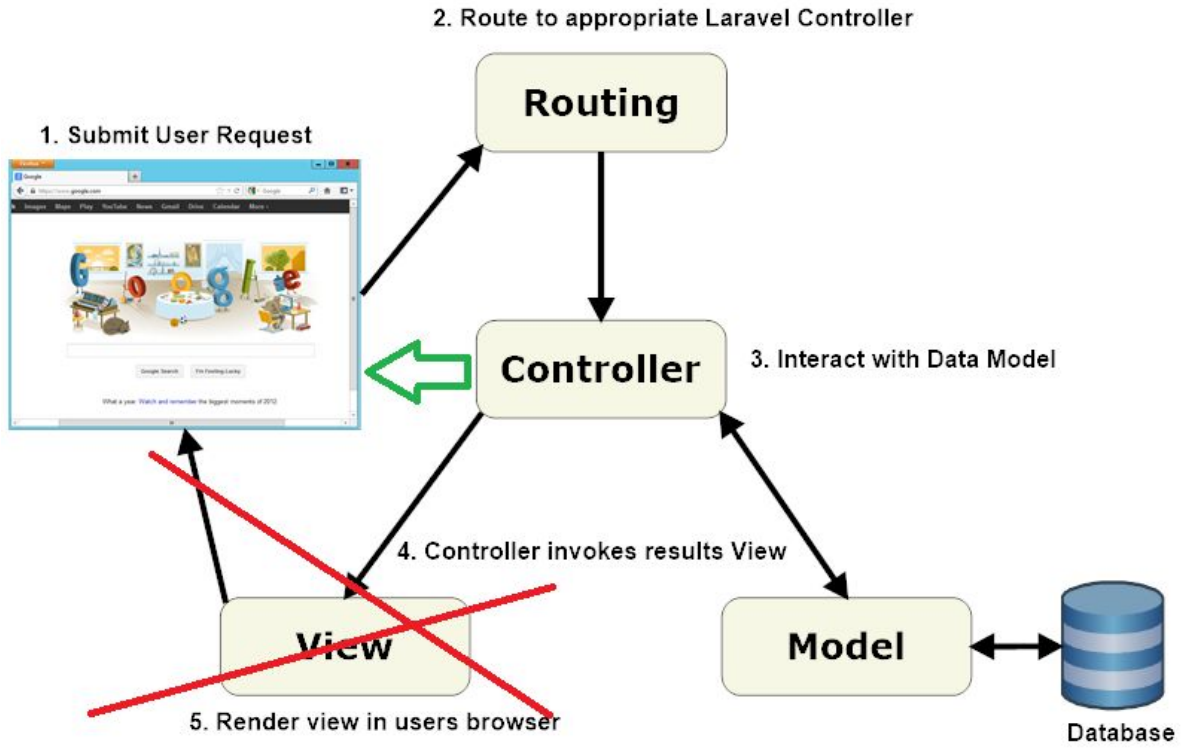
LARAVEL ตอนที่ 12 API

API ย่อมาจาก Application Programming Interface คือส่วนที่ใช้จัดการ Route url ที่ User จะใช้ขอข้อมูลต่างๆโดยติดต่อ API ผ่าน มี concept เหมือนกันกับ Web เพียงแต่จะไม่ผ่าน view แต่จะ return ข้อมูลเป็น json หรือ xml



ยกตัวอย่างข้อมูลที่ร้องขอผ่าน API เช่น

1. ข้อมูล Profile
2. API ผู้ให้บริการส่ง SMS
3. API ผู้ให้บริการส่ง Email
4. Login API

โดยการที่จะสามารถร้องขอข้อมูลได้นั้น จะต้องใช้ token ในการเป็นเหรียญยืนยันตัวตน ยกตัวอย่าง API ของ facebook ในการร้องขอข้อมูล Profile เช่น

https://graph.facebook.com/me?access_token=xxx

ตัวอย่างข้อมูล array

```
$arr["course"] = "Laravel 5.6";
$arr["day"] = "3";
```

ตัวอย่างข้อมูล json

```
{
"course":"Laravel 5.6",
"day":"3"
}
```

ตัวอย่างข้อมูล array

```
$arr[0]["CustomerID"]="C001";
$arr[0]["Name"]="Panupong Kongarn";
$arr[0]["Email"]="kongarn@gmail.com";
$arr[0]["CountryCode"]="TH";
$arr[0]["Budget"]="1000000";
$arr[0]["Used"]="600000";
```

ตัวอย่างข้อมูล json

```
[{
"CustomerID":"C001",
"Name":"Panupong Kongarn",
"Email":"kongarn@gmail.com",
"CountryCode":"TH",
"Budget":"1000000",
"Used":"600000"
}]
```

ในส่วนของ Laravel นั้น Routing แบบ API จะอยู่ที่ (**routes/api.php**) โดยจะถูกกำหนด prefix ของ route ในการเรียกใช้งานเริ่มต้นเป็น /api อยู่แล้ว

12.1 เริ่มสร้าง Product API

สร้าง ProductController สำหรับ Provide ข้อมูลแบบ **Public**

```
php artisan make:controller Api/ProductController --resource
```

แก้ไขไฟล์ (**routes/api.php**) เพิ่ม (prefix api อัตโนมัติ)

```
Route::resource('products', 'Api\ProductController');
```

วิธี clear config cache

php artisan config:cache

แก้ไขไฟล์ ProductController ในฟังก์ชัน index ให้เรียกข้อมูล Product มาแสดงสินค้า

```
use App\Model\Product as ProductMod;
```

```
public function index()
   {

       $products = ProductMod::all();
       return response()->json([
             'data' => $products
          ]);

   }
```

12.2 สร้าง LoginAPI

เราจะมาลองทำ LoginAPI โดยการทำทุกอย่างคล้ายๆกับ LoginWeb เพียงแค่ return จาก View เปลี่ยนเป็น json โดยมีข้อมูล **api_token** อยู่ภายใน

แก้ไข /database/migration/create_users_table **เพิ่มบรรทัด** การสร้าง column **api_token**

```
$table->string('api_token', 60)->unique()->nullable();
```

หลังจากนั้นให้รันคำสั่ง

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

สร้าง LoginController ผ่าน php artisan

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

แก้ไขไฟล์ (**routes/api.php**)

```
Route::post('login', 'Api\LoginController@authenticate');
```

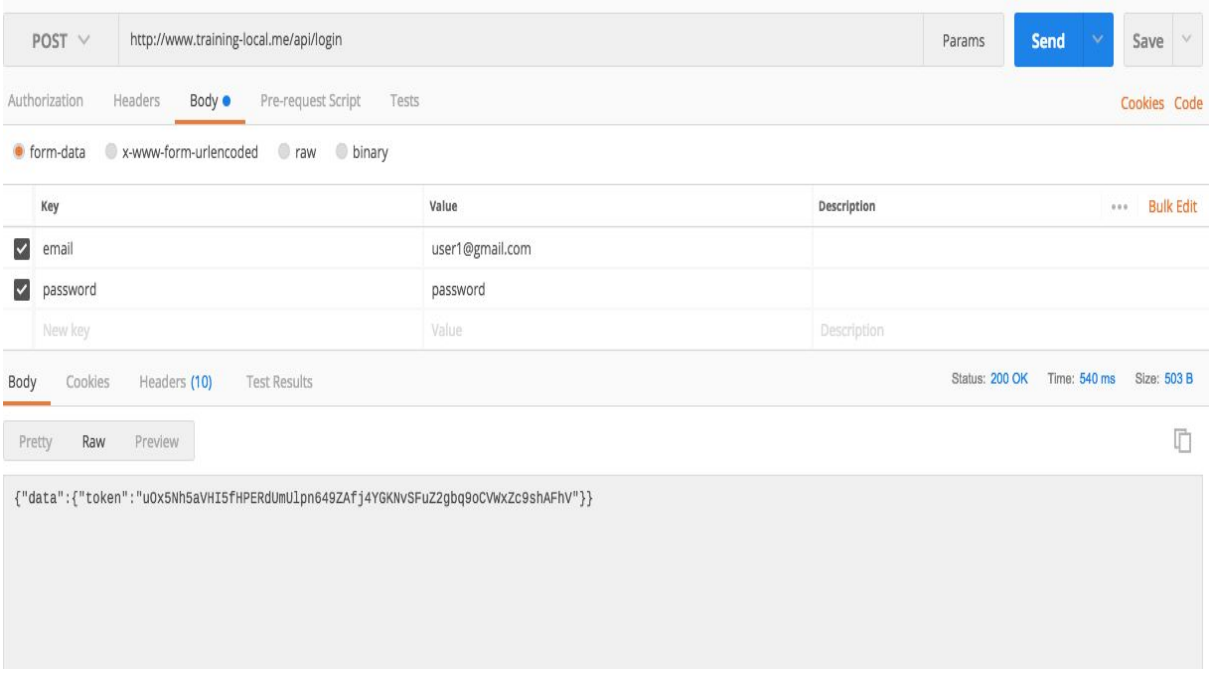
เพิ่ม method authenticate ใน /Api/LoginController

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

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
class LoginController extends Controller
{
    /**
     * Handle an authentication attempt.
     *
     * @param  \Illuminate\Http\Request $request
     *
     * @return Response
     */
    public function authenticate(Request $request)
    {

        $credentials = $request->only('email', 'password');

        if (Auth::attempt($credentials)) {
            // Authentication passed...
            // Get the currently authenticated user...

            $user = Auth::user();

            $user->generateToken();

            $output = array("token" => $user->api_token);
            //return json response
            return response()->json([
                'data' => $output ,
            ]);

        }else{
            return response()->json([
                'error' => 'Invalid Authenticated',
            ],401);
        }
    }


}
```

เพิ่ม method generateToken() สำหรับสร้าง api_token ใน /app/User.php

```
public function generateToken()
   {
      $this->api_token = str_random(60);
      $this->save();

      return $this->api_token;
   }
```

ทดสอบโดยการใช้ POSTMAN



12.3 สร้าง Private Resource API

สร้าง ShopController สำหรับ Provide ข้อมูลแบบ **Private**

```
php artisan make:controller Api/ShopController --resource
```

แก้ไขไฟล์ (**routes/api.php**)

```
Route::middleware('auth:api')->group(function () {
    Route::resource('shop', 'Api\ShopController');
});
```

middleware('auth:api') จะทำการตรวจสอบค่า api_token ซึ่งใช้มาตรฐาน Authorization Type Bearer Token Standard (https://tools.ietf.org/html/rfc6750)

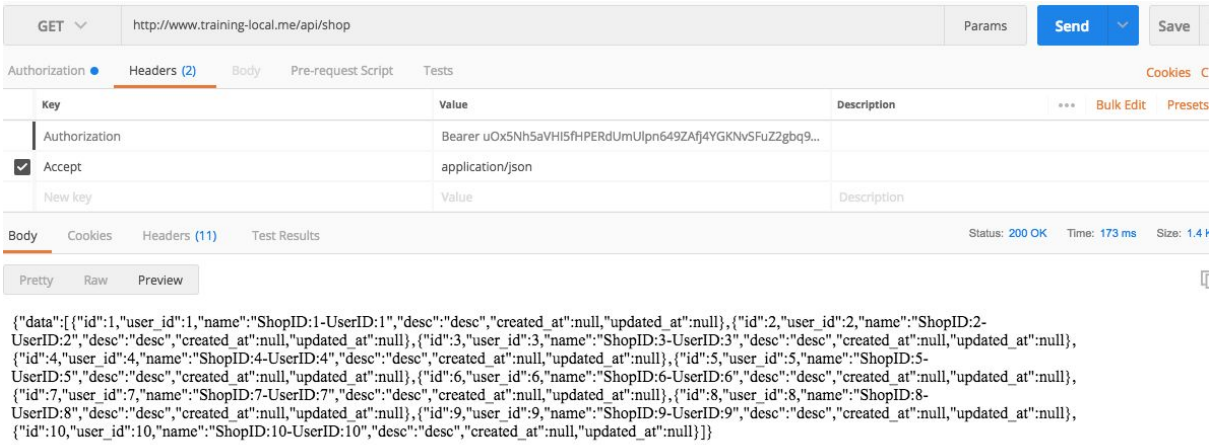
เพิ่มการดึงข้อมูล Shop Controller ใน method index()

```
use App\Model\Shop as ShopMod;
```

```
public function index()
   {

      $shops = ShopMod::all();
      return response()->json([
            'data' => $shops
         ]);

   }
```

ทดสอบโดยการใช้ POSTMAN

1. ไปคลิกที่ส่วน Authorization เลือก type เป็น Bearer-type ใส่ token
2. ไปคลิกที่ส่วน Header และกด Bulk Edit ใส่ Accept:application/json และกด Key-Value Edit กลับ



หากค่า email/password ส่งมาไม่ถูกต้อง เราสามารถ handle middleware Exception ได้ที่ เพิ่มโค้ด (app/Exceptions/Handler.php) ดังนี้

เพิ่ม method unauthenticated สำหรับ handle Auth::attempt retrun False

```
protected function unauthenticated($request, AuthenticationException $exception)
    {
        //when Auth::attempt not passed
        //for api return json 401
        //for web redirect to route as you want
        return $request->expectsJson()
                    ? response()->json(['error' => 'Unauthenticated'], 401)
                    : redirect()->guest(route('login'));
    }
```

ที่หัว class เพิ่ม

```
use Illuminate\Auth\AuthenticationException;
```

ทดสอบใส่ token ผิด โดยการใช้ POSTMAN

LARAVEL ตอนที่ 13 รู้จัก Unit Test

Unit Test เป็นการทดสอบโค้ดในระดับย่อยที่สุดว่าทำงานได้ถูกต้อง ตามที่เราคาดหวัง เพื่อป้องกันการผิดพลาดจากการเขียนโค้ด ไม่ว่าจะทำงานคนเดียวหรือเป็นทีม

ข้อดีทาง Unit Test

1. Finds Software Bugs Early

ทำให้เราเจอบัคได้รวดเร็ว เนื่องจากเราจะต้องรัน Unit Test ก่อนการ Deployment เสมอ เพื่อช่วยป้องกันเรื่องการแก้โค้ดที่กระทบส่วนอื่นๆ ทำให้ที่มีบางสิ่งผิดพลาด

2. Provides Documentation

Unit test ที่เราสร้างนั้นจะถือเป็นเอกสารของระบบ นักพัฒนาคนอื่นๆ สามารถเรียนรู้จากตรงนี้ได้ง่าย

3. Repeat Test

การเทสซ้ำเดิมๆ มี 1-100 งาน มี 101-1000 งาน ก็ต้องเทสตั้งแต่ 1-1000

โดย Laravel นั้นจะมีไฟล์ **phpunit.xml** เป็น config ของ Unit Test และโฟลเดอตัวอย่าง Test จะอยู่ใน /tests

คำสั่งรัน Unit Test (หากไม่ map environment variable ใน window)

(ต้อง cd เข้าไปที่ root path ของ project)

เราควรจะเคลีย config ระบบของ laravel ก่อน

```
php artisan config:cache
```

Windows

```
vendor\bin\phpunit
```

MacOS

```
./vendor/bin/phpunit
```

13.1 PHP Code Coverage

PHPUnit ในบางครั้งเราอาจเขียน unit test ซ้ำๆที่เดิม ตัวนี้จะมาช่วยตรวจสอบโค้ด Line , Method ต่างๆว่าเราเทสผ่านบรรทัดไหนบ้าง คิดเป็นกี่ % ซึ่งเราควรเขียน unit test ให้วิ่งผ่านทุกๆจุดของโปรแกรมของเรานั่นเอง (ใช้เป็นเงื่อนไขการ deployment ได้)

ติดตั้ง phpunit code coverage

```
composer require phpunit/php-code-coverage --dev
```

phpunit code coverage จะใช้ xedug ในการสร้าง report ดังนั้นให้เราตรวจสอบ xdebug extension ใน php.ini หากยังไม่มีให้ใส่ตามนี้ หากมีให้ uncomment ออกมา

เข้าไปที่ https://gist.github.com/odan/1abe76d373a9cbb15bed

- Copy the file `php_xdebug-2.6.0-7.2-vc15.dll` to: `C:\xampp\php\ext`
- Open the file `C:\xampp\php\php.ini` with Notepad++

```
[XDebug]
zend_extension = "c:\xampp\php\ext\php_xdebug-2.6.0-7.2-vc15.dll"
```

หลังจากนั้น Restart XAMPP และทดสอบตรวจสอบ phpversion และ xdebug โดยพิมพ์คำสั่ง

```
php -v
```

จะต้องพบคำว่า with **Xdebug v2.6.0**

สำหรับ Wamp

[xdebug]
zend_extension="c:/wamp/bin/php/php7.1.16/zend_ext/php_xdebug-2.6.0-7.1-vc14.dll"

ที่ไฟล์ phpunit.xml เพิ่มคำสั่งด้านล่างนี้ ไว้ล่างสุดก่อนปิด </phpunit> เพื่อบอกให้สร้างรายงาน phpunit code coverage เป็น html ให้ด้วย ที่ report folder

```
<logging>
        <log type="coverage-html" target="./report" lowUpperBound="50"
highLowerBound="80" />
  </logging>
```

ทดสอบรัน unit test อีกครั้ง

```
vendor\bin\phpunit
```

13.1 Mockery

Mockery ใช้สำหรับจำลอง(mock) object ในการทำ Unit test เราจะไม่มีการต่อ database ในการทำ Unit test ยกตัวอย่างเช่น การ mock object ในการต่อ database , object ของคลาสต่างๆ

วิธีติดตั้ง mockery

```
composer require mockery/mockery --dev
```

ตัวอย่างการ Mock ตัวแปร Auth Object

(Unit Test LoginWeb)

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

```
    public function testLoginSuccess()
    {
        $credential = [
            'email' => 'kongarn@gmail.com',
            'password' => '11111111'
        ];

        $this->withoutMiddleware();

        Auth::shouldReceive('attempt')->once()->withAnyArgs()->andReturn(true);
        Auth::shouldReceive('user')->once()->withAnyArgs()->andReturn(true);

        $response = $this->post('api/login',$credential);

        $response->assertRedirect('admin/user');
    }

    public function testLoginFail()
```

```
    {
        $credential = [
            'email' => 'user@ad.com',
            'password' => 'incorrectpass'
        ];

        $this->withoutMiddleware();

        $response = $this->post('/login',$credential);

        $response->assertRedirect('/login');
    }
```

ในการเทสนั้น จะต้องปิด `middleware` ทิ้ง เพื่อให้ปิด `csrf_token , auth , auth:api`

```
use Illuminate\Foundation\Testing\WithoutMiddleware;
```

```
$this->withoutMiddleware();
```

(Unit Test Get Resource User API)

```
public function testGetResourceAPISuccess()
    {
        $user = factory(User::class)->create([
            'surname' => 'pizzamuncher'
        ]);

        $this->withoutMiddleware();

        Auth::shouldReceive('user')->once()->withAnyArgs()->andReturn($user);

        $response = $this
            ->json('GET','/api/resource/user');

        $response->assertStatus(200);
        $response->assertSee($user->surname,'pizzamuncher');

    }
```

```
public function testGetResourceAPIFail()
{

    $response = $this
       ->json('GET','/api/resource/user');

    $response->assertStatus(401);

}
```

Factory เป็น class สำหรับจำลอง Object ต่างๆใน unittest โดยสามารถระบุค่าลงไปได้ เนื่องจากบาง column นั้นบังคับระบุค่า จากตัวอย่าง Table User surname column ของเรานั้น require ค่า

ทดสอบ PHP Unittest โดยการรันคำสั่ง

```
vender\bin\phpunit
```

LARAVEL ตอนที่ 14 Dependency

Dependency คือ external library ที่เราจำเป็นต้องใช้ในโปรเจ็คของเรา ในภาษา PHP นั้น เราใช้ความสามารถของ composer ในการโหลดเข้ามาใช้งาน

14.1 ติดตั้ง Lib Excel (https://github.com/Maatwebsite/Laravel-Excel)

```
composer require maatwebsite/excel
```

14.2 เพิ่ม method testexcel ใน DemoController และอย่าลืม use BladeExport Class สำหรับ

```
use \App\Exports\BladeExport;
use App\User as UserMod;
```

```
public function testexcel(){
    $user = UserMod::all();
    return \Excel::download(new BladeExport($user->toArray()), 'invoices.xlsx');
}
```

14.3 สร้าง excel template ใน /resources/views/excel/template.blade.php

```
<table>
        <thead>
        <tr>
            @foreach($data[0] as $key => $value)
```

```
                    <th>{{ ucfirst($key) }}</th>
                @endforeach
            </tr>
            </thead>
            <tbody>
            @foreach($data as $row)
                <tr>
                @foreach ($row as $value)
                    <td>{{ $value }}</td>
                @endforeach
                </tr>
            @endforeach
            </tbody>
        </table>
```

14.4 สร้างไฟล์ BladeExport สำหรับโหลด excel template (/app/Exports/BladeExport.php)

```
<?php
namespace App\Exports;

use Illuminate\Contracts\View\View;
use Maatwebsite\Excel\Concerns\FromView;

class BladeExport implements FromView
{

    private $data;
```

```
    public function __construct($data)
    {
        $this->data = $data;
    }

    public function view(): View
    {
        return view('/excel/template', [
            'data' => $this->data
        ]);
    }
}
?>
```

ติดตั้ง Lib PDF (https://github.com/niklasravnsborg/laravel-pdf)

ติดตั้ง Lib QRCODE (https://github.com/werneckbh/laravel-qr-code)