الموضوع الأول

التوجيه Routing

❖ **ماذا نعني بالتوجيه Routing في لارافل ؟**

التوجيه Routing باختصار هو آلية تقرر ما سينفذ عند طلب URL معين .

ليتضح لك مفهوم Routing بشكل أكبر ، دعنا نستعرض الموقع التالي كمثال :

أمامك الآن الصفحة الرئيسية للموقع laravel.com ، سأقوم بطلب صفحة معينة على هذا الموقع و لكن دعنا قبل ذلك نشغل أدوات المطور لنعرف تفاصيل ما يحدث ، و لفتح أدوات المطور انقر بالزر الأيمن على المتصفح ثم اختر فحص ، فعل علامة التبويب الشبكة .

شاهد .. سأقوم طلب عنوان (URL) معين و ليكن **docs/5.5** ، النتيجة أنه قام بعرض صفحة معينة عندما قمت بطلب رابط معين ، فلنكرر العملية مع عنوان آخر و ليكن **docs/views** ، تم التوجيه لصفحة أخرى مختلفة ، هذا هو ما نعنيه بالتوجيه ، فـ باختصار هو استجابة لما نقوم بطلبه على المتصفح .

لاحظ هنا تظهر تفاصيل الطلب الذي قمنا به ، فعنوان الطلب هو **docs/5.5** و نوعه هو **GET** ، و قد تم الاستجابة بنجاح للطلب و أرسل الصفحة المطلوبة .

نظام التوجيه في لارافل يوفر طريقة لطلب الصفحات بطرق واضحة و سهلة القراءة ،

الشكل العام للمسار سيكون كالتالي:

```
Route::get ( الـ URL , ما سيتم تنفيذه )
```

يحول الطلب عادةً إلى function بداخل Controller معين ليقوم بتنفيذها و إرجاع النتيجة كالتالي :

```
Route::get('/page', 'controllername@function')
```

و يمكن أيضاً تضمين المهمة بداخل ملف الـRoutes مباشرة دون المرور على الـcontroller ، بالشكل التالي :

```
Route::get('/page', function () {
        return ' Welcome to Laravel Tutorial ' ;
});
```

لاحظ هنا أن المهمة مضمنة في نفس ملف الroute ، فعند طلب العنوان **page** سيعيد الجملة الموضحة أمامك .

يتم إنشاء المسارات بداخل المجلد Routes ، حيث أن مسارات موقع الويب تعرف في web.php ، أما مسارات الـ api فتكون بداخل api.php .

❖ **تطبيق عملي :**

سنضيف الآن عدد من التوجيهات على مشروع لارافل الذي قمنا بإنشاءه مسبقاً ، نشغل المشروع أولاً باستخدام الأمر php artisan serve .

فلنقم الآن بطلب الصفحة الرئيسية ، لاحظ أنه يعرض هذه الصفحة عند طلب عنوان الرئيسية، و هي عبارة عن view ، ملفات العرض الـview موجودة بداخل المجلد views الذي يحوي الملف **welcome** بشكل افتراضي ، و إذا لاحظت فستعرف أنها نفس الصفحة التي تعرض بعد تشغيل المشروع ، فكيف تم استدعاؤها و إظهارها ؟ ، لنعد لملف web.php و نرى

هذا السطر هو المسؤول عن طلب الصفحة السابقة ، فكما ترى ، عند طلب الصفحة الرئيسية و التي تمثل بالشرطة المائلة / ، يتم إعادة الـ view المسمى (**welcome**).

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

أو يمكن كتابة نفس الأمر السابق بطريقة أخرى أكثر بساطة

```
Route::view('/', 'welcome');
```

هذا السطر يعني قم بجلب الـview المسمى welcome عند طلب عنوان الرئيسية / .

الأمر جداً بسيط و واضح .

كذلك يمكنك تمرير مصفوفة من البيانات ، إلى العرض view ، بالطريقة التالية :

```
Route::view('/pass', 'welcome', ['key' => 'content']);
```

حيث **welcome** هو اسم الـview ، **key** رمز مفتاح في المصفوفة ، **content** القيمة (البيانات) ، و إذا قرأنا التعليمة من جهة اليمين ستكون مرر القيمة **content** إلى **الـview** المسمى **welcome** عند طلب العنوان **pass** .

الآن كيف سأظهر هذه المحتويات على الصفحة التي مررت لها المصفوفة ،

بالطبع سأقوم بكتابة أمر طباعة قيمة الـ$key على الصفحة ، و لعلك ستقوم مباشرة بكتابته بالطريقة التالية:

```
<?php  echo $key   ?>
```

بالتأكيد التعليمة السابقة سيعمل ، و لكن هناك طريقة أخرى أكثر بساطة ، باستخدام الأقواس المجعدة ، كالتالي:

```
{{ $key }}
```

كما ترى أصبح أكثر ترتيباً و أسهل للقراءة ، و هذا أحد المزايا التي يقدمها نظام الـblade (سنتحدث عنه في الجزء التالي من هذه المحاضرة).

❖ **الإشارة إلى View موجود بداخل مجلد :**

حسناً سأقوم الآن بإنشاء **view** آخر ، و لنسمه **about** ، و سأضعه بداخل **مجلد** بالاسم **front** ، و أريد استدعاء هذا الـ **view** عند طلب العنوان **about** .

سيكتب هذا بالطريقة التالية

```
Route::view(' about ','front.about');
```

بما أن العرض about هو بداخل مجلد front و المتفرع من **resources/views** ، إذاً يجب أن يذكر اسم المجلد أولاً ، فيكون **front.about**

فلنشاهد النتيجة على المتصفح ، لاحظ تم عرض الصفحة about .

❖ **استخدام المعلمات (Parameters) في التوجيه :**

في كثير من الأحيان نكون بحاجة إلى التقاط بعض الأجزاء على URL ، مثلاً رقم المستخدم أو اسمه ، لنلق نظرة على طريقة استخدام الـparameter :

```
Route::get('user/{id}', function ($id) {
    return 'User Id :'.$id;
});
```

التعليمة السابقة تعني أنه عند طلب العنوان user متبوعاً بأي قيمة كانت فأعد النص **User Id** مع قيمة البراميتر **الممرر** ، لاحظ أن ما يكون بين القوسين المجعدين هو عبارة قيمة متغيره .
و الآن على سبيل المثال قم بطلب العنوان **user**/3 و شاهد النتيجة على المتصفح .

❖ **تسمية الـ Route :**

يمكنك تعيين إسم لمسار Route معين ، بواسطة إضافة التعليمة name ثم الاسم الذي ترغبه ، كما يلي :

```
Route::get('user/profile', function () {
    Return view('user_profile');
})->name('profile');
```

و بهذه الطريقة يمكنك استخدام اسم المسار **فقط** لإعادة التوجيه إلى الـview المسمى **user_profile** عوضاً عن كتابة المسار بالكامل ، و أحد مزايا تسمية المسارات أنه إذا أردت تغيير العنوان URL مثلاً من **user/profile** إلى **member/profile** فسيلزمك تغييره في ملف المسارات فقط ، و لن تضطر إلى تعديل كل ما يشير إليه ، ستكون الإشارة لهذه المسارات بأحد الطرق التالية :

```
return redirect()->route('profile');

$url = route('profile');
```

❖ **طرق التوجيه الأخرى المتاحة :**

جميع الأمثلة التي ذكرناها كانت بنوع الطلب GET ، و هذا لا يعني أنها الطريقة الوحيدة المتاحة ، فالأنواع التالية أيضاً يمكن استخدامها حسب حالة الطلب .

```
Route::post($uri, $callback);

Route::put($uri, $callback);

Route::patch($uri, $callback);

Route::delete($uri, $callback);

Route::options($uri, $callback);
```

معرفة المزيد حول التوجيه (Routing) يمكنك زيارة الرابط التالي

https://laravel.com/docs/5.5/routing

الموضوع الثاني

العروض Views

في الجزء السابق من المحاضرة و الذي تحدثنا فيه عن المسارات ، تعاملنا أيضاً مع ملفات العرض **views** ، و سنعود الآن للتعرف عليها بشكل أكثر تفصيلاً :

❖ **الـ View :**

هي ملفات تحوي شيفرات html الخاصة بالتطبيق ، و هي الواجهة التي سيتعامل معها المستخدم النهائي ، توجد ملفات العرض بداخل الدليل `resources/views` ، و يتواجد ملف العرض **welcome** بشكل افتراضي ضمنه ، و هي تمثل الصفحة الرئيسية التي تظهر عند تشغيل تطبيق لارافل كما ذكرنا سابقاً.

❖ **طريقة إنشاء View جديد:**

لإنشاء view ، انقر بالزر الأيمن على المجلد Views ← ثم new file ← قم بتسمية الملف و احفظه بالامتداد blade.php .

ملفات العرض تنتهي بالإمتداد **blade.php** ، و هي نظام يقدم عدد من المزايا :

١ـ يمكنك كتابة كود php ضمن الـView بطريقة سهلة و مختصرة ، كما سترى في الأمثلة التالية :

عرض جملة على الصفحة :

بدلاً من كتابتها بالشكل المعتاد `<?php echo "hello world" ?>` **، استخدم الشكل التالي:**

```
{{"hello world"}}
```

أو

```
{!!"hello world"!!}
```

و الفرق بينهما أن الأولى تمنع هجمات XSS ، حيث تقوم بتمرير البيانات إلى الدالة **htmlspecialchars** قبل عرضها على المتصفح، بينما الثانية تتجاهل ذلك ، **إذاً** فاستخدامك للطريقة الأولى سيكون أكثر أماناً ، و خصوصاً في حالة كان المحتوى هو مدخلات من المستخدم ،

إذا لم يتضح لك الفرق ، فجرب الكود التالي :

```
@php
     $user_input = "<h1>content</h1>";
@endphp
```

و الآن جرب عرض محتوى **$user_input** ، مرة باستخدام **{{**$user_input **}}** و أخرى بـ **{!!** $user_input **!!}** ثم استنتج الفرق .. على افتراض أن **$user_input** هي مدخلات من المستخدم .

استخدام أكواد php ضمن الـview :

```
@php
  f_num=20;
  s_num=600;
  sum=f_num + s_num;
@endphp
```

طريقة استخدام حلقات التكرار ضمن الـview :

```
@for ($i = 0; $i < 10; $i++)
   The current value is {{ $i }}
@endfor

@foreach ($users as $user)
   <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
   <li>{{ $user->name }}</li>
@empty
   <p>No users</p>
```

```
@endforelse



@while (true)
  <p>I'm looping forever.</p>
@endwhile
```

❖ **نظام القوالب template engine:**

يوفر لارافيل نظام قوالب سهل و قوي ، تكمن ميزة استخدام القوالب في أنها تسهل من عملية التصميم ، حيث تمكنك من إعادة استخدام تصميم صفحة A ، في عدد لا نهائي من الصفحات B و C و D .. إلخ

فمثلاً لديك الصفحة master.blade.php ، المطلوب استخدام نفس الشكل للهيدر و الفوتر في ثلاث صفحات أخرى ، مع تغيير المحتوى فقط ، كل ما عليك فعله هو توريث الصفحة Master إلى بقية الصفحات

الخطوات التالية ستوضح الطريقة :

١- ينشىء الـview القالب ، و الذي يتضمن التصميم العام للصفحات .

٢- على الصفحة السابقة ، يُشار إلى مكان المحتوى (و الذي سيختلف من صفحة لصفحة) بالتعليمة **@yield** ، المكان المشار إليه بـ **@yield** سيخلى ليعبأ بالمحتوى من الصفحات **child** الوارثة.

٣- على الصفحة **الـchild** ، تستخدم التعليمة **@extends** لتعيين القالب الذي سيورث الى هذا **الـchild** ، و أيضاً سنحدد موضع المحتوى بالتعليمة **@section** .

سيتضح لك الأمر بالمثال التالي:

سأنشئ الآن القالب الرئيسي و الذي سيورث فيما بعد لعدة صفحات ، يفضل وضعه بداخل مجلد folder ، سأسمي المجلد master ، ننشيء الـ view القالب ، و سأعطيه الاسم app .

على الصفحة app سأضع شيفرة html ، فلنقم باستعراض الصفحة لنرى الشكل العام .

حسناً ، لدي هنا ثلاث صفحات **home about contact**، لكلٍ منها محتوى مختلف ، و ما نريده هو استخدام نفس التصميم الموجود في القالب السابق ، حسب الخطوات التي استعرضناها مسبقاً فإنه يجب الإشارة لموضع المحتوى في الصفحة القالب ، إذاً ننتقل للقالب الرئيسي app ، و نشير لجزء المحتوى بالتعليمة @yield ، هذا هو المكان الذي سيظهر فيه المحتوى و الذي سيختلف من صفحة لأخرى .

```
<div class="container-fluid text-center">
  <div class="row content">
    <div class="col-sm-12 text-left">
```

```
        @yield('content')


    </div>
  </div>
</div>
```

و الآن سننتقل للصفحة **home** و نورث لها القالب السابق

نكتب (**'master.app'**)@extends ، و بما أن الصفحة app هي بداخل مجلد ، إذاً لا بد من ذكر اسم المجلد أولاً كما رأيت.

و لا ننسى أن نضع المحتوى بين

@section('content')

@endsection

و الآن فلنشاهد النتيجة على المتصفح للصفحة home ، و التي قمنا بإعطائها نفس تصميم القالب app

و لا تنسى أولاً أن تعرف مساراً للصفحة **home** في ملف المسارات web.php .

الشيفرة النهائية للصفحة الـChild :

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


    @section('content')
      <p>This is just content.</p>
    @endsection
```

في حالة أردت إضافة أي view جديد بنفس تصميم القالب الرئيسي ، فما عليك سوى استخدام نفس الشيفرة السابقة مع تغيير المحتوى حسب الهدف من الصفحة .

❖ **إضافة title لكل صفحة :**

كما ترى فالوسم <title> موجود في القالب الرئيسي ، و في حالة وضعت عنواناً في القالب الرئيسي ، فسوف ترث جميع الصفحات نفس العنوان و هذا ما لا نريده ، الحل هو استخدام yield في جزء الـtitle بالشكل التالي :

```
<title>@yield('title')</title>
```

من المؤكد أنك تعرف الآن ما عليك إضافته في الصفحة الوارثة

```
@section('title','page title')
```

❖ **إنشاء تعليمات مخصصة Custom Directive :**

ميزة أخرى من مزايا Laravel Blade ، و هي إمكانية إنشاء تعليمات مخصصة ضمن الـviews .

Custom Directive هي توابع function تنفذ مهام معينة ، و تظهر ميزتها في إخفاء تفاصيل الكود و اختصارها ، ما يحسن من شكل الأسطر البرمجية و يجعلها أكثر قابلية للقراءة .

على سبيل المثال ما رأيك باستبدال

```
<?php
      echo "<center><u>welcome to our website </u></center>";
 ?>
```

بهذا فقط

```
@welcome
```

يتم الإعلان عن التعليمات المخصصة في الملف AppServiceProvider الموجود في الدليل App\Providers .

❖ **الخطوات لإنشاء Directive مخصص :**

انتقل للملف AppServiceProvider ، و بداخل التابع boot() قم بتعريف التعليمة خاصتك كالتالي :

```
Blade::directive('welcome', function () {
return "<center><u>welcome to our website </u></center>";
});
```

يحدد اسم التعليمة ، ثم المهمة التي ستقوم بها عند استدعائها .

لا تنسى أن تعرف الواجهة Blade في أعلى الملف

```
use Blade;
```

و الآن يمكننا استخدام التعليمة welcome ضمن الـview هكذا **@welcome** .

يمكنك أيضاً تمرير قيم بهذا الشكل **@welcome**('student')

ليتم استخدامها بداخل التابع

```
Blade::directive('welcome', function ($name) {
    return "<center><u>welcome to our website
".$name."</u></center>";
 });
```

❖ **Directive 'IF' :**

من المزايا الجديدة في الإصدار 5.5 من لارافيل إمكانية إنشاء جمل **شرط** مخصصة باستخدام الجملة **Blade::if.**

بالنظر بداخل ملف welcome الموجود ضمن الدليل views ، ستلاحظ وجود هذه التعليمة

```
  @auth

  @else

  @endauth
```

هذه التعليمة هي في الأصل

```
<?php

     if(auth()->guard()->check()):

     else:

     endif;
?>
```

إنما تم تعريف التعليمة @auth لتقوم بنفس المهمة ، و عمل الكود السابق هو التحقق من كون المستخدم قد سجل دخوله أم لا .

مثال :

لنقم بالإعلان عن التعليمة admin@ ليتحقق من أن المستخدم النشط هو admin أم لا

ستعرف التعليمة في ملف AppServiceProvider ، بشكل شبيه للتالي :

```
   Blade::if('admin', function () {
        return auth()->check() && auth()->user()->isAdmin();
    });
```

السطر السابق يعني أنه في حالة تحقق الشرط هنا

`auth()->check() && auth()->user()->isAdmin()` سيتم إرجاع true ، مالم سيعيد false

و الآن التعليمة admin جاهزة لاستخدامها في الـview بالشكل:

```
@admin
    {{"user is admin"}}
@else
    {{"user not admin"}}
@endadmin
```

(بعد كل عملية تعديل على ملفات الـblade يتم تجميعها إلى شكلها الأصلي و حفظها في الدليل Storage\framework\views كتخزين مؤقت)

يمكنك حذفها يدوياً أو استخدام الأمر php artisan view:clear

و كما رأيت فإن استخدامك للتعليمات المخصصة في مشروعك سيحسن من الشيفرة البرمجية خاصتك بشكل كبير و يجعلها أسهل للقراءة و التعديل .

هناك العديد من المزايا الأخرى التي يقدمها نظام الـblade في لارافل ، يمكنك التعرف عليها من خلال زيارة الرابط التالي

https://laravel.com/docs/5.5/blade