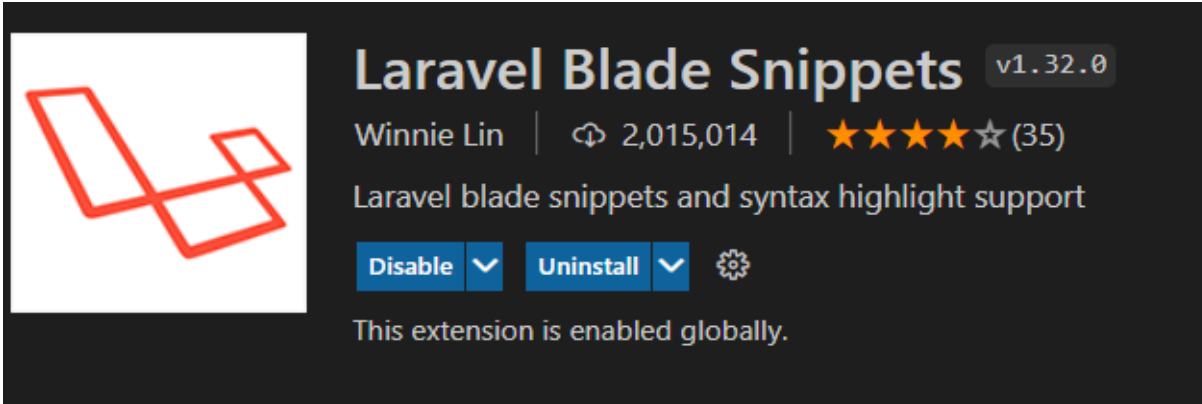


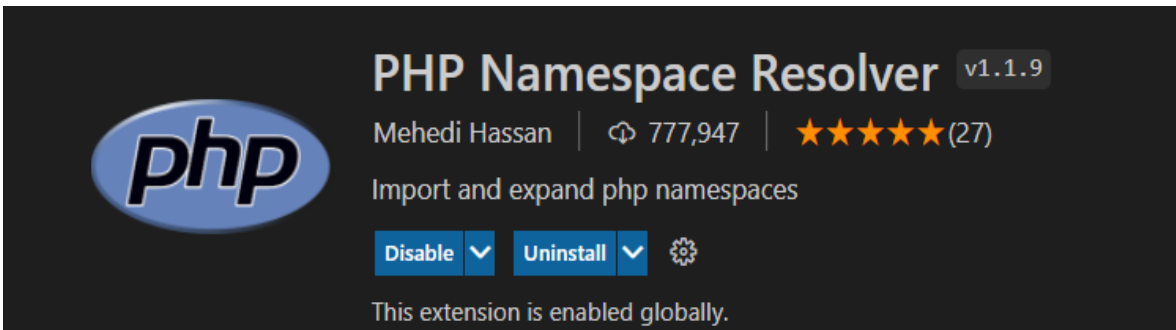
TP1 : Gestion des produits

Partie II :

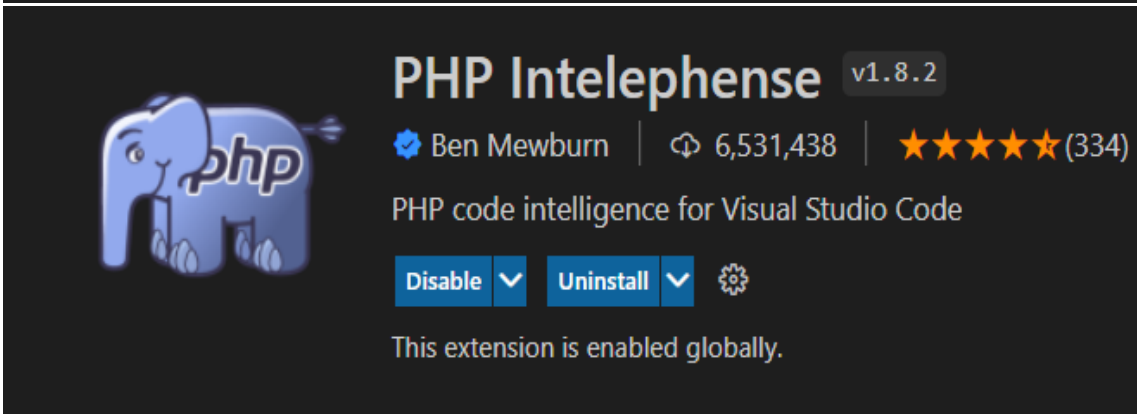
1. Quelques extensions utiles de vs code:



Laravel Blade Snippets v1.32.0
Winnie Lin | 2,015,014 | ★★★★★ (35)
Laravel blade snippets and syntax highlight support
Disable Uninstall ⚙️
This extension is enabled globally.



PHP Namespace Resolver v1.1.9
Mehedi Hassan | 777,947 | ★★★★★ (27)
Import and expand php namespaces
Disable Uninstall ⚙️
This extension is enabled globally.



PHP Intelephense v1.8.2
Ben Mewburn | 6,531,438 | ★★★★★ (334)
PHP code intelligence for Visual Studio Code
Disable Uninstall ⚙️
This extension is enabled globally.

1. Installer l'outil de gestion de dépendances composer <https://getcomposer.org/>:

2. Créer un projet laravel en utilisant composer:

```
>composer create-project laravel/laravel nom_projet
```

Ou bien, vous pouvez créer de nouveaux projets Laravel en installant globalement le programme d'installation de Laravel via Composer :

```
>composer global require laravel/installer
```

Et créer le projet :

```
>laravel new nom_projet
```

Documentation <https://laravel.com/docs/9.x#your-first-laravel-project>

3. Lancement dans le serveur de développement local en utilisant la commande CLI Artisan de Laravel:

```
>cd nom_projet =>pour se placer dans le dossier du projet
```

```
>php artisan serve
```

Remarque :

- **Composer** est un gestionnaire de dépendances pour PHP qui simplifie le processus d'installation et de gestion des bibliothèques nécessaires au développement. Dans le contexte de Laravel, Composer est souvent utilisé pour installer les dépendances du projet et maintenir la cohérence des versions entre les différentes bibliothèques.
- **Artisan** est l'interface de ligne de commande CLI (command line interface) incluse avec Laravel. Artisan existe à la racine de votre application en tant que script artisan et fournit un certain nombre de commandes utiles qui peuvent vous aider lors de la création de votre application.

Documentation : <https://laravel.com/docs/9.x/artisan>

4. Utiliser la commande artisan serve pour démarrer un serveur de développement intégré pour l'application Laravel que vous êtes en train de développer. Lorsque vous exécutez cette commande, Laravel lance un serveur HTTP local, généralement sur le port 8000, qui peut être utilisé pour tester et développer votre application localement.

```
>php artisan serve
```

Par défaut, le serveur sera accessible à l'adresse `http://localhost:8000` dans votre navigateur (`http://127.0.0.1:8000/`).

5. Le point d'entrée de votre application est le fichier de routage. Allez au dossier `Routes>web.php` :

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

Le mécanisme de routage fait correspondre un URL saisi dans la barre d'adresse du navigateur avec une vue (une interface graphique : page web) :

Requête http => routage => contrôleur :action =>vue

Les routes Laravel les plus élémentaires acceptent un URI et une fermeture (closures, fonction anonyme) fournissant une méthode très simple et expressive de définition des routes et du comportement sans fichiers de configuration de routage compliqué.

La méthode View prend en paramètre le nom de vue définie dans le dossier **ressources/views** sous le nom **welcome.blade.php**.

Blade est un moteur de template simple mais puissant fourni avec Laravel. Il vous simplifier l'intégration du code PHP dans vos vues (code html).

Le nom d'une vue qui utilise blade doit se terminer par : **.blade.php**

Documentation : <https://laravel.com/docs/5.3/blade>

Exercice :

Allez dans `welcome.blade.php` et remplacer le contenu de body avec :

```
<h1>Salut.C'est mon premier projet laravel.</h1>
```

Lancer le serveur web et saisir l'URL : <http://127.0.0.1:8000> dans le navigateur.

Exemples :

- 1) **Fonctions anonyme qui retourne un résultat correspondant à une requête get sur un url /hello avec des headers personnalisés :**

```
Route::get('/hello', function() {  
    return response('<h1>Hello World</h1>', 200)  
        ->header('Content-Type', 'text/plain')  
        ->header('foo', 'bar');  
});
```

- Ajouter ce code a web.php
- Taper l'url : <http://127.0.0.1:8000/hello> dans la barre d'adresse de votre navigateur chrome
- Inspecter les headers de la réponse avec dev tools de chrome.

2) Url contenant un paramètre :

```
Route::get('/posts/{id}', function($id){  
    return response('Post ' . $id);  
})->where('id', '[0-9]+');
```

- Ajouter ce code a web.php
- Taper l'url : <http://127.0.0.1:8000/posts/a12> dans la barre d'adresse de votre navigateur chrome. Commenter!
- Taper l'url : <http://127.0.0.1:8000/posts/12> dans la barre d'adresse de votre navigateur chrome. Commenter!

```
Route::get('/search', function(Request $request)  
    dd($request);  
});
```

Remarque :

En MVC le routage est un processus consistant à mapper la demande du navigateur à l'action du contrôleur et à renvoyer la réponse.

Documentation : <https://laravel.com/docs/9.x/routing>

Exercice :

L'objectif de cet exercice est d'afficher une liste des produits. Pour adopter une approche progressive nous allons commencer la démonstration par la création de liste dans la mémoire (in memory) puis dans un deuxième temps nous allons utiliser un modèle et base de données.

- 1) Ajouter cette route à web.php :

```
Route::get('/', function () {  
    $articles=[  
        ["title"=>"Produit 1",  
         "prix"=>200.5],  
        ["title"=>"Produit 2",  
         "prix"=>89],  
        ["title"=>"Produit 3",  
         "prix"=>78.5],  
    ];  
    return view('catalogue',["articles"=>$articles]);  
});
```

- 2) Créons la vue correspondante catalogue avec blade et qui va afficher notre liste des produits :

Vue catalogue.blade.php

```
<h1>Nos produits</h1>  
<ul>  
    @foreach ($articles as $item)  
        <li>{{ $item['title'] }} prix : <strong>{{ $item['prix'] }} MAD</strong></li>  
    @endforeach  
</ul>
```

- 3) Taper l'url : `http://127.0.0.1:8000/posts/12` dans la barre d'adresse de votre navigateur chrome!
- 4) Vous avez remarques que ne n'avons pas respecter le pattern mvc qui consiste à mettre la logique métier (ici la préparation de la liste des produits) dans un contrôleur et non pas dans le fichier de routage. C'est pourquoi nous allons créer un contrôleur pour les produits :

Dans le terminal utiliser la commande artisan CLI suivante :

```
PS C:\2022 2023\Laravel\tp1> PHP artisan make:controller ProduitController

INFO Controller created successfully.
```

Le contrôleur est créer dans le dossier `app>http>controllers` :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ProduitController extends Controller
{
    //
}
```

NB : Comme vous pouvez le remarquer un contrôleur une classe qui hérite de la classe de base Controller.

Déplaçons la logique définie dans le fichier de routage dans le contrôleur :

```
class ProduitController extends Controller
{
    public function index(){
        $articles=[
            ["title"=>"Produit 1",
            "prix"=>200.5],
            ["title"=>"Produit 2",
            "prix"=>89],
            ["title"=>"Produit 3",
            "prix"=>78.5],
        ];
        return view('catalogue',["articles"=>$articles]);
    }
}
```

6. Configurons la route vers la fonction index du contrôleur :

```
use App\Http\Controllers\ProduitController;
Route::get('/',[ProduitController::class,'index']);
```

7. Maintenant, nous souhaitons utiliser une base de données pour stocker les données des produits. C'est le rôle d'un model :

Ajouter le model produit en utilisant la commande artisan suivante :

```
PS C:\2022 2023\Laravel\tp1> PHP artisan make:model Produit

INFO Model created successfully.
```

8. Changeons le code de la méthode index du contrôleur pour utiliser le model :

```
<?php
use App\Models\Produit;
namespace App\Http\Controllers;

use App\Models\Produit;
use Illuminate\Http\Request;

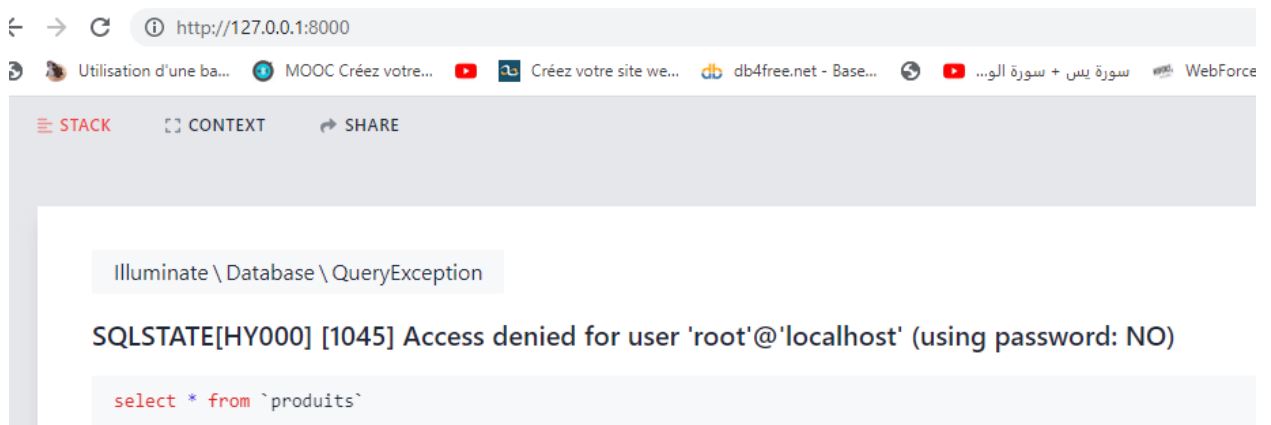
class ProduitController extends Controller
{
    public function index(){
        return view('catalogue',["articles"=>Produit::all()]);
    }
}
```

Remarque :

Produit::all() est une syntaxe propre à **Eloquent**.

Laravel inclut Eloquent, un mappeur relationnel objet (ORM) qui rend agréable l'interaction avec votre base de données **sans écrire du code sql**. Lors de l'utilisation d'Eloquent, chaque table de base de données à un "modèle" correspondant qui est utilisé pour interagir avec cette table. En plus de récupérer des enregistrements de la table de base de données, les modèles Eloquent vous permettent également d'insérer, de mettre à jour et de supprimer des enregistrements de la table.

9. Rechargeons la page dans le navigateur :



Il y a une exception lors de l'accès à la base de données (absence du mot de passe) :

Nous allons spécifier le mot de passe dans le fichier de configuration **.env** :

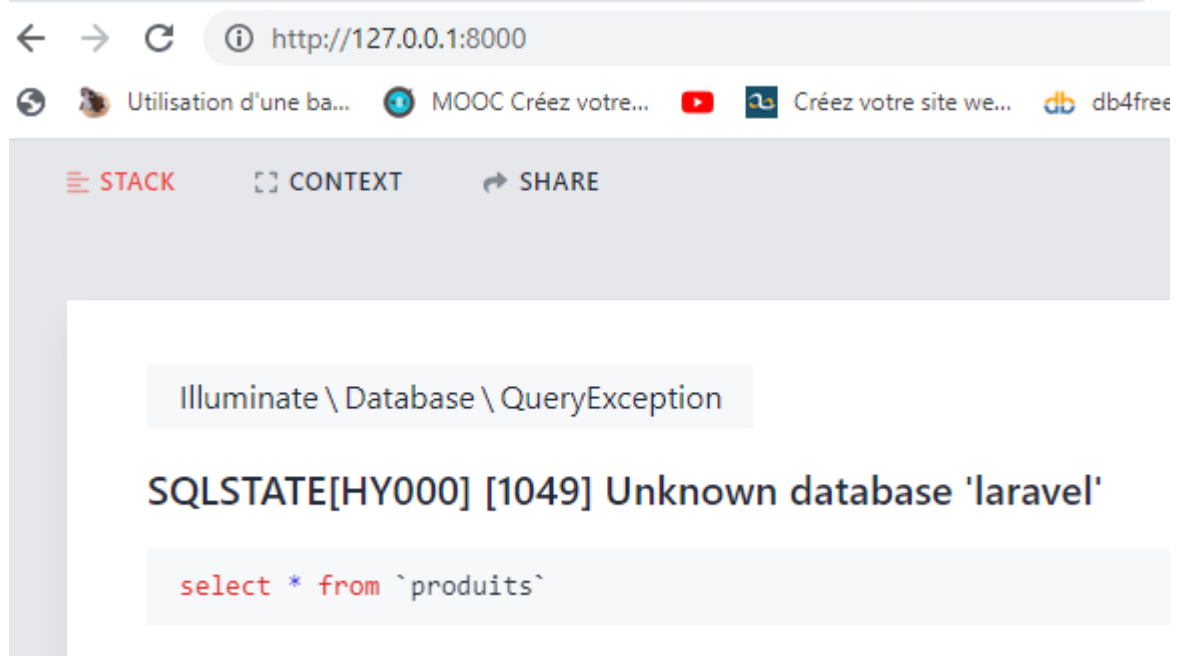

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=root
```

Remarque :

Laravel utilise une bibliothèque .env pour la gestion des paramètres de configuration.

Documenttation : <https://laravel.com/docs/9.x/configuration>

En chargeant la page il y aura une autre exception :



NB : Eloquent a généré le code sql correspondant à la méthode statique all :

Select * from produits

10. Nous devons créer la base de données laravel (ou autre nom à modifier dans .env)

11. Maintenant il faut créer la table produite qui correspond au model Produit. Pour cela nous allons créer un fichier de migration Elequent avec la commande Artisan suivante :

```
PS C:\2022 2023\Laravel\tp1> php artisan make:migration create_produits_table
```

```
INFO Created migration [2022_08_12_141631_create_produits_table].
```

Cette commande crée un fichier de migration dans **databases/migrations** :

Ce fichier contient le code d'une classe qui hérite de la classe Migration.

Cette classe contient :

- Une méthode up est utilisée pour ajouter de nouvelles tables, colonnes ou index à votre base de données,
- Une méthode down doit inverser les opérations effectuées par la méthode up (rollback).

```
public function up()
{
    Schema::create('produits', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::dropIfExists('produits');
}
```

Remarques :

- Les migrations est une technique pour gérer l'historiques des modifications apportées a la base de données. Ce qui très utile dans le contexte de collaboration de plusieurs développeurs.
- La façade Laravel Schema fournit une prise en charge indépendante de la base de données pour créer et manipuler des tables sur tous les systèmes de base de données pris en charge par Laravel. En règle générale, les migrations utilisent cette façade pour créer et modifier des tables et des colonnes de base de données.

Documentation : <https://laravel.com/docs/9.x/migrations>

12. Définissant les colonnes de la table produits :

```
public function up()  
{  
    Schema::create('produits', function (Blueprint $table) {  
        $table->id();  
        $table->string("title");  
        $table->decimal("prix");  
        $table->timestamps();  
    });  
}
```

13. Exécuter les migrations :

```
PS C:\2022 2023\Laravel\tp1> php artisan migrate  
  
INFO Preparing database.
```

La table est créée dans la base de données.

14. Ajouter des lignes à la table produits en utilisant phpmyadmin

15. Recharger la page :

```
PS C:\2022 2023\Laravel\tp1> php artisan serve  
  
INFO Server running on [http://127.0.0.1:8000].
```

Nos produits

- Fugiat iure officia. prix : **29.57 MAD** [description](#)
- Est ex similique. prix : **672567.82 MAD** [description](#)
- Et in aut. prix : **0.07 MAD** [description](#)
- Ullam consequuntur maiores. prix : **423.19 MAD** [description](#)
- Expedita dignissimos quia. prix : **168.50 MAD** [description](#)
- Error nam labore. prix : **20.62 MAD** [description](#)

Conclusion :

Dans cette partie vous avez appris à mettre en place une architecture MVC en définissant

- Un modèle qui représente une table dans la base de données;
- Des migrations pour mettre à jour la structure de la base de données.
- Une vue pour afficher les données;
- Un contrôleur pour traiter les différentes requêtes http;
- Des routes pour mapper la demande du navigateur (requêtes http) à l'action du contrôleur.

Partie II :

1. Lecture et filtrage:

```
$produits = Produit::all();  
$produit = Produit::find(1);  
$produit = Produit::findOrFail(1);  
$produits = Produit::where('designation', 'Clavier')->get();  
$produits = Produit::where('designation', 'Clavier')->orWhere('designation', 'Sourie')->get();  
$produits = Produit::orderBy('designation', 'desc')->get();  
$count = Produit::where('active', 1)->count();  
$produit = Produit::where('designation', 'Clavier')->first();  
$produits = Produit::findMany([1, 2, 3]);  
$produits = Produit::whereBetween('age', [18, 30])->get();  
$produits = Produit::whereNull('deleted_at')->get();  
$produit = Produit::where('designation', 'Clavier')->firstOrFail();  
$produits = Produit::whereDate('created_at', '=', now()->toDateString())->get();
```

2. Création :

```
$produit = Produit::create([  
    'designation' => 'Clavier ',  
    'prix' => 70,  
    'password' => bcrypt('password'),
```

```
]);
```

Ou :

```
$newProduit = new Produit;
```

```
$newProduit->designation = 'Clavier ';
```

```
$newProduit->prix = 70;
```

```
$newProduit->save();
```

3. Modification :

```
$produit = Produit::find(1);
```

```
$produit->update(['designation' => 'New Name']);
```

Ou

```
$produit = Produit::find(1);
```

```
$produit->designation = 'Nouveau Nom';
```

```
$produit->save();
```

4. Suppressions:

```
$produit = Produit::find(1);
```

```
$produit->delete();
```

Ou:

```
Produit::where('active', '=', 0)->delete(); // Supprime tous les produits inactifs
```

Ou :

```
Produit::destroy(1); // Supprime le produit avec la clé primaire 1
```

```
Produit::destroy([1, 2, 3]); // Supprime les produits avec les clés primaires 1, 2 et 3
```

Partie III



All Categories ▾

Search

SEARCH

[HOME](#)[PRODUCTS](#)[BLOGS](#)[CONTACT US](#)[BRANDS](#)

TOP CATEGORY

I Desktops

[Web Cameras](#)[Cameras](#)[Components](#)[Phones & PDAs](#)[Laptops](#)

SALE



Canon EOS 1300D DSL...

€ 69 00

SALE



Canon CS100 Connect ...

€ 69 00

SALE



Apple iPhone 6 (Gold, ...

€ 69 00