลาด เช่น ไม่กรอกข้อมูล Title

4. หากต้องการเปลี่ยนข้อความแจ้ง Error ให้กำหนดใน function message() ของ Form Request Validation

ในไฟล์ /app/Http/Requests/ArticleRequest.php เพิ่ม function message()

```
public function messages(){
  return [
       'required' => 'You have to enter some data on :attribute field',
       'title.required' => 'Please enter the title of this article'
  ];
}
```

ทดสอบการสร้าง Article โดยไม่กรอกข้อมูล Title และ Body สังเกตข้อความแจ้ง Error ที่แสดง

ปฏิบัติการที่ 9	View Partial
วัตถุประสงค์	เพื่อให้เข้าใจ View Partial และวิธีการสร้าง View Partial เพื่อนำกลับมาใช้ซ้ำ

1. ในการแก้ไข (Update) Article มักจะใช้ Form เดียวกับการสร้าง (Create) Article

   สร้างไฟล์ /resources/views/articles/_form.blade.php

```
<div class="form-group">
    {!! Form::label('title', 'Article Title: ') !!}
    {!! Form::text('title', null, ['class'=>'form-control']) !!}
</div>
<div class="form-group">
    {!! Form::label('body', 'Body: ') !!}
    {!! Form::textarea('body', null, ['class'=>'form-control']) !!}
</div>
<div class="form-group">
    {!! Form::label('published_at', 'Publish on: ') !!}
    {!! Form::input('date', 'published_at',
                \Carbon\Carbon::now()->format('Y-m-d'),
                ['class'=>'form-control']) !!}
</div>
<div class="form-group">
    {!! Form::submit($submitButtonText,
                ['class' => 'btn btn-primary form-control']) !!}
</div>
```

2. สร้างไฟล์ /resources/views/errors/list.blade.php สำหรับแสดง Error ของ Form Validation

```
@if($errors->any())
<ul class="alert alert-danger">
    @foreach($errors->all() as $error)
    <li>{{ $error }}</li>
    @endforeach
</ul>
@endif
```

3. แก้ไข view articles/create (/resources/views/articles/create.blade.php) ให้เรียก View Partial ที่สร้างไว้ในข้อ 1. และ 2. โดยใช้คำสั่ง @include()

```
@extends('layouts.main')
@section('content')
    <h1 class="page-title">Write a New Article</h1>
    {!! Form::open(['url' => 'articles']) !!}
    @include('articles._form', ['submitButtonText' => 'Add Article'])
    {!! Form::close() !!}
    @include('errors.list')
@stop
```

ทดสอบสร้าง Article จาก URL http://localhost:8000/articles/create

4. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function edit() และ update()

```
public function edit($id){
      $article = Article::find($id);
      if(empty($article))
            abort(404);
      return view('articles.edit', compact('article'));
}

public function update($id, ArticleRequest $request){
      $article = Article::findOrFail($id);
      $article->update($request->all());
      session()->flash('flash_message', 'Edit Article Complete');
      return redirect('articles');
}
```

5. สร้างไฟล์ /resources/views/articles/edit.blade.php

```
@extends('layouts.main')
@section('content')
    <h1 class="page-title">Edit: {{ $article->title }}</h1>
    {!! Form::model($article, ['method' => 'PATCH',
        'action' => ['ArticlesController@update', $article->id]]) !!}
    @include('articles._form', ['submitButtonText' => 'Update Article'])
    {!! Form::close() !!}
    @include('errors.list')
@stop
```

6. สร้าง Link เพื่อ Edit Article ในไฟล์ /resources/views/articles/show.blade.php

```
@extends('layouts.main')
@section('content')
    <div class="page-header" id="banner">
         <h1>{{ $article->title }}</h1>
         <a class="btn btn-primary"
            href="{{ url("articles/{$article->id}/edit") }}">
                 Edit
         </a>
    </div>
    <div class="panel panel-default">
        <div class="panel-body">{{ $article->body }}</div>
    </div>
@stop
```

ทดสอบการแก้ไข Article

ปฏิบัติการที่ 10	Authentication
วัตถุประสงค์	เพื่อให้เข้าใจ Authentication

1. เมื่อเริ่มติดตั้ง Laravel 5 มีไฟล์ที่เกี่ยวข้องกับ Authentication ดังนี้

   /database/migrations/2014_10_12_000000_create_users_table.php
   /config/auth.php
   /app/User.php
   /app/Http/Controllers/Auth/AuthController.php
   /resources/views/auth/

   และ Route

```
Route::controllers([
      'auth' => 'Auth\AuthController',
      'password' => 'Auth\PasswordController'
]);
```

2. ในไฟล์ /app/Http/Controllers/Auth/AuthController.php เพิ่ม attribute $redirectTo เพื่อเปลี่ยนเส้นทางหลัง login

```
protected $redirectTo = '/articles';
```

3. หลังจากที่ Migrate users table แล้ว ให้สร้างผู้ใช้งาน โดยไปที่ URL http://localhost:8000/auth/register กรอกข้อมูล Name, Email Address, Password, Confirm Password เมื่อลงทะเบียนสำเร็จ ระบบจะ redirect ไปที่หน้า /articles

4. Logout ที่ URL http://localhost:8000/auth/logout

   Login ที่ URL http://localhost:8000/auth/login

5. สามารถตรวจสอบว่าผู้ใช้งานนั้น login หรือไม่ ด้วย Auth::check()

   เรียกดูข้อมูลผู้ใช้งานที่ login ด้วย Auth::user() เช่น

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

ปฏิบัติการที่ 11	Authorization
วัตถุประสงค์	เพื่อให้เข้าใจ Authorization

1. เมื่อเริ่มติดตั้ง Laravel 5 มีไฟล์ที่เกี่ยวข้องกับ Authorization ดังนี้

   /app/Http/Middleware/Authenticate.php
   /app/Http/Middleware/RedirectIfAuthenticated.php

2. ในไฟล์ /app/Http/Controllers/ArticlesController.php ให้เพิ่มการเรียกใช้ middleware auth เพื่อกำหนดสิทธิ์การสร้างและแก้ไข article เฉพาะผู้ใช้งานที่ login เท่านั้น

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

   (ต้องระบุ use Illuminate\Support\Facades\Auth; ด้วย)

3. ทดสอบเข้า URL สำหรับสร้าง Article (http://localhost:8000/articles/create) ทั้งกรณีที่ login และไม่ login

ปฏิบัติการที่ 12	User hasMany Articles
วัตถุประสงค์	เพื่อให้เข้าใจความสัมพันธ์ของ Model แบบ Many-to-one

1. ในไฟล์ /app/User.php เพิ่ม function articles()

```
public function articles(){
      return $this->hasMany('App\Article');
}
```

2. ในไฟล์ /app/Article.php เพิ่ม function user()

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

3. สร้าง Migration เพื่อเพิ่ม field user_id ในตาราง articles ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:migration articles_add_user_id --table="articles"
```

จะได้ไฟล์ /database/migrations/[วันเวลา]_articles_add_user_id.php

4. ในไฟล์ Migration ที่ได้จากข้อ 3. แก้ไข function up() และ down()

```
public function up(){
      Schema::table('articles', function(Blueprint $table){
            $table->integer('user_id')->unsigned()->default(1);
            $table->foreign('user_id')
                  ->references('id')
                  ->on('users')
                  ->onDelete('cascade');
      });
}
public function down(){
      Schema::table('articles', function(Blueprint $table){
            $table->dropColumn('user_id');
      });
}
```

จากนั้น Migrate ผ่าน CLI ด้วยคำสั่ง

```
php artisan migrate
```

5. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function store()

```
public function store(ArticleRequest $request){
      $article = new Article($request->all());
      Auth::user()->articles()->save($article);
      return redirect('articles');
}
```

สร้าง Article ผ่าน URL http://localhost:8000/articles/create

6. ในไฟล์ /resources/views/articles/index.blade.php เพิ่มคำสั่งเพื่อแสดงชื่อของผู้สร้าง Article

```
...
       <div class="panel-footer">
           By <strong>{{ $article->user->name }}</strong>
           {{ $article->published_at->diffForHumans() }}
       </div>
...
```

ปฏิบัติการที่ 13	article_tag
วัตถุประสงค์	เพื่อให้เข้าใจความสัมพันธ์ของ Model แบบ Many-to-many

1. ในไฟล์ app/Article.php เพิ่ม function tags()

```
public function tags(){
      Return $this->belongsToMany('App\Tag');
}
```

2. สร้าง Model Tag ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:model Tag
```

จะได้ไฟล์ /app/Tag.php และ /database/migrations/[วันเวลา]_create_tags_table.php

3. ในไฟล์ /app/Tag.php เพิ่ม attribute $fillable และ function articles()

```
protected $fillable = ['name'];
public function articles(){
      return $this->belongsToMany('App\Article');
}
```

4. ในไฟล์ Migration create_tags_table.php แก้ไข function up() และ down()

```
public function up(){
      Schema::create('tags', function(Blueprint $table){
            $table->increments('id');
            $table->string('name');
            $table->timestamps();
      });
      Schema::create('article_tag', function(Blueprint $table){
            $table->integer('article_id')->unsigned()->index();
            $table->foreign('article_id')->references('id')
                  ->on('articles')->onDelete('cascade');
            $table->integer('tag_id')->unsigned()->index();
            $table->foreign('tag_id')->references('id')->on('tags')
                                    ->onDelete('cascade');
            $table->timestamps();
      });
}
public function down(){
      Schema::drop('tags');
      Schema::drop('article_tag');
}
```

Migrate ผ่าน CLI ด้วยคำสั่ง `php artisan migrate`

5. ใช้ tinker (คำสั่งผ่าน CLI: `php artisan tinker`) สร้าง Tag จำนวนหนึ่ง เช่น

```
>>> App\Tag::create(['name'=>'sports']);
```

6. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function create()

```
public function create(){
      $tag_list = Tag::lists('name', 'id');
      return view('articles.create', compact('tag_list'));
}
```


ต้องระบุ use App\Tag; ด้วย

7. ในไฟล์ /resources/views/articles/_form.blade.php เพิ่ม selection สำหรับ Tag

```
<div class="form-group">
    {!! Form::label('tag_list', 'Tags:'); !!}
    {!! Form::select('tag_list[]', $tag_list, null,
                          ['class'=>'form-control', 'multiple']) !!}
</div>
```

8. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function store()

```
public function store(ArticleRequest $request){
      $article = new Article($request->all());
      Auth::user()->articles()->save($article);
      $tagsId = $request->input('tag_list');
      if(!empty($tagsId))
          $article->tags()->sync($tagsId);
      return redirect('articles');
}
```

9. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function edit()

```
public function edit($id){
      $article = Article::find($id);
      $tag_list = Tag::lists('name', 'id');
      if(empty($article))
          abort(404);
      return view('articles.edit', compact('article', 'tag_list'));
}
```

10. ในไฟล์ /app/Article.php เพิ่ม function getTagListAttribute()

```
public function getTagListAttribute(){
      return $this->tags->lists('id');
}
```

11. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function update()

```
public function update($id, ArticleRequest $request){
      $article = Article::findOrFail($id);
      $article->update($request->all());
      $tagsId = $request->input('tag_list');
      if(!empty($tagsId))
          $article->tags()->sync($tagsId);
      else
          $article->tags()->detach();
      session()->flash('flash_message', 'Edit Article Complete');
      return redirect('articles');
}
```

12. ในไฟล์ /resources/views/articles/show.blade.php เพิ่มการแสดง Tag ของ Article

```
@unless($article->tags->isEmpty())
      <div>Tags:
            <ul>
                  @foreach($article->tags as $tag)
                        <li>{{ $tag->name }}</li>
                  @endforeach
            </ul>
        </div>
@endunless
```

13. เมื่อทดสอบสร้าง Article ที่มี Tag จะพบว่าข้อมูลในตาราง article_tag ไม่มีการอัพเดทค่า created_at และ updated_at แก้ไขโดยในไฟล์ app/Article.php เพิ่ม withTimestamps() เพื่อให้ระบบอัพเดทค่าอัตโนมัติ

```
public function tags(){
      Return $this->belongsToMany('App\Tag')->withTimestamps();
}
```

ปฏิบัติการที่ 14	File Upload
วัตถุประสงค์	เพื่อให้เข้าใจวิธีการอัพโหลดไฟล์

1. สร้าง Migration เพื่อเพิ่ม field image ในตาราง articles ผ่าน CLI ด้วยคำสั่ง

```
php artisan make:migration articles_add_image --table=articles
```

จะได้ไฟล์ /database/migrations/[วันเวลา]_articles_add_image.php

2. ในไฟล์ Migration ที่ได้จากข้อ 1. แก้ไข function up() และ down()

```
public function up(){
      Schema::table('articles', function(Blueprint $table){
            $table->string('image')->nullable();
      });
}
public function down(){
      Schema::table('articles', function(Blueprint $table){
            $table->dropColumn('image');
      });
}
```

จากนั้น Migrate ผ่าน CLI ด้วยคำสั่ง

```
php artisan migrate
```

3. ในไฟล์ /app/Article.php เพิ่ม attribute $fillable

```
protected $fillable = ['title', 'body', 'published_at', 'image'];
```

4. แก้ไข Form ในไฟล์ /resources/views/articles/create.blade.php

```
@extends('layouts.main')
@section('content')
    <h1 class="page-title">Write a New Article</h1>
    {!! Form::open(['url' => 'articles',
                    'files' => true ]) !!}
    @include('articles._form',
                        ['submitButtonText' => 'Add Article'])
    {!! Form::close() !!}
    @include('errors.list')
@stop
```

5. เพิ่ม Form Input เพื่อรับอัพโหลดไฟล์ ในไฟล์ /resources/views/articles/_form.blade.php

```
...
   {!! Form::label('image', 'Article Image: ') !!}
   {!! Form::file('image', null) !!}
...
```

6. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function store()

```
public function store(ArticleRequest $request){
   $article = new Article($request->all());
   if($request->hasFile('image')){
      $image_filename = $request->file('image')
                                        ->getClientOriginalName();
       $image_name = date('Ymd-His-').$image_filename;
       $public_path = 'images/articles/';
       $destination = base_path() . "/public/" . $public_path;
       $request->file('image')->move($destination, $image_name);
       $article->image = $public_path . $image_name;
   }
    Auth::user()->articles()->save($article);
    $tagsId = $request->input('tag_list');
    if(!empty($tagsId))
        $article->tags()->sync($tagsId);
    return redirect('articles');
}
```

สร้างโฟลเดอร์ /public/images/articles/

7. แก้ไข Form ในไฟล์ /resources/views/articles/edit.blade.php

```
@extends('layouts.main')
@section('content')
    <h1 class="page-title">Edit: {{ $article->title }}</h1>
    {!! Form::model($article, ['method' => 'PATCH',
        'action' => ['ArticlesController@update', $article->id],
        'files' => true]) !!}
    @include('articles._form', ['submitButtonText' => 'Update Article'])
    {!! Form::close() !!}
    @include('errors.list')
@stop
```

8. ในไฟล์ /app/Http/Controllers/ArticlesController.php แก้ไข function update()

```
public function update($id, ArticleRequest $request){
   $article = Article::findOrFail($id);
   $article->update($request->all());
   if($request->hasFile('image')){
      $image_filename = $request->file('image')
                                        ->getClientOriginalName();
      $image_name = date('Ymd-His-').$image_filename;
       $public_path = 'images/articles/';
       $destination = base_path() . "/public/" . $public_path;
       $request->file('image')->move($destination, $image_name);
       $article->image = $public_path . $image_name;
       $article->save();
   }
    $tagsId = $request->input('tag_list');
    if(!empty($tagsId))
        $article->tags()->sync($tagsId);
    return redirect('articles');
}
```

9. ในไฟล์ /app/Http/Requests/ArticleRequest.php แก้ไข function rules()

```
public function rules(){
      return [
            'title' => 'required|min:5|max:255',
            'body' => 'required',
            'published_at' => 'required|date',
            'image' => 'mimes:png,jpeg,jpg'
      ];
}
```

10. ในไฟล์ /resources/views/articles/show.blade.php เพิ่มการแสดงรูป

```
<img src="{{ url($article->image) }}" alt="Article Image" />
```

ปฏิบัติการที่ 15	Localization
วัตถุประสงค์	เพื่อให้เข้าใจ Localization และวิธีการทำเว็บหลายภาษา

1. สร้างไฟล์ /resources/lang/en/site.php และกำหนดคำสำหรับภาษาอังกฤษ

```
<?php
return [
      'articles' => 'Articles',
      'all_articles' => 'All articles',
      'add_article' => 'Add Article'
];
```

2. สร้างไฟล์ /resources/lang/th/site.php และกำหนดคำสำหรับภาษาไทย

```
<?php
return [
      'articles' => 'บทความ',
      'all_articles' => 'บทความทั้งหมด',
      'add_article' => 'เพิ่มบทความ'
];
```

3. ในไฟล์ /resources/views/articles/index.blade.php ใช้คำสั่ง trans() เพื่อแปลภาษา

```
@extends('layouts.main')
@section('content')
    <div class="page-header" id="banner">
        <div class="row">
            <div class="col-xs-12">
                <h1>{{ trans('site.articles') }}</h1>
                <p class="lead">{{ trans('site.all_articles') }}</p>
            </div>
        </div>
    </div>
...
```

4. สร้าง Route สำหรับการเปลี่ยนภาษา ในไฟล์ /app/Http/routes.php

```
Route::get('lang/{lang}', function($lang){
    $available = ['en', 'th'];
    Session::put('locale', in_array($lang, $available)? $lang :
                                                Config::get('app.locale'));
    return redirect()->back();
});
```

5. สร้าง Middleware Locale เพื่อการเปลี่ยนภาษา ผ่าน CLI ด้วยคำสั่ง

   php artisan make:middleware Locale

   จะได้ไฟล์ /app/Http/Middleware/Locale.php ให้แก้ไข function handle()

```
use Config;
use Illuminate\Support\Facades\App;
...
public function handle($request, Closure $next){
      if(session()->has('locale')){
            App::setLocale(session()->get('locale'));
      }else{
            session('locale', Config::get('app.locale'));
      }
      return $next($request);
}
```

6. ในไฟล์ /app/Http/Kernel.php เพิ่ม 'App\Http\Middleware\Locale' ที่ attribute $middleware

```
      'App\Http\Middleware\Locale'
```

7. ในไฟล์ /resources/views/layouts/main.blade.php สร้างลิงก์สำหรับเปลี่ยนภาษา

```
<a href="{{ url('lang/en') }}" class="btn">EN</a>
<a href="{{ url('lang/th') }}" class="btn">TH</a>
```