Projet Laravel : Produit

Rapport du Projet : Développer une application CRUD

Toutes les fichiers source sont disponibles sur le dépôt GitHub associé à l'article.

Par : MAZTAOUI ILIAS

Lien GitHub :

https://github.com/ilias-coder/Produit

Sommaire :

Remerciement :

Je tiens à vous remercier pour votre initiative pour nous donner un projet par lequel j'ai pu personnellement maitriser plusieurs notions et leurs nécéssités (OOP, MVC, Laravel…).

J'espère que ce travail a été à la hauteur de vos prédictions.

Travail réalisé par : Maztaoui Ilias

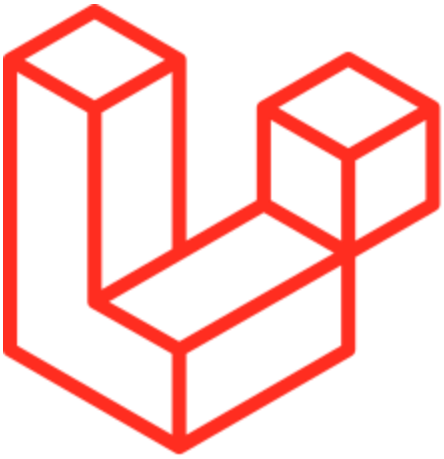
Cordialement , Portez-vous bien.

Présentation d’application :

L’application du CRUD, que j’ai développé pendant ce projet s’agit d’un utilisateur qui peut ajouter des produits (nom, description, prix, date d’ajout de ce produit) et donner son avis là-dessus, sur le produit qu’il vient de tester et qu’il ajoute son commentaire sur le produit, tout en lui gardant la possibilité de supprimer un produit qu’on vient d’ajouter, ou de modifier quelques données qui étaient ajoutées là-dessus.

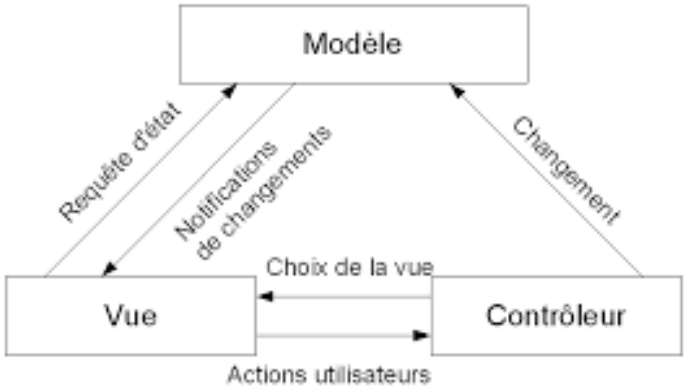
Tout d’abord, la première question qui se pose :

C’est quoi Laravel ?



Laravel est un framework web open-source écrit en PHP1 respectant le principe **MVC (modèle-vue-contrôleur)** et entièrement développé en programmation orientée objet. Laravel est distribué sous licence MIT, avec ses sources hébergées sur GitHub.

Mais un MVC, c'est quoi plus précisement ?



Modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques lancé en 1978 et très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : les modèles, les vues et les contrôleurs.

Un **modèle (Model)** contient les données à afficher.

Une **vue (View)** contient la présentation de l'interface graphique.

Un **contrôleur (Controller)** contient la logique concernant les actions effectuées par l'utilisateur.

Ce motif est utilisé par de nombreux frameworks pour applications web tels que **Laravel**, Ruby on Rails, Grails, ASP.NET MVC, Spring, Struts, Symfony, Apache Tapestry, ou AngularJS.

En général on résume en disant que le modèle gère la base de données, la vue produit les pages HTML et le contrôleur fait tout le reste. Dans Laravel :

- le modèle correspond à une table d'une base de données. C'est une classe qui étend la classe Model qui permet une gestion simple et efficace des manipulations de données et l'établissement automatisé de relations entre tables.
- le contrôleur se décline en plusieurs catégories : contrôleur classique, contrôleur RESTfull et contrôleur de ressource (je détaillerai évidemment tout ça dans le cours).
- la vue est soit un simple fichier avec du code HTML, soit un fichier utilisant le système de template Blade de Laravel.

Laravel propose ce modèle mais ne l'impose pas. Nous verrons d'ailleurs qu'il est parfois judicieux de s'en éloigner parce qu'il y a des tas de chose qu'on arrive pas à caser dans ce modèle. Par exemple si je dois envoyer des emails où vais-je placer mon code ? En général ce qui se produit est l'inflation des contrôleurs auxquels on demande des choses pour lesquelles ils ne sont pas faits.

Pourquoi Laravel :

- **Constitution de Laravel**

Laravel, créé par Taylor Otwel, initie une nouvelle façon de concevoir un framework en utilisant ce qui existe de mieux pour chaque fonctionnalité. Par exemple toute application web a besoin d'un système qui gère les requêtes HTTP. Plutôt que de réinventer quelque chose le concepteur de Laravel a tout simplement utilisé celui de **Symfony** en l'étendant pour créer un système de routage efficace. De la même manière l'envoi des emails se fait avec la bibliothèque **SwiftMailer**. En quelque sorte Otwel a fait son marché parmi toutes les bibliothèques disponibles. Nous verrons dans ce cours comment cela est réalisé. Mais Laravel ce n'est pas seulement le regroupement de bibliothèques existantes, c'est aussi de nombreux composants originaux et surtout une orchestration de tout ça.

Vous allez trouver dans Laravel :

- un système de routage perfectionné (RESTFul et ressources),
- un créateur de requêtes SQL et un ORM performants,
- un moteur de template efficace,
- un système d'authentification pour les connexions,
- un système de validation,
- un système de pagination,
- un système de migration pour les bases de données,
- un système d'envoi d'emails,
- un système de cache,
- une gestion des sessions...

Et bien d'autres choses encore que nous allons découvrir ensemble. Il est probable que certains éléments de cette liste ne vous évoque pas grand chose, mais ce n'est pas important pour le moment, tout cela deviendra plus clair au fil des chapitres.

- **Utilisation du meilleur du php :**

Plonger dans le code de Laravel c'est recevoir un cours de programmation tant le style est clair et élégant et le code merveilleusement organisé. La version actuelle de Laravel est la 5.1, elle nécessite au minimum la version 5.5.9 de PHP. Pour aborder de façon efficace ce framework il serait souhaitable que vous soyez familiarisé avec ces notions :

- **Les espaces de noms** : c'est une façon de bien ranger le code pour éviter des conflits de nommage. Laravel utilise cette possibilité de façon intensive. Tous les composants sont rangés dans des espaces de noms distincts, de même que l'application créée.
- **Les fonctions anonymes** : ce sont des fonctions sans nom (souvent appelées closures) qui permettent d'améliorer le code. Les utilisateurs de Javascript y sont habitués. Les utilisateurs de PHP un peu moins parce qu'elle y sont plus récentes. Laravel les utilise aussi de façon systématique.
- **Les méthodes magiques** : ce sont des méthodes qui n'ont pas été explicitement décrites dans une classe mais qui peuvent être appelées et résolues.
- **Les interfaces** : une interface est un contrat de constitution des classes. En programmation objet c'est le sommet de la hiérarchie. Tous les composants de Laravel sont fondés sur des interfaces. La version 5 a même vu appraraître un lot de contrats pour étendre de façon sereine le framework.
- **Les traits** : c'est une façon d'ajouter des propriétés et méthodes à une classe sans passer par l'héritage, ce qui permet de passer outre certaines limitation de l'héritage simple proposé par défaut par PHP.

➢ **Utilisation du POO (Programmation orienté objet) :**

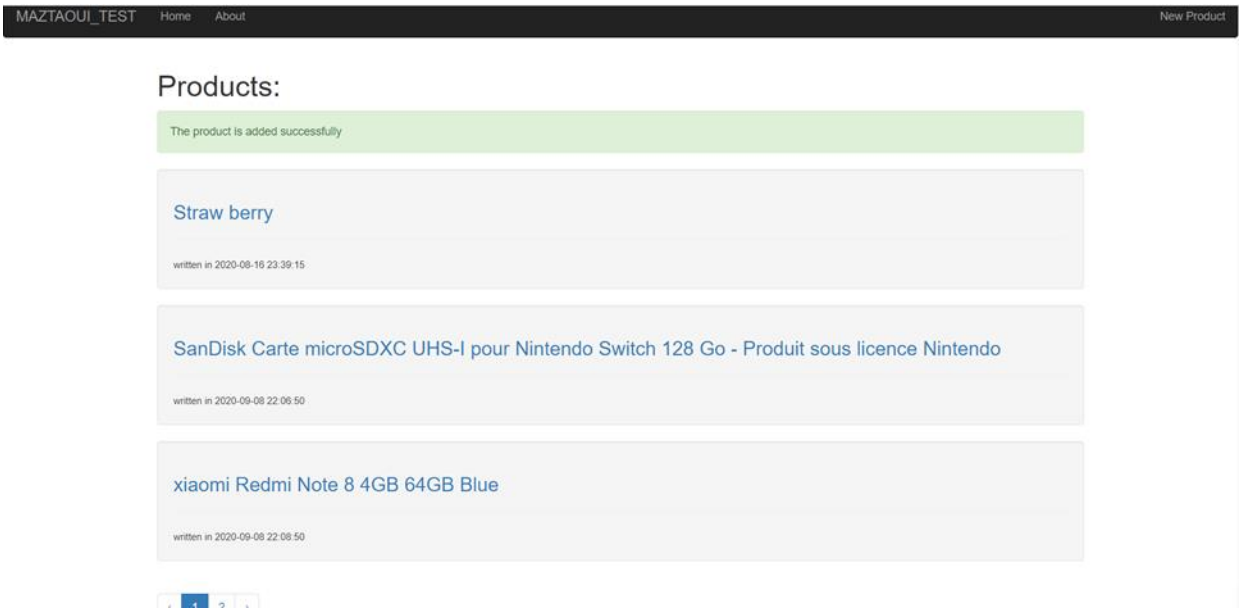
Laravel est fondamentalement orienté objet. La POO est un design pattern qui s'éloigne radicalement de la programmation procédurale. Avec la POO tout le code est placé dans des classes qui découlent d'interfaces qui établissent des contrats de fonctionnement. Avec la POO on manipule des objets.

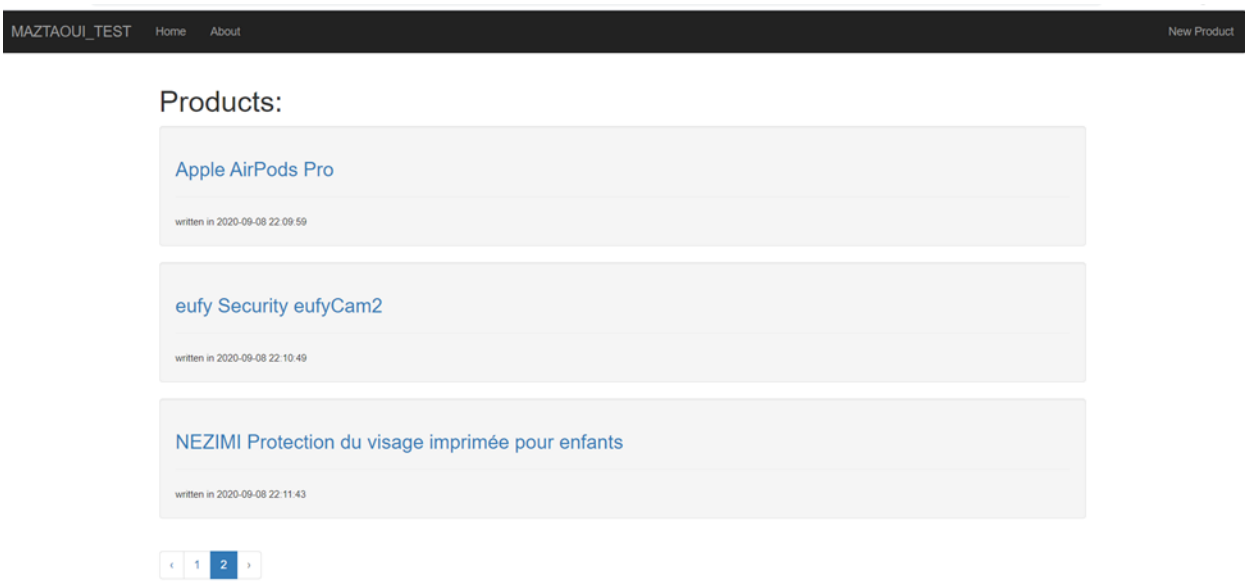
Avec la POO la responsabilité du fonctionnement est répartie dans des classes alors que dans l'approche procédurale tout est mélangé. Le fait de répartir la responsabilité évite la duplication du code qui est le lot presque forcé de la programmation procédurale. Laravel pousse au maximum cette répartition en utilisant l'injection de dépendance.

L'utilisation de classes bien identifiées, dont chacune a une rôle précis, pilotées par des interfaces claires, dopées par l'injection de dépendances : tout cela crée un code élégant, efficace, lisible, facile à maintenir et à tester. C'est ce que Laravel propose. Alors vous pouvez évidemment greffer là dessus votre code approximatif, mais vous pouvez aussi vous inspirer des sources du framework pour améliorer votre style de programmation.

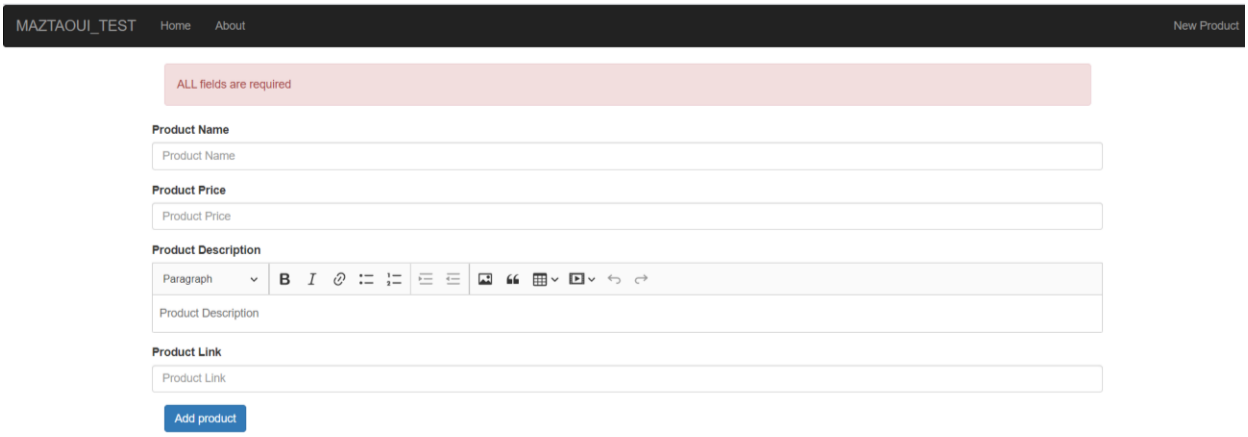
Présentation du fonctionnement de l’application :

Voyons tout d’abord le résultat du code développé :

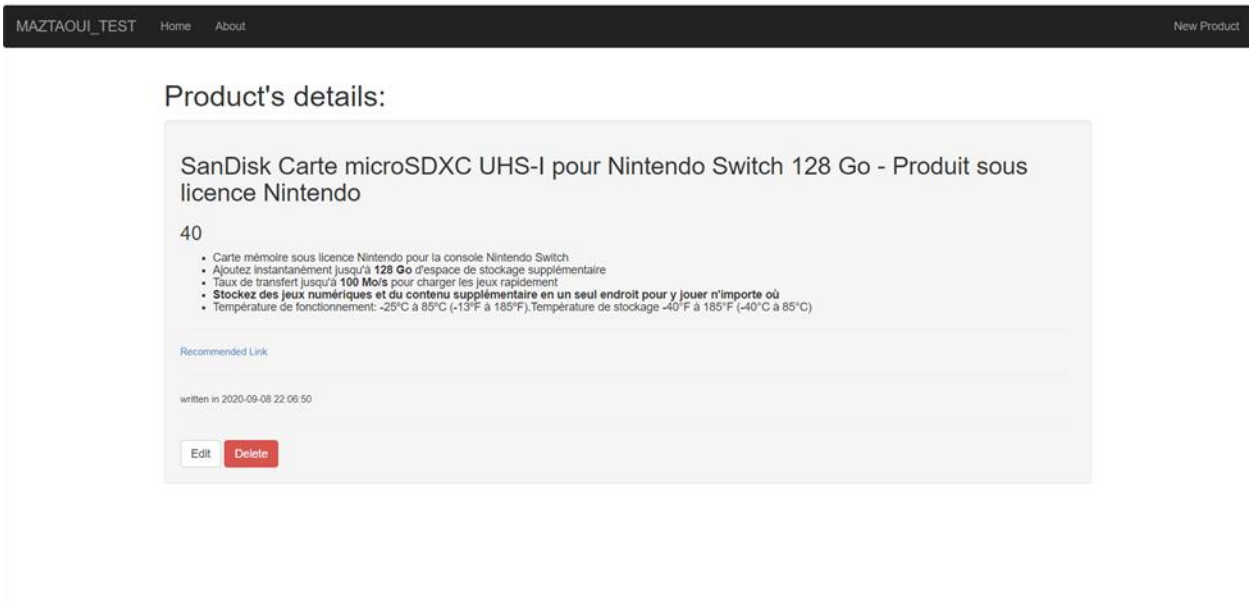




Quand on clique sur New Product :



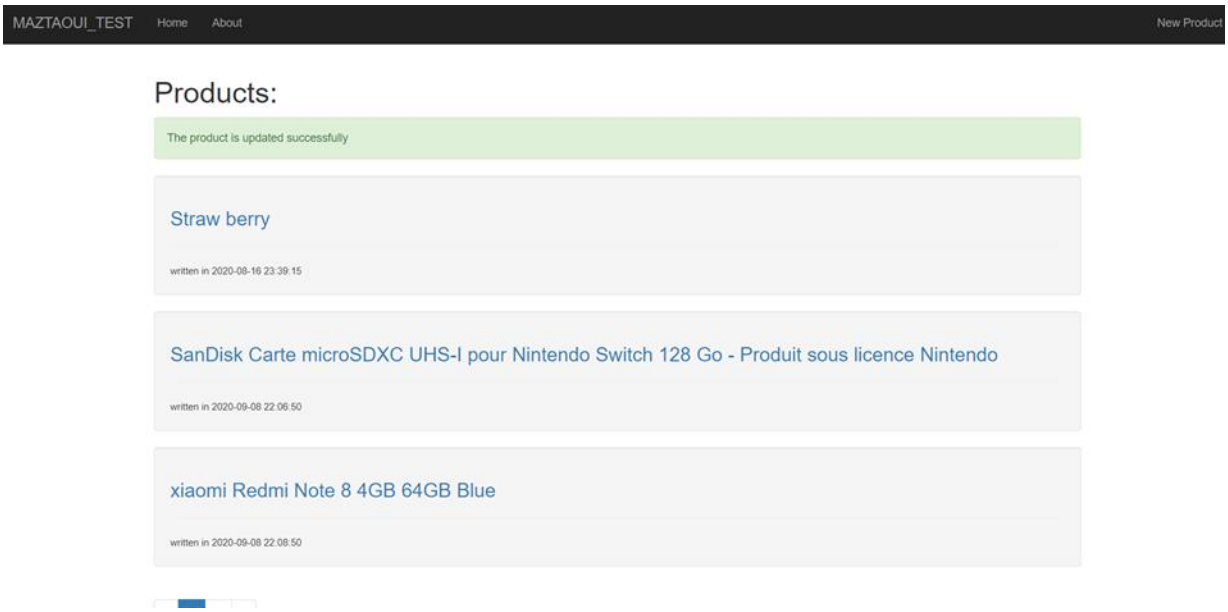
En revenant à la page officiel, et en cliquant sur le nom d'un produit quelconque:



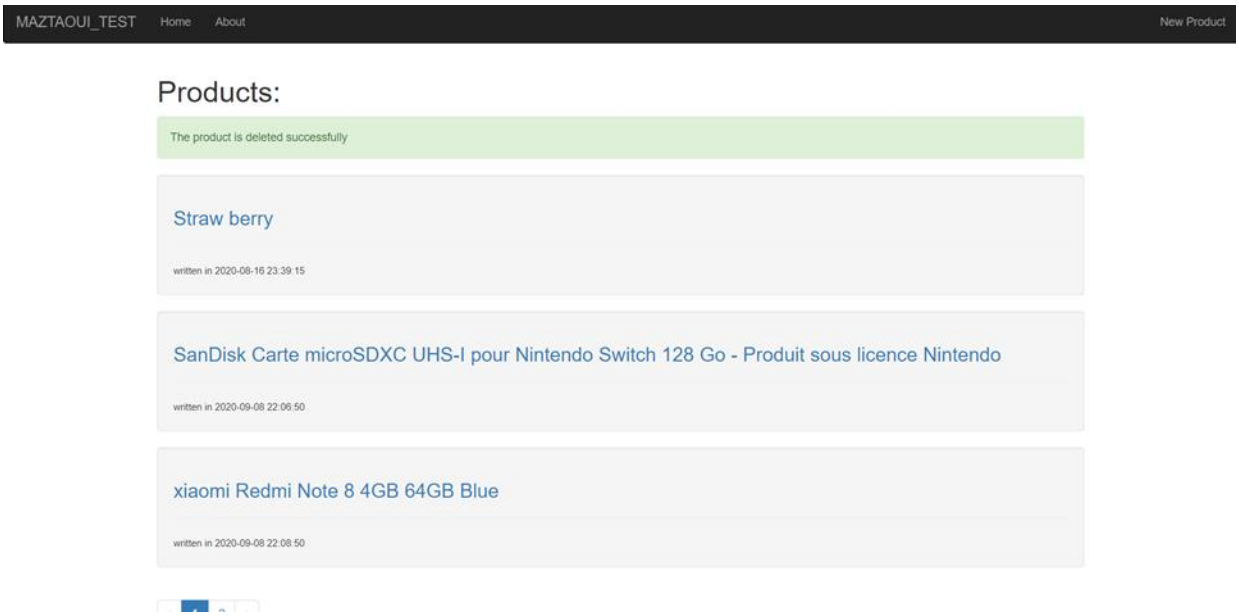
Si on clique sur Edit, on modifiera le contenu affiché aux clients :

Voilà ce qui se passe en cliquant là-dessus :

En cliquant sur update product, le produit est modifié avec un message en page d'accueil que le produit a été bien modifié:



Si l'opération était delete :



Maintenant, Il nous reste d'expliquer :

- Home : càd, page d'accueil où les produits sont affichés
- About : détail le projet.

De la part de Ilias Maztaoui

L'application du CRUD, que j'ai développé pendant ce projet s'agit d'un utilisateur qui peut ajouter des produits (nom, description, prix, date d'ajout de ce produit) et donner son avis là-dessus, sur le produit qu'il vient de tester et qu'il ajoute son commentaire sur le produit, tout en lui gardant la possibilité de supprimer un produit qu'on vient d'ajouter, ou de modifier quelques données qui étaient ajoutées là-dessus.

Explication du fonctionnement du code :

Tout d'abord, on commence notre projet par le dossier Produit/ressources/routes/web.php

Où on a le contenu suivant :

```php
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/','PagesController@index');
Route::get('/about','PagesController@about');
Route::get('/services','PagesController@services');
Route::get('/show/{id}','Pagescontroller@show');
route::get('/create','Pagescontroller@create');
route::post('/saveproduct','Pagescontroller@saveproduct');
route::get('/edit/{id}','PagesController@editproduct');
route::post('/update','Pagescontroller@updateproduct');
route::get('/delete/{id}','PagesController@deleteproduct');
```

On remarque qu'il y a la premier route que laravel accède quand elle se connecte au dossier public, c'est la basic route d'application

```php
Route::get('/','PagesController@index');
```

- ✓ Route pour utiliser la classe route ;
- ✓ get est souvent utiliser pour visualiser et présenter un contenu ;
- ✓ PagesController Pour nommer le contrôleur et la classe où la prochaine requête se situe.
- ✓ Index : la méthode qu'on veut appliquer à ce moment-là.

Routes statiques :

```
Route::get('/about','PagesController@about');
```

Cette ligne nous dirigera vers la méthode about car on a tapé la route basique en plus on ajouté (about).

```
route::post('/update','Pagescontroller@updateproduct');
```

Cette ligne nous dirigera vers la méthode updateproduct car on a tapé la route basique en plus on ajouté (update).

Mais ce qui différent ici, c'est qu'on a utilisé **post** qui est souvent utilisé pour changer et envoyer des données en formulaire…

Routes dynamiques:

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

Ici on veut voir les détails d'un produit dont son id change avec le changement du produit ; donc on ne pourra pas créer de multiples route statiques ; donc on crée une qui est différenciable avec le numéro d'id de chaque produit car il est unique.

Et, c'est la même chose pour les autres routes.

Maintenant, on se déplacera vers app/http/controllers/PagesController.php dont le contenu est :

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use DB;
use App\product;
use Session;

class PagesController extends Controller
{
    //

        public function index()
    {
        $products = Product::paginate(3);
        //$products = DB::table('products')->get();
        return view('pages.index')->with('products',$products);
    }

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


    public function show($id)
    {
        $product=Product::find($id);

        //$product=DB::table('products')->where('id',$id)->first();
        return view('pages.show')->with('product',$product);

    }

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

    public function saveproduct(request $request)
    {
```

```
        /*$product = new Product();
        $product->product_name=$request->product_name;
        $product->product_price=$request->product_price;
        $product->product_description=$request->product_description;
        $product->save();
        */
        if (!$request->product_name || $request->product_price ||$request-
>product_description || $request->Link)
        {
            Session::put('error','ALL fields are required');
        }
        $this->validate($request,['product_name' => 'required',
                                  'product_price' => 'required',
                                      'product_description' => 'required',
                                          'Link' => 'required']);

        $data = array();
        $data['product_name']=$request->product_name;
        $data['product_price']=$request->product_price;
        $data['product_description']=$request->product_description;
        $data['Link']=$request->Link;
        $data['created_at']=now();

        DB::table('products')->insert($data);
        Session::put('success','The product is added successfully');
        return redirect('/');

    }

    public function editproduct($id)
    {
        $product=Product::find($id);
        return view('pages.edit')->with('product',$product);

    }

    public function updateproduct(request $request)
    {
        /*
        $product=Product::find($request->id);

        $product->product_name=$request->input('product_name');
        $product->product_price=$request->input('product_price');
        $product->product_description=$request->input('product_description');
```

```
        $product->update();
        */

        if (!$request->product_name || $request->product_price ||$request-
>product_description||$request->Link  )
        {
            Session::put('error','ALL fields are required');
        }
        $this->validate($request,['product_name' => 'required',
                                  'product_price' => 'required',
                                      'product_description' => 'required',
                                          'Link' => 'required']);

        $data=array();
        $data['product_name']=$request->product_name;
        $data['product_price']=$request->product_price;
        $data['product_description']=$request->product_description;
        $data['Link']=$request->Link;

        DB::table('products')->where('id',$request->id)
                             ->update($data);

        Session::put('success','The product is updated successfully');
        return redirect('/');
    }

    public function deleteproduct($id)
    {
        $product=Product::find($id);

        $product->delete();

        Session::put('success','The product is deleted successfully');
        return redirect('/');
    }
}
```

On crée une classe avec les méthodes dont on aura besoin :

```
public function index()
    {
```

```
        $products = Product::paginate(3);
        //$products = DB::table('products')->get();
        return view('pages.index')->with('products',$products);
    }
```

Cette méthode est la méthode qui utilise la classe Product qui a relation à nos produits qui se situe à la base de données :

Product.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    //
}
```

Paginate pour limiter le nombre de produits affichés en page en 3

Le message en commentaire, on pouvait l'utiliser au lieu de la ligne qui le précede ; et qui utilise la classe DB qui interroge la table products et faisant un appel (get).

La dernière ligne pour se diriger vers le dossier pages et plus précisément le **fichier index.blade.php** (avec les produits qui sont stockés en base de données) et dont le contenu est :

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

@section('title')
Home
@endsection

<?php
    //$products=DB::table('products')->get();
?>
```

```
@section('content')
    <div class="container">
        <h1>Products: </h1>
        @if (Session::has('success'))
        <p class="alert alert-success">{{Session::get('success')}}
                                    {{Session::put('success',null)}}</p>

        @endif
       @foreach ($products as $product)
       <div class="well">
       <h3><a href="/Produit/public/show/{{$product->id}}">{{$product-
>product_name}}</a></h3>
       <hr>
       <small>written in {{$product->created_at}}</small>
       </div>

       @endforeach
       {{$products->links()}}

  @endsection
```

Cette ligne :

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

Elle est utilisée pour pourvoir utiliser le dossier layout et le fichier **app.blade.php** plus précisément. Ce fichier nous permet d'éviter le code qui se répète.

```
<!DOCTYPE html>

<html lang="en">
<head>
  <title>@yield('title')</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/cs
s/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.j
s"></script>
</head>
<body>

@include('navbar.nav')

@yield('content')

<script src="https://cdn.ckeditor.com/ckeditor5/16.0.0/classic/ckeditor.js"></scr
ipt>
<script>
  ClassicEditor
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
          console.error( error );
      } );
</script>

</body>
</html>
```

Maintenant,

```
  @yield('title')
```

Pour déterminer l'emplacement du contenu qui se répète en plusieurs autres fichiers (create,add, index…)

```
<html lang="en">
<head>
  <title>@yield('title')</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

```
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/cs
s/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.j
s"></script>
</head>
<body>
```

Pour afficher le design de notre site web ;

```
@include('navbar.nav')
```

Inclure un autre template de la barre où il y en a home, about, add product et le nom du site.

```
<nav class="navbar navbar-inverse">
    <div class="container-fluid">
      <div class="navbar-header">
        <a class="navbar-brand" href="#">MAZTAOUI_TEST</a>
      </div>
      <ul class="nav navbar-nav">
          <li><a href="{{URL::to('/')}}">Home</a></li>
          <li><a href="{{URL::to('/about')}}">About</a></li>
      </ul>
      <ul class="nav navbar-nav navbar-right">
        <li><a href="/Produit/public/create">New Product</a></li>
      </ul>
    </div>
  </nav>
```

```
          <li><a href="{{URL::to('/')}}">Home</a></li>
```

En cliquant sur Home, on se dirige vers la route de la page d’accueil.

```
<script src="https://cdn.ckeditor.com/ckeditor5/16.0.0/classic/ckeditor.js"></scr
ipt>
<script>
  ClassicEditor
```

```
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
          console.error( error );
      } );
</script>
```

Pour pouvoir ajouter du style à la description du produit (ajouter une barre par laquelle on peut rendre le texte en gras italien ….);

Maintenant, revenons à notre vue index.blade.php:

```
@section('title')
Home
@endsection
```

Ces lignes permettent de déterminer l'emplacement du template qu'on a écrit avant avec le nom qu'on avait choisi (title).

```
@if (Session::has('success'))
        <p class="alert alert-success">{{Session::get('success')}}
                                      {{Session::put('success',null)}}</p>

        @endif
```

Pour la gestion des messages de la réussite de la suppression, modification…

```
@foreach ($products as $product)
       <div class="well">
       <h3><a href="/Produit/public/show/{{$product->id}}">{{$product-
>product_name}}</a></h3>
       <hr>
       <small>written in {{$product->created_at}}</small>
       </div>

       @endforeach
```

On créera une boucle tout en déterminant les informations qu'on veut afficher à partir de la base de données.

```
       <h3><a href="/Produit/public/show/{{$product->id}}">{{$product-
>product_name}}</a></h3>
```

En cliquant sur ce produit(nom du produit), on se dirigera vers la route /Produit/public/show/ (id du produit)

```
{{$products->links()}}
```

Cette ligne pour pouvoir afficher en utilisant paginate que j'ai expliqu » auparavant en rapport.

En se dirigeant vers cette route, on se redirige automatiquement vers le contrôleur PagesController et plus précisément la méthode show :

```
public function show($id)
    {
        $product=Product::find($id);

        //$product=DB::table('products')->where('id',$id)->first();
        return view('pages.show')->with('product',$product);

      //print('the product id is:' .$id);
    }
```

Maintenant, on essaiera de trouver le produit avec l'id donné par (find)

Puis on retourne la vue show.blade.php avec le produit trouvé.

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

@section('title')
    Products infos
@endsection

@section('content')
<div class="container">
    <h1>Product's details:</h3>
    <div class="well">
        <h2>{{$product->product_name}}</h2>
        <h3>{{$product->product_price}} Dhs</h3>
        <h5>{!!$product->product_description!!}</h5>
        <hr>
        <h6><a href="{{$product->Link}}">Recommended Link</a></h6>
        <hr>
        <small>written in {{$product->created_at}}</small>
        <hr>
        <a href="/Produit/public/edit/{{$product->id}}" class="btn btn-
default">Edit</a>
```

```
        <a href="/Produit/public/delete/{{$product->id}}" class="btn btn-
danger">Delete</a>
    </div>

</div>
@endsection
```

Les deux écritures ci-dessous sont différents la première permet à laravel de prendre en considération le code html généré dedans par contre la deuxième non!

```
        <h3>{!!$product->product_description!!}</h3>
```

```
        <h2>{{$product->product_name}}</h2>
```

En cliquant sur edit:

On active :

```
public function editproduct($id)
    {
        $product=Product::find($id);
        return view('pages.edit')->with('product',$product);

    }
```

Qui nous redirige vers edit.blade.php :

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

@section('title')
 Create
@endsection

<?php
    //$products=DB::table('products')->get();
?>

@section('content')
    <div class="container">

        @if (Session::has('error'))
```

```
        <p class="alert alert-danger">{{Session::get('error')}}
                                    {{Session::put('error',null)}}</p>

        @endif

        {!!Form::open(['action' => 'Pagescontroller@updateproduct','method' => 'P
OST', 'class' => 'form-horizontal'])!!}
        {{ csrf_field() }}

        {{Form::hidden('id', $product->id )}}

        <div class="form-group">
            {{Form::label('','Product Name')}}
            {{Form::text('product_name', $product-
>product_name, ['placeholder' => 'Product Name ', 'class' =>'form-control'])}}
        </div>

        <div class="form-group">
            {{Form::label('','Product Price')}}
            {{Form::number('product_price', $product-
>product_price, ['placeholder' => 'Product Price ', 'class' =>'form-control'])}}
        </div>

        <div class="form-group">
            {{Form::label('','Product Description')}}
            {{Form::textarea('product_description', $product-
>product_description, ['id'=>'editor','placeholder' => 'Product Description ', 'c
lass' =>'form-control'])}}
        </div>

        <div class="form-group">

            {{Form::label('','Product Link')}}
            {{Form::text('Link',$product-
>Link, ['placeholder' => 'Product Link ', 'class' =>'form-control'])}}
        </div>

        {{Form::submit('Update product', ['class' => 'btn btn-primary'])}}
        {!!Form::close()!!}

    </div>
  @endsection
```

J’ai écrit ce code en utilisant un formulaire par laravel collective ;

```
Pour active le champ où on veut le script de la gestion du texte'id'=>'editor'
```

Pour la gestion des erreurs, si un utilisateur n'a pas rempli les champs de formulaire que j'ai choisi comme obligatoire, j'ai appliqué :

```
@if (Session::has('error'))
        <p class="alert alert-danger">{{Session::get('error')}}
                                      {{Session::put('error',null)}}</p>

        @endif
```

Qui est contrôlé par en cliquant sur update :

```
if (!$request->product_name || $request->product_price ||$request-
>product_description||$request->Link  )
        {
            Session::put('error','ALL fields are required');
        }
        $this->validate($request,['product_name' => 'required',
                                  'product_price' => 'required',
                                     'product_description' => 'required',
                                        'Link' => 'required']);
```

Puis si tout se passe bien :

```
$data=array();
        $data['product_name']=$request->product_name;
        $data['product_price']=$request->product_price;
        $data['product_description']=$request->product_description;
        $data['Link']=$request->Link;

        DB::table('products')->where('id',$request->id)
                             ->update($data);

        Session::put('success','The product is updated successfully');
        return redirect('/');
```

On crée un tableau on on stockera les éléments dont on a besoin avec un message à la page d'accueil `'The product is updated successfully'`;

De la même manière on crée un produit, en cliquant sur new product en page d'accueil :

```
public function create()
```

```
    {
        return view('pages.create');
    }
```

Vers la vue create.blade.php :

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

@section('title')
 Create
@endsection

<?php
    //$products=DB::table('products')->get();
?>

@section('content')
    <div class="container">
        @if (Session::has('error'))
        <p class="alert alert-danger">{{Session::get('error')}}
                                      {{Session::put('error',null)}}</p>

        @endif

        {!!Form::open(['action' => 'Pagescontroller@saveproduct','method' => 'POS
T', 'class' => 'form-horizontal'])!!}
        {{ csrf_field() }}

        <div class="form-group">
            {{Form::label('','Product Name')}}
            {{Form::text('product_name','', ['placeholder' => 'Product Name ', 'c
lass' =>'form-control'])}}
        </div>

        <div class="form-group">
            {{Form::label('','Product Price')}}
            {{Form::number('product_price','', ['placeholder' => 'Product Price '
, 'class' =>'form-control'])}}
        </div>

        <div class="form-group">
            {{Form::label('','Product Description')}}
            {{Form::textarea('product_description','', ['id'=>'editor','placehold
er' => 'Product Description ', 'class' =>'form-control'])}}
```

```
        </div>

        <div class="form-group">
            {{Form::label('','Product Link')}}
            {{Form::text('Link','', ['placeholder' => 'Product Link ', 'class' =>
'form-control'])}}
        </div>

        {{Form::submit('Add product', ['class' => 'btn btn-primary'])}}
        {!!Form::close()!!}

    </div>
  @endsection
```

Vers le contrôleur pour sauvegarder les données en base de données :

```
public function saveproduct(request $request)
    {
        /*$product = new Product();
        $product->product_name=$request->product_name;
        $product->product_price=$request->product_price;
        $product->product_description=$request->product_description;
        $product->save();
        */
        if (!$request->product_name || $request->product_price ||$request-
>product_description || $request->Link)
        {
            Session::put('error','ALL fields are required');
        }
        $this->validate($request,['product_name' => 'required',
                                  'product_price' => 'required',
                                      'product_description' => 'required',
                                          'Link' => 'required']);


        $data = array();
        $data['product_name']=$request->product_name;
        $data['product_price']=$request->product_price;
        $data['product_description']=$request->product_description;
        $data['Link']=$request->Link;
        $data['created_at']=now();

        DB::table('products')->insert($data);
        Session::put('success','The product is added successfully');
        return redirect('/');
```

```
    }
```

On peut soit créer un tableau, soit faire ce qui est en commentaire.

Et finalement delete de la même manière

```
public function deleteproduct($id)
    {
        $product=Product::find($id);

        $product->delete();

        Session::put('success','The product is deleted successfully');
        return redirect('/');
    }
```

Finalement, c’est ça le résultat de notre application CRUD

Conclusion :

Finalement, J’ai aimé le pouvoir de laravel à simplifier le travail de PHP, orienté objet ; j’ai pu maitriser le Framework MVC par ce travail en améliorant mes connaissances sur le rôle de chacun (MVC).

Pour cela, j’ai tenté de faire une autre application CRUD en utilisant Resource et les log middlewares un peu, mais j’ai toujours des erreurs, j’essaierai de faire un rapport rapide et vous l’envoyer le plus tôt possible.