محتوى الملحق:

- حلول تمرين الlaravel بالإضافة لشرحها وبعض الملاحظات الهامة.
- ارقام المحاضرات التي يوجد فيها التمرين 3-4-5.

Exercise 3

1-Create end point to get list of user's names after capitalize all names.

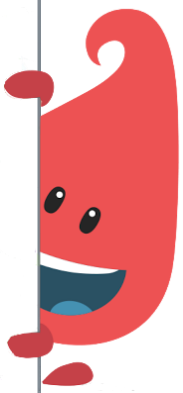
خطوات الحل:

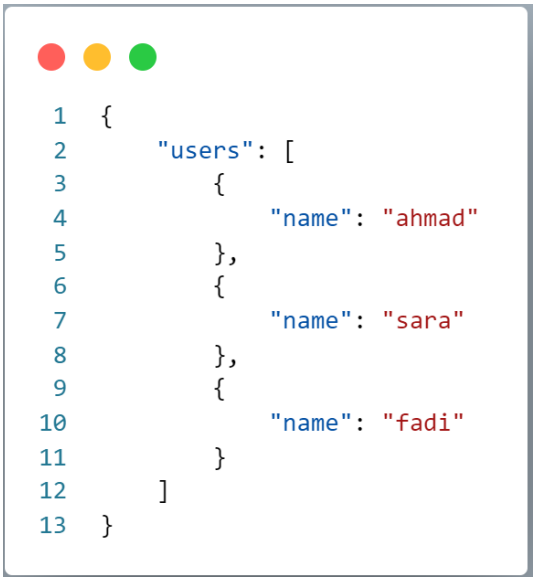
- قمنا بإنشاء controller وأسميناه UsersController بحيث يحتوي على تابع يدعى showUsersNames والمهمة الأساسية لهذا التابع هي أخذ أسماء المستخدمين من ملف json مكتوب ومن ثم جعل أحرف كل اسم من أسماء المستخدمين أحرف كبيرة.

```php
<?php

namespace App\Http\Controllers;

class UsersController extends Controller{
    public function ShowUsers(){
        #code
    }
}
```





```
{
    "users": [
        {
            "name": "ahmad"
        },
        {
            "name": "sara"
        },
        {
            "name": "fadi"
        }
    ]
}
```

- نقوم بإنشاء ملف json ولنسميه فرضاً users.json، قمنا بوضع ملف الـ json ضمن مجلد الـ public، ومن ثم قمن بكتابة أسماء الـ users الافتراضية ضمن هذا الملف.

- بدايةً نقوم بتعريف متغير ولنسميه usersJsonData فرضاً بحيث يحتوي هذا المتغير على المعلومات الموجودة ضمن ملف الـ json الذي قمنا بكتابة أسماء الـ users ضمنه:

```
$ usersJsonData = file_get_contents(public_path('users.json'));
```

- ومن ثم نقوم بتحويل هذه المعلومات إلى مصفوفة لنستطيع التعامل معها والتعديل عليها بسهولة:

```
$usersArray = json_decode($ usersJsonData, true);
```

- ثانياً نعرف مصفوفة ولنسمها capitalizedUsers لتقوم بتخزين أسماء الـ users ضمنها بعد أن نقوم بعملية تحويل الأحرف، ومن ثم نقوم بالمرور على جميع الأسماء الموجودة ضمن المصفوفة ونحول أحرف كل اسم لأحرف كبيرة عن طريق تعليمة foreach:

```
foreach ($usersArray as $users) {
    foreach ($users as $user) {
        $user_name = $user['name'];
        $user_name = strtoupper($user_name);
        array_push($capitalizedUsers, $user_name);
    }
}
```

- نلاحظ أننا قمنا بالمرور على عناصر الـ usersArray بداية ومن ثم قمنا بالمرور على كل عنصر من عناصرها كمصفوفة وذلك لأنه عندما قمنا بكتابة ملف الـ json عرفنا مصفوفة أوبجيكتات أي أن كل عنصر من عنا صر المصفوفة هو مصفوفة أيضاً.

- في تعليمة الـ arraypush نقوم بإضافة الـ user_name بعد تكبير أحرفه على المصفوفة capitalizedUsers عند كل مرور.

وهكذا يكون قد أصبح لدينا مصفوفة تحوي على الأسماء بالشكل المطلوب، وللحصول على هذه الأسماء لدينا عدة طرق (إما طباعتها على شاشة المستخدم عند وضع الـ عقم المحدد في الـ route أو نقوم بإعادته كـ response json ولكن الطريقة الصحيحة أن نعيده كـ json response لأنها نتيجة من الـBackend وعلى الـFrontend عرضها بالطريقة التي يراها مناسبة.

- نقوم بإنشاء الـ route الخاص بهذا التابع كالتالي في الملف routes/web.php:

```
Route::get('/showusers', [UsersController::class, 'ShowUsersNames']);
```

وتكون النتيجة النهائية للـController على الشكل التالي:

```php
public function ShowUsers(){
        $capitalizedUsers = [];
        $usersJsonData = file_get_contents(public_path('users.json'));
        $usersArray = json_decode($usersJsonData, true);
        foreach ($usersArray as $users) {
            foreach ($users as $user) {
                $user_name = $user['name'];
                $user_name = strtoupper($user_name);
                array_push($capitalizedUsers, $user_name);
            }
        }
        return response()->json([
            'users name'=>$capitalizedUsers
        ]);
    }
```

2-Add a new JSON file with array of object, each one contains name, email and phone number, and return a success message only if user sent an existed email or /and phone number.

خطوات الحل:

بداية نقوم بإضافة الـ fields التالية email و phone لملف الـ json السابق، ومن ثم ننشئ تابع باسم checkUsere() وضمن هذا التابع نعرف متغير باسم $user_input يقوم بأخذ الـ input من المستخدم، ومن ثم نقوم بنفس الخطوات التي قمنا بها في التمرين السابق لتخزين ملف الـ json ومن ثم تحويله لمصفوفة.

```json
{
    "users": [
        {
        "name": "ahmad",
        "phone": "0912233444",
        "email": "ahmad@gmail.com"
        },
        {
            "name": "sara",
            "phone": "0912233555",
            "email": "sara@gmail.com"
        },
        {
            "name": "fadi",
            "phone": "0912233777",
            "email": "fadi@gmail.com"
        }
    ]
}
```

شكل ملف الـJSON =>

```php
public function CheckUserByEmailOrPhone(){
        $usersJsonData = file_get_contents(public_path('users.json'));
        $usersArray = json_decode($usersJsonData, true);
    }
```

- وللبحث ضمن ملف الـ json قمنا بإنشاء حلقتين للبحث ضمن مصفوفة الـ objects كما في المثال السابق وأضفنا شرط (إذا وجد الـ user يعيد البرنامج json data).
- قمنا بتعريف متغير $find قيمته الابتدائية false وتتغير قيمته لـ true عندما يجد الـ user المطلوب، الهدف من هذا المتغير معرفة فيما إذا وجد الـ user أم لا فإذا لم يجده تبقى قيمته $false وبالتالي يعيد json data مختلفة عن السابقة كما هو موضح في الكود التالي:

```php
public function CheckUserByEmailOrPhone(){
        $user_input = request()->query('user_input');
        $usersJsonData = file_get_contents(public_path('users.json'));
        $usersArray = json_decode($usersJsonData, true);
        $find = false;
        foreach ($usersArray as $users) {
            foreach ($users as $user) {
                if ($user['email'] == $user_input || $user['phone'] == $user_input) {
                    $find = true;
                    return response()->json([
                        'message' => 'successfully enter'
                    ]);
                }
            }
        }
        if (!$find) {
            return response()->json(
                [
                    'message' => 'undefined phone number or email'
                ]
            );
        }
    }
```

Exercise 2

تعلمنا في هذه المحاضرة الـRequest Methos وطبقنا كل من الـget/post/delete على مثال المنتجات Products وتعمنا كيفية استخدام الـpostman وكيف يساعدنا في التعامل مع الـAPI حيث لا يمكننا تجربة الـrequest methods من الـWeb Browser مباشرة.

تذكرة:

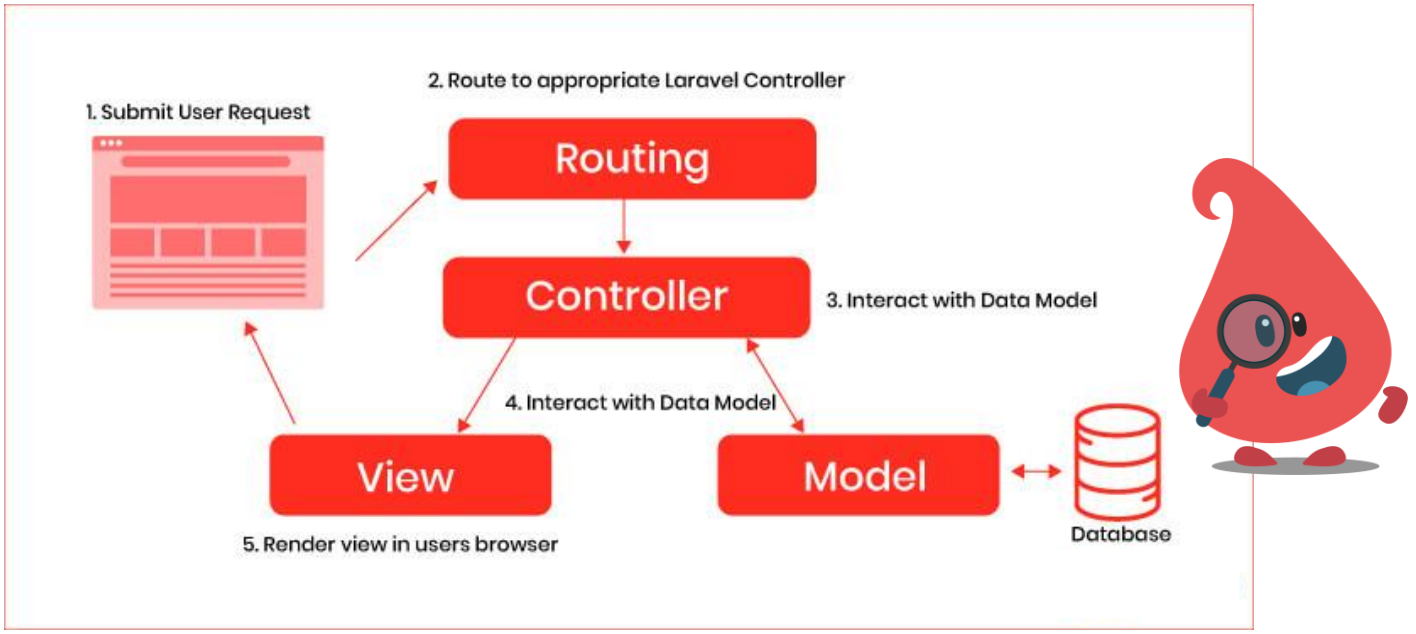
GET: تستخدم هذه الـMethod بشكل كبير لطلب data دون القيام بأي تعديل على Database أو الـServer.

POST: تستخدم بشكل كبير لإنشاء data جديدة أو التعديل على data موجودة في Database أو الـServer ويمكن استخدامها عوضاً عن الـDelete او الـPut حيث هذه الـMethods مشتقة منها.

DELETE: هو نوع من الـPOST Methods، ولكنها تستخدم بشكل أساسي لحذف Data من Database أو الـServer.

وسنكمل بحل الطلبات الإضافية معاً.

الـMVC في Laravel تمثل بالتالي:



- Add a new API to the previous example, which update the name of an existing product by getting "product index number" and the "new name" as parameters, and replace the product old name with the new one.
- Note that the new API should return an error if it could not find the product by its number.

خطوات الحل:

- أضافة Function UpdateName للـController يحتوي على الـLogic المطلوب (تعديل اسم مستخدم عن طريق الـID الخاص به والتأكد من انه مستخدم موجود لدينا).
- أضافة end point للـApi Route وستكون من نوع POST كما نلاحظ لأننا نريد التعديل على الـData.
- ملف الـJson الذي يحتوي على الـData حيث أنشأناه في بداية التمرين.

أنشاء الـFunction UpdateName:

- سوف ننشأ هذا الـfunction في الـProductController الذي أنشأناه في بداية التمرين في المسار app/Http/Controllers.
- يمتلك هذا الـFunction على parameter وحيد هو : Request request حيث نريد إرسال المعلومات عن طريق الـRequest Body وهذه الطريقة أفضل من أرسال المعلومات في الـURL لأنها أكثر أماناً.
- الأسطر (4<-2) من أجل الوصول إلى الى ملف الـJSON الخاص بالمنتجات وهي على الترتيب:

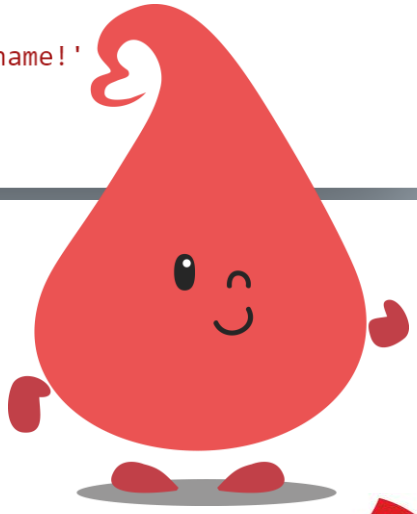
filePath: سنخزن بداخله مسار الملف json للوصول إليه.

fileContent: سنخزن بداخله ملف الـJson بعد إحضاره عن طريق الـMethod الجاهزة file_get_content.

jsonContent: سنخزن بداخله ملف الـJson ولكن بعد اجراء عملية decode له للتعامل مع محتوياته كمتحولات والتعديل عليها إذا ما اردنا.

- الأسطر (8<-6) التأكد من أن الـID موجود لدي أو نعيد رسالة error أذا ما طلب المستخدم تعديل اسم منتج غير موجود في الأصل.
- الأسطر (10<-9) التعديل على الاسم.
- السطر (11) إعادة تخزين ملف json المعدل (ملف المنتجات) مع إجراء عملية encode عليه.
- الأسطر (14<-12) إعادة response يدل على نجاح العملية للمستخدم.

```php
public function UpdateName(Request $request){
        $filePath = 'C:Users\RBCs\PhpProjects\lap_2\storage\app\products_list.json';
        $fileContent = file_get_contents($filePath);
        $jsonContent = json_decode($fileContent,true);
        $productId = $request->input('id');
        if($productId < 0 || $productId > count($jsonContent)){
            return response()->json(['message' => 'Invalid ID'],400);
        }
        $newName = $request->input('newName');
        $jsonContent[$productId-1]['name'] = $newName;
        file_put_contents($filePath,json_encode(array_values($jsonContent)));
        return response()->json([
                'message' => 'Product $productId has a new name!'
        ]);
    }
```



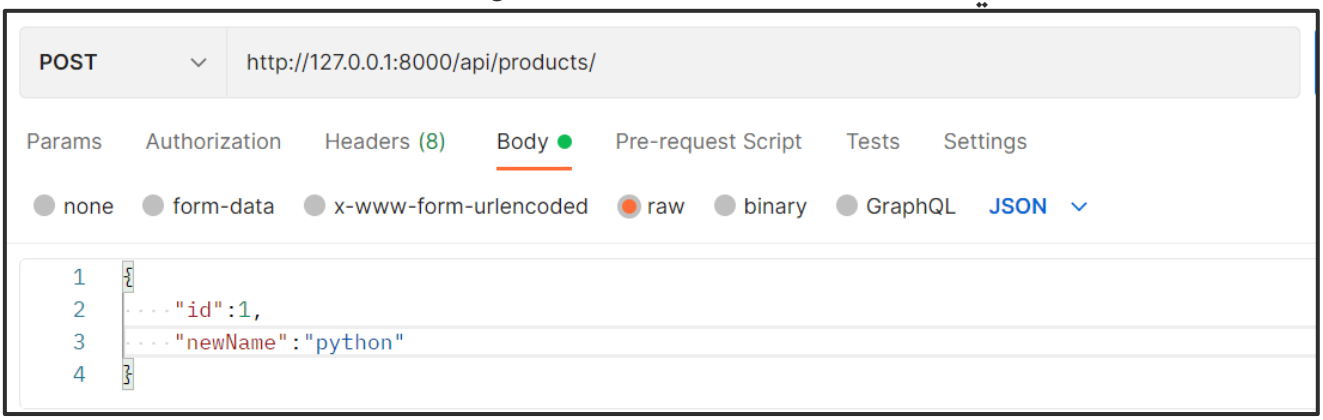
أنشاء Route للUpdateName:

- كما ذكرنا ستكون الـMethod في الـRoute من نوع POST لأننا نريد التعديل على الـData.

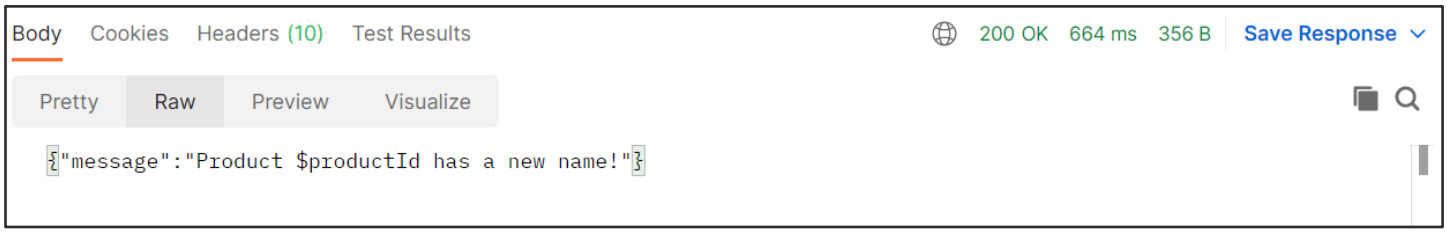
```php
//edit the name of a product
Route::post('products',[ProductController::class,'UpdateName']);
```

النتيجة على الـPostman:

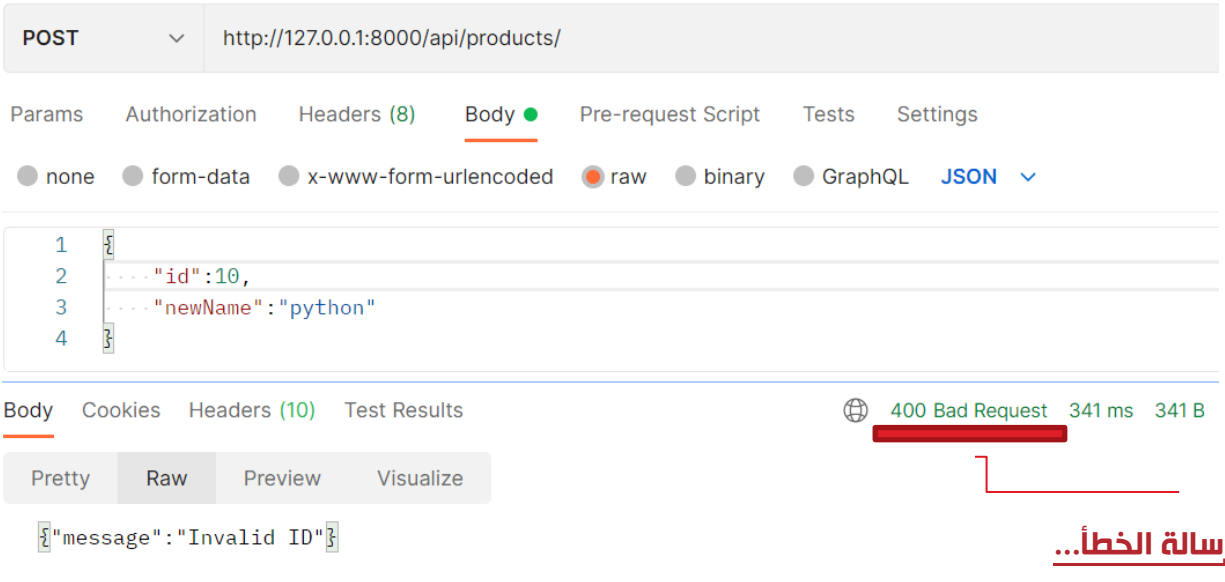
- نريد ارسال المعلومات في الـRequest body وسنرسل ID صحيح هذه المرة وستكون النتيجة:



- وسيكون الـResponse:



- بنما لو ارسلنا ID خاطئ ستكون النتيجة:

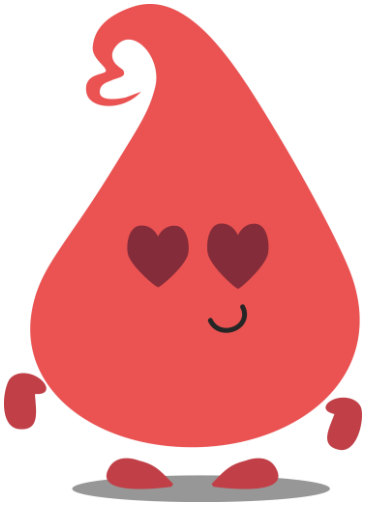


نلاحظ رسالة الخطأ...

طريقة التعامل مع الRoute بالشكل الحديث vs الطريقة القديمة

الميزات عند الضغط على زر Ctrl + الضغط على اسم الكلاس او الميثود يمكننا الدخول إلى ملف الClass واذا ما كان الاسم فيه خطأ فلن نتمكن من الدخول وهذا يساعدنا من التأكد من صحة اسم الMethod والController.

```php
//the old way
Route::post('product','ProductController@UpdateName');
//the new way
Route::post('products',[ProductController::class,'UpdateName']);
```

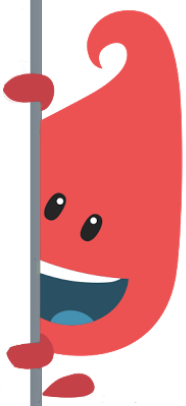


Exercise 3

1. We have a list of products, each one consists of: name, price, owner email. [JSON FILE]

-قمنا بإنشاء ملف json ولنفترض أسميناه data.json وقمنا بوضعه ضمن C:\xampp\htdocs ومن ثم قمنا بوضع معلومات ال users من اسم وايميل وسعر المنتج ووصف لهذا المنتج ونوعه.

```json
[
    {
        "name":"Ali",
        "brand":"brandd",
        "desc":"d",
        "price":"5252",
        "email":"Ali@gmail.com"
    },
    {
        "name":"Ahmad",
        "brand":"branddd",
        "desc":"dd",
        "price":"5252",
        "email":"Ahmad@gmail.com"
    }
]
```



2. We have a list of user credentials [Email, Password] for many users in a new JSON FILE.

-قمنا بانشاء ملف json وأسميناه Users.json و قمنا بوضعه ضمن C:\xampp\htdocs ومن ثم بوضع معلومات الـ Users الموجودين وكل user لديه Email و Password.

```json
[
    {
        "email": "Ali@gmail.com",
        "password": "5252"
    },
    {
        "email": "Ahmad@gmail.com",
        "password": "5252"
    }
]
```

3. We have 2 end-points:
   1. Login using a correct email & password, the response of success login should be a valid token for that user.

خطوات الحل:

-قمنا بانشاء Middleware وليكن CheckMiddleware ونقوم بكتابة الكود ضمن التابع handle

-بداية قمنا بتعريف متحولين $error, $valid ونقوم بوضع 'X-TOKEN' ضمن ال header ضمن برنامج ال Postman وفي السطر 13 قمنا باختبار فيما إذا كان ال header يحوي هذه ال key ومن ثم خزننا ضمن ال $token هذه ال key.

```php
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Http\Request;
class CheckTokenMiddleware
{

    public function handle(Request $request, Closure $next)
    {
        $error = false;
        $valid = false;

        if(!$request->hasHeader('X-TOKEN')){
            $error = true;
        }
        $token = $request->header('X-TOKEN');
```

```php
try{
        $jsonstr = base64_decode($token);
        $jsonPayLoad = json_decode($jsonstr, true);
        $file = file_get_contents('C:\xampp\htdocs /Users.json');
        $decodedFile = json_decode($file, true);

        if(!$jsonPayLoad){
            $error = true;
        }
        if(!isset($jsonPayLoad['email'])){
            $error = true;
        }
        for($i = 0;$i<count($decodedFile);$i++){
            if($decodedFile[$i]['email']==$jsonPayLoad['email'] &&
             $decodedFile[$i]['password']==$jsonPayLoad['password']){
                $valid = true;
            break;
            }
        }
    }
```

- من ثم سنختبر فيما إذا كان هذا ال token صحيح ام لا ونقوم بفك تشفيره عن طريق base64_decode ومن ثم اختبار فيما اذا كان يحوي بيانات أم لا.
- وفي السطر 10 اختبرنا فيما اذا كان ملف الـ json يحوي email.
- في حلقة ال for اختبرنا فيما اذا كان ملف الـ json مطابق لمعلومات هذا الuser ام لا وإذا كان مطابق هذا يعني ان $valid=true

```php
    catch(\Exception $exeption){
            $error = true;
        }
        if($error){
            return response()->json([
                'massage' => 'Invaid token'
            ]);
        }
        if(!$valid){
            return response()->json([
                'massage' => 'Incorect Account'
            ]);
        }
        return $next($request);
    }
}
```

-في الخطوة الأخيرة لدينا Catch فاذا كان هنالك أي خطا فسيكون هنالك response وهو على سبيل المثال 'Invalid token'.

-أو اذا كان $valid=false سيكون هنالك response وهو 'Incorrect Account'

-وأخيرا اذا لم يكن هنالك أخطاء سيسمح لنا الـ Middleware بتنفيذ التابع الموجود في الـ Controller.

-يجب علينا أيضاً كتابة هذه التعليمة ضمن ملف ال Kernel.php ويمكن اختيار أي اسم وفي مثالنا وضعنا 'check_token'.

```php
'check_token' => \App\Http\Middleware\CheckTokenMiddleware::class,
```

-والآن نذهب وننشئ controller جديد ونضع الدالة المراد تنفيذها ولتكن check وتعيد لنا response وليكن 'Login Success'.

```php
public function check(Request $request){
        return response()->json([
            'massage' => 'Login Success'
        ]);
    }
```

```php
Route::get('/check_id', [MainController::class, 'check'])->middleware(['check_token']);
```

أخيراً في ملف api.php ننشئ Route جديد واستخدمنا ->middleware(['check_token'])

2. Users with a valid token can delete products only if the user email matches the owner email of the requested product

خطوات الحل:

```php
<?php
namespace App\Http\Middleware;
use Closure;
use Illuminate\Http\Request;
class ProductsEdit
{
    public function handle(Request $request, Closure $next)
    {

        $error = false;
        $valid = false;
        $valid1 = false;
        $pr =  $request->route('id');
        if(!$request->hasHeader('X-TOKEN')){
            $error = true;
        }
        $token = $request->header('X-TOKEN');
```

-نقوم بإنشاء Middleware وليكن Productsedit.php وذلك للتحقق فيما اذا كان صاحب هذا الايميل مخول بحذف هذا المنتج أم لا

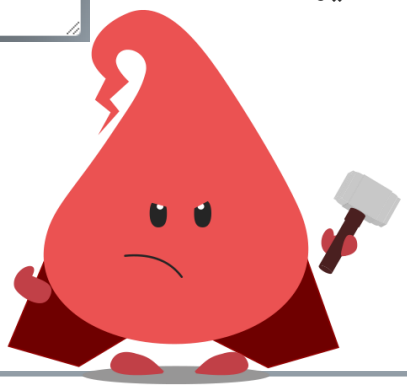
وكما فعنا سابقنا ننشئ ثلاث متحولات:

$error, $valid, $valid1,$pr

وضمن ال pr خزننا ال id الخاص بهذا المنتج

واختبرنا فيما إذا كان ال header يحوي ال X-TOKEN.

وخزننا الـtoken ضمن المتغير $token.



```php
try{
        $jsonstr = base64_decode($token);
        $jsonPayLoad = json_decode($jsonstr, true);
        $file = file_get_contents('C:\xampp\htdocs/Users.json');
        $decodedFile = json_decode($file, true);
        $file1 = file_get_contents('C:\xampp\htdocs/data.json');
        $decodedFile1 = json_decode($file1, true);

        if(!$jsonPayLoad){
            $error = true;
        }
        if(!isset($jsonPayLoad['email'])){
            $error = true;
        }
        for($i = 0;$i<count($decodedFile);$i++){
            if($decodedFile[$i]['email']==$jsonPayLoad['email'] &&
               $decodedFile[$i]['password']==$jsonPayLoad['password']){
                $valid = true;
            }
        }
        if ($decodedFile['email'] == $decodedFile1[$pr]['email']) {
            $valid1 = true;
        }

    }
```

- وأيضاً لدينا try وكما فعلنا سابقاً نفك تشفير ملف ال json ونخزن محتوى ملفات ال json التي لدينا ضمن $file و $file1
- ونختبر في شرطي ال If كما فعلنا سابقاً
- اما في حلقة ال For وشرط ال if الأخير اختبرنا فيما اذا كان ال email وال password المدخلين متطابقين مع معلومات ال user المخول بحذف المنتج.

```php
catch(\Exception $exeption){
            $error = true;
        }
        if($error){
            return response()->json([
                'massage' => 'Invaid token'
            ]);
        }
        if(!$valid&&!$valid1){
            return response()->json([
                'massage' => 'Incorect Account'
            ]);
        }
        return $next($request);
    }
}
```

- أخيراً لدينا Catch للاكتشاف فيما اذا كان هنالك أخطاء أم لا واذا كان كل شيئ صحيح يتم تنفيذ التابع الموجود في الـ Controller.

- وهنا لدينا التابع وليكن deleteByID وقمنا باختبار تحقق وجود هذا الـ id المراد حذفه واذا كان موجود يتم حذفه عن طريق التعليمة في السطر 10 وبعد ذلك يكون هنالك response بتحقق عملية الحذف.

```php
public function deleteByID($id)
{
    $file = file_get_contents('C:\xampp\htdocs/data.json');
    $decodedFile = json_decode($file, true);
    if ($id < 1 || $id > count($decodedFile)) {
        return response()->json([
            'message' => 'Invalid ID',
        ], 400);
    }
    unset($decodedFile[$id - 1]);
    file_put_contents('C:\xampp\htdocs/data.json',
     json_encode(array_values($decodedFile)));
    return response()->json([
        'message' => 'product has been deleted successfully'
    ]);
}
```

أخيراً نعدل ملف api.php وملف الـKernel.php كما فعلنا سابقاً:

```php
'edit_products' => \App\Http\Middleware\ProductsEdit::class,
```

```php
Route::delete('/delete/{id}',[MainController::class,'deleteByID'])
->middleware(['edit_products']);
```

تذكرة:

Authentication: تعني المصادقة وهي عملية تحقق من أن هذه البيانات تم إدخالها في فورم ما أو تسجيل الدخول لتطبيق ما هي لمستخدم موجود فعلاً (أي إثبات هوية الشخص).

Authorization: تعني التخويل أو الصلاحيات وهي تأتي في المرتبة الثانية بعد إتمام عملية تسجيل الدخول حيث في هذه العملية يتم إعطاء كل مستخدم صلاحيته والعمليات المسموحة والممكنة له.

Middleware : هو عبارة عن Class تتعامل مع الـ Request قبل وصولها إلى الـ Controller ، وهي المسؤولة عن تنفيذ عمليتي Authentication و Authorization الفائدة منها تعترض الطلبات وتقوم بتصفيتها في حال وجود مشكلات أمنية غير صالحة ، في حال كانت صالحة ترسل الطلب إلى ال Controller .

يمكن صنع Middleware عبر التعليمة:

```
php artisan make:middleware middlewarename
```

-The End-