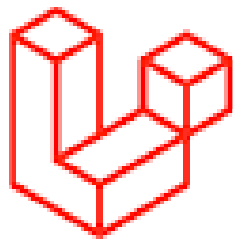


CCI - SIO Atelier 2 - SACCHETTO Vladimir le 04/07/2022

## Installation et utilisation de Laravel

Rédaction de documentation utilisateur efficace



**Laravel**

# SOMMAIRE

1. INTRODUCTION. . . . .	3
2. INSTALLER LE FRAMEWORK LARAVEL. . . . .	4
3. ROUTING DE BASE. . . . .	6
4. LES VUES AVEC BLADE. . . . .	7
5. LES CONTROLEURS . . . . .	8
6. VUES ET CONTROLEURS. . . . .	9
7. LES MIGRATIONS ET LES MODELES. . . . .	11
8. L'ORM ELOQUENT. . . . .	13
9. CRUD. . . . .	15

## 1. INTRODUCTION

Laravel est un Framework web open-source écrit en PHP respectant le principe **modèle-vue-contrôleur** et entièrement développé en programmation orientée objet. Laravel est distribué sous licence MIT, avec ses sources hébergées sur [GitHub](#). Laravel a été créé par Taylor Otwell en juin 2011.

En peu de temps, une communauté d'utilisateurs du Framework s'est constituée, et il est devenu en 2016 le projet PHP le mieux noté de GitHub. Laravel reste pourtant basé sur son grand frère Symfony, pour au moins 30 % de ses lignes (utilisation de "Symfony component").

## 2. INSTALLER LE FRAMEWORK LARAVEL

Nous allons installer le Framework Laravel à travers le gestionnaire de dépendances **composer** que nous avons installé lors de la mise en place de notre environnement de travail (*doc 3*).

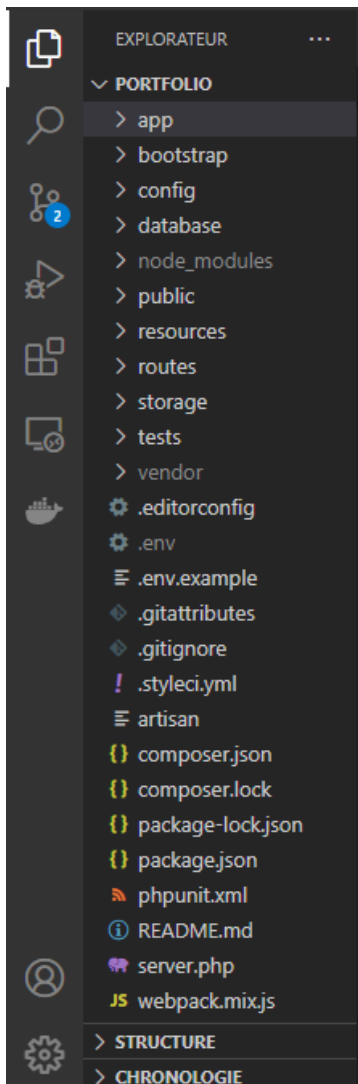
Pour ce faire nous allons nous rendre dans notre machine de développement dans **/var/www** et lançons la commande **composer create-project laravel/laravel** (*nom du répertoire app par défaut*) (**CCI-SIO21-Portfolio dans mon cas**)

```
ubuntu@devslam:/var/www$ composer create-project laravel/laravel CCI-SIO21-Portfolio
Creating a "laravel/laravel" project at "../CCI-SIO21-Portfolio1"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v8.6.12)
As there is no 'unzip' nor '7z' command installed zip files are being unpacked using the PHP zip extension.
This may cause invalid reports of corrupted archives. Besides, any UNIX permissions (e.g. executable) defined in the archives will be lost.
Installing 'unzip' or '7z' (21.01+) may remediate them.
- Installing laravel/laravel (v8.6.12): Extracting archive
Created project in /var/www/CCI-SIO21-Portfolio
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 110 installs, 0 updates, 0 removals
- Locking asm89/stack-cors (v2.1.1)
- Locking brick/math (0.9.3)
- Locking dflydev/dot-access-data (v3.0.1)
- Locking doctrine/inferno (2.0.4)
- Locking doctrine/instantiator (1.4.1)
- Locking doctrine/lexer (1.2.3)
- Locking dragonmantank/cron-expression (v3.3.1)
- Locking egulias/email-validator (2.1.25)
- Locking facade/flare-client-php (1.9.1)
- Locking facade/ignition (2.17.6)
- Locking facade/ignition-contracts (1.0.2)
- Locking fakerphp/faker (v1.19.0)
- Locking filp/whoops (2.14.5)
- Locking fruitcake/laravel-cors (v2.2.0)
```

```
- Installing sebastian/code-unit-reverse-lookup (2.0.3): Extracting archive
- Installing phpunit/php-code-coverage (9.2.15): Extracting archive
- Installing doctrine/instantiator (1.4.1): Extracting archive
- 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 (9.5.21): Extracting archive
74 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Package swiftmailer/swiftmailer is abandoned, you should avoid using it. Use symfony/mailer instead.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/sail
Discovered Package: laravel/sanctum
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
77 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.
```

```
ubuntu@devslam:/var/www$ ll
total 16
drwxr-xr-x  4 ubuntu www-data 4096 Jul  6 16:13 ./
drwxr-xr-x 14 root    root    4096 May  4 21:10 ../
drwxrwxr-x 14 ubuntu ubuntu  4096 Jul  6 14:56 CCI-SIO21-Portfolio/
drwxr-xr-x  2 ubuntu www-data 4096 May 31 09:59 html/
ubuntu@devslam:/var/www$
```

Notre projet vient d'être installé. Si nous ouvrons **Visual Studio Code** nous pouvons observer le contenu du dossier avec tous les répertoires et les fichiers de Laravel.



Nous pouvons retrouver toutes les informations sur le Framework sur leur page officielle [Laravel](https://laravel.com).

### 3. ROUTING DE BASE

Afin de coder en Laravel nous devons d'abord comprendre le principe des routes.

Dans le dossier **routes** nous avons différents fichiers PHP.

Nous allons ouvrir le fichier *web.php* qui sera celui qui nous servira à afficher nos pages web. Nous pouvons observer que le fichier contient des class PHP "*use Illuminate\Support\Facades\Route;*" et une fonction **Route::get** que Laravel propose par défaut.

```
Portfolio > routes > web.php > ...
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5
6  /*
7  |-----
8  | Web Routes
9  |-----
10 |
11 | Here is where you can register web routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | contains the "web" middleware group. Now create something great!
14 |
15 */
16
17 Route::get('/', function () {
18     return view('welcome');
19 });
20
```

La **Route::get** commence à la racine "/" et contient une fonction qui affiche la vue "*welcome*". Cette vue fait référence au fichier *welcome.blade.php* que Laravel possède par défaut dans le dossier **/resources/views**.

Nous allons créer une route qui retourne la vue "*test*" pour l'exemple de ce document

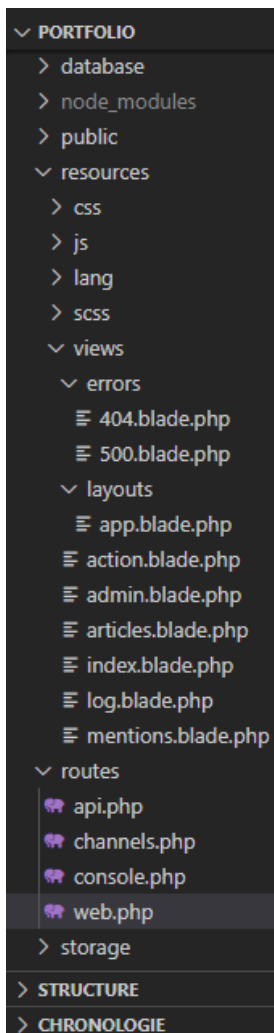
```
routes > web.php > ...
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5
6  /*
7  |-----
8  | Web Routes
9  |-----
10 |
11 | Here is where you can register web routes for you
12 | routes are loaded by the RouteServiceProvider wit
13 | contains the "web" middleware group. Now create s
14 |
15 */
16
17 Route::get('/', function () {
18     return view('test');
19 });
20
21 // Route::get('test', function () {
22 //     return view('test');
23 // });
24
```

## 4. LES VUES AVEC BLADE

Comme nous l'avons vu dans le fichier *web.php* les routes en Laravel affichent des vues qui sont stockés dans le dossier **/resources/views**.

Les vues sont des fichiers en format PHP qui ont l'extension *".blade.php"*. Dans ces fichiers pouvons coder en HTML et PHP à l'aide des propriétés de **Blade**. Blade est le moteur de Template utilisé par Laravel. Son but est de permettre d'utiliser du PHP sur notre vue mais d'une manière assez particulière. Pour créer un fichier qui utilise le moteur de Template Blade nous sommes obligés d'ajouter l'extension *".blade.php"*.

Voici les vues créées pour le projet « **Portfolio** »

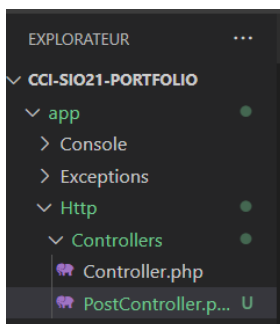


## 5. LES CONTROLEURS

Afin que tout fonctionne correctement, Laravel utilise les **contrôleurs**. Le contrôleur est un moyen pour faire afficher les pages qui sont dans le répertoire **/ressources/views** de manière optimisée. Ce processus utilise une logique bien définie.

Nous pouvons retrouver les contrôleurs dans le répertoire **app\Http\Controllers**. Afin de créer un nouveau contrôleur Laravel nous propose une commande **php artisan make:controller (nom du contrôleur)** nous créons pour l'exemple de ce document un fichier *PostController.php*.

```
ubuntu@devslam:/var/www/CCI-SIO21-Portfolio$ php artisan make:controller PostController
Controller created successfully.
ubuntu@devslam:/var/www/CCI-SIO21-Portfolio$
```



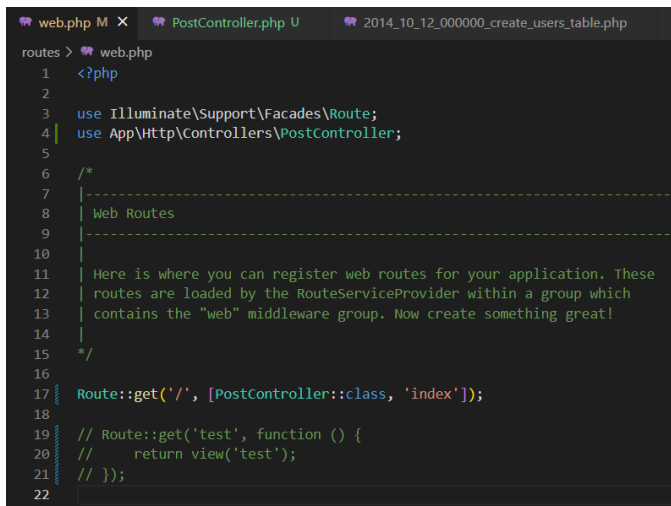
C'est donc dans ce fichier que nous allons insérer une fonction qu'on appelle *index* qui va rappeler la vue "test".

```
PostController.php U x 2014_10_12_000000_create_users_table
app > Http > Controllers > PostController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  class PostController extends Controller
6  {
7      public function index()
8      {
9          return view('test');
10     }
11 }
12
```

Ce fichier php contient le **namespace** avec le chemin où le fichier se trouve **app\Http\Controllers**; ainsi que la **class PostController** qui étend la classe de base de Laravel *Controller.php* dans **app\Http\Controllers\Controller.php**



Une fois que nous avons indiqué la fonction dans le contrôleur nous allons modifier la route en indiquant la *class* du contrôleur et le nom de la fonction.



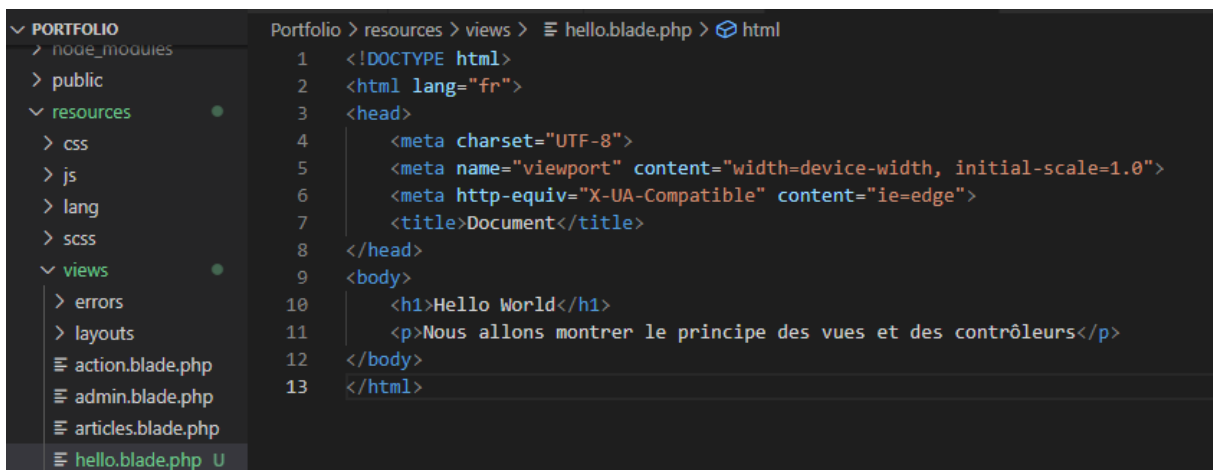
```

1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\PostController;
5
6  /*
7   |-----
8   | Web Routes
9   |-----
10  |
11  | Here is where you can register web routes for your application. These
12  | routes are loaded by the RouteServiceProvider within a group which
13  | contains the "web" middleware group. Now create something great!
14  |
15  */
16
17  Route::get('/', [PostController::class, 'index']);
18
19  // Route::get('test', function () {
20  //     return view('test');
21  // });
22

```

## 6. VUES ET CONTROLEURS

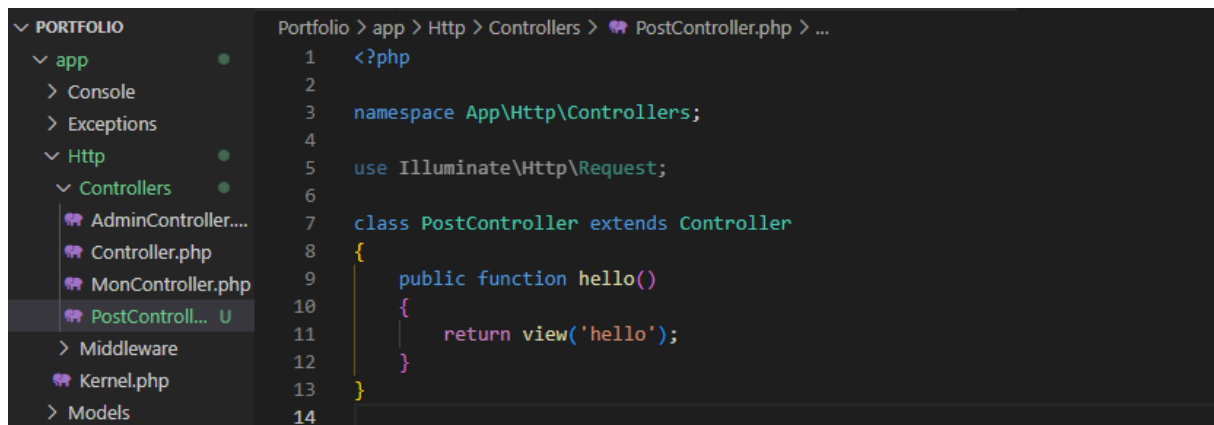
Les vues sont directement liées à la route et le contrôleur. Comme nous l'avons expliqué dans les précédents paragraphes, nous pouvons créer une vue où nous insérons ce que la page va nous afficher en HTML et/ou PHP (avec les propriétés Blade) et rappeler dans le contrôleur la fonction qui sera affiché par la vue.



```

Portfolio > resources > views > hello.blade.php > html
1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <title>Document</title>
8  </head>
9  <body>
10     <h1>Hello World</h1>
11     <p>Nous allons montrer le principe des vues et des contrôleurs</p>
12 </body>
13 </html>

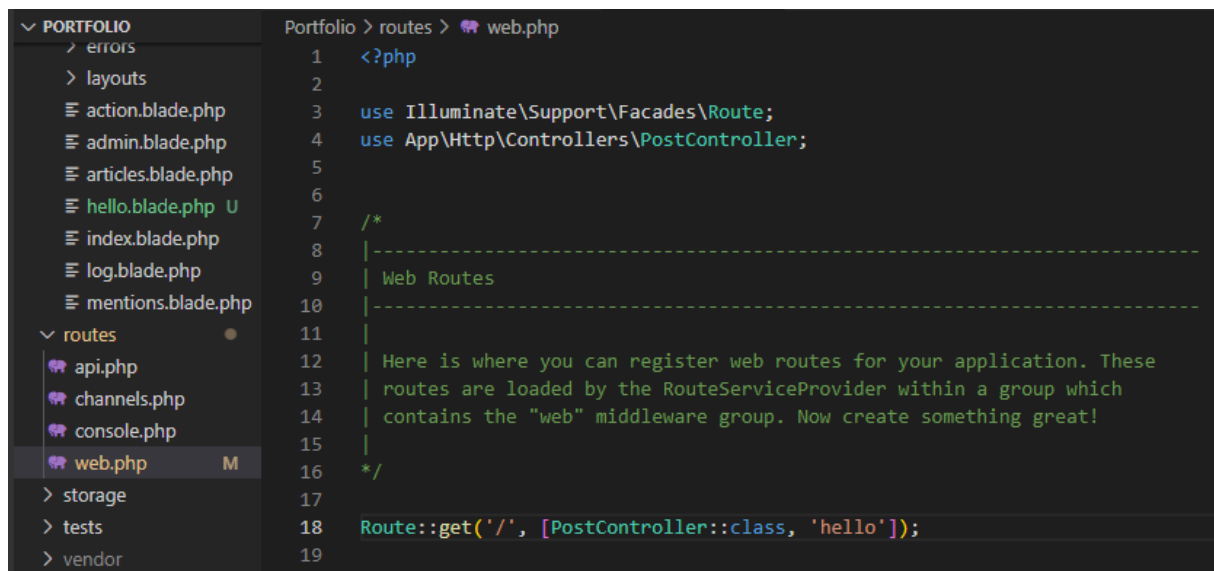
```



```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class PostController extends Controller
8  {
9      public function hello()
10     {
11         return view('hello');
12     }
13 }
14

```

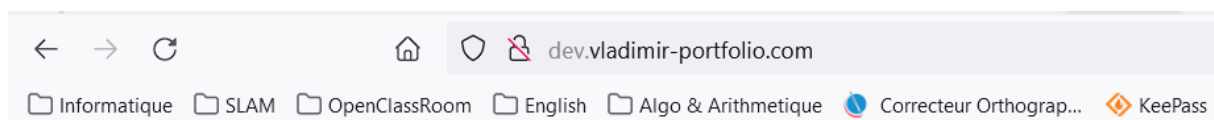


```

1  <?php
2
3  use Illuminate\Support\Facades\Route;
4  use App\Http\Controllers\PostController;
5
6
7  /*
8   |-----
9   | Web Routes
10  |-----
11  |
12  | Here is where you can register web routes for your application. These
13  | routes are loaded by the RouteServiceProvider within a group which
14  | contains the "web" middleware group. Now create something great!
15  |
16  */
17
18 Route::get('/', [PostController::class, 'hello']);
19

```

De cette manière si nous nous rendons sur notre page web nous allons afficher **Hello World**



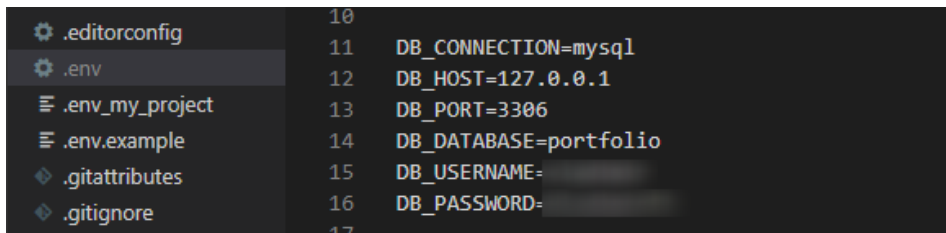
## Hello World

Nous allons montrer le principe des vues et des contrôleurs

## 7. LES MIGRATIONS ET LES MODELES

Les migrations nous permettent de nous connecter à la Base de données.

Nous allons modifier le fichier **.env** en ajoutant les informations sur la BDD qu'on utilise (*dans mon cas mysql*). Ainsi nous allons indiquer le nom de la BDD, l'user et le mot de passe pour la connexion.

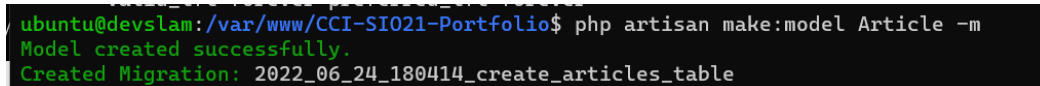


```

10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=portfolio
15 DB_USERNAME=
16 DB_PASSWORD=
17

```

Nous allons ensuite utiliser la commande **php artisan make:model (nom de la table) -m** afin de créer notre migration ainsi que notre modèle.

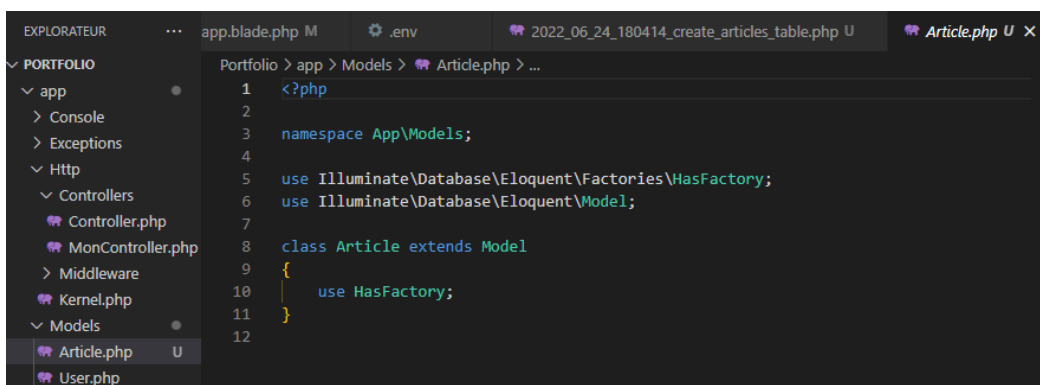


```

ubuntu@devslam:/var/www/CCI-SI021-Portfolio$ php artisan make:model Article -m
Model created successfully.
Created Migration: 2022_06_24_180414_create_articles_table

```

Nous aurons un modèle dans le dossier **app/Models** avec le nom de la table (dans mon cas **Article.php**) et une migration qui va se créer automatiquement dans le dossier **database/migration**

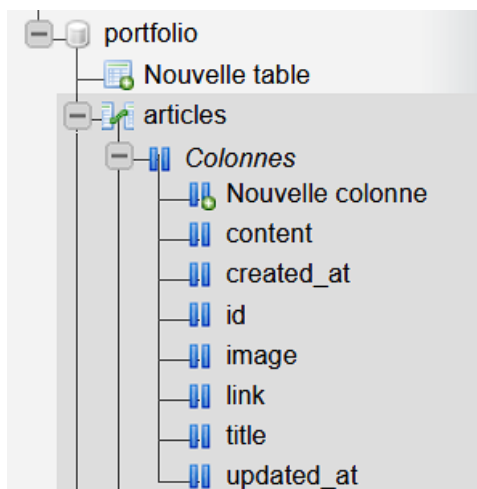


```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Article extends Model
9 {
10     use HasFactory;
11 }
12

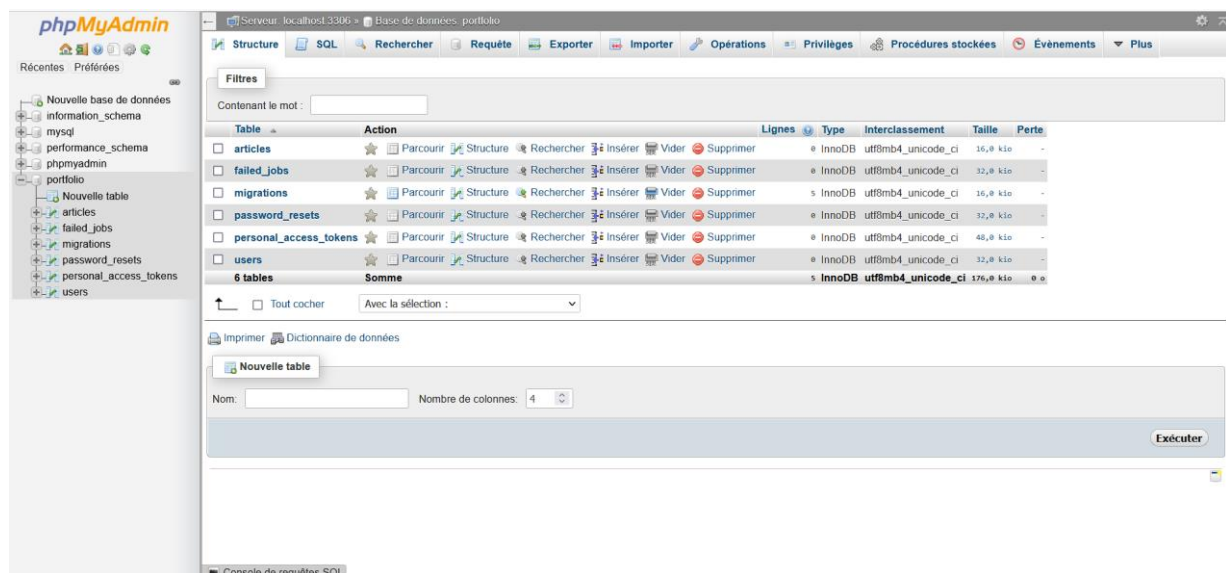
```

C'est dans la migration que nous allons rajouter des champs et des tables. Dans notre cas on rajoute un **id**, **title** « **string** », **image** « **mediumText** », **content** « **mediumText** » et **link** « **mediumText** ». Par défaut des paramètres sont déjà présents dans la BDD dont **created\_at** et **updated\_at**.



Nous allons ensuite utiliser la commande **php artisan migrate** pour communiquer avec la base de données et ajouter tous nos fichiers de migration

```
ubuntu@devslam:/var/www/CCI-SI021-Portfolio$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (60.09ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (47.93ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (58.63ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated: 2019_12_14_000001_create_personal_access_tokens_table (71.80ms)
Migrating: 2022_06_24_180414_create_articles_table
Migrated: 2022_06_24_180414_create_articles_table (28.65ms)
```



Nous pouvons effectuer la commande **php artisan migrate:fresh** ainsi tout le contenu de nos tables est effacé et toutes les migrations sont réécrites à vide.

## 8. L'ORM ELOQUENT

Laravel offre un environnement de développement très fonctionnel, ainsi que des interfaces de ligne de commande intuitives et expressives. En outre, Laravel utilise la cartographie **objet-relationnel (ORM)** pour simplifier l'accès et la manipulation des données.

Nous allons utiliser ce principe afin d'afficher les articles présents dans notre BDD sur une page web que je vais nommer *articles.blade.php*

Dans cette page grâce à une propriété *blade* je peux utiliser un *@foreach* qui me permettra d'afficher les articles présents dans ma BDD que j'ai appelé également dans mon contrôleur avec le paramètre *compact()*.

```

1  @extends('layouts.app')
2
3  @section('css')
4  <link rel="stylesheet" href="{{ asset('css/stylePort.css') }}">
5  <link rel="stylesheet" href="{{ asset('css/styleLegal.css') }}">
6  <link rel="stylesheet" href="{{ asset('css/styleArticle.css') }}">
7  @endsection
8  @section('title')
9  Veille technologique
10 @endsection
11 @section('content')
12 <header>
13 <a href="/"
14 ></a>
16 <a href="/login">Articles</h1>
18 </header>
19
20 <div class="contain-all-articles">
21
22     @foreach ($articles as $article)
23     <div class="contain-article" data-id="{{ $article->id }}">
24
25         <div class="title">{{ $article->title }}</div>
26         <div class="image">{{ $article->content }}</div>
28         <div class="link"><a href="{{ $article->link }}" target="_blank">Con
29         {{-- {{substr($article->content, 0, 62)}} ... --}}
30     </div>
31     @endforeach

```

```

24 public function articles()
25 {
26     $articles = Article::all();
27
28     return view(
29         'articles',
30         compact('articles')
31     );
32 }
33

```

Ainsi ma page affichera les articles présents dans la BDD qui sont définis dans le modèle *Article.php*. Le contrôleur en effet présente la **class** *use app\Models\Article* afin de récupérer les informations

```
use App\Models\Article;
```

## 9. CRUD

En utilisant le principe des migrations et des modèles ainsi que le MVC (modèle-vue-contrôleur), nous pouvons créer des liaisons avec la base de données et ainsi mettre en place un **CRUD**. L'acronyme informatique anglais CRUD (*pour Create, Read, Update, Delete*) désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Soit :

- **create** : créer
- **read** : lire
- **update** : mettre à jour
- **delete** : supprimer

Plus généralement, il désigne les opérations permettant la gestion d'une collection d'éléments. Ce terme est aussi un jeu de mot en anglais sur l'adjectif **crude** (en français **brut** ou **rudimentaire**).

Nous allons pouvoir créer, afficher, modifier et supprimer nos articles directement depuis notre page web.

Le CRUD va donc être établi dans le contrôleur ainsi que la vue.

J'ai créé un contrôleur qui va contenir des fonctions **new**, **edit** et **delete** qui feront en sorte de pouvoir **ajouter, modifier et supprimer** les articles directement depuis la BDD

▼ Controllers	45	}
AdminController....	46	
Controller.php	47	public function new()
MonController.php	48	{
> Middleware	49	\$message = false;
Kernel.php	50	if (Session::get('connected') === false) {
> Models	51	return abort('404');
> Providers	52	}
> bootstrap	53	if (count(\$_POST) > 3) {
> config	54	Article::create([
▼ database	55	'title' => \$_POST['title'],
> factories	56	'image' => \$_POST['image'],
> migrations	57	'content' => \$_POST['content'],
2014_10_12_00000...	58	'link' => \$_POST['link'],
2014_10_12_10000...	59	]);
2019_08_19_00000...	60	\$message = true;
2019_12_14_00000...	61	}
2022_06_24_18041...	62	return view(
	63	'action'
	64	)->with('action', 'new')->with('message', \$message);
	65	}

▼ Controllers	66	public function edit(\$id, Request \$request)
AdminController....	67	{
Controller.php	68	\$message = false;
MonController.php	69	if (Session::get('connected') === false) {
> Middleware	70	return abort('404');
Kernel.php	71	}
> Models	72	\$article = Article::find(\$id);
> Providers	73	
> bootstrap	74	if (count(\$_POST) > 3) {
> config	75	\$update = Article::find(\$id);
▼ database	76	
> factories	77	\$update->title = \$request->input("title");
> migrations	78	\$update->image = \$request->input("image");
2014_10_12_00000...	79	\$update->content = \$request->input("content");
2014_10_12_10000...	80	\$update->link = \$request->input("link");
2019_08_19_00000...	81	\$update->save();
2019_12_14_00000...	82	\$message = true;
2022_06_24_18041...	83	}
> seeders	84	return view(
	85	'action'
	86	)->with('action', 'edit')->with('article', \$article)->with('message', \$mes.
	87	}

▼ Controllers	88	public function delete(\$id)
AdminController....	89	{
Controller.php	90	if (Session::get('connected') === false) {
MonController.php	91	return abort('404');
> Middleware	92	}
Kernel.php	93	\$delete = Article::find(\$id);
> Models	94	\$delete->delete();
> Providers	95	return view(
> bootstrap	96	'action'
> config	97	)->with('action', 'delete');
▼ database	98	}
	99	
	100	

J'ai ensuite indiqué dans une vue les trois fonctions qui envoient un formulaire pour chacun des actions et crée des chemins dans la route.



```

10
11
12 @if($action === 'new')
13     <header class="bloc">
14         <h1>Création d'un article</h1>
15
16         <form action="/admin" method="POST" >
17             @csrf
18             <button type="submit" class="deconnexion" name="pseudo" name="password">Se deconnecter</button>
19         </form>
20         <a href="/login" class="deconnexion">Se deconnecter</a>
21     </header>
22
23     @if($message)
24         <p class="textaction_"> L'action a bien été réalisée</p>
25     @endif
26
27     <form action="/admin/new" method="post" class="w-50 mx-auto">
28         @csrf
29         <div class="conteneur-pseudo form-floating mb-3">
30             <input type="text" class="form-control" name="title" id="title" placeholder="Titre">
31             <label for="title"></label>
32         </div>

```

```

48
49 @elseif($action === 'edit')
50     <header class="bloc">
51         <h1>Modification d'un article</h1>
52
53         <form action="/admin" method="POST" >
54             @csrf
55             <button type="submit" class="deconnexion" name="pseudo" name="password">Se deconnecter</button>
56         </form>
57         <a href="/login" class="deconnexion">Se deconnecter</a>
58     </header>
59
60     @if($message)
61         <p class="textaction_"> L'action a bien été réalisée</p>
62     @endif
63
64     <form action="/admin/edit/{{ $article->id }}" method="post" class="w-50 mx-auto">
65         @csrf
66         <div class="conteneur-pseudo form-floating mb-3">
67             <label for="title" class="lab">Titre</label>
68             <input type="text" class="form-control" name="title" id="title" placeholder="Titre">
69         </div>
70

```

```

90 @elseif($action === 'delete')
91     <header class="bloc">
92         <h1>Suppression d'un article</h1>
93         <form action="/admin" method="POST" >
94             @csrf
95             <button type="submit" class="deconnexion" name="pseudo" name="password">Se deconnecter</button>
96         </form>
97         <a href="/login" class="deconnexion">Se deconnecter</a>
98     </header>
99     <p class="textaction">L'article a bien été supprimé</p>
100 @endif

```

```

Route::post('/admin/new', [AdminController::class, 'new'])->name('new');
Route::post('/admin/edit/{id}', [AdminController::class, 'edit'])->name('edit');
Route::post('/admin/delete/{id}', [AdminController::class, 'delete'])->name('delete');

```