Centre universitaire de Mila
Première Année Master STIC

Cours Développement Web Avancé Chapitre 2

Présenté par Mr :
Boukhechem Nadhir

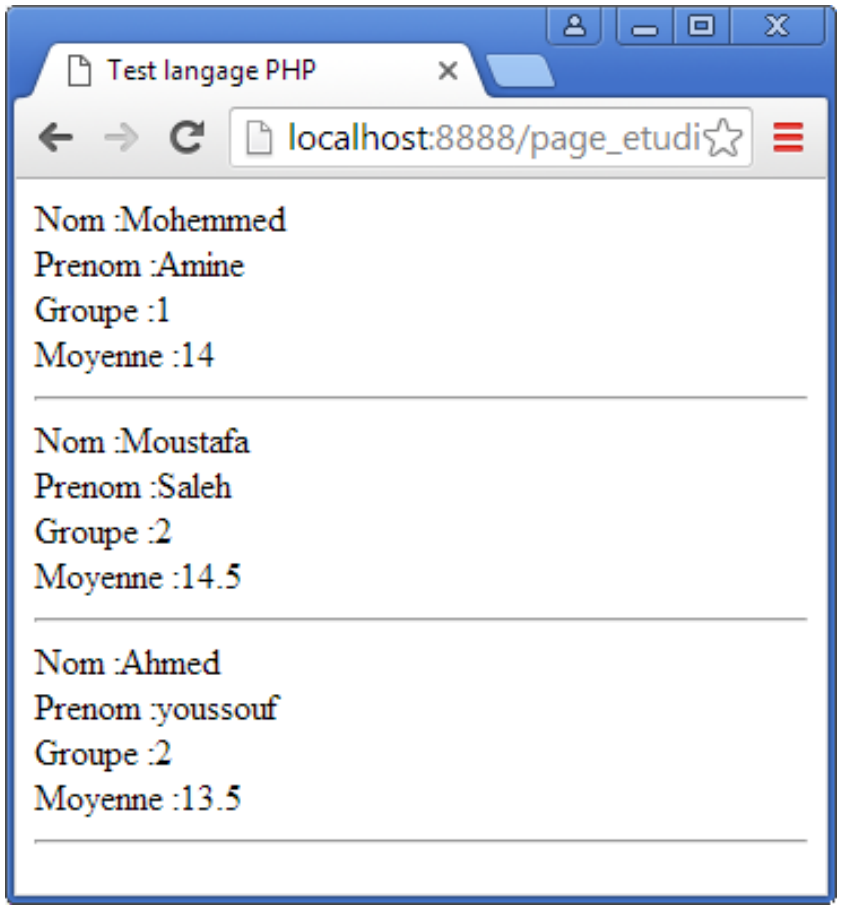
Contenu du module

Chapitre 2 : Les frameworks de développement Web

- Le modèle MVC (Modèle-Vue-Contrôleur)
- Les frameworks de développement Web
- Installation et configuration (framework Laravel)
- Le système de routage
- Manipulation des bases de données
- Gestion des formulaires
- Le système d’authentification
- Les middlewares

(Modèle-Vue-Contrôleur) : *Exemple Introductif*

Pour commencer nous allons prendre un exemple d'un programme PHP qui permet d'afficher une liste d'étudiants a partir d'une base de données. Nous allons voir plusieurs version du programme et déterminer la version la plus efficace.



(Modèle-Vue-Contrôleur) : *Exemple Introductif*

```php
<?php
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $resultat = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

?>


<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
</body>
</html>
```

Version 1 : Un seul fichier

« page_etudiants.php »

(Modèle-Vue-Contrôleur) : *Exemple Introductif*

```php
<?php
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $resultat = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
    mysql_close(); //fermer la connexion
    require ('vue1.php');
?>
```

« page_etudiants.php »

```php
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
</body>
</html>
```

« vue1.php »

Version 2 : deux fichiers

(Modèle-Vue-Contrôleur) : *Exemple Introductif*

```php
<?php
    function get_etudiants()
    {
    $ma_connexion = mysql_connect ('localhost', 'root', '');   //Connexion au SGBD MySQL
    mysql_select_db ('universite', $ma_connexion) ; // selection de la base université
    //Préparer la requette SQL
    $sql = 'SELECT * FROM etudiants;';
    //Exécuter la requette SQL
    $data = mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

    return $data;   // retourner le resultat extrait de la base de données
    }
?>
```

« modele_etudiants.php »
le modèle

Version 3 : trois fichiers

```php
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
        <head>
            <title>Test langage PHP</title>
            <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
        </head>
<body>
    <?php
    //Percourir chaque tuple du résultat et l'afficher
    while($etud = mysql_fetch_row($resultat))
    {
    echo 'Nom :'.$etud['0'].'<br/>';
    echo 'Prenom :'.$etud['1'].'<br/>';
    echo 'Groupe :'.$etud['2'].'<br/>';
    echo 'Moyenne :'.$etud['3'].'<br/>';
    echo '<hr/>';
    }
    ?>
</body>
</html>
```

« vue1.php »
la vue

```php
<?php
    require 'modele_etudiants.php';
    $resultat = get_etudiants();
    require 'vue1.php';
?>
```

« page_etudiants.php »
le contrôleur

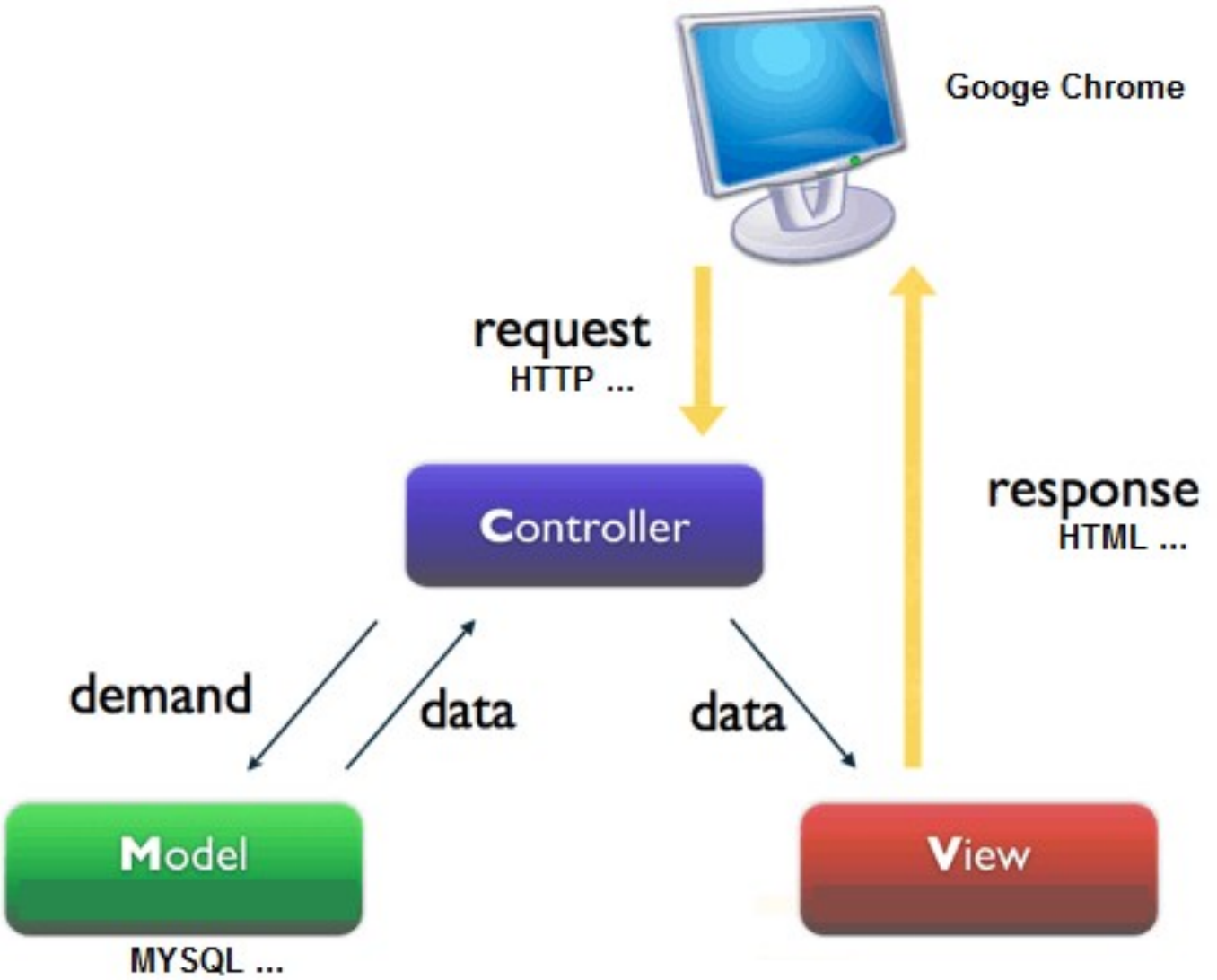
Modèle-Vue-Contrôleur

Modèle : gestion des données de l'application

Vue : Affichage et interaction avec l'utilisateur

Contrôleur : gestion de la logique du code (il prend les décisions)

Modèle-Vue-Contrôleur



L'architecture MVC (Modèle-Vue-Contrôleur)

Modèle-Vue-Contrôleur

Avantages du MVC

- Une conception claire et efficace
- Un gain de temps de maintenance et d'évolution du site (l'ajout et mise à jour des fonctionnalités est très facile)
- Une modularité qui offre une grande souplesse pour organiser le développement du site entre différents développeurs
- Une rapidité de développement

Les frameworks de développement Web

Le framework

≈

« cadre de travail » ou « cadre de développement »

≈

« architecture prête à l'emploi »

≈

« ensemble d'outils constituant les fondations d'une application »

≈

« c'est à la fois une sorte de boîte à outils et une méthodologie de travail »

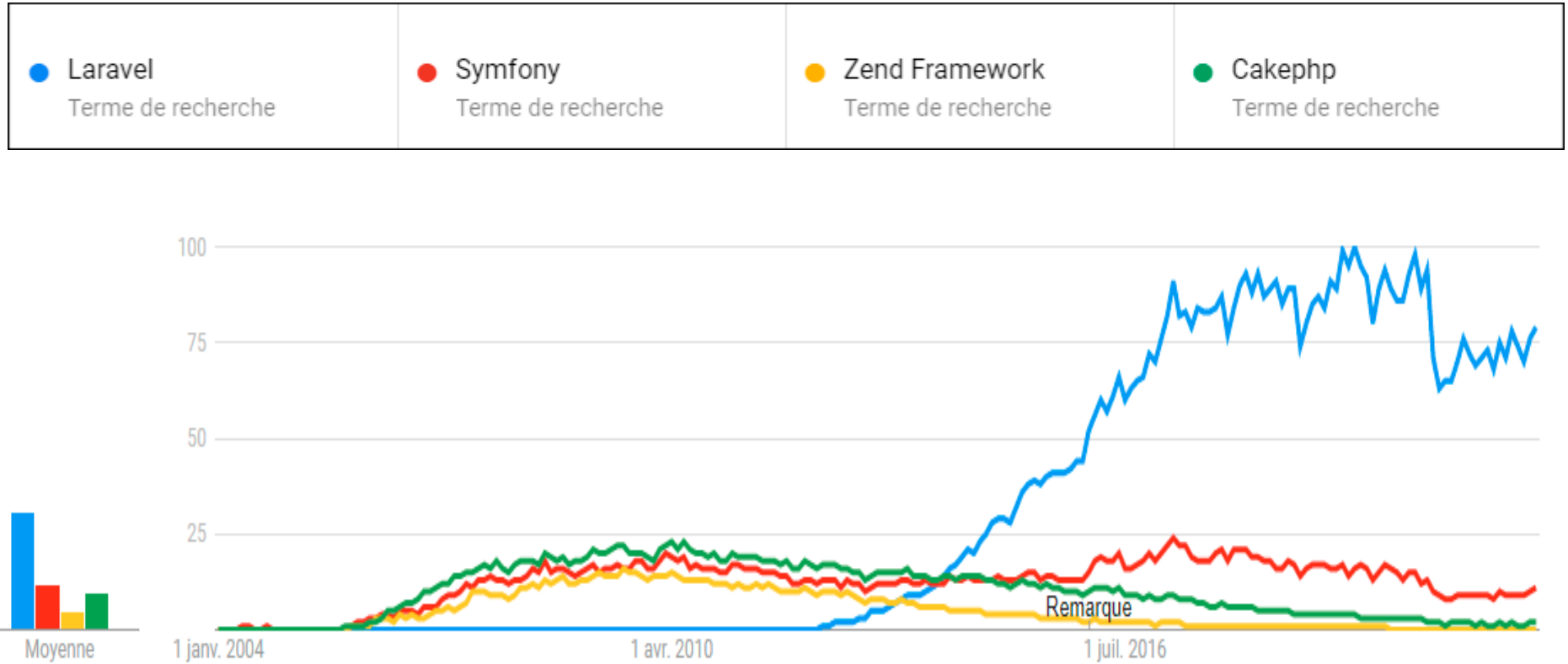
Les frameworks de développement Web

Avantages des frameworks

- La rapidité de développement des projets grâce aux outilles disponibles
- Une très bonne organisation des projets
- Plusieurs frameworks sont basé sur l'architecture MVC, ils bénéficient donc de ses avantages
- Composants réutilisables
- Respecter les norme actuelles (sécurité , nouvelles technologies)

Les frameworks de développement Web

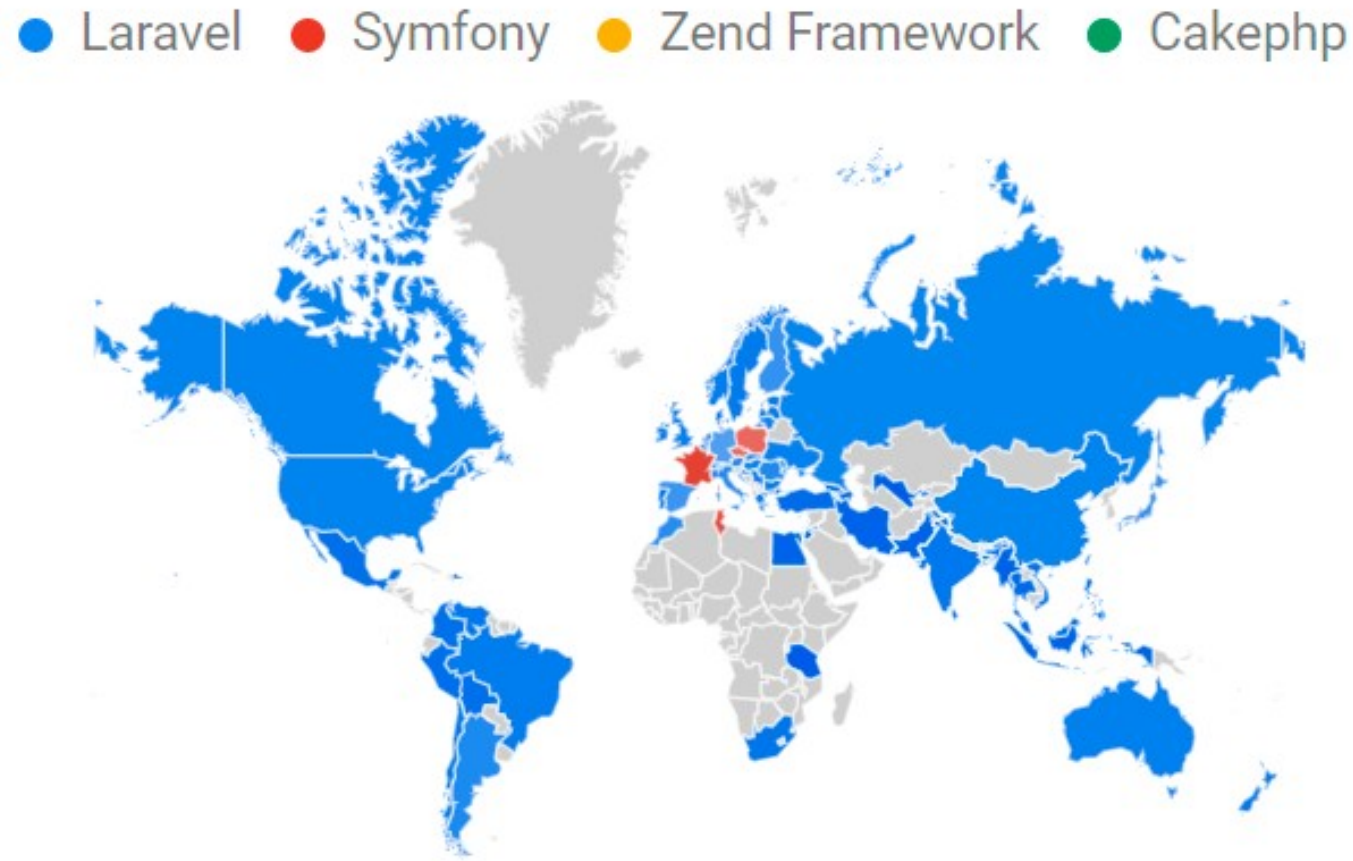
Tendance des frameworks Web PHP



« Statistiques du 01/01/2014 au 21/03/2022 (Google Trends) »

Les frameworks de développement Web

Tendance des frameworks Web PHP



« Statistiques du 22/04/2018 au 21/03/2022 (Google Trends) »

Le frameworks Laravel (présentation)

Le framework « Laravel »

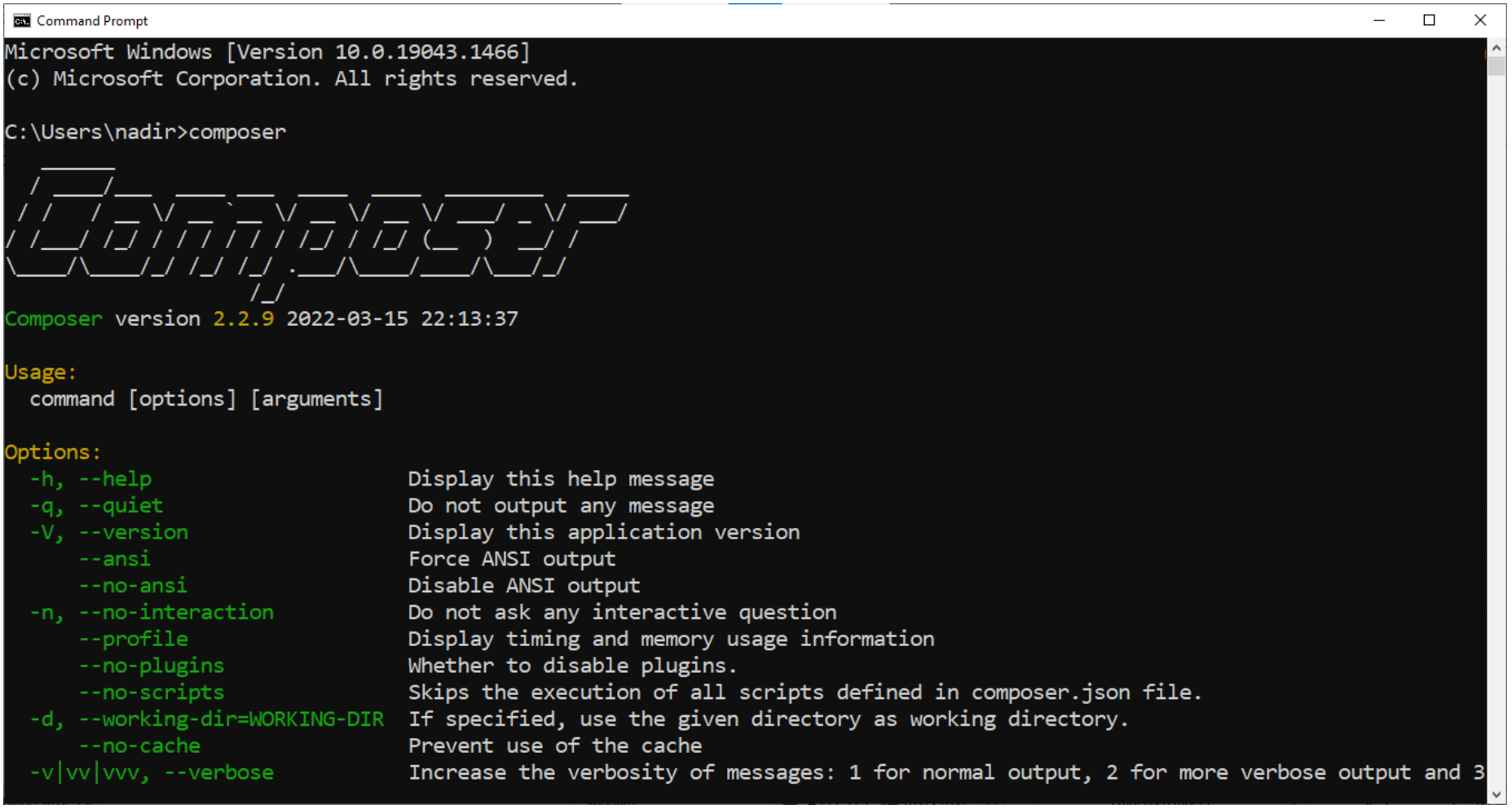
C'est un famewok PHP open sources basé sur l'architecture MVC. Sortie en juin 2011, Ils est actuellement l'un des frameworks PHP les plus populaires.

Le framework Laravel (Installation)

1) Installer d'abord un **interpréteur PHP** et tous les composants nécessaires a Laravel (on installer par exemple **Wampserver** ou **Xampp** qui contiens tous les composants nécessaires a Laravel)

Le framework Laravel (Installation)

2) Installer « **Composer** », celui-ci est un outils qui permet de gérer les dépendances des paquetages PHP,



Le framework Laravel (Installation)

3) Installer « **Laravel** » en exécutant la commande suivante :

```
composer create-project laravel/laravel monprojet1
```

```
C:\>cd dev_web

C:\dev_web>composer create-project laravel/laravel monprojet1
```

Le framework Laravel (Installation)

Command Prompt

```
  - Installing phpspec/prophecy (v1.15.0): Extracting archive
  - Installing phar-io/version (3.2.1): Extracting archive
  - Installing phar-io/manifest (2.0.3): Extracting archive
  - Installing myclabs/deep-copy (1.11.0): Extracting archive
  - Installing phpunit/phpunit (9.5.19): Extracting archive
  - Installing spatie/backtrace (1.2.1): Extracting archive
  - Installing spatie/flare-client-php (1.1.0): Extracting archive
  - Installing spatie/ignition (1.2.5): Extracting archive
  - Installing spatie/laravel-ignition (1.1.1): Extracting archive
73 package suggestions were added by new dependencies, use `composer suggest` to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: laravel/sail
Discovered Package: laravel/sanctum
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Discovered Package: spatie/laravel-ignition
Package manifest generated successfully.
78 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
No publishable resources for tag [laravel-assets].
Publishing complete.
> @php artisan key:generate --ansi
Application key set successfully.
```

Le framework Laravel (Installation)

Pour connaître le version PHP installé on tape la commande :

```
php -v
```

```
C:\>php -v
PHP 8.1.4 (cli) (built: Mar 16 2022 09:33:31) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.4, Copyright (c) Zend Technologies
```

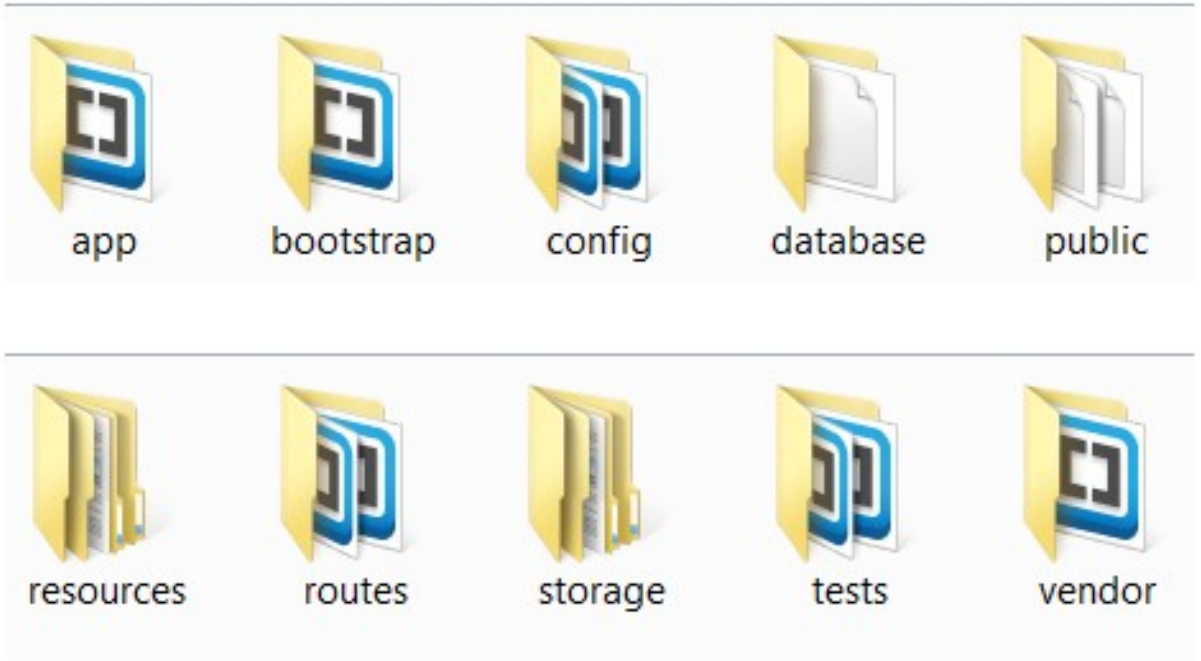
Pour connaître le version Laravel installé on tape la commande :

```
php artisan --version
```

```
C:\>cd dev_web\monprojet1

C:\dev_web\monprojet1>php artisan --version
Laravel Framework 9.5.1
```

Le framework Laravel (Installation)

Les dossiers de « Laravel »

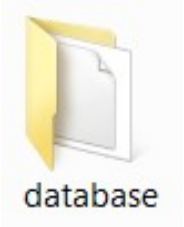


Le framework Laravel (descriptions des dossiers)

Contient toutes les classes PHP de l'application (Controllers, Middleware ..etc).

Contient les scriptes d'initialisation de Laravel (à ne pas confondre avec le framework css bootstrap).

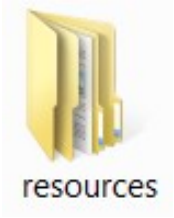
Contient les configurations nécessaires a l'application (base de données, authentification, langue par défaut ..etc).



Contient les différents fichiers qui décrivent les tables de la base de données (les migrations).

Le framework Laravel (descriptions des dossiers)

Contient les fichiers de l'application accessibles depuis l'extérieur de Laravel (images, fichiers CSS et Js ... etc). On peut éventuellement copier les fichiers du framework Bootstrap4 dans ce dossier si on veut l'utiliser.



Contient les fichiers de l'application qui ne sont pas écrits en php (html, fichiers de traduction de la langue ..etc). Il contient entre autre le dossier « views » dans le quel on met les différentes vues de l'application (model MVC).

Permet de gérer les URLs d'entré a l'application et leurs redirections (entre autres les URLs web avec le fichier « web.php »)

Le framework Laravel (descriptions des dossiers)

Contient les données temporaire générés par l'application.

Utilisé pour faire des testes sur l'application.

Contient les fichiers nécessaires au fonctionnement de Laravel.

Le framework Laravel (Démarrage du serveur)

Après avoir installer Laravel on se déplace ver le dossier d'installation et on exécute la commande ci dessous pour démarre le serveur :

```
php artisan serve
```

ou

```
php artisan serve –port= numéro_port
```

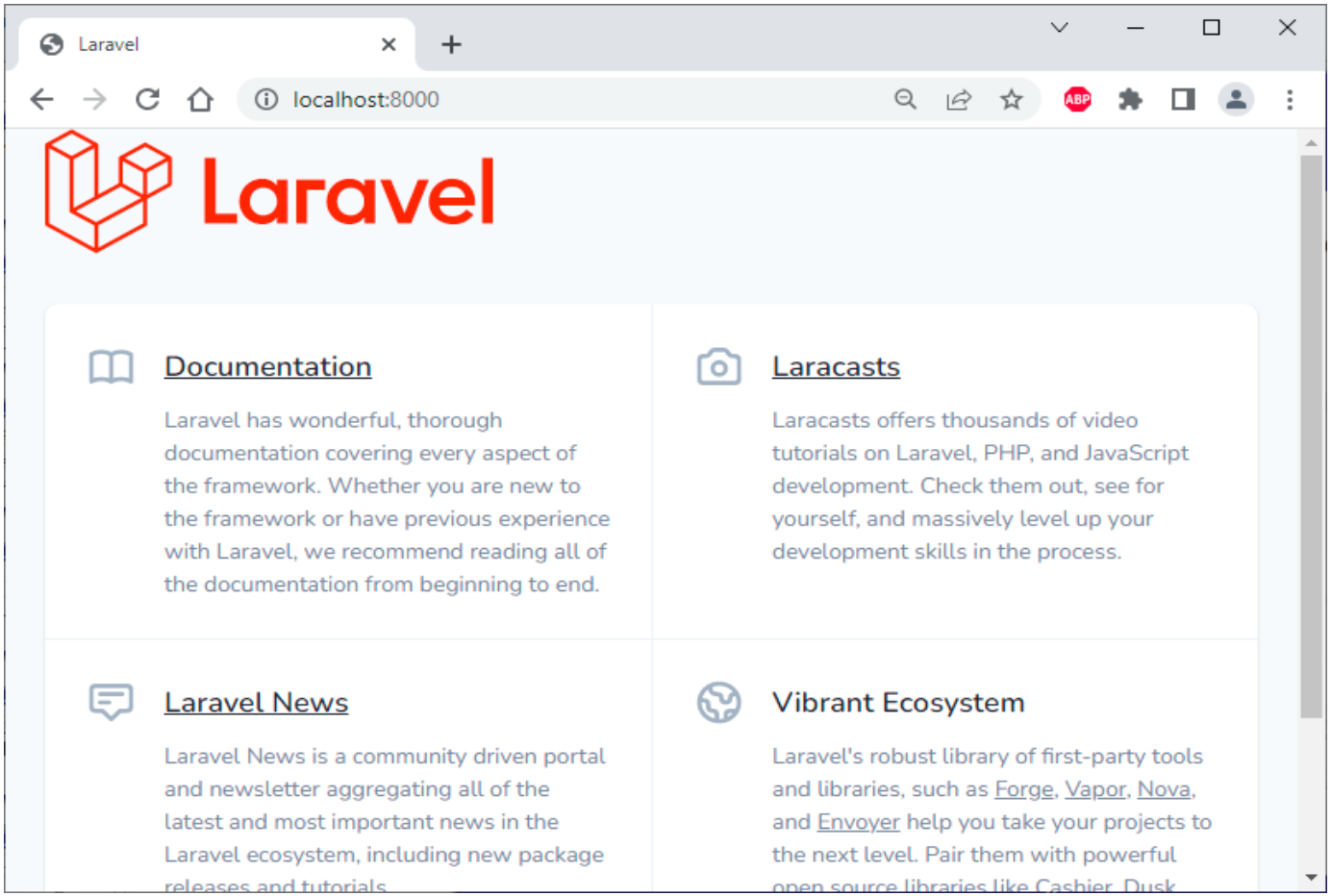
Après avoir exécuté la commande le lien ver le serveur s'affiche sur l'écran.

```
C:\dev_web\monprojet1>php artisan serv
Starting Laravel development server: http://127.0.0.1:8000
[Mon Mar 21 19:47:06 2022] PHP 8.1.4 Development Server (http://127.0.0.1:8000) started
```

Le framework Laravel (Démarrage du serveur)

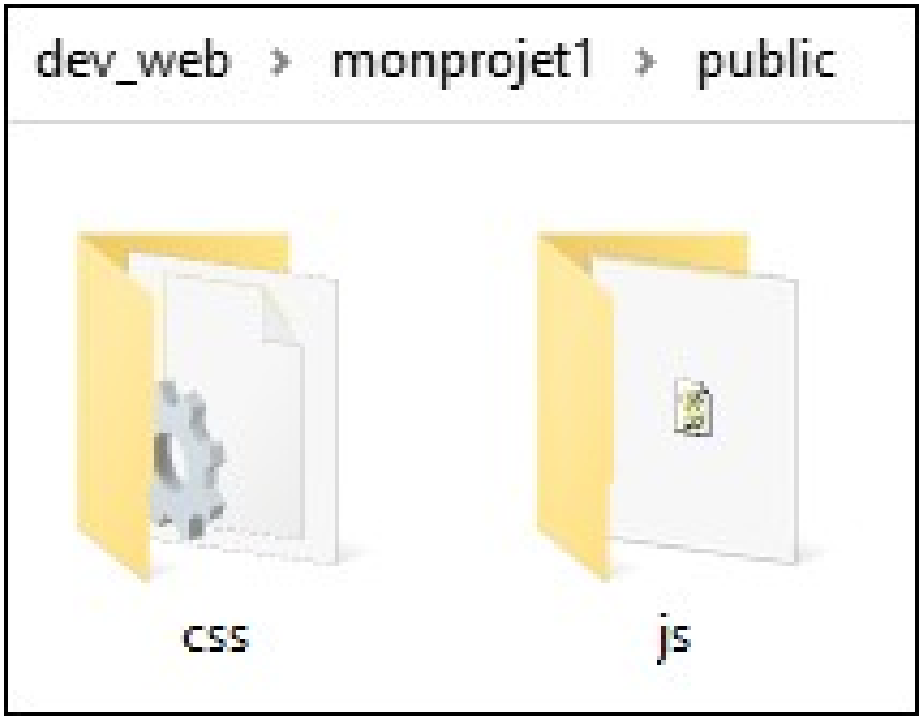
Pour vérifier que l'installation s'est bien déroulé, on ouvre avec un navigateur le lien : **http://localhost:8000/**

Dans l'exemple ci dessous le port utilisé par Laravel est 8000.



Le framework Laravel (Démarrage du serveur)

Remarque : Pour utiliser Bootstrap en local avec Laravel il faut copiez les fichiers css et javascript dans le dossier **public**.



Le framework Laravel (système de routage)

Les routes dans Laravel sont gérés par le fichier « **/routes/web.php** »

```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('/', function () {
    return view('welcome');

});
```

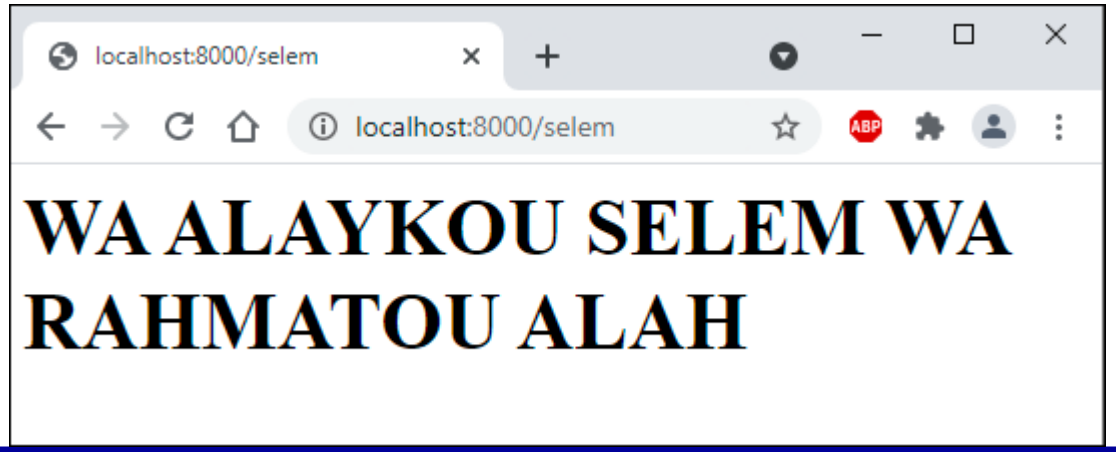
Le framework Laravel (système de routage)

Une route est définit comme suit :

Route::méthode_HTTP('URI' , fonction_appelé);

Exemple :
On veut créer une route vers l'URI : « /selem » qui peut être accédé avec la méthode « get », pour cela on ajoute une route au fichier « web.php » comme suit :

```php
Route::get('selem', function () {

    $resultat = '<h1> WA ALAYKOU SELEM WA RAHMATOU ALAH <h1>';
    return $resultat;

});
```



Le framework Laravel (système de routage)

Une route est définit comme suit :

Route::méthode_HTTP('URI' , fonction_appelé);

On peut utiliser les méthodes HTTP suivantes :

```
Route::get();
Route::post();
Route::put();
Route::delete();
Route::patch();
Route::delete();
Route::any();          « n'importe quelle méthode »
```

Le framework Laravel (système de routage)

Une route est définit comme suit :

Route::méthode_HTTP('URI' , fonction_appelé);

On peut aussi combiner les méthodes comme dans l'exemple suivant :

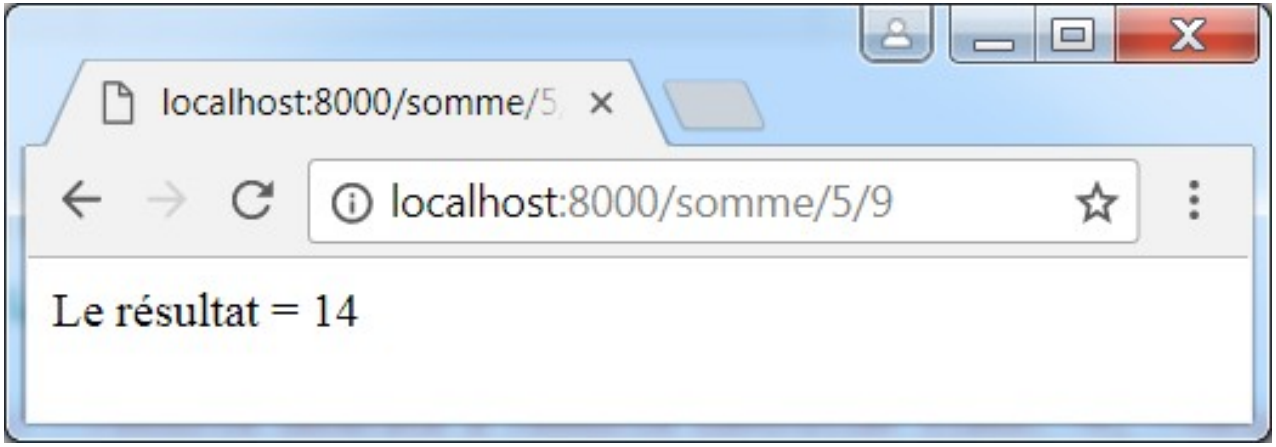
```php
Route::match(['get' ,'post'] , 'selem', function () {

    $resultat = '<h1> WA ALAYKOU SELEM WA RAHMATOU ALAH <h1>';
    return $resultat;

});
```

Le framework Laravel (système de routage :: paramètres)

On peut passer des paramètre avec le lien URI comme suit :

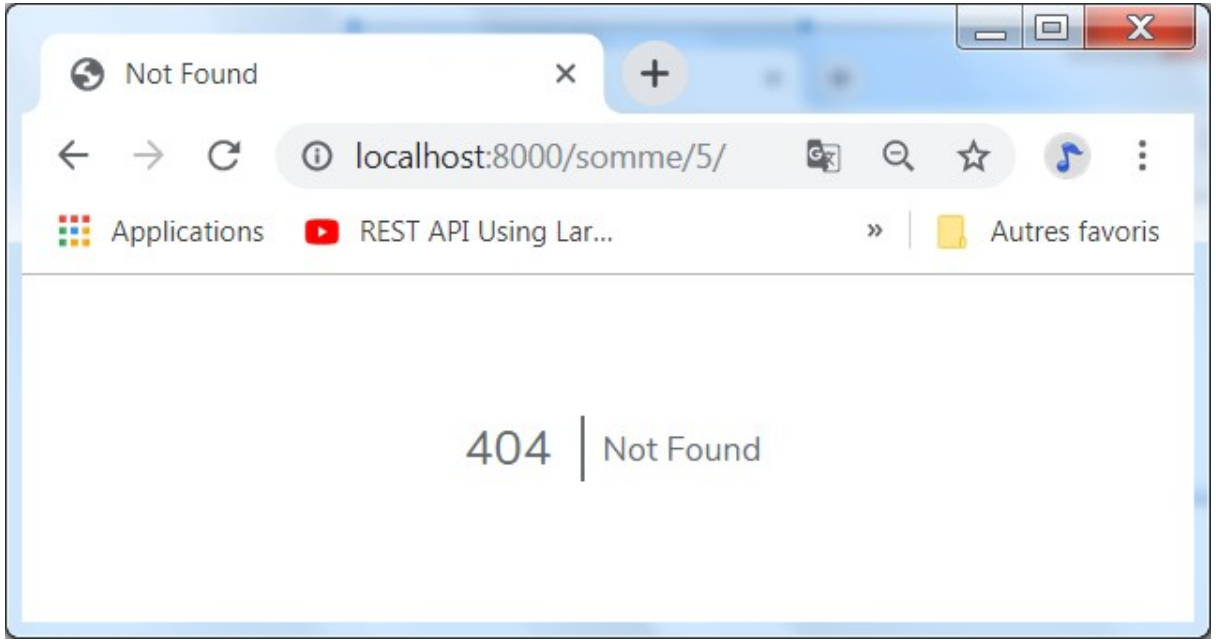
```php
Route::get('somme/{a}/{b}', function($a,$b) {
$result = $a+$b;
return 'La Somme = ' . $result;
});
```

routes/web.php



Le framework Laravel (les erreurs)

Si on oublie un paramètre dans le lien du navigateur un message d'erreur s'affiche :



Le framework Laravel (les erreurs)

Si on fait une erreur dans le code, par exemple oublier un point virgule, Laravel affiche l'emplacement de l'erreur :

ParseError

syntax error, unexpected 'return' (T_RETURN)

```
22    route::get('somme/{a}/{b}', function($a,$b) {
23
24    $result = $a+$b
25
26    return 'Le resultat = ' . $result;
27    });
```

Le framework Laravel (Les Contrôleurs)

Les **contrôleurs** se trouvent dans le dossier « **App/Http/Controllers** ». Un contrôleur est représenté par une classe PHP contenant plusieurs méthodes. Pour créer un nouveau contrôleur on tape la commande suivante :

```
php artisan make:controller NomControleur
```

Le framework Laravel (Les Contrôleurs)

Exemple :

1) On veut créer un contrôleur « CalculController » qui s'occupe des calculs mathématiques. On commence par créer un nouveau contrôleur comme suit :

```
C:\dev_web>cd monprojet1

C:\dev_web\monprojet1>php artisan make:controller CalculController
Controller created successfully.
```

app/Http/Controllers/CalculController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class CalculController extends Controller
{
    //



}
```

Le framework Laravel (Les Contrôleurs)

Exemple :
2) On ajoute les méthodes du contrôleur :

app/Http/Controllers/CalculController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class CalculController extends Controller
{
    //

public function somme($a,$b)
{
$resultat = $a * $b;
return 'La somme = ' . $result;
}


public function produit($a,$b)
{
$resultat = $a * $b;
return 'Le Produit = ' . $result;
}
```

Le framework Laravel (Les Contrôleurs)

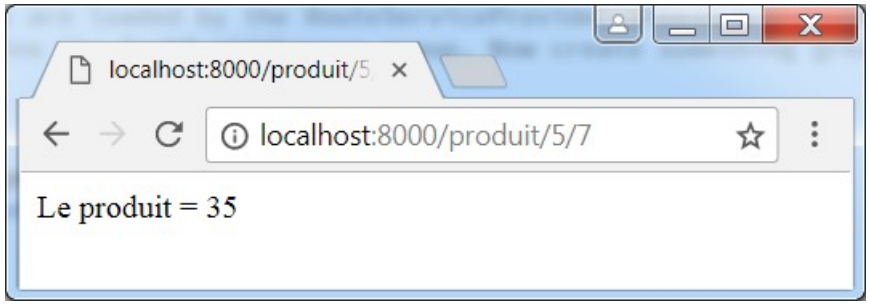
Exemple :

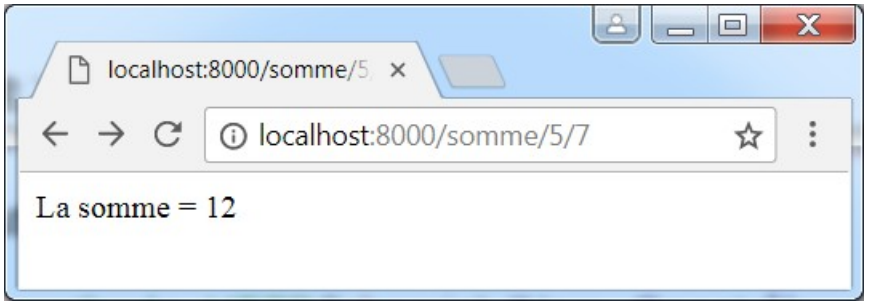
2) On ajoute les routes nécessaires dans le fichier « web.php » :

Une route ver un contrôleur est définit comme suit :

Route::methode_HTTP('chemin', 'Conntrôleur@fonction_contrôleur');

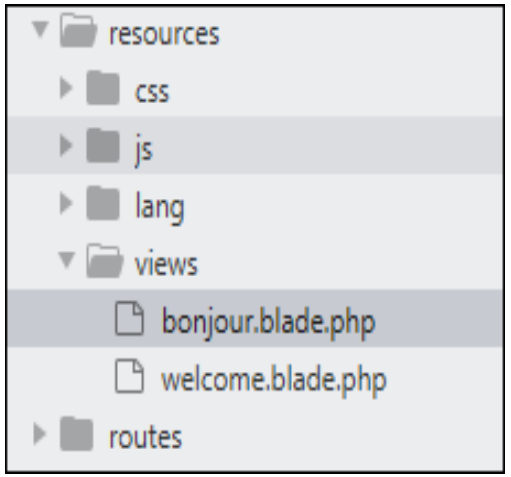
```
Route::get('somme/{a}/{b}', 'App\Http\Controllers\CalculController@somme');

Route::get('produit/{a}/{b}', 'App\Http\Controllers\CalculController@produit');
```





Le framework Laravel (Les Vues)

Les Vues se trouvent dans le dossier « **/resources/views** ». On donne ci-dessous un exemple d'une vue qui affiche « Bonjour » dans le navigateur , la vue s'appelle « bonjour.blade.php »




```html
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="/css/bootstrap.min.css" >


  </head>
  <body>

    <div class="text-center  text-info display-3">
    BONJOUR, C'EST LARAVEL
  </div>

        <!-- à la fin du body -->
        <script src="/js/bootstrap.bundle.min.js" > </script>

  </body>
</html>
```

Le framework Laravel (Les Vues)

Pour afficher la vue ont doit déclarer une route ver celle-ci dans le fichier « **web.php** ». La vue peut être retourner **directement par une fonction** dans le fichier « **web.php** » , ou par un contrôleur.

1

```php
Route::get('bonjour', function () {
    return view('bonjour');}
);
```

routes/web.php

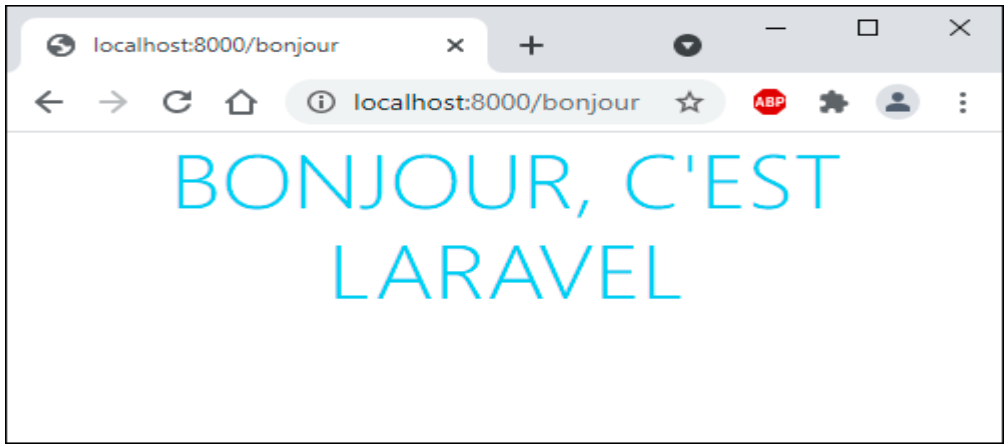
OU

1

```php
Route::view('/bonjour', 'bonjour');
```

routes/web.php

2



Si la vue est dans un dossier on fait : return view('**nom_dossier.**nom_de_la_vue') ;

Le framework Laravel (Les Vues)

Pour afficher la vue ont doit déclaré une route ver celle-ci dans la fichier « web.php ». la vue peut être retourner directement par une fonction dans le fichier « web.php » , ou par **un contrôleur**.

1

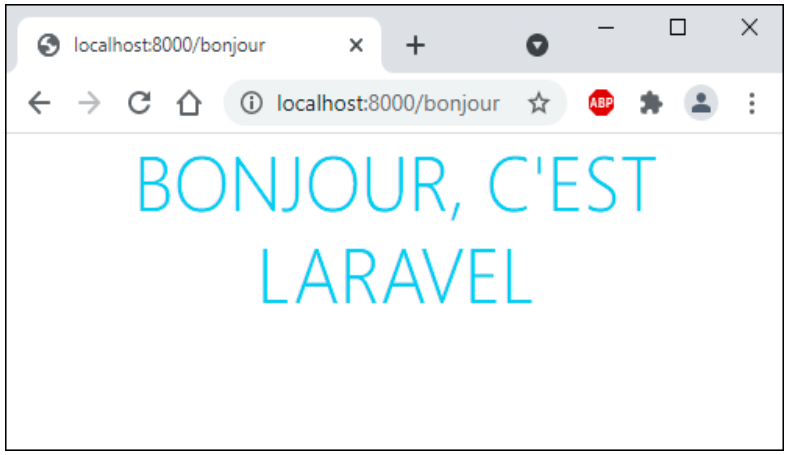
```
php artisan make:controller MonControleur
```

app/Http/Controllers/MonControleur.php

2

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class CalculController extends MonControlleur
{


public function affichage1 () {
    return view('bonjour');
}
```

4



routes/web.php

3

```php
Route::get('test2', 'App\Http\Controllers\MonControleur@affichage1');
```

Le framework Laravel (Passage de paramètres a la vue)

On veut programmer une application web qui calcul la factorielle d'un nombre :

1) On commence par créer une route ver un contrôleur « CalculControleur » qui appelle une fonction « factorielle »

routes/web.php

```
Route::get('factorielle/{n}', 'App\Http\Controllers\CalculController@factorielle');
```

Le framework Laravel (Passage de paramètres a la vue)

On veut programmer une application web qui calcul la factorielle d'un nombre :

2) On ajoute la fonction « factorielle » au contrôleur Calcul :

app/Http/Controllers/CalculController.php

```php
public function factorielle($n)
{

    $fact = 1;
    for ($i=1; $i<=$n;$i++)
    {
        $fact = $fact * $i;
    }

    return view('affichage',compact('fact'));
}
```

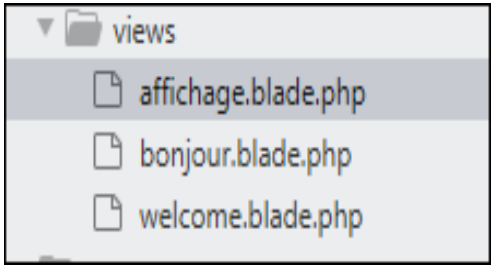
«« Transférer le résultat »

Le framework Laravel (Passage de paramètres a la vue)

On veut programmer une application web qui calcul la factorielle d'un nombre :

3) On ajoute la vue « affichage.blade.php » dans « **/resources/views** » :

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="/css/bootstrap.min.css" >

  </head>
  <body>

            <div class="text-center  text-info display-3">
            Le résultat de la factorielle  =   {{$fact}}
            </div>

            <!-- On peut aussi utiliser la synctaxe suivante : -->

            <div class="text-center  text-info display-3">
            Le résultat de la factorielle  =   <?php echo $fact ?>
            </div>

        <!-- à la fin du body -->
        <script src="/js/bootstrap.bundle.min.js" > </script>

  </body>
</html>
```



«« récupérer et afficher le résultat »

Le framework Laravel (Passage de paramètres a la vue)

On veut programmer une application web qui calcul la factorielle d'un nombre :

3) On ajoute la vue « affichage.blade.php » dans « **/resources/views** » :

resources/views/affichage.blade.php

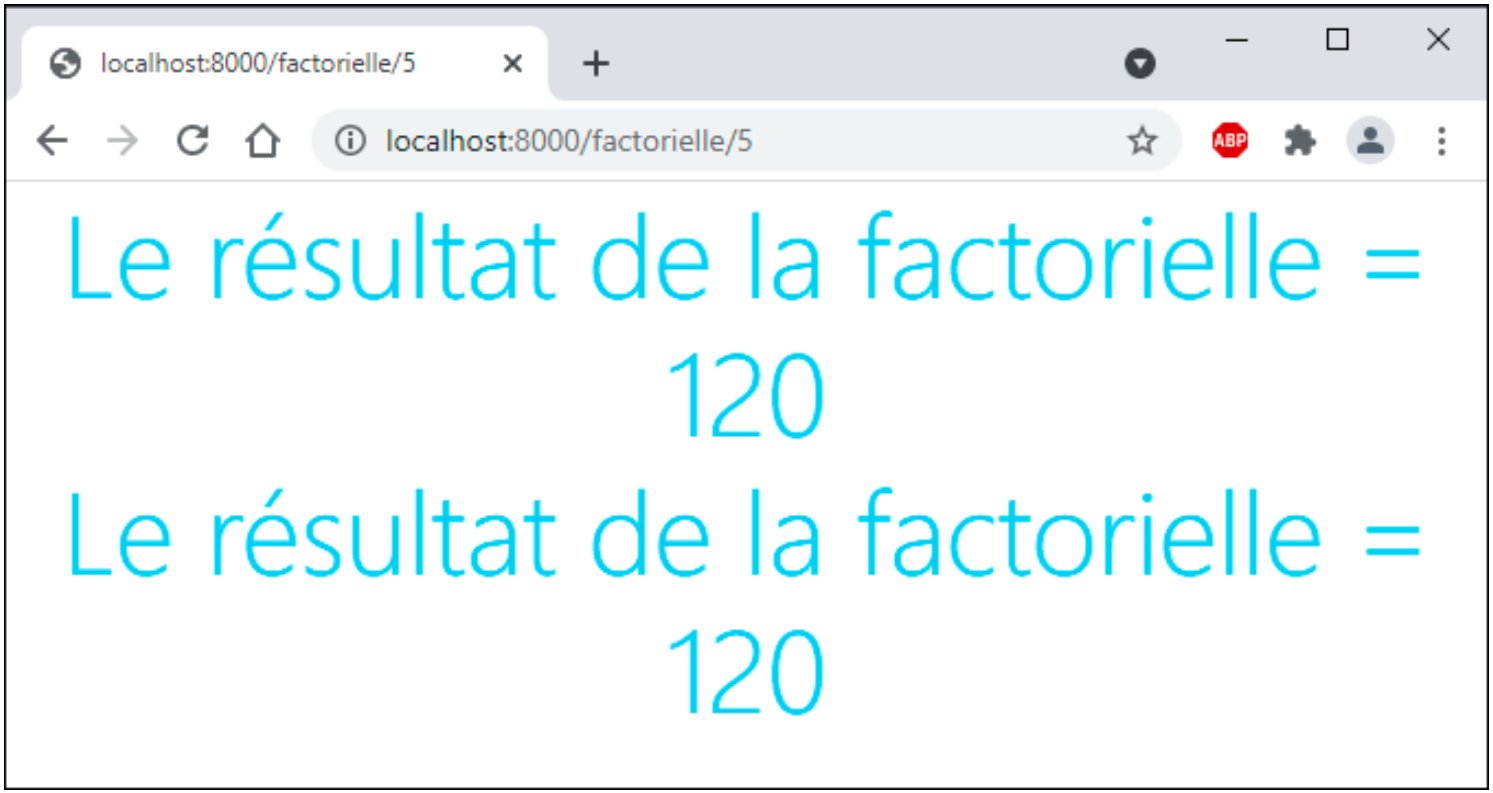
```
<div class="text-center  text-info display-3">
Le résultat de la factorielle  =   {{$fact}}
</div>

<!-- On peut aussi utiliser la synctaxe suivante : -->

<div class="text-center  text-info display-3">
Le résultat de la factorielle  =   <?php echo $fact ?>
</div>
```

«« **récupérer et afficher le résultat »**

Le framework Laravel (Passage de paramètres a la vue)

On veut programmer une application web qui calcul la factorielle d'un nombre :

3) On accède a l'application en utilisant le navigateur :



Le framework Laravel (Passage de paramètres a la vue)

« Passage de paramètres du contrôleur ver la Vue »

On peut passer plusieurs paramètres a une vue comme suit :

Méthode 1 :

app/Http/Controllers/CalculController.php

```php
public function calcul($a,$b)
{

    $s= $a+$b;
    $p = $a*$b;
    $d = $a/$b;

    return view('affichage', [
        'somme'=> $s,
        'produit'=>$p,
        'division'=>$d

    ] );
}
```

Le framework Laravel (Passage de paramètres a la vue)

« Passage de paramètres du contrôleur ver la Vue »

On peut passer des paramètres a une vue comme suit :

Méthode 2 :

app/Http/Controllers/CalculController.php

```php
public function calcul($a,$b)
{

    $data=[];
    $data['somme']= $a+$b;
    $data['produit'] = $a*$b;
    $data['division'] = $a/$b;

    return view('affichage', $data );
}
```

Le framework Laravel (Passage de paramètres a la vue)

« Passage de paramètres du contrôleur ver la Vue »

On peut passer des paramètres a une vue comme suit :

Méthode 3 :

app/Http/Controllers/CalculController.php

```php
public function calcul($a,$b)
{


    $somme= $a+$b;
    $produit = $a*$b;
    $division = $a/$b;

    return view('affichage',compact('somme', 'produit','division' ));
}
```

Le framework Laravel (Passage de paramètres)

« Passage de paramètres du contrôleur ver la Vue »

On peut récupérer les paramètres dans la vue comme suit :

resources/views/affichage.blade.php

```
<div class="text-center  text-info display-3">
Le résultat de la somme  =   {{  $somme  }}
</div>

<div class="text-center  text-info display-3">
Le résultat du produit  =   {{   $produit  }}
</div>

<div class="text-center  text-info display-3">
Le résultat de la division  =   {{  $division }}
</div>
```

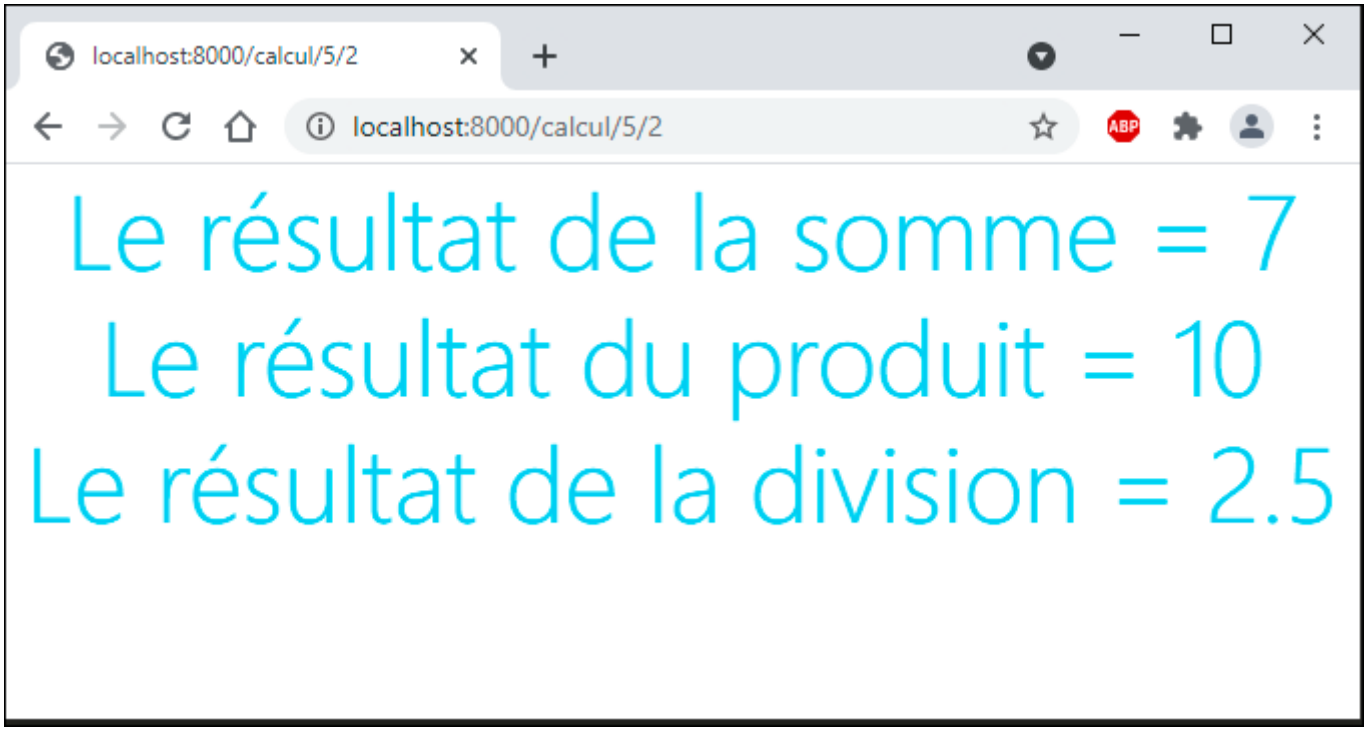
« Récupérer les données par la vue »

Le framework Laravel (Passage de paramètres a la vue)

« Passage de paramètres du contrôleur ver la Vue »

routes/web.php

```
Route::get('calcul/{a}/{b}', 'App\Http\Controllers\CalculController@calcul');
```



Le framework Laravel (Passage de paramètres a la vue)

« Passage d'un tableau du contrôleur ver la Vue »

Le passage d'un tableau se fait de la même façon que les autres variables.

```php
public function etudiant()
{

    $data=[];
    $data['nom']= 'Ahmed';
    $data['prenom'] = 'Moustafa';
    $data['age'] = 23;
    $data['modules'] = ['Développement Web', 'Réseaux', 'Optimisation'];

    return view('affichage', $data );
}
```

Le framework Laravel (Passage de paramètres a la vue)

« Passage d'un tableau du contrôleur ver la Vue »

```
<div class=" text-info display-4">
Nom :    {{  $nom  }}
</div>

<div class="  text-info display-4">
Prénom :    {{  $prenom  }}
</div>

<div class=" text-info display-4">
Age :    {{  $age  }}
</div>

<div class=" text-info display-4">
Modules :

        @foreach ($modules as $element)

        <li>{{ $element }} </li>

        @endforeach

</div>
```

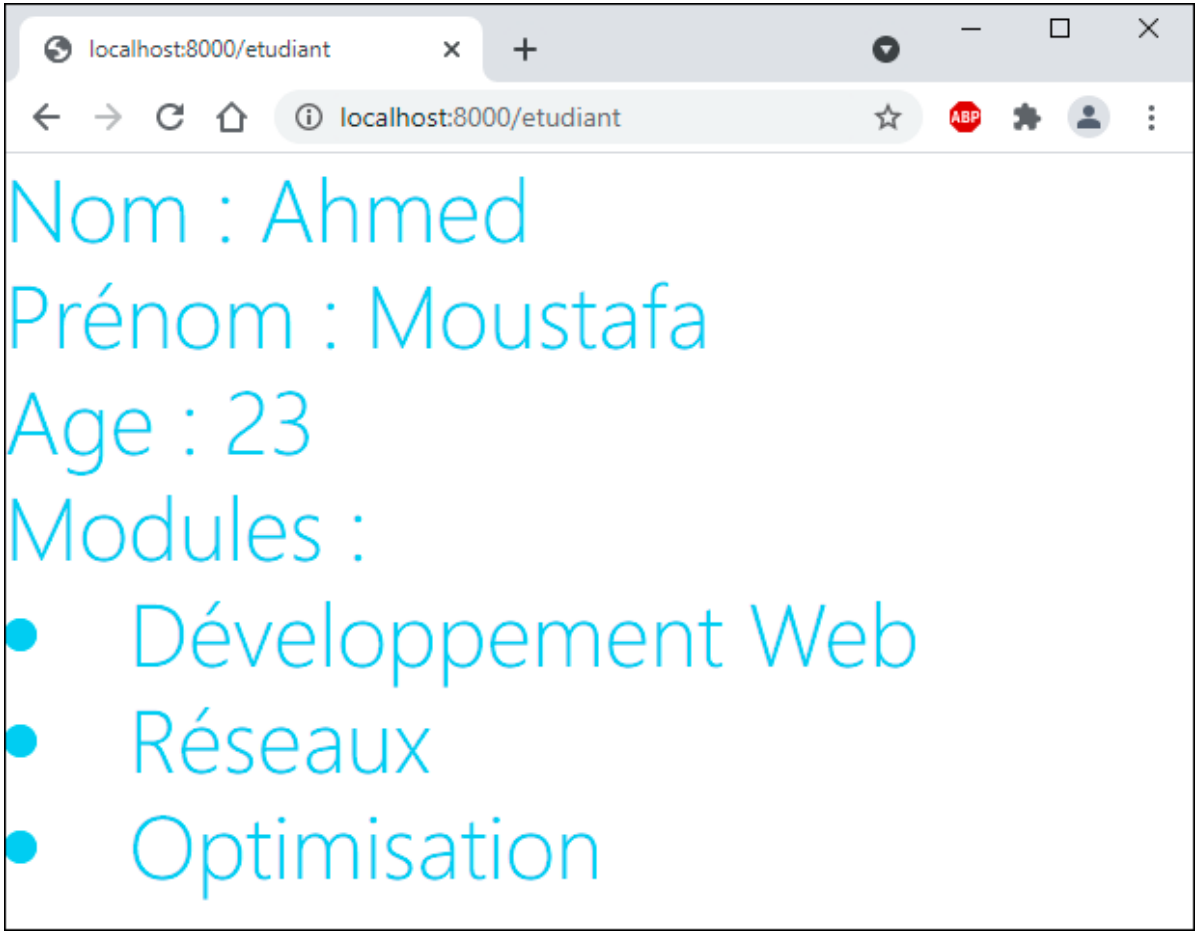
resources/views/affichage.blade.php

« « affichage du tableau

Le framework Laravel (Passage de paramètres a la vue)

« Passage d'un tableau du contrôleur ver la Vue »

```
Route::get('etudiant', 'App\Http\Controllers\MonControleur@etudiant');
```



Le framework Laravel (Le moteur de template Blade)

Le moteur de template blade (fichier avec l'extension « .blade.php ») permet de faire l'extension d'une page secondaire à partir d'une page principale.

Exemple :

resources/views/page_principale.blade.php

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
     <link rel="stylesheet" href="/css/bootstrap.min.css" >
   </head>

  <body>
    <div class = "container">

            @yield('contenu')

    </div>

     <!-- à la fin du body -->
    <script src="/js/bootstrap.bundle.min.js" > </script>

  </body>
</html>
```

Le framework Laravel (Le moteur de template Blade)

Le moteur de templete blade (fichier avec l'extension « .blade.php ») permet de faire l'extension d'une page secondaire a partir d'une page principale.

Exemple :

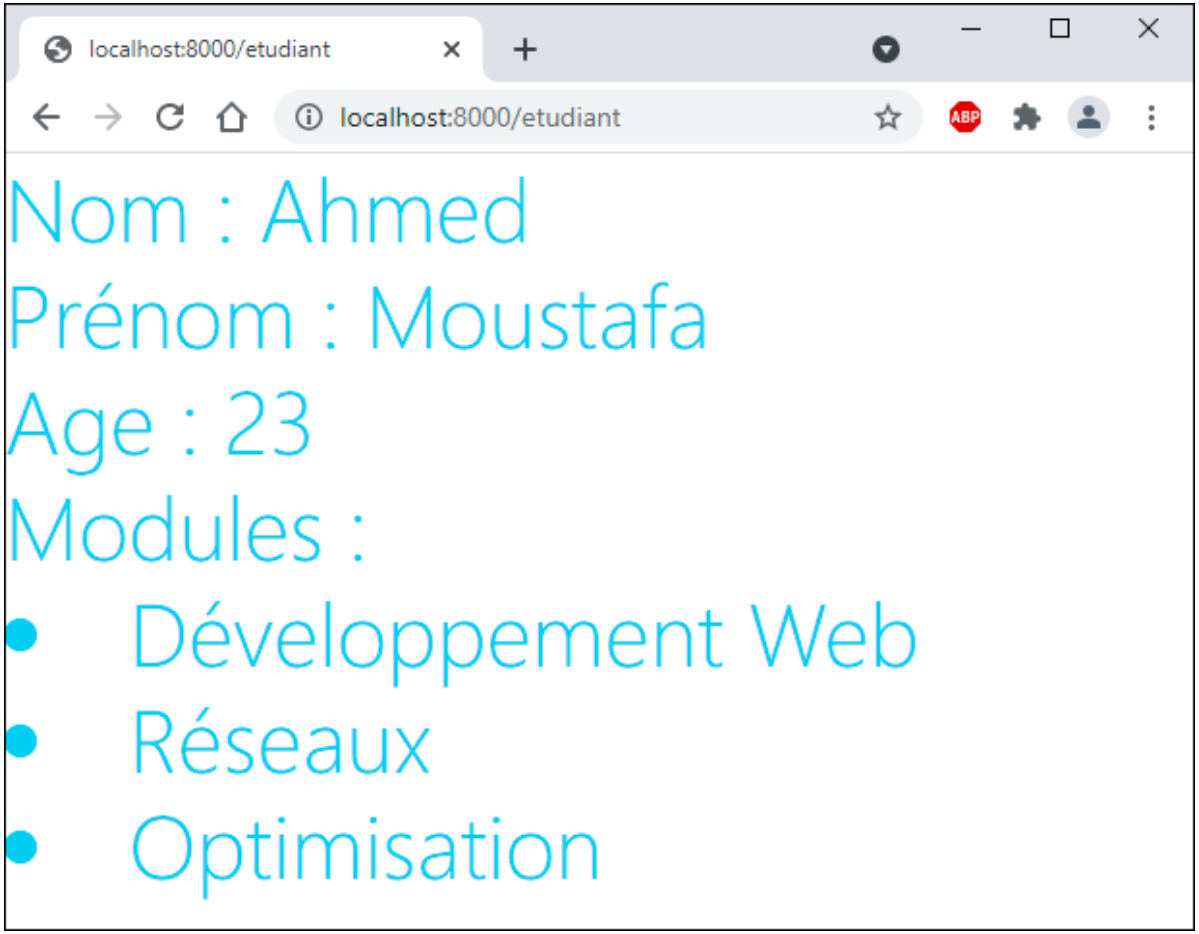
resources/views/affichage.blade.php

```
@extends ('page_principale')

@section('contenu')
            <div class=" text-info display-4">   Nom :    {{  $nom  }}  </div>

            <div class="  text-info display-4">  Prénom :    {{  $prenom  }}  </div>

            <div class=" text-info display-4">   Age :    {{  $age  }}  </div>

            <div class=" text-info display-4">   Modules :

                @foreach ($modules as $element)
                    <li>{{ $element }} </li>
                @endforeach
            </div>
@endsection
```

Le framework Laravel (Le moteur de template Blade)

Le moteur de template blade (fichier avec l'extension « .blade.php ») permet de faire l'extension d'une page secondaire a partir d'une page principale.

Exemple :



Le framework Laravel (Les formulaires)

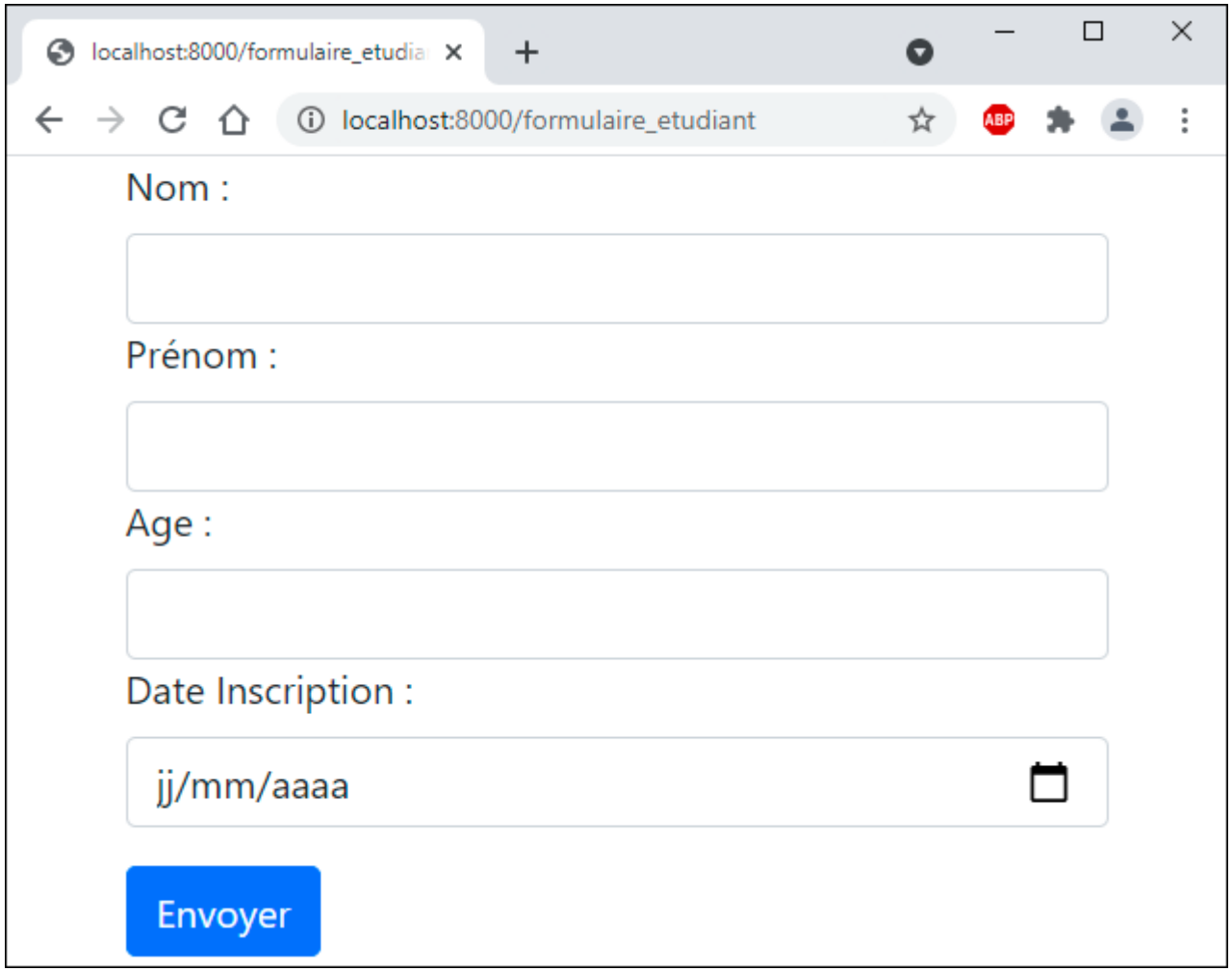
« Comment récupérer les paramètres d'un formulaire »

resources/views/formulaire_etudiant.blade.php

```
@extends ('page_principale')

@section('contenu')

<form method="POST" action="gestion_informations_etudiant">

    @csrf

<div class= "input-groupe mb-3 justify-content-center">
    <label for="id_nom" class="form-label">Nom :</label>
    <input type="text" class="form-control" id="id_nom" name="nom">

    <label for="id_prenom" class="form-label">Prénom :</label>
    <input type="text" class="form-control" id="id_prenom" name="prenom">

    <label for="id_age" class="form-label">Age :</label>
    <input type="number" class="form-control" id="id_age" name="age">

    <label for="id_date_inscription" class="form-label">Date Inscription :</label>
    <input type="date" class="form-control" id="id_date_inscription" name="
    date_inscription">
</div>

    <button type="submit" class="btn btn-primary" name ="informations">Envoyer</button>
</form>

@endsection
```

Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »



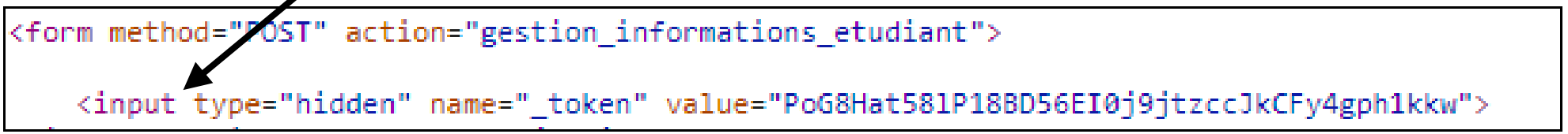
Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »

La partie **@csrf** est très important, elle permet de s'assurer que les informations parviennent d'un formulaire du site Laravel et évite le piratage. La partie @csrf est remplacer par un code généré automatiquement par Laravel. Ce code qui est retourné par les formulaire à chaque envoie des données.

resources/views/formulaire_etudiant.blade.php

```
<form method="POST" action="gestion_informations_etudiant">

    @csrf
```

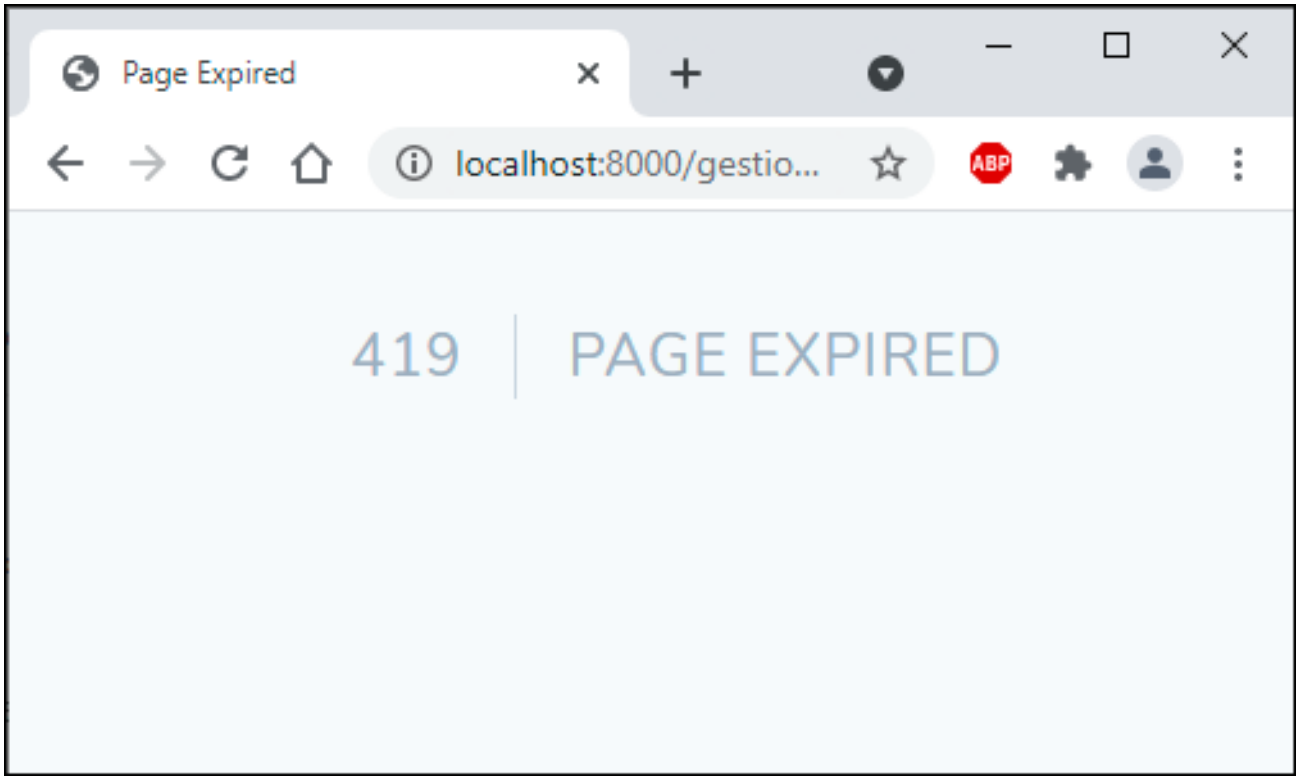


« « Navigateur

Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »

Le code **@csrf** a une période de validité, si celle-ci est dépassé Laravel ne traitera pas les données du formulaire.



Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »

On doit créer le contrôleur qui va gérer le formulaire, et aussi les routes nécessaires.

```
C:\dev_web\monprojet1>php artisan make:controller GestionEtudiantsController
Controller created successfully.
```

routes/web.php

```
Route::get('formulaire_etudiant', '
    App\Http\Controllers\GestionEtudiantsController@gestion_formulaire_etudiant');

Route::post('gestion_informations_etudiant', '
    App\Http\Controllers\GestionEtudiantsController@gestion_informations_etudiant');
```

Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »

On ajoute les fonctions « gestion_formulaire » et « gestion_informations_etudiant » dans le contrôleur.

App\Http\Controllers\GestionEtudiantController

```
public function gestion_formulaire_etudiant()

{
return view("formulaire_etudiant");

}
```

Le framework Laravel (Les formulaires)

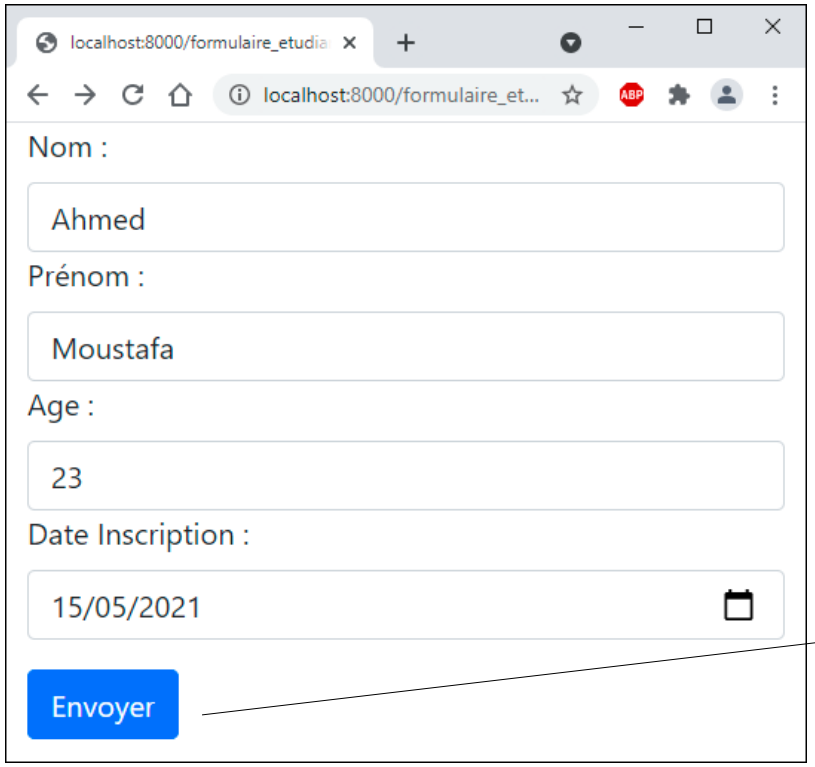
« Comment récupérer les paramètres d'un formulaire »

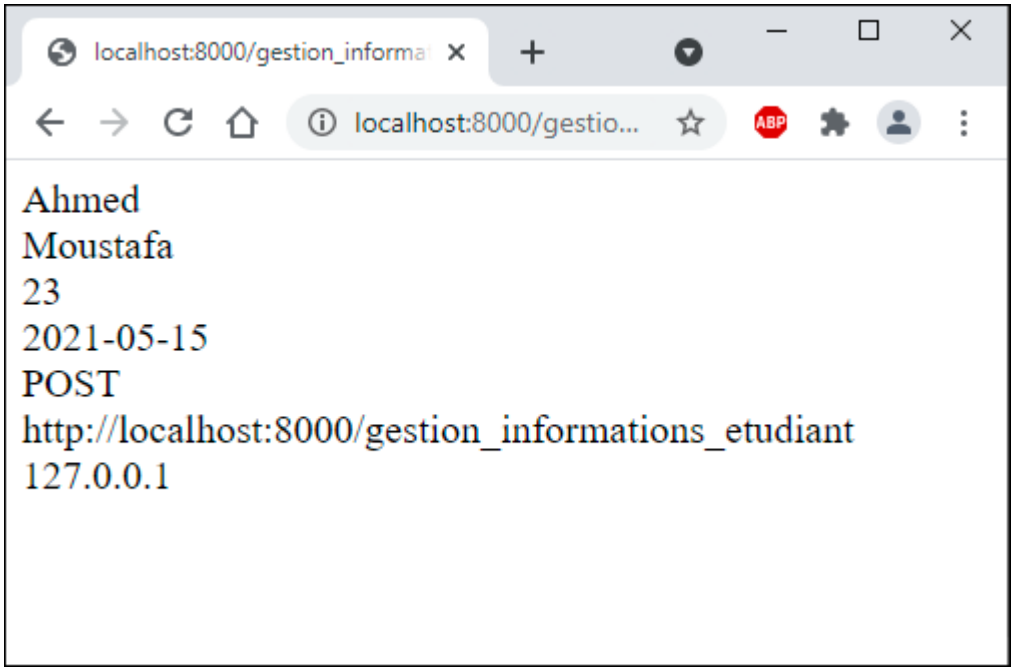
App\Http\Controllers\GestionEtudiantController

```php
public function gestion_informations_etudiant(Request  $request)

{
$informations   =  $request-> input('nom')
                . '</br>'
                . $request-> input('prenom')
                . '</br>'
                . $request-> input('age')
                . '</br>'
                . $request-> input('date_inscription')
                . '</br>'
                . $request-> method()
                . '</br>'
                . $request-> url()
                . '</br>'
                . $request-> ip();

return $informations;
}
```

Le framework Laravel (Les formulaires)

« Comment récupérer les paramètres d'un formulaire »





Le framework Laravel (Configuration de la base de données)

Configuration du fichier « /config/database.php »

config/database.php

```php
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'url' => env('DATABASE_URL'),
        'database' => env('DB_DATABASE', database_path('database.sqlite')),
        'prefix' => '',
        'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
    ],

    'mysql' => [
        'driver' => 'mysql',
        'url' => env('DATABASE_URL'),
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'unix_socket' => env('DB_SOCKET', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'prefix_indexes' => true,
        'strict' => true,
        'engine' => null,
        'options' => extension_loaded('pdo_mysql') ? array_filter([
            PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
        ]) : [],
    ],
```

Le framework Laravel (Configuration de la base de données)

1) Configuration du fichier « **/config/database.php** »

Exemple :

config/database.php

```
'default' => env('DB_CONNECTION', 'mysql'),
```

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

Le framework Laravel (Configuration de la base de données)

2) Configuration du fichier «.env »

Exemple :

.env

```
1   APP_NAME=Laravel
2   APP_ENV=local
3   APP_KEY=base64:erDsjLw43i09tEsQoE+2WHeECcbLKlF6E/bROEceRNQ=
4   APP_DEBUG=true
5   APP_URL=http://localhost
6
7   LOG_CHANNEL=stack
8   LOG_DEPRECATIONS_CHANNEL=null
9   LOG_LEVEL=debug
10
11  DB_CONNECTION=mysql
12  DB_HOST=127.0.0.1
13  DB_PORT=3306
14  DB_DATABASE=universite_mila
15  DB_USERNAME=root
16  DB_PASSWORD=''
17
```

configuration <<----------

Le framework Laravel (La migration des BDD)

Une migration est une manipulation sur le schéma de la BDD. Pour créer une nouvelle migration qui ajoute une nouvelle table à la base de données on exécute la commande :

```
php artisan make:migration nom_fichier_migration
```

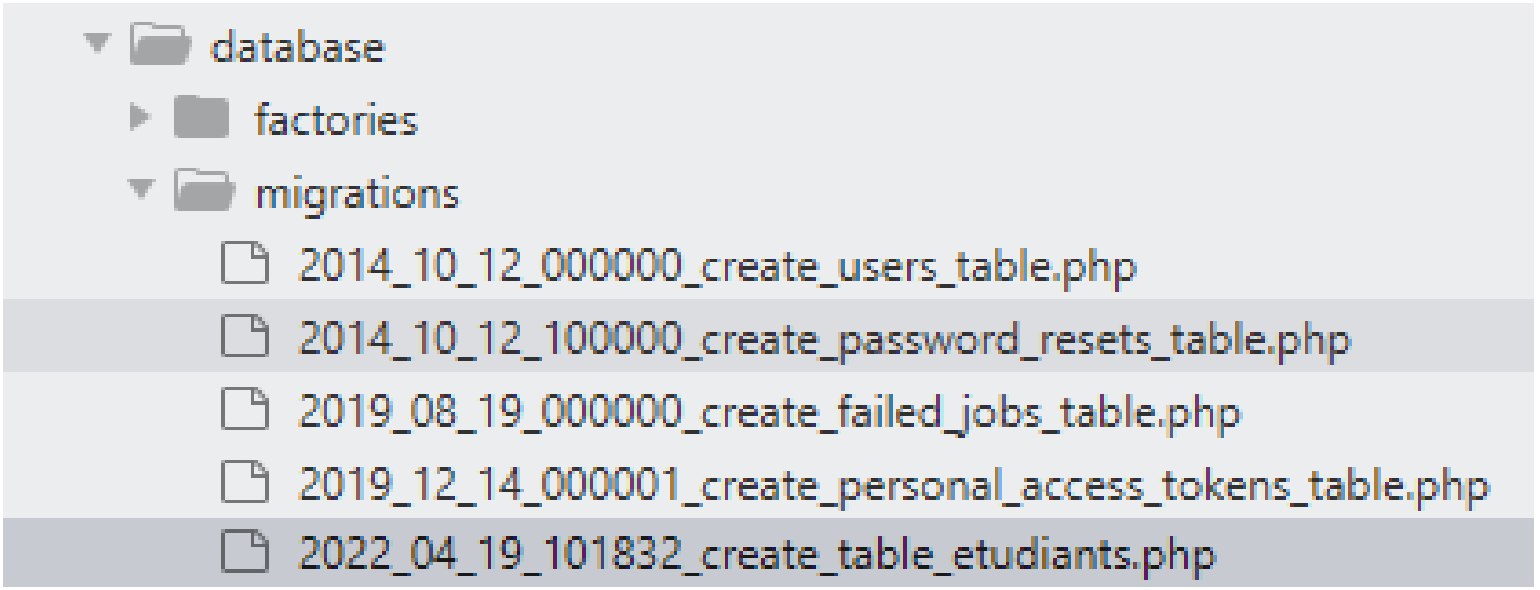
Les migration sont par défaut créés dans le dossier :
« **/app/database/migations/** »

Remarque : le nom de la table doit être en minuscule et avec un « **s** » a la fin.

Le framework Laravel (La migration des BDD)

Exemple :

```
c:\dev_web\monprojet1>php artisan make:migration create_table_etudiants
Created Migration: 2022_04_19_101832_create_table_etudiants
```



Le framework Laravel (La migration des BDD)

Le fichier « **/app/database/migations/**xyz_**create_table_etudiants.php**»

database\migrations\2022_04_19_094800_create_table_etudiants.php

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('table_etudiants', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

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

«« Instructions pour définir le schema

«« Instructions pour annuler la migration.

Le framework Laravel (La migration des BDD)

Ajouter des attributs au schéma de la table «table_ etudiants »

database\migrations\2022_04_19_094800_create_table_etudiants.php

```php
public function up()
{
    Schema::create('table_etudiants', function (Blueprint $table) {
        $table->id();

        $table->timestamps(); //insère les colones "updated_at"  et "created_at"

        $table->string('nom',50);//une colonne de type chaine de 50 caractères
        $table->string('prenom',80); //une colonne de type chaine de 80 caractères
        $table->integer('age');  //une colonne de type entier
        $table->date('date_inscription');  //une colonne de type date
        $table->string('specialite',50);//une colonne de type chaine de 50 caractères
    });
}
```

```php
public function down()
{
    Schema::dropIfExists('table_etudiants');
}
```

Le framework Laravel (La migration des BDD)

Pour exécuter les migrations on exécute la commande :

```
php artisan migrate
```

Pour annuler la dernière migrations on exécute la commande :

```
php artisan migrate:rollback
```

Pour annuler toutes les migrations on exécute la commande :

```
php artisan migrate:reset
```

Pour annuler toutes les migrations et les exécuter a nouveau on exécute la commande :

```
php artisan migrate:refresh
```

Pour supprimer toutes les tables et exécuter les migrations a nouveau on exécute la commande :

```
php artisan migrate:fresh
```

Le framework Laravel (La migration des BDD)

Exemple : création d'une table « **etudiants** » :

2) Exécution de la migration : `php artisan migrate`

```
c:\dev_web\monprojet1>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (50.90ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (44.54ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (39.39ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (73.85ms)
Migrating: 2022_04_19_094800_create_table_etudiants
Migrated:  2022_04_19_094800_create_table_etudiants (27.01ms)
```

Le framework Laravel (La migration des BDD)

Exemple : création d'une table « **etudiants** » :

2) Exécution de la migration : `php artisan migrate`

#	Nom	Type	Interclassement	Attributs
1	id	bigint(20)		UNSIGNED
2	created_at	timestamp		
3	updated_at	timestamp		
4	nom	varchar(50)	utf8mb4_unicode_ci	
5	prenom	varchar(80)	utf8mb4_unicode_ci	
6	age	int(11)		
7	date_inscription	date		
8	specialite	varchar(50)	utf8mb4_unicode_ci	

Le framework Laravel (La migration des BDD)

En cas d'erreur éditez le fichier « AppServiceProvider.php » comme suit :

app/Providers/AppServiceProvider.php

```php
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use Illuminate\support\Facades\schema;  <-------- Ajouter la ligne

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        Schema::defaultStringLength(191); <------- Ajouter la ligne
    }

```

```
C:\dev_web\monprojet1>php artisan migrate:fresh
```

Le framework Laravel (La migration des BDD)

Remarque importante :

Ne pas supprimer manuellement les fichiers de migration, si vous voulez refaire les migration, modifiez seulement les schémas et exécuter la commande « php artisan migrate:fresh » celle-ci vas supprimer toutes les tables et exécuter les migrations a nouveau par ordre de création.

```
c:\dev_web\monprojet1>php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (37.93ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (35.25ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (42.20ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (55.86ms)
Migrating: 2022_04_19_094800_create_table_etudiants
Migrated:  2022_04_19_094800_create_table_etudiants (17.72ms)
```

Le framework Laravel (La manipulation des BDD) : CRUD

Nous allons voir dans cette partie comment ont peut faire les différentes manipulation (CRUD) sur une base de données :

CRUD (**C**reate, **R**ead, **U**pdate, **D**elete)

Nous allons faire des manipulations sur la table « **table_etudiants** » vue précédemment en utilisant les modèles (Laravel Eloquent).

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent »

On peut manipuler les base de données en utilisant Eloquent (les modèles)
Ce dernier fait le mapping entre le modèle relationnelle et le modèle objet.

Nous allons premièrement créer un nouveau modèle sur la table « table_etudiants » appelé « Etudiants » en utilisant la commande suivante :

```
php artisan make:model Etudiants
```

Un nouveau modèle « Etudiants » sera créer dan le dossier « *app/Models* »

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent »

Le résultat est comme suit :

```
C:\dev_web\monprojet1>php artisan make:model Etudiants
Model created successfully.
```

app\Models\Etudiants.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Etudiants extends Model
{
    use HasFactory;


}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent »

On associe le modèle « Etudiant » a la « table_etudiants » comme suit :

app\Models\Etudiants.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Etudiants extends Model
{
    use HasFactory;


    protected $table = 'table_etudiants';


}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (CREATE)

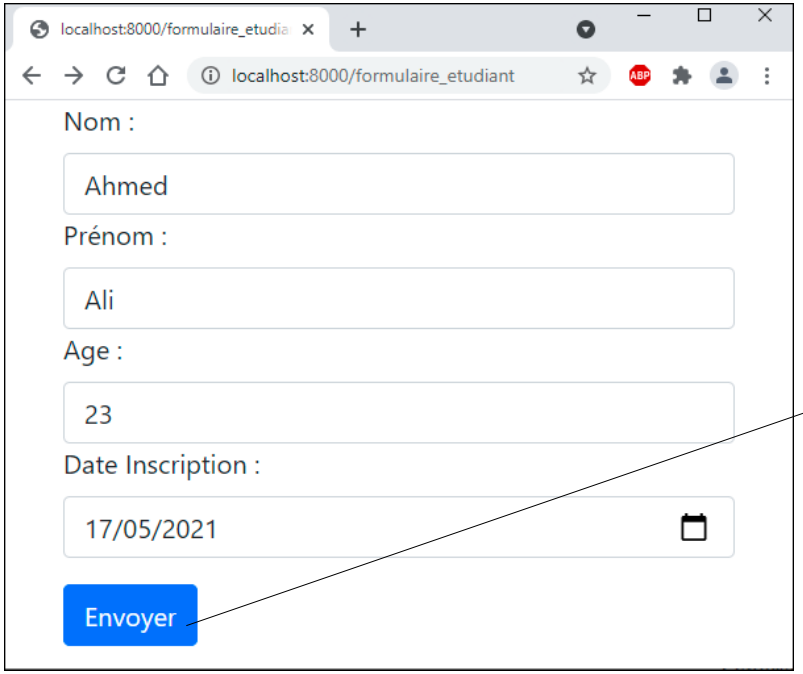
On insère les données avec le modèle comme suite : `use App\Models\Etudiants;`

```php
public function create_etudiant(Request  $request)

{

    $E = new Etudiants();

    $E->nom = $request-> input('nom');;
    $E->prenom = $request-> input('prenom');
    $E->age = $request-> input('age');
    $E->date_inscription = $request-> input('date_inscription');
    $E->save();

return '<h1> Etudiant Ajouté Avec Succès </h1>';
}
```

« created_at » et **« updated_at »** sont ajoutés automatiquement.

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (CREATE)



id	created_at	updated_at	nom	prenom	age	date_inscription
1	2021-05-17 05:27:58	2022-03-17 05:27:58	Ahmed	Ali	23	2022-03-17

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (UPDATE)

On modifie les données avec le modèle comme suite : `use App\Models\Etudiants;`

```php
public function update_etudiant(Request  $request)

{

    $id = $request-> input('id');  // Récupérer l'id de l'étudiant

    $E = Etudiants::find($id);  //rechercher l'étudiant avec son id


    $E->nom = $request-> input('nom');
    $E->prenom = $request-> input('prenom');
    $E->age = $request-> input('age');
    $E->date_inscription = $request-> input('date_inscription');

    $E->save();   // enregistrer les modifications

return '<h1> Etudiant Modifié Avec Succès </h1>';
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (DELETE)

On Supprime un enregistrement comme suite :

```php
use App\Models\Etudiants;
```

```php
public function delete_etudiant($id)

{

    $E = Etudiants::find($id);  //rechercher l'étudiant avec son id

    $E->delete();   // supprimer l'étudiant

return '<h1> Etudiant Supprimé Avec Succès </h1>';
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (READ)

On récupère les données avec le modèle comme suite : `use App\Models\Etudiants;`

```php
public function read_etudiant($id)

{

    $data = Etudiants::find($id);  //rechercher l'étudiant avec son id

    return view( 'gestion_etudiants_view' , compact('data'));
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

« Tous les étudiants »

```php
$data = Etudiants::all();

return view( 'gestion_etudiants_view' , compact('data'));
```

« Tous les étudiants avec age>25 »

```php
$data = Etudiants::where('age','>', 25)->get();
```

« Tous les étudiants avec age>25 et spécialité informatique»

```php
$data = Etudiants::where('age','>', 25)
        -> where('specialite','informatique') ->get();
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

« Le premier étudiant avec age>25 et spécialité informatique»

```
$data = Etudiants::where('age','>', 25)
                -> where('specialite','informatique')
                -> first();
```

« Tous les étudiants avec age>25 et spécialité informatique»

```
$data = Etudiants::where('age','>', 25)
                -> where('specialite','informatique')
                -> orgerBy('updated_at','desc') // ascendant('asc')  , descendant ('desc')
                -> get();
```

Prendre seulement les 10 premiers étudiants

```
$data = Etudiants::where('age','>', 25)
                -> where('specialite','informatique')
                -> orgerBy('nom','asc')
                -> take(10)  //prendre 10 étudiants
                -> get();
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

Modification

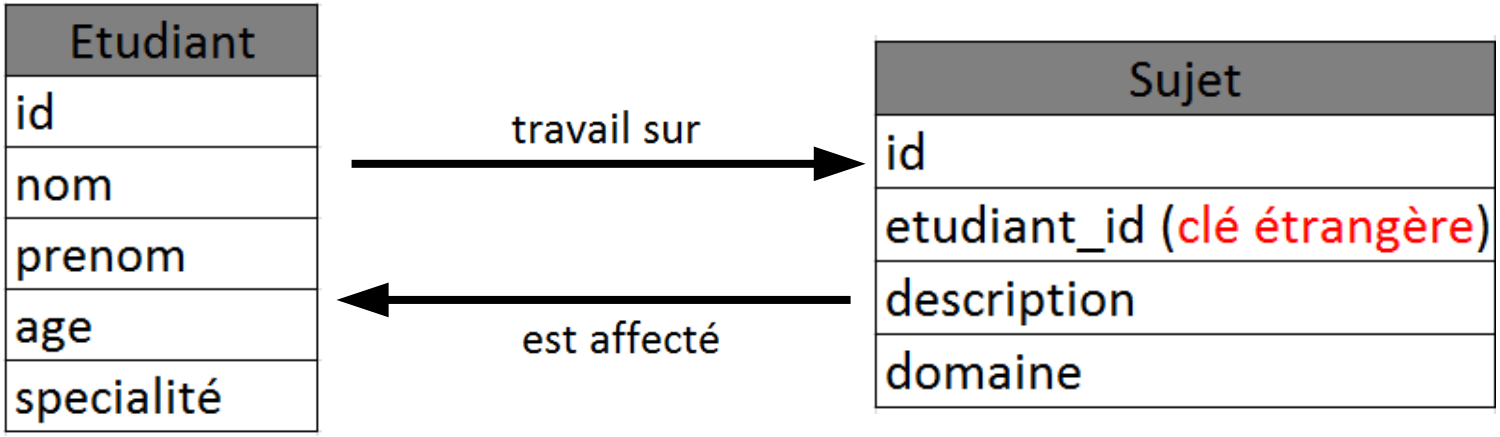
```
Etudiants::where('age','>', 25)
        -> where('specialite','informatique')
        -> orgerBy('nom','asc')
        -> take(2)
        -> update (['specialite' => 'Mathématique']);  //modifier la specialité
```

Suppression

```
Etudiants::where('age','>', 25)
        -> where('specialite','informatique')
        -> delete();  //supprimer les étudiants
```

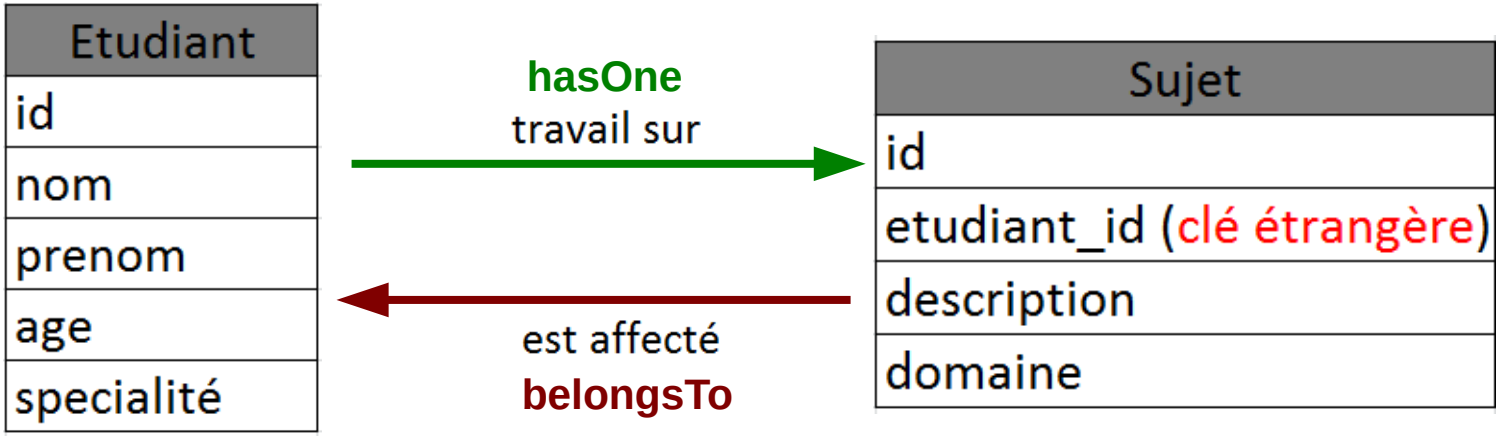
Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:1)



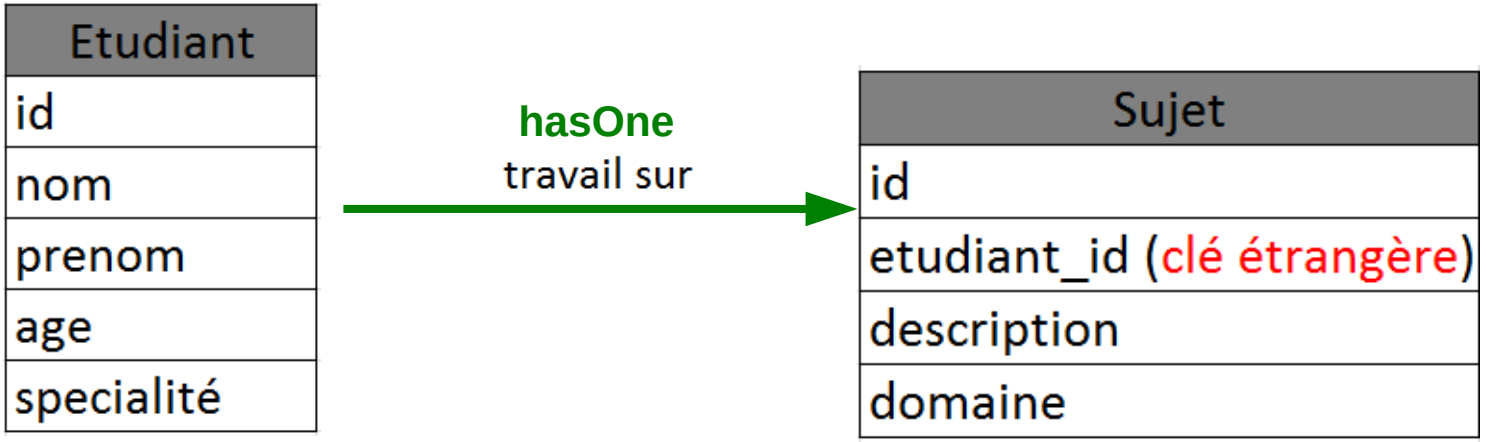
Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:1)



Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:1)




```php
class Etudiants extends Model
{
    use HasFactory;

    protected $table='table_etudiants';

public function sujet()
    {
        //return $this->hasOne(Sujets::class, 'foreign_key', 'local_key');

        return $this->hasone(Sujets::class , 'etudiant_id', 'id');
    }

}
```

Le modèle

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:1)

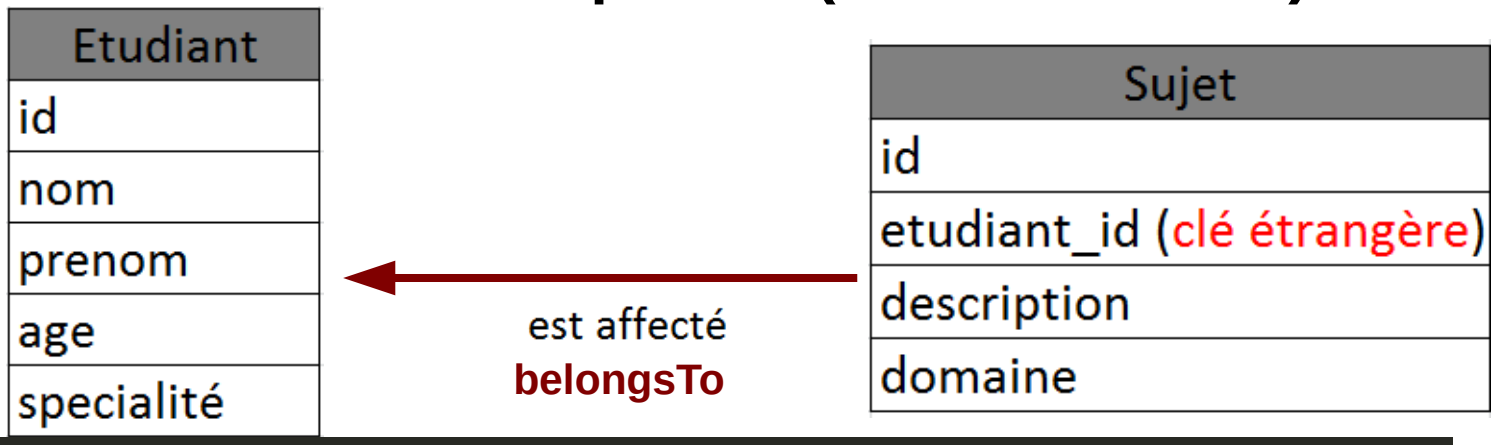
Sujet
id
etudiant_id (clé étrangère)
description
domaine

La migration

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

        $table->unsignedBigInteger('etudiant_id');
        $table->foreign('etudiant_id')->references('id')->on('table_etudiants');

        $table->string ('titre', 100);
        $table->string ('description', 100);

    });
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:1)




```php
class Sujets extends Model
{
    use HasFactory;

    protected $table ='table_sujets';

    public function etudiant()

    {
      //return $this->belongsTo(Etudiants::class, 'foreign_key', 'owner_key');

        return $this->belongsTo(Etudiants::class, 'etudiant_id', 'id');
    }

}
```

Le modèle

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

```php
use App\Models\Etudiants;
use App\Models\Sujets;
```

Trouver la description du sujet de l'étudiant qui a comme identifiant 9

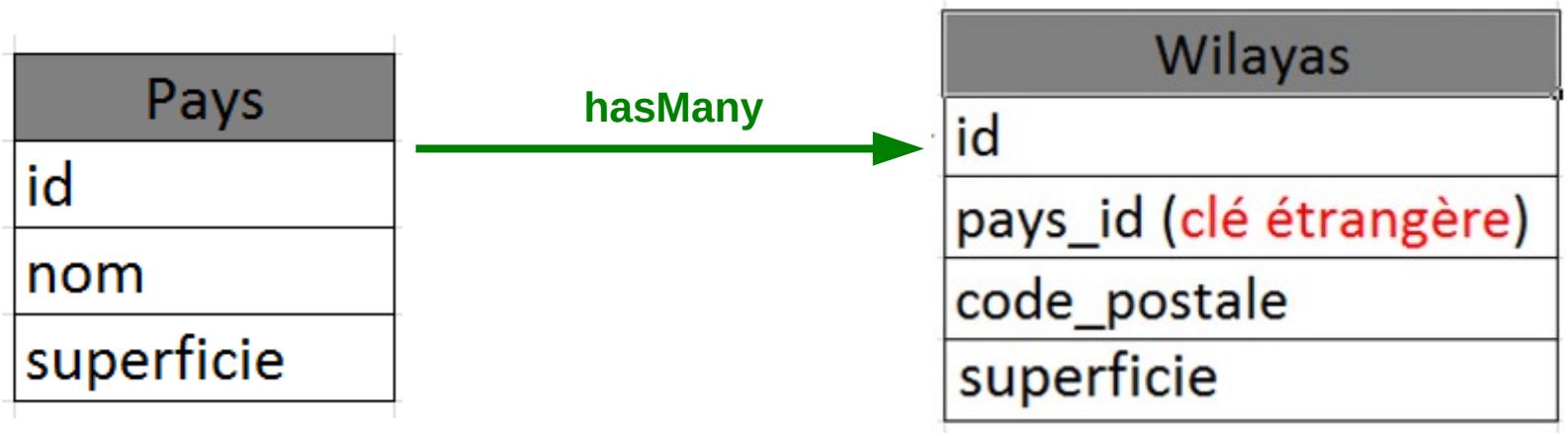
```php
$data = Etudiants::find(9)->sujet->description;
```

Trouver le nom de l'étudiant qui travail sur le premier sujet

```php
$data = Sujets::first()->etudiant->nom;
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation 1:n)




```php
class Pays extends Model
{
    use HasFactory;

    protected $table='table_pays';

public function wilayas()
    {
        //return $this->hasMany(Comment::class, 'foreign_key', 'local_key');
        return $this->haMany(Wilayas::class , 'pays_id', 'id');
    }
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

```php
use App\Models\Pays;
use App\Models\Wilayas;
```

Trouver les wilayas du pays qui a comme identifiant 2

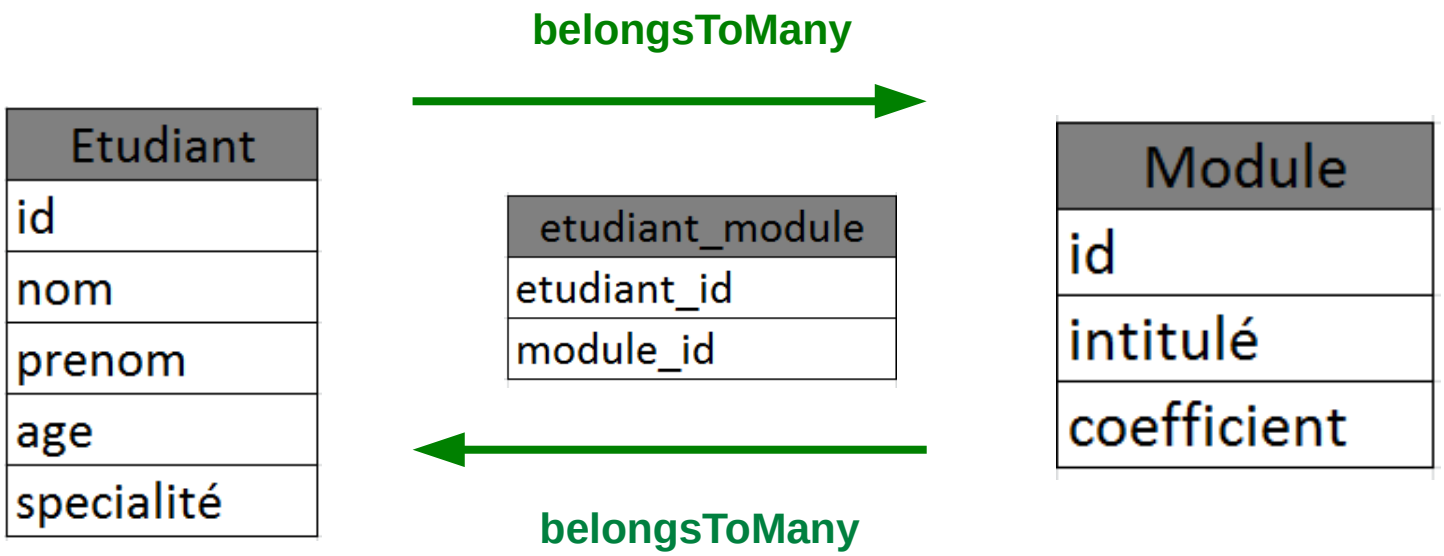
```php
$data = Pays::find(2)->wilays;
```

Trouver les wilayas qui ont une superficie > 250000 et qui son dans pays qui a comme identifiant 2

```php
$data = Pays::find(2)->wilays()->where('superficie', '>', 50000)->get();
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation n:n)



Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation n:n)

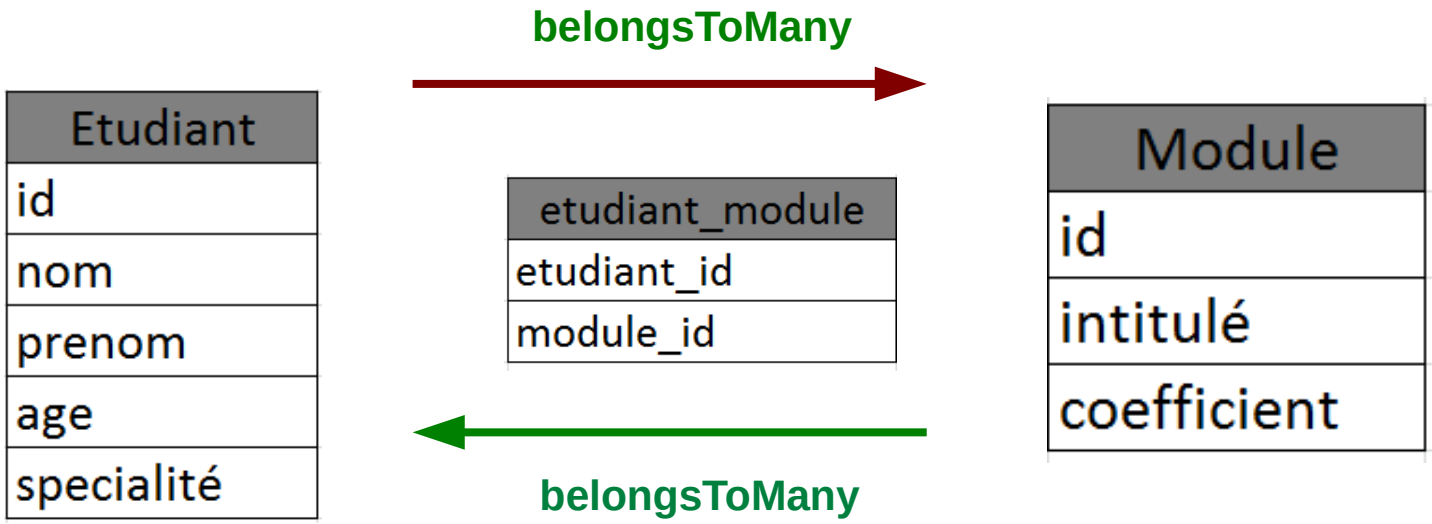
etudiant_module
etudiant_id
module_id

La migration

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

        $table->unsignedBigInteger('etudiant_id');
        $table->foreign('etudiant_id')->references('id')->on('table_etudiants');

        $table->unsignedBigInteger('module_id');
        $table->foreign('module_id')->references('id')->on('table_modules');

    });
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation n:n)

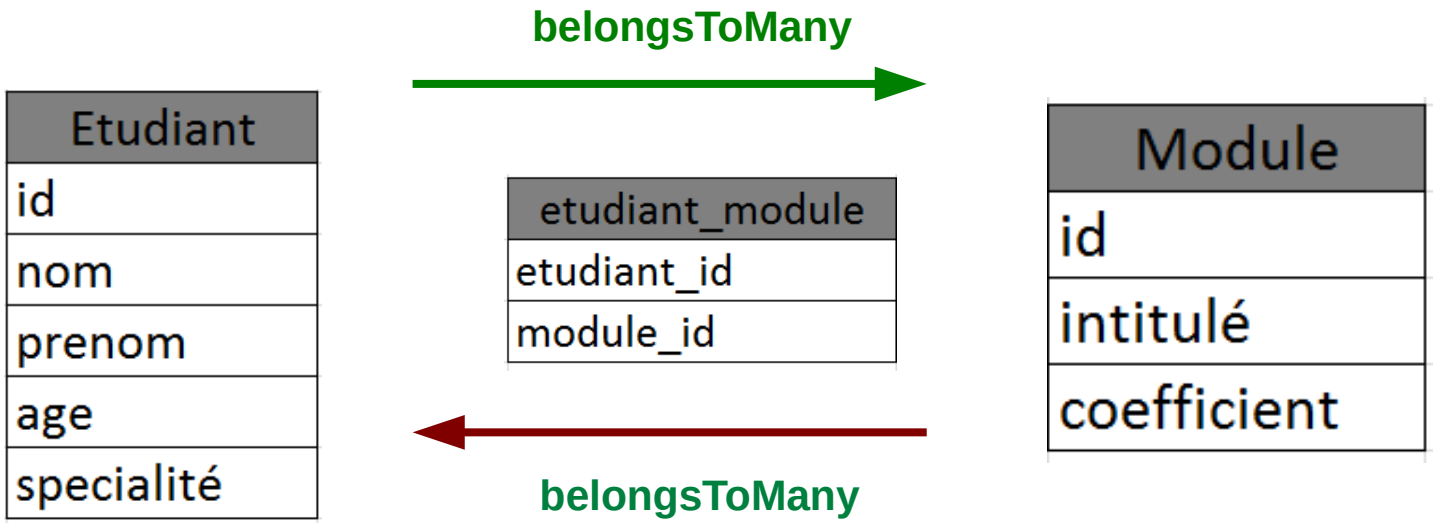


Le modèle Etudiants

```php
public function modules()
    {
        return $this->belongsToMany(Modules::class , 'table_etudiants_modules', '
            etudiant_id', 'module_id');
    }
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (La relation n:n)



Le modèle Modules

```php
public function etudiants()
{
    return $this->belongsToMany(Etudiants::class , 'table_etudiants_modules', '
        module_id', 'etudiant_id');
}
```

Le framework Laravel (La manipulation des BDD) : CRUD

« Laravel Eloquent » (Exemples de Manipulations)

```php
use App\Models\Etudiants;
use App\Models\Modules;
use App\Models\Etudiants_Modules;
```

Tous les module de l'étudiant qui a comme identifiant 13 ordonnés par coefficient

```php
$data = Etudiants::find(13)->modules()->orgerBy('coefficient','asc')->get();
```

Tous les étudiant qui apprennent le premier module

```php
$data = Modules::first->etudiants;
```

Le framework Laravel (La validation de formulaires)

La validation de formulaire permet de vérifier la cohérence les données d'un formulaire avant de les sauvegarder dans la base de données et cela en définissant un ensemble de règle à respecter.

On va prendre comme exemple l'ajout d'un nouveau étudiant dans la base de données :

```
function create_etudiant(Request $request) {

$E = new Etudiants();
$E->nom =  $request->input('no');
$E->prenom =  $request->input('pr');
$E->age =  $request->input('ag');
$E->date_inscription = $request->input('di');
$E->specialite =  $request->input('sp');

$E->save();

return redirect('/page_gestion_etudiants');
}
```

Le framework Laravel (La validation de formulaires)

Avant d'ajouter un nouveau étudiant on doit vérifier si les données du formulaire sont valides comme suit :

app/Http/Controllers/EtudiantsController.php

```php
function create_etudiant(Request $request) {

 $request->validate([
'no' => 'required|max:255',
'pr' => 'required',
'ag' => 'required | numeric',
'di' => 'date',
]);


$E = new Etudiants();
$E->nom =  $request->input('no');
$E->prenom =  $request->input('pr');
$E->age =  $request->input('ag');
$E->date_inscription = $request->input('di');
$E->specialite =  $request->input('sp');

$E->save();

return redirect('/page_gestion_etudiants');
}
```

« « « validation du formulaire

Le framework Laravel (La validation de formulaires)

Available Validation Rules

Below is a list of all available validation rules and their function:

Accepted	Ends With	Nullable
Active URL	Exclude If	Numeric
After (Date)	Exclude Unless	Password
After Or Equal (Date)	Exists (Database)	Present
Alpha	File	Prohibited
Alpha Dash	Filled	Prohibited If
Alpha Numeric	Greater Than	Prohibited Unless
Array	Greater Than Or Equal	Regular Expression
Bail	Image (File)	Required
Before (Date)	In	Required If
Before Or Equal (Date)	In Array	Required Unless
Between	Integer	Required With
Boolean	IP Address	Required With All
Confirmed	JSON	Required Without
Date	Less Than	Required Without All
Date Equals	Less Than Or Equal	Same
Date Format	Max	Size
Different	MIME Types	Sometimes

Remarque : pour connaître les possibilité disponible vous pouvez consulter la page :
https://laravel.com/docs/8.x/validation#available-validation-rules

Le framework Laravel (La validation de formulaires)

Les messages d'erreur sont en anglais. On peut cependant personnaliser les messages en modifiant le fichier : « ressources\lang\en\validation.php »

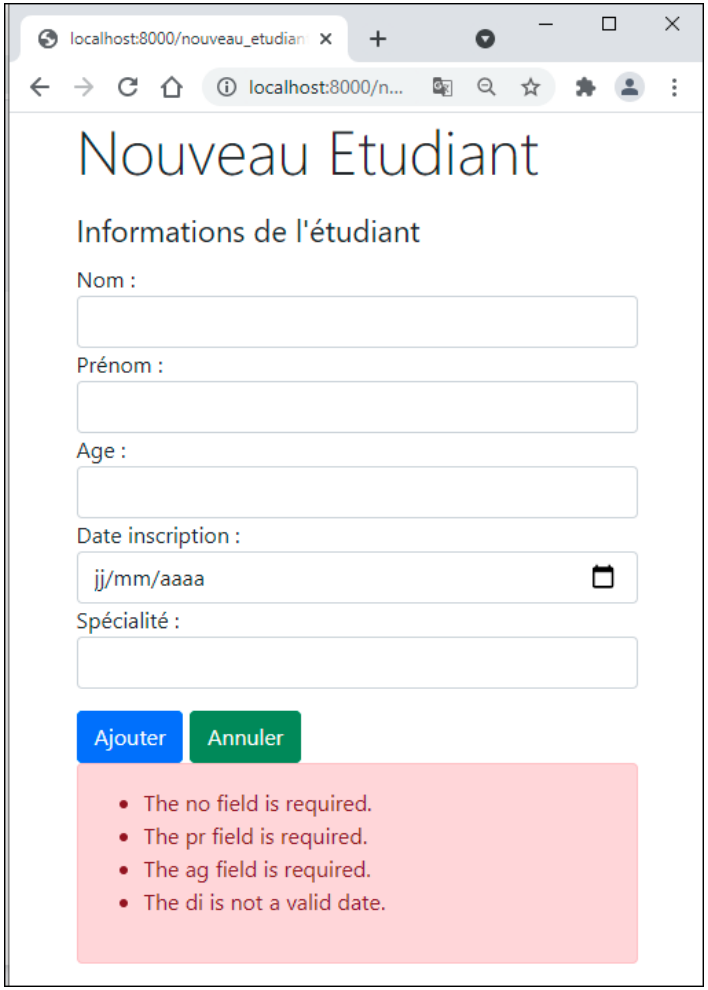
```
'accepted' => 'The :attribute must be accepted.',
'active_url' => 'The :attribute is not a valid URL.',
'after' => 'The :attribute must be a date after :date.',
'after_or_equal' => 'The :attribute must be a date after or equal to :date.',
'alpha' => 'The :attribute must only contain letters.',
'alpha_dash' => 'The :attribute must only contain letters, numbers, dashes and
    underscores.',
'alpha_num' => 'The :attribute must only contain letters and numbers.',
'array' => 'The :attribute must be an array.',
'before' => 'The :attribute must be a date before :date.',
'before_or_equal' => 'The :attribute must be a date before or equal to :date.',
'between' => [
    'numeric' => 'The :attribute must be between :min and :max.',
    'file' => 'The :attribute must be between :min and :max kilobytes.',
    'string' => 'The :attribute must be between :min and :max characters.',
    'array' => 'The :attribute must have between :min and :max items.',
],
'boolean' => 'The :attribute field must be true or false.',
'confirmed' => 'The :attribute confirmation does not match.',
'date' => 'The :attribute is not a valid date.',
'date_equals' => 'The :attribute must be a date equal to :date.',
```

Le framework Laravel (La validation de formulaires)

Pour afficher les messages d'erreurs on modifie la vue du formulaire « nouveau_etudiant_view.blade.php » en ajoutant le morceau de code suivant :

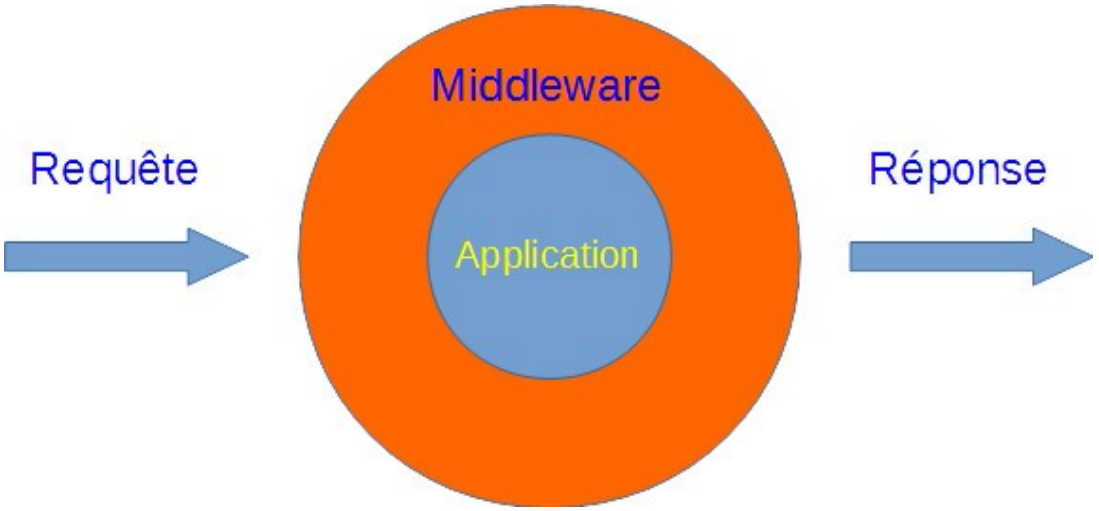
resources/views/nouveau_etudiant_view.blade.php

```
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```



Le framework Laravel (Les Middleware)

Les Midelware permettent de filtrer les raquettes HTTP ver l'application.



Plusieurs Middleware sont inclus dans Laravel , il se trouvent dans le dossier : « **app/Http/Middleware** »
Pour créer un nouveau Middleware on exécute la commande suivante :

```
php artisan make:middleware nom_middleware
```

Le framework Laravel (Les Middleware)

Exemple : Un Middleware pour vérifier l'adresse Ip d'un utilisateur :

```
php artisan make:middleware VerifierIp
```

Http\Middleware\VerifierIp.php

```php
class VerifierIp
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        return $next($request);
    }
}
```

Le framework Laravel (Les Middleware)

On modifie la fonction « handle » du Middleware comme suit :

Http\Middleware\VerifierIp.php

```php
public function handle(Request $request, Closure $next)
{

if (strcmp(  $request->Ip() , '192.0.0.20') !== 0)
    dd ('Accès Interdit');

else
    return $next($request);
}
```

Dans ce cas si l'utilisateur n'a pas l'adresse ip : 192.0.0.20 on luis affiche le message **« Accès interdit »**. La fonction **dd()** affiche le message et termine l'exécution du scripte PHP. On peut cependant le redirigé var une autre page ou afficher un message d'erreur.

Le framework Laravel (Les Middleware)

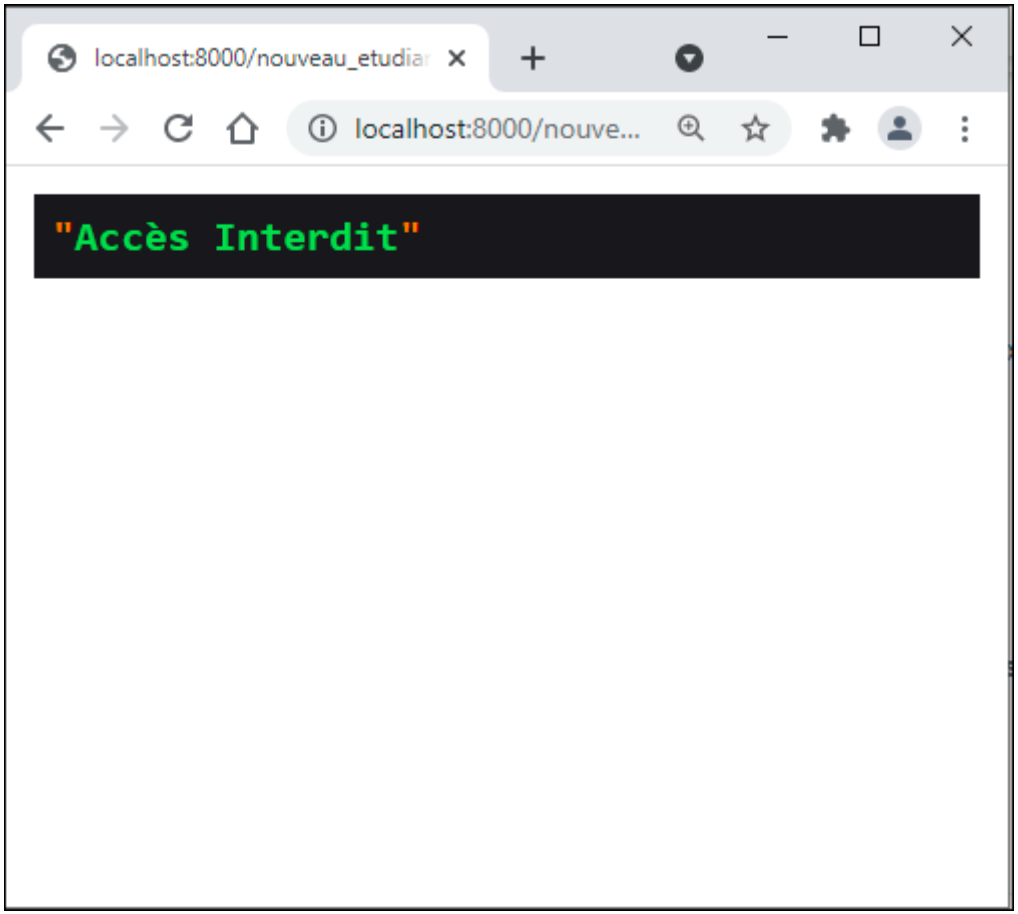
Si on veut que ce Middleware soit exécuté pour toutes les requêtes de l'application on ajoute la classe du Middleware au fichier « app/Http/Kernel.php » comme suit :

\app\Http\Kernel.php

```php
protected $middleware = [
    // \App\Http\Middleware\TrustHosts::class,
    \App\Http\Middleware\TrustProxies::class,
    \Fruitcake\Cors\HandleCors::class,
    \App\Http\Middleware\PreventRequestsDuringMaintenance::class,
    \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
    \App\Http\Middleware\TrimStrings::class,
    \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
                                                « « «  Notre Middleware
    \App\Http\Middleware\VerifierIp::class,
];
```

Le framework Laravel (Les Middleware)

Si on veut que ce Middleware soit exécuté pour toutes les requêtes de l'application on ajoute la classe du Middleware au fichier « app/Http/Kernel.php » comme suit :



Le framework Laravel (Les Middleware)

Si par contre on veut que ce Middleware soit exécuté pour une route précise on ajoute premièrement un alias dans le fichier « app/Http/Kernel.php » comme suit :

app\Http\Kernel.php

```php
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

    'vip' => \App\Http\Middleware\VerifierIp::class, « « «  Notre Middleware


];
```

Le framework Laravel (Les Middleware)

Si par contre on veut que ce Middleware soit exécuté pour une route précise on ajoute premièrement un alias dans le fichier « app/Http/Kernel.php » comme suit :

Puis on ajoute le Middleware à la route désiré comme suit :

```
Route::get('/supprimer_etudiant/{id}', '
    App\Http\Controllers\EtudiantsController@delete_etudiant')->middleware('vip');
```

Dans cet exemple on peut supprimer un étudiant seulement si on a l'adresse ip = 192.0.0.10

Le framework Laravel (Les Middleware)

Pour protéger un contrôler avec le middleware on lui ajoute une méthode « __construct » . Dans l'exemple suivant je protège le contrôleur « **Etudiant_Controller** » :

```
class EtudiantsController extends Controller
{
    //

public function __construct()
{

  $this->middleware('vip');
}
```

Si l'utilisateur n'a pas l'adresse n'a pas l'adresse ip : 192.0.0.10 on lui affiche le message **« Accès interdit ».**

Le framework Laravel (L'authentification)

L'authentification permet de protéger des pages du site web par mot de passe. Laravel intègre un système d'authentification basé sur les adresses mail et les mots de passes , il suffit simplement de l'installer comme suit :

1) Installer npm , vous pouvez utiliser le lien suivant :

https://www.npmjs.com/get-npm

Le framework Laravel (L'authentification)

L'authentification permet de protéger des pages du site web par mot de passe. Laravel intègre un système d'authentification basé sur les adresses mail et les mots de passes , il suffit simplement de l'installer comme suit :

2) Exécuter successivement les commande suivantes :

```
composer require laravel/jetstream
```

```
php artisan jetstream:install livewire
```

```
npm install
```

```
npm run dev
```

```
php artisan migrate
```

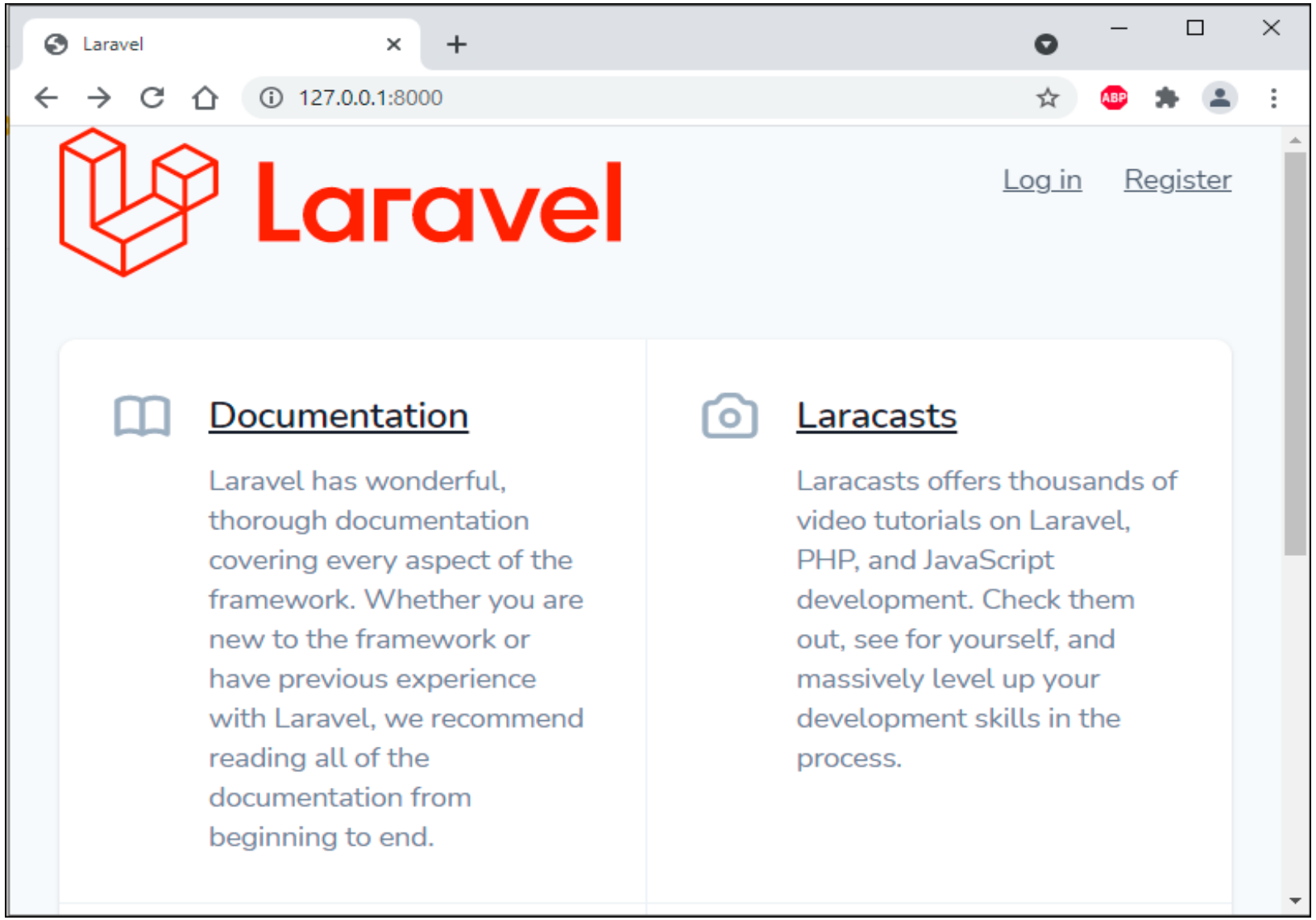
Le framework Laravel (L'authentification)

L'authentification permet de protéger des pages du site web par mot de passe. Laravel intègre un système d'authentification basé sur les adresses mail et les mots de passes , il suffit simplement de l'installer comme suit :

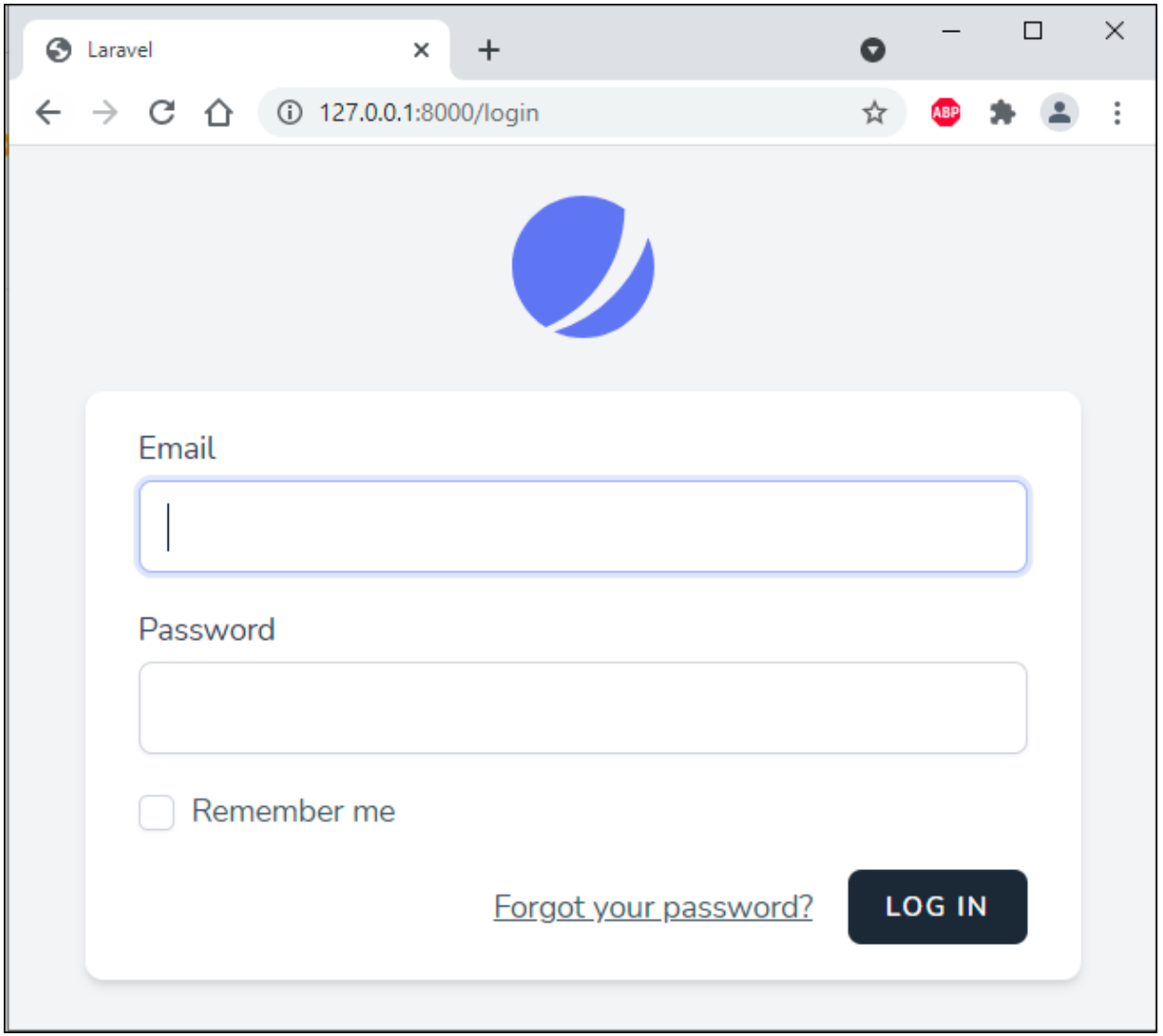
3) Démarrer le serveur Laravel et accéder a la page d'accueil

```
C:\dev_web\exemple_CRUD>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
```

Le framework Laravel (L'authentification)

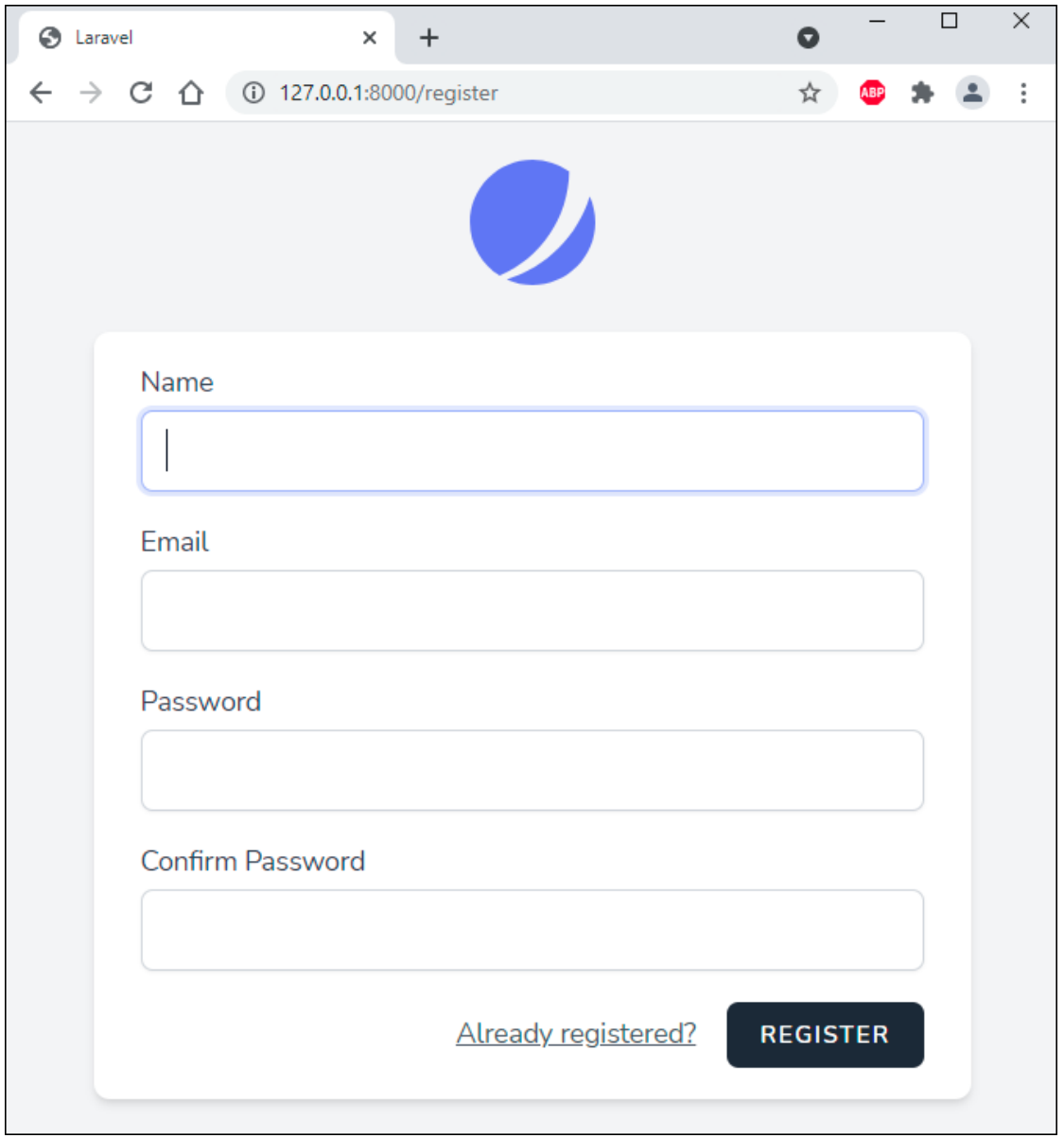


Le framework Laravel (L'authentification)



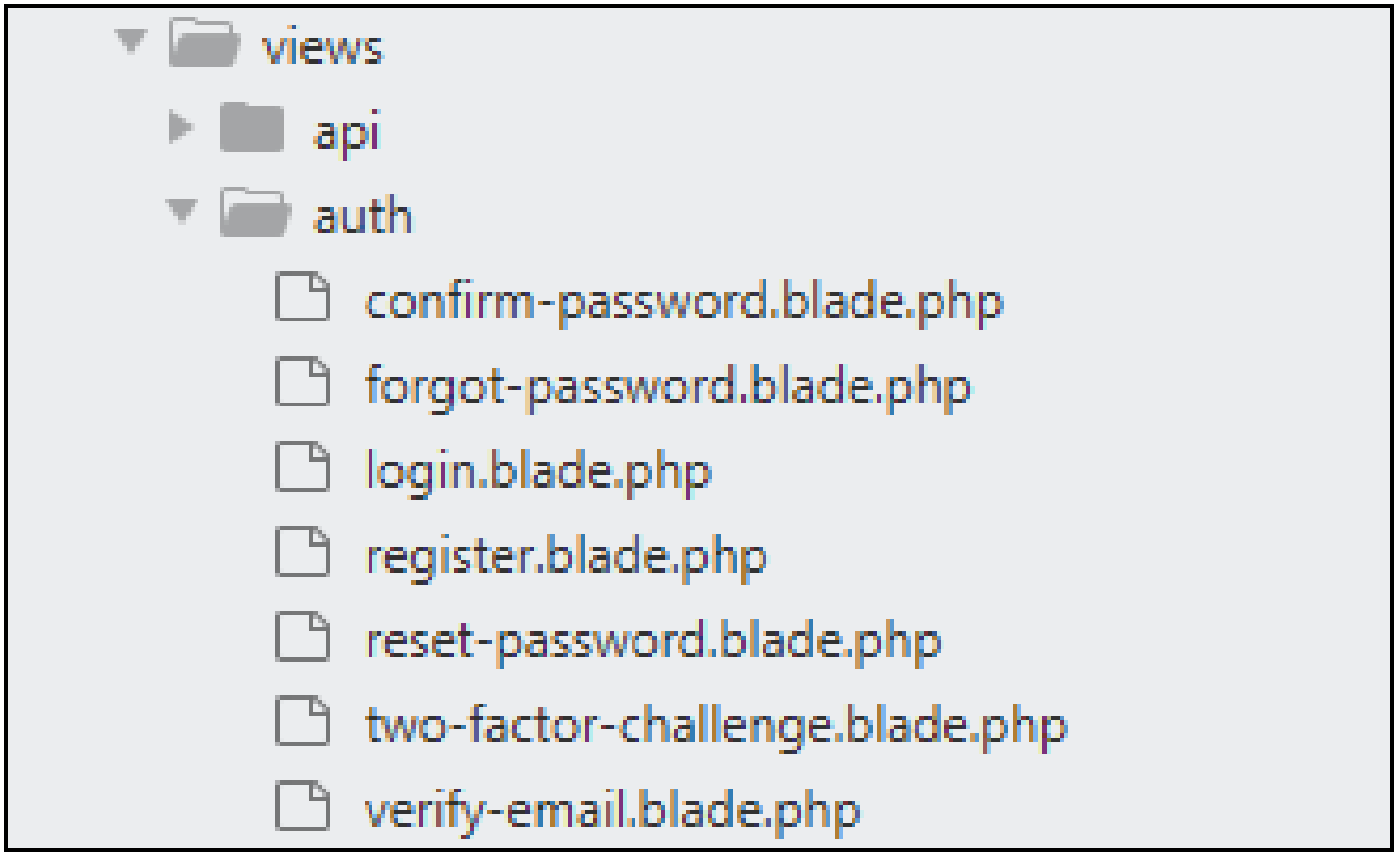
« « « Login

Le framework Laravel (L'authentification)



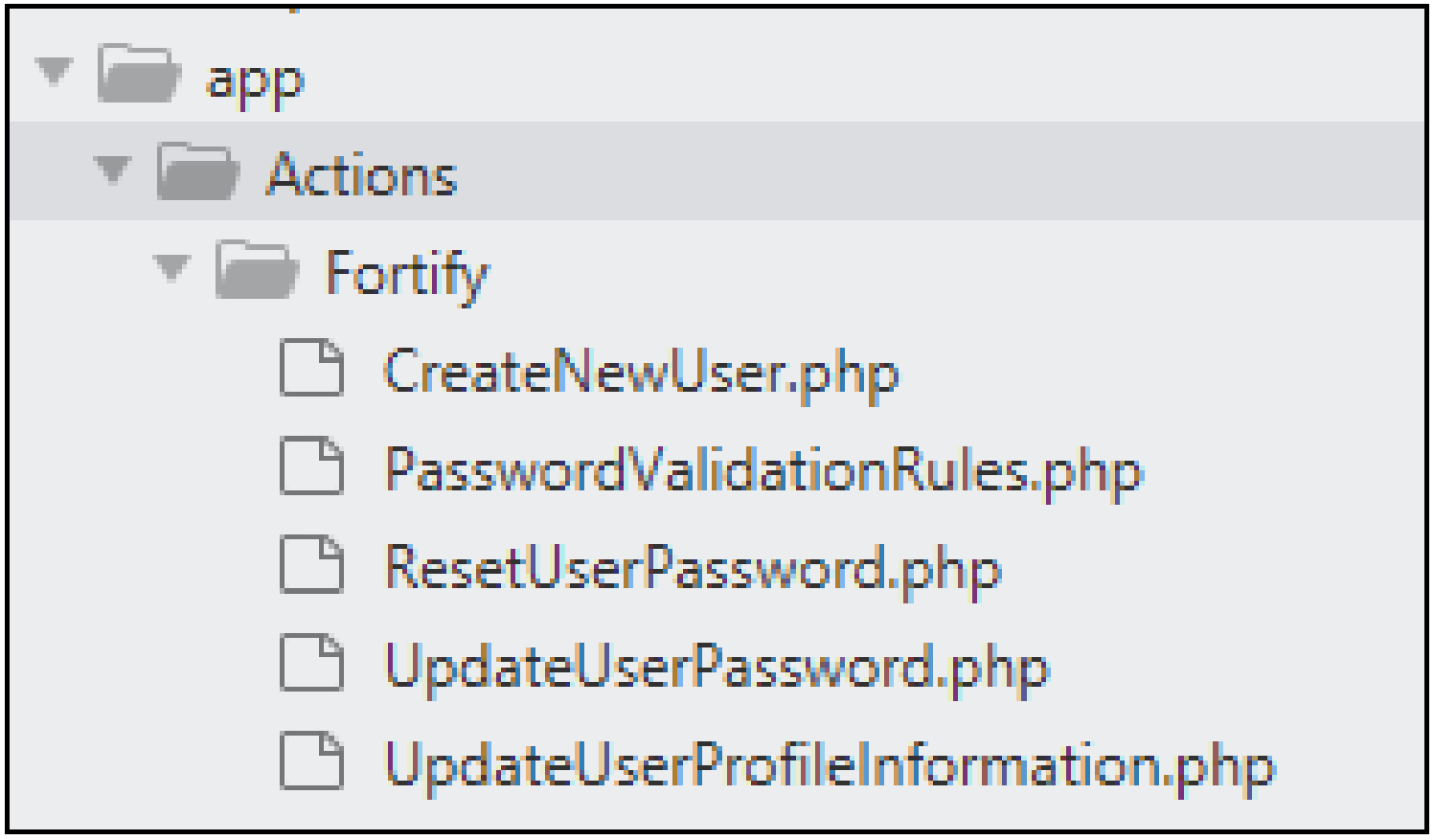
« « «Register

Le framework Laravel (L'authentification)



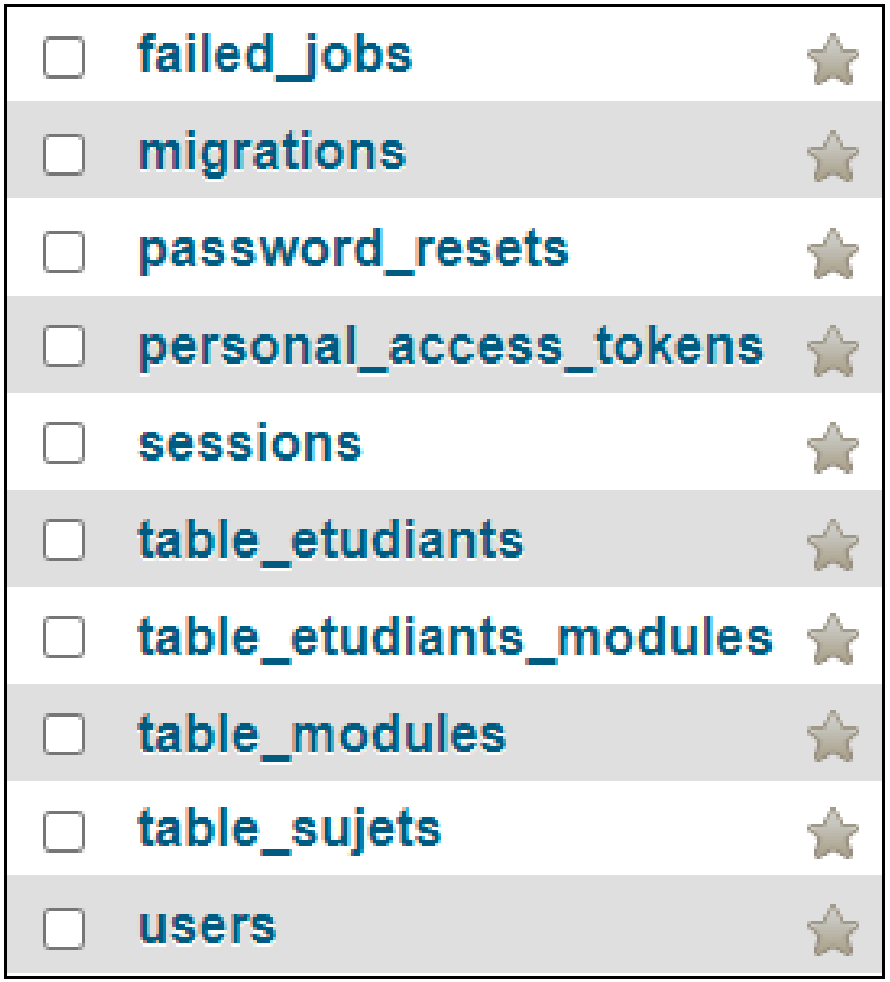
« « « Le Vues

Le framework Laravel (L'authentification)



« « « Les Controleurs

Le framework Laravel (L'authentification)



« « « Les tables

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

On veut contrôler l'accès aux fonctionnalités du site web selon le rôle des utilisateurs. Par exemple on aura trois rôle différents :

- Un rôle Administrateur : il aura toutes les autorisations
- Un rôle Enseignant
- Un rôle Étudiant

Une méthode simple consiste à ajouter un champ « Rôle » a la table user et contrôler les routes par les middlewares. Pour cela on suivra les étapes suivantes :

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

1) Modification de la table User : On modifie premièrement la migration de la table user en ajoutant la colonnes **user_role** comme suit :

database\migrations\2014_10_12_000000_create_users_table.php

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->foreignId('current_team_id')->nullable();
        $table->string('profile_photo_path', 2048)->nullable();
        $table->timestamps();

        $table->string('user_role');



    });
}
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

2) Exécuter la migration

```
php artisan migrate:fresh
```

1	id 🔑	bigint(20)
2	name	varchar(191)
3	email	varchar(191)
4	email_verified_at	timestamp
5	password	varchar(191)
6	two_factor_secret	text
7	two_factor_recovery_codes	text
8	remember_token	varchar(100)
9	current_team_id	bigint(20)
10	profile_photo_path	varchar(2048)
11	created_at	timestamp
12	updated_at	timestamp
13	user_role	varchar(191)

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

3) On modifie le modèle user comme suit :

app\Models\User.php

```php
protected $fillable = [
    'name',
    'email',
    'password',
    'user_role',
];
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

4) Ajouter un controlleur « GestionUtilisateursControlleur » et lui ajouter les fonctions nécessaires : `php artisan make:controller GestionUtilisateursController`

\app\Http\Controllers\GestionUtilisateursController.php

```php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class GestionUtilisateursController extends Controller
{

function espace_administrateur() {

return view('/espace_administrateur_view');

    }

function espace_enseignant() {

return view('/espace_enseignant_view');

    }
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

4) Ajouter un contrôleur « GestionUtilisateursControlleur » et lui ajouter les fonctions nécéssaires :

\app\Http\Controllers\GestionUtilisateursController.php

```
function espace_etudiant() {

return view('/espace_etudiant_view');

    }


    function gestion_welcome() {

    return view('/page_inscription_view');
    }
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

4) Ajouter un controlleur « GestionUtilisateursControlleur » et lui ajouter les fonctions nécéssaires :

\app\Http\Controllers\GestionUtilisateursController.php

```php
function gestion_dashboard() {

if(Auth::check())
{

    if (Auth::user()->user_role === "Administrateur")
        return redirect('/espace_administrateur');
    else
    if (Auth::user()->user_role === "Enseignant")
        return redirect('/espace_enseignant');
    else
    if (Auth::user()->user_role === "Etudiant")
        return redirect('/espace_etudiant');
    else
        return redirect('/page_inscription');

}
else  return redirect('/page_inscriptions');

}
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

5) Ajouter les vues :

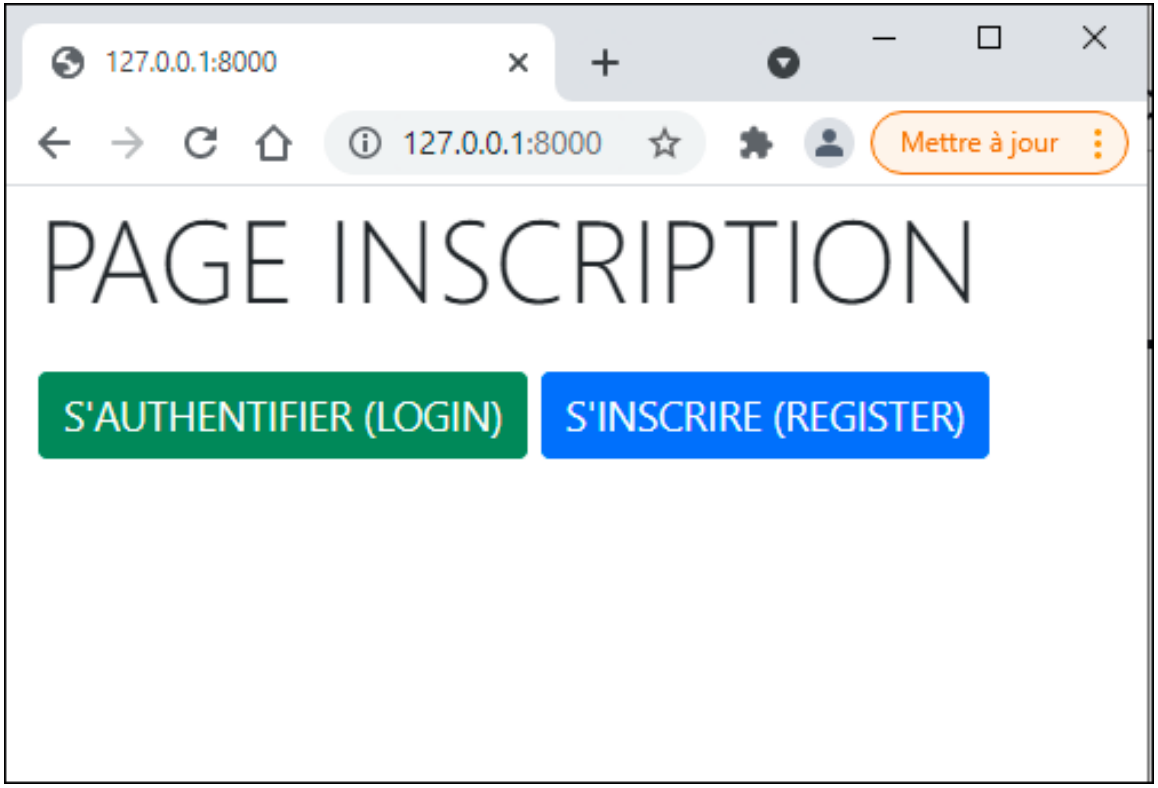
resources\views\page_inscription_view.blade.php

```
@extends('page_principale')

@section('contenu')

<div class="row  justify-content-center">
    <p class="display-2">PAGE INSCRIPTION</p>
</div>

<a href="/login" class="btn btn-success btn-sm ">S'AUTHENTIFIER (LOGIN)</a>

<a href="/register" class="btn btn-primary btn-sm">S'INSCRIRE (REGISTER)</a>

@endsection
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

5) Ajouter les vues :

resources\views\page_inscription_view.blade.php



Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

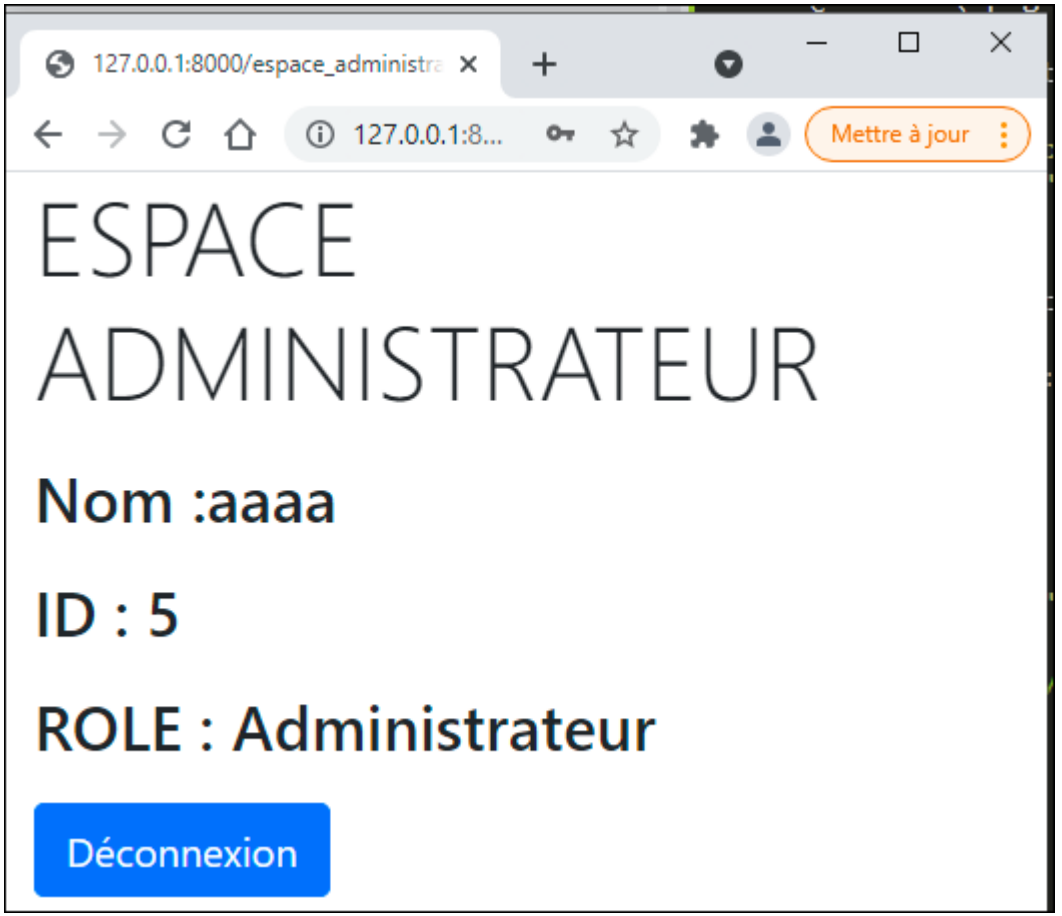
5) Ajouter les vues : resources\views\espace_administrateur_view.blade.php

```
@extends('page_principale')

@section('contenu')

<div class="row  justify-content-center">
    <p class="display-2">ESPACE ADMINISTRATEUR</p>
</div>

@if(Auth::check())
<h2>
    <p>  Nom :{{ Auth::user()->name }} </p>
    <p>  ID : {{ Auth::user()->id }}   </p>
    <p>  ROLE : {{ Auth::user()->user_role }}</p>
</h2>
@endif

<form method="POST" action="/logout">
    @csrf
     <input type="submit" class="btn btn-primary" name="logout"  value="Déconnexion"
     />
</form>

@endsection
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

5) Ajouter les vues :

resources\views\espace_administrateur_view.blade.php



Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

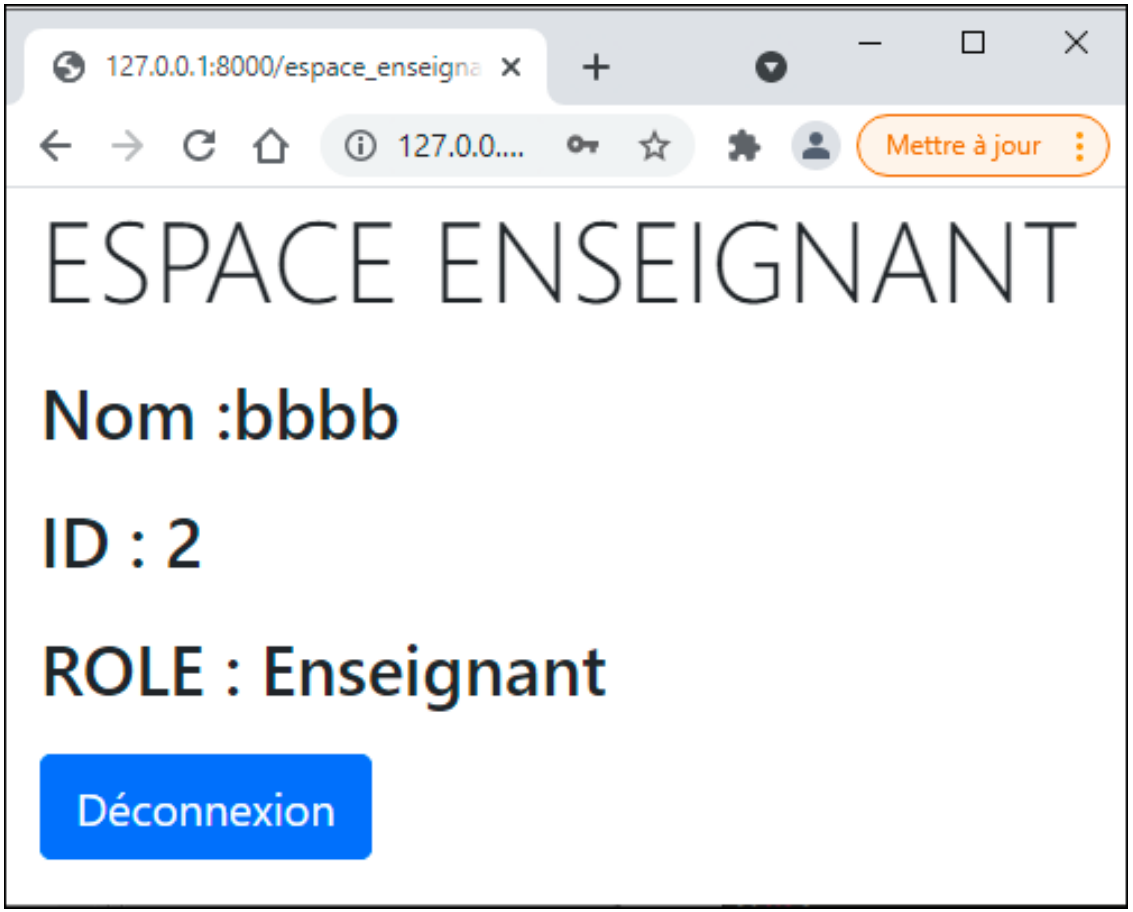
5) Ajouter les vues : resources\views\espace_enseignant_view.blade.php

```
@extends('page_principale')

@section('contenu')

<div class="row  justify-content-center">
    <p class="display-2">ESPACE ENSEIGNANT</p>
</div>

@if(Auth::check())
<h2>
    <p>  Nom :{{ Auth::user()->name }} </p>
    <p>  ID : {{ Auth::user()->id }}   </p>
    <p>  ROLE : {{ Auth::user()->user_role }}</p>
</h2>
@endif

<form method="POST" action="/logout">
    @csrf
     <input type="submit" class="btn btn-primary" name="logout"  value="Déconnexion"
     />
</form>

@endsection
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

5) Ajouter les vues :

resources\views\espace_enseignant_view.blade.php



Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

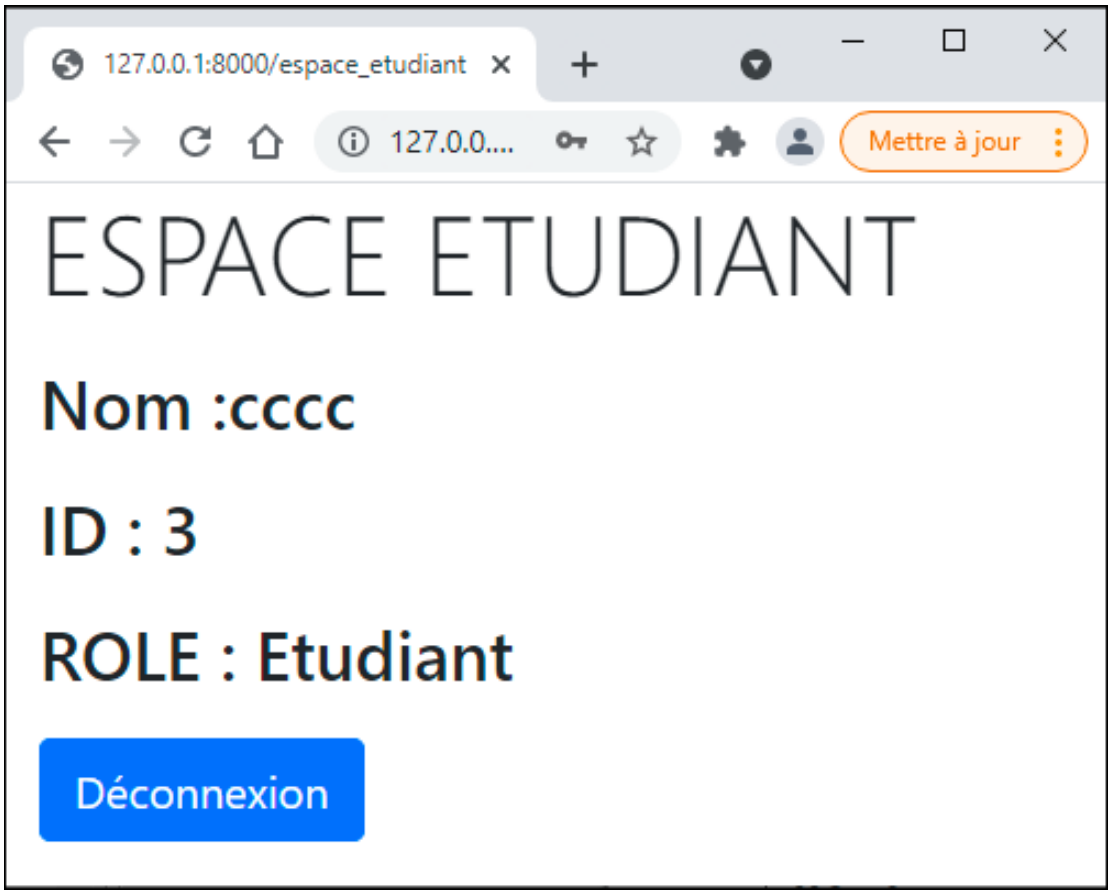
5) Ajouter les vues : resources\views\espace_etudiant_view.blade.php

```
@extends('page_principale')

@section('contenu')

<div class="row  justify-content-center">
    <p class="display-2">ESPACE ETUDIANT</p>
</div>

@if(Auth::check())
<h2>
    <p>  Nom :{{ Auth::user()->name }} </p>
    <p>  ID : {{ Auth::user()->id }}   </p>
    <p>  ROLE : {{ Auth::user()->user_role }}</p>
</h2>
@endif

<form method="POST" action="/logout">
    @csrf
     <input type="submit" class="btn btn-primary" name="logout"  value="Déconnexion"
     />
</form>

@endsection
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

5) Ajouter les vues :

resources\views\espace_etudiant_view.blade.php



Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

6) Modifier la vue du login en ajoutant la champ Rôle :

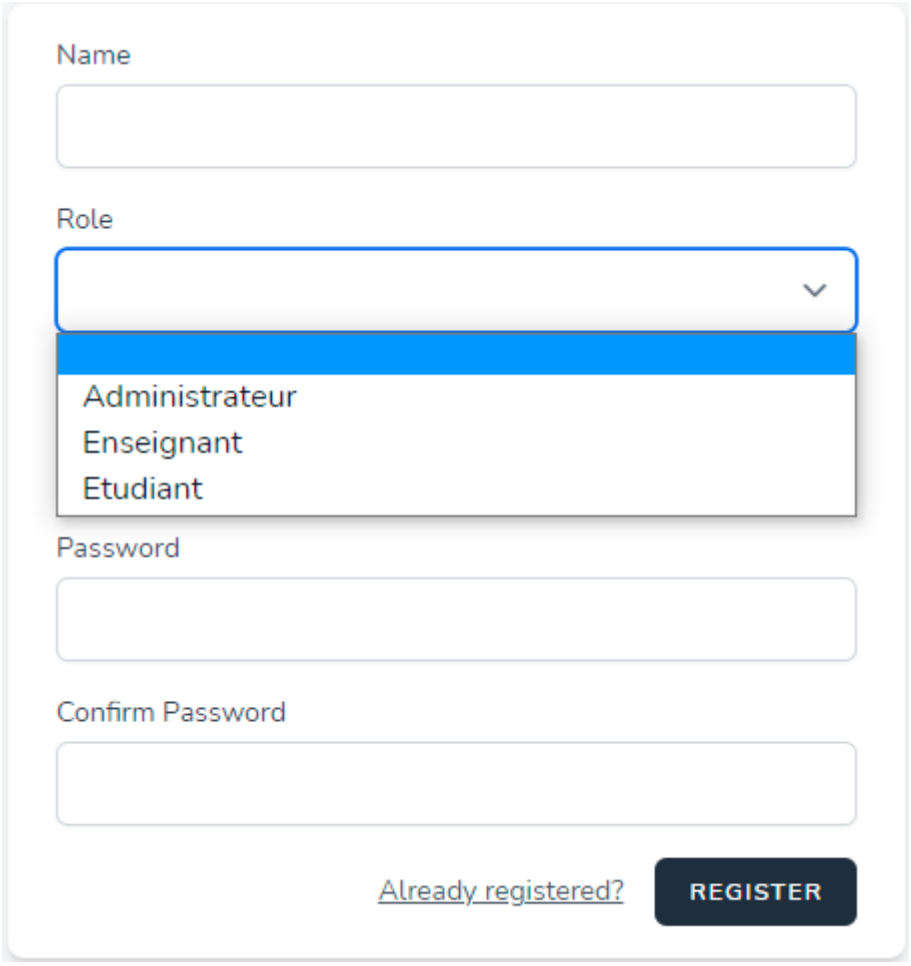
resources\views\auth\register.blade.php

```
<div>
    <x-jet-label for="name" value="{{ __('Name') }}" />
    <x-jet-input id="name" class="block mt-1 w-full" type="text" name="
    name" :value="old('name')" required autofocus autocomplete="name" />
</div>


<div class="mt-4">
<x-jet-label for="user_role" value="{{ __('Role') }}" />
<select  id="user_role" name="user_role" class="block mt-1 w-full
border-gray-300 rounded-md">
<option selected></option>
<option value="Administrateur">Administrateur</option>
<option value="Enseignant">Enseignant</option>
<option value="Etudiant">Etudiant</option>
</select>
</div>
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

6) Modifier la vue du login en ajoutant la champ Role :

resources\views\auth\register.blade.php



Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

6) Ajouter le Rôle au validateur

app\Actions\Fortify\CreateNewUser.php

```php
Validator::make($input, [
    'name' => ['required', 'string', 'max:255'],
    'email' => ['required', 'string', 'email', 'max:255',
        'unique:users'],

    'user_role' => ['required', 'string' ,
            'in:Administrateur,Enseignant,Etudiant'],

    'password' => $this->passwordRules(),
    'terms' => Jetstream::hasTermsAndPrivacyPolicyFeature(
        ) ? ['required', 'accepted'] : '',
])->validate();
```

Pas d'espace

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

6) Ajouter le Rôle au validateur

app\Actions\Fortify\CreateNewUser.php

```php
return User::create([
    'name' => $input['name'],
    'email' => $input['email'],
    'password' => Hash::make($input['password']),
    'user_role' => $input['user_role'],

]);
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

7) Ajouter les middleware pour protéger les espaces :

```
php artisan make:middleware VerifierEnseignant
```

```
php artisan make:middleware VerifierEtudiant
```

```
php artisan make:middleware VerifierAdministrateur
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

7) Ajouter les middleware pour protéger les espaces :

app\Http\Middleware\VerifierAdministrateur.php

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

```php
public function handle(Request $request, Closure $next)
{


    if (Auth::check() && (Auth::user()->user_role === "Administrateur"))
        return $next($request);
    else
        return redirect('/login');
}
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

8) Ajouter les routes et les protéger par les middlewares :

routes\web.php

```
Route::get('/dashboard', '
    App\Http\Controllers\GestionUtilisateursController@gestion_dashboard');


Route::get('/','App\Http\Controllers\GestionUtilisateursController@gestion_welcome');
```

Remarque : Après un **« logout »** Laravel dirige l'utilisateur a la racine **« / »**. Aussi après un **« login »** ou un **« register »** Laravel dirige l'utilisateur à la route **« /dashboard »**. On peut définir ces routes pour les contrôler selon nos besoins à l'aide des fonctions du contrôleur.

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

7) Ajouter les middleware pour protéger les espaces :

app\Http\Middleware\VerifierEnseignant.php

```php
use Illuminate\Support\Facades\Auth;
public function handle(Request $request, Closure $next)
{

    if (Auth::check() && (Auth::user()->user_role === "Enseignant"))
        return $next($request);
    else
        return redirect('/login');
}
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

7) Ajouter les middleware pour protéger les espaces :

app\Http\Middleware\VerifierEtudiant.php

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

```php
public function handle(Request $request, Closure $next)
{
    if (Auth::check() && (Auth::user()->user_role === "Etudiant"))
        return $next($request);
    else
        return redirect('/login');
}
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

7) Ajouter les middleware pour protéger les espaces :

app\Http\Kernel.php

```
'verif_admin' => \App\Http\Middleware\VerifierAdministrateur::class,
'verif_etudiant' => \App\Http\Middleware\VerifierEtudiant::class,
'verif_enseignant' => \App\Http\Middleware\VerifierEnseignant::class,
```

Le framework Laravel (Le contrôle selon les rôles des utilisateurs)

8) Ajouter les routes et les protéger par les middlewares :

routes\web.php

```
Route::get('/espace_administrateur', '
    App\Http\Controllers\GestionUtilisateursController@espace_administrateur')->
    middleware('verif_admin');

Route::get('/espace_enseignant', '
    App\Http\Controllers\GestionUtilisateursController@espace_enseignant')->
    middleware('verif_enseignant');;

Route::get('/espace_etudiant', '
    App\Http\Controllers\GestionUtilisateursController@espace_etudiant')->middleware(
    'verif_etudiant');;
```

Remarque : On peut contrôler les routes par le middleware « **auth** » pour s ‘assurer que l’utilisateur est authentifié.