

**Développement
Web**

Tutorial **LARAVEL**

Niveau Approfondi

DCLIC

«Formez-vous au
numérique avec l'OIF »



Tutoriel 1 : Présentation, Installation et mise en place du Framework Laravel

A la fin de ce tutoriel les objectifs atteints sont :

- Découvrir les notions fondamentales des Framework PHP
- Se familiariser avec l'architecture MVC
- Préparer l'environnement de développement Laravel
- Créer un nouveau projet Laravel
- Comprendre la structure et le rôle des fichiers dans le projet

I. Présentation du Framework Laravel

1. Qu'est-ce qu'un Framework web ?

Un Framework est une boîte à outils pour aider les développeurs dans la réalisation de leurs tâches. Frame (cadre) et work (travail).

Un Framework contient des composants autonomes qui permettent de résoudre les problèmes souvent rencontrés par les développeurs (CRUD, arborescence, normes, sécurités, etc.).

Un Framework n'est pas uniquement une boîte à outils. Il peut aussi désigner une méthodologie.

Un Framework est donc perçu comme un squelette, regroupant chaque os et articulations (ici composants/librairies) de manière harmonieuse mais surtout de manière fiable !

2. Pourquoi utiliser un Framework ?

L'objectif de l'utilisation d'un Framework est de :

- Maximiser la productivité du développeur qui l'utilise.
- Faciliter la maintenance de l'application Web réalisée.

Les trois raisons à utiliser un Framework :

- Rapidité : le Framework permet un gain de temps et une livraison beaucoup plus rapide qu'un développement à zéro.
- Organisation : l'architecture d'un Framework favorise une bonne organisation du code source (modèle MVC par exemple).
- Maintenabilité : l'organisation du Framework facilite la maintenance du logiciel et la gestion des évolutions.

3. Les critères de choix d'un Framework PHP

- La courbe d'apprentissage du Framework ne devrait pas être trop dure.
- Le Framework PHP doit répondre aux exigences techniques de votre projet. (Version PHP minimale, supporter la ou les bases de données du projet, ...)
- Un bon équilibre de fonctionnalités du Framework vis-à-vis du projet. Choisir un Framework minimal pour un petit projet.
- Une bonne documentation et un bon support sont importants pour que vous puissiez tirer le meilleur parti de votre Framework PHP

4. Les Framework PHP les plus populaires

Il est difficile d'obtenir une liste définitive des Framework PHP. Wikipédia liste 40 Framework PHP. Voici quelques-uns des meilleurs Framework PHP en usage aujourd'hui :

1. Laravel
2. Symfony
3. Zend Framework / Laminas Project

5. Laravel

Laravel est un Framework très puissant qui suit la structure **MVC**. Il est conçu pour les développeurs Web qui ont besoin d'une boîte à outils simple, élégante et puissante pour créer un site Web complet.

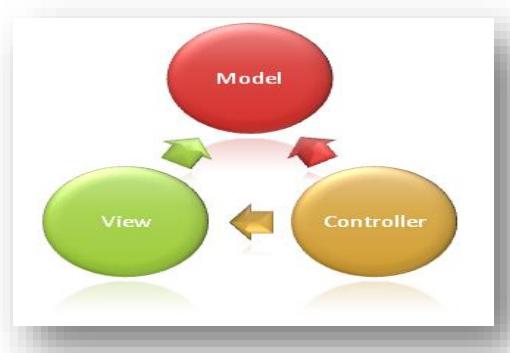
Laravel est un Framework d'applications Web basé sur PHP, il fournit des outils pour créer des applications puissantes et robustes, c'est un Framework open source, qui fournit une structure qui permet de gagner beaucoup de temps pour créer et planifier des applications volumineuses. C'est l'une des plateformes les plus sécurisées utilisant une base PHP. Il fournit des fonctionnalités intégrées pour l'autorisation des utilisateurs, telles que la connexion, l'enregistrement et l'oubli du mot de passe.

II. MVC

Les Framework PHP suivent généralement le plus célèbres design patterns qui s'appelle **MVC**, qui signifie Modèle - Vue - Contrôleur.

Le but du pattern **MVC** est de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère la logique métier. Le modèle communique avec la BDD. C'est lui qui s'occupe de récupérer un article en base par exemple. On y trouve des algorithmes complexes et des requêtes SQL.
- **Vue** : cette partie se concentre sur l'affichage. Elle affiche à l'utilisateur des données contenues dans des variables. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples.
- **Contrôleur** : cette partie gère les échanges avec l'utilisateur. C'est en quelque sorte l'intermédiaire entre l'utilisateur, le modèle et la vue. Le modèle lui transmet les données récupérées en base de données, puis il transmet ces données à la vue.



Dans Laravel :

- Le modèle correspond à une table d'une base de données. C'est une classe qui étend la classe Model qui permet une gestion simple et efficace des manipulations de données et l'établissement automatisé de relations entre tables.
- Le contrôleur se décline en deux catégories : contrôleur classique et contrôleur de ressource.
- La vue est soit un simple fichier avec du code HTML, soit un fichier utilisant le système de template Blade de Laravel.

Laravel propose ce patron mais ne l'impose pas.

III. Installation des outils nécessaires

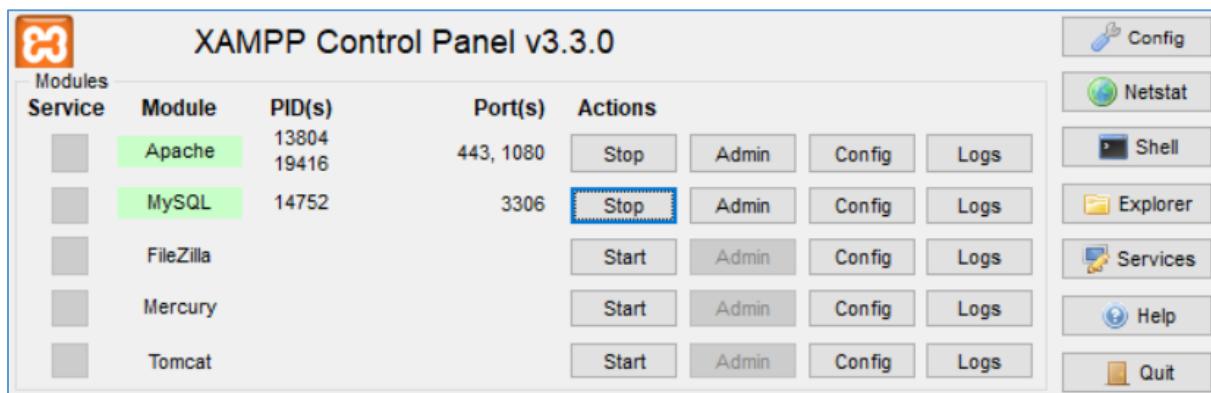
1. Serveur APACHE

Il vous faudra obligatoirement un serveur Apache ou Nginx permettant l'exécution de PHP ainsi qu'un serveur de base de données.

Installez XAMPP ou Laragon ou WAMP si vous êtes sous Windows, MAMP si vous êtes sous MacOS ou XAMPP pour Linux, ce sont des logiciels qui regroupent serveur apache et serveur base de données avec une interface pour gérer vos bases de données, le module PHP, etc.

Assurez-vous d'avoir une version PHP supérieure ou égale à 8.0 car c'est le minimum demandé par Laravel pour fonctionner.

Démarrer les serveurs :



2. Le composer



Composer est un gestionnaire de dépendances pour PHP, et il vous aide à installer Laravel et toutes les autres dépendances dont vous avez besoin pour construire un site Web. Composer c'est concrètement du code PHP, il vous faudra donc installer php sur votre machine par l'intermédiaire de logiciel comme Wamp, Mamp, xammp, Laragon, etc.

Pour vérifier que php est installé correctement sur votre machine, vous pouvez exécuter la commande suivante :

```
php -v
```

Vous devriez normalement avoir le résultat suivant :

```
Windows PowerShell
PS C:\Users\Anis> php -v
PHP 8.1.10 (cli) (built: Aug 30 2022 18:05:49) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.10, Copyright (c) Zend Technologies
PS C:\Users\Anis>
```

Le lien correspondant permet l'installation du Composer-Setup :

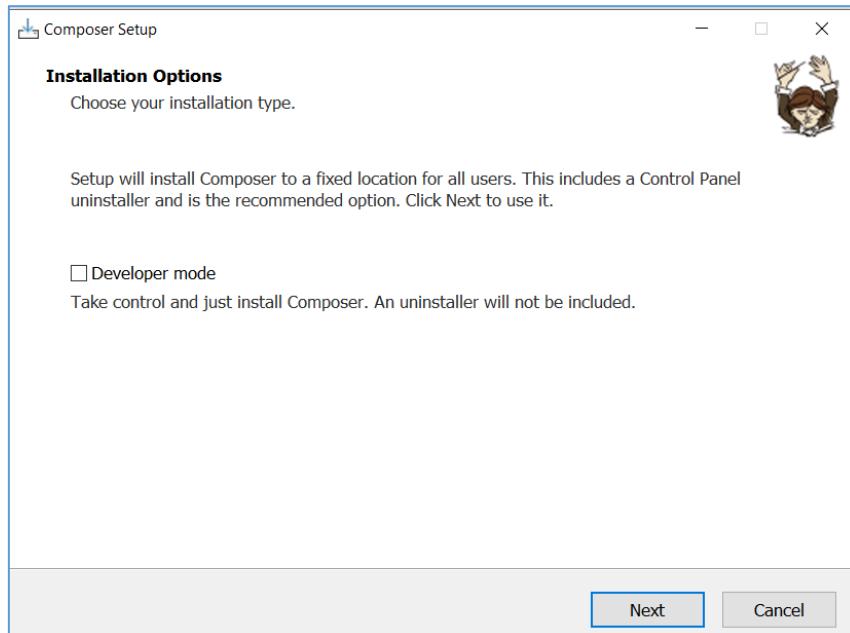
<https://getcomposer.org/download/>

Windows Installer

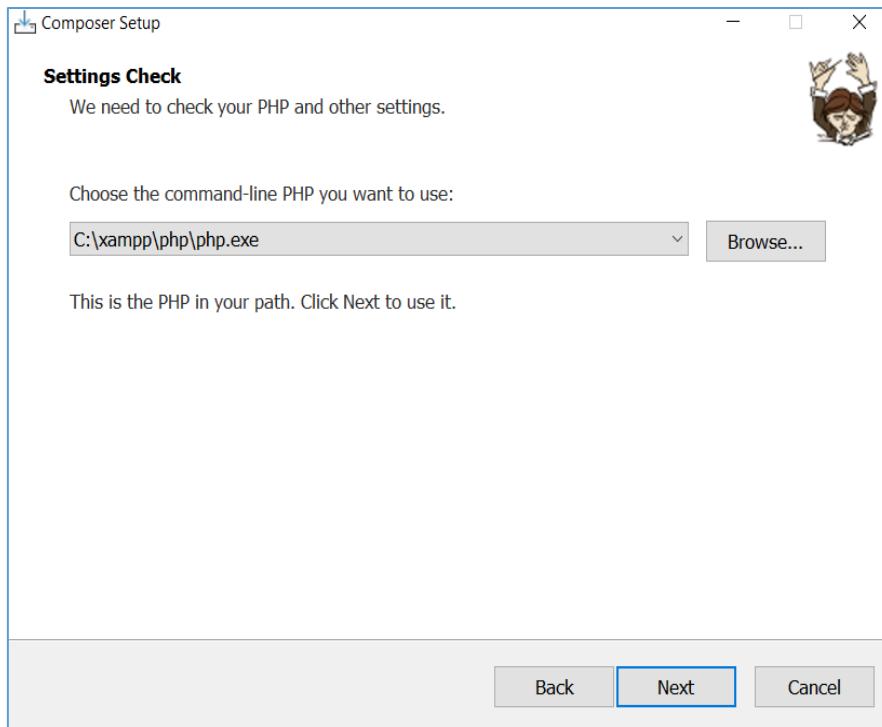
The installer - which requires that you have PHP already installed - will download Composer for you and set up your PATH environment variable so you can simply call `composer` from any directory.

Download and run [Composer-Setup.exe](#) - it will install the latest composer version whenever it is executed.

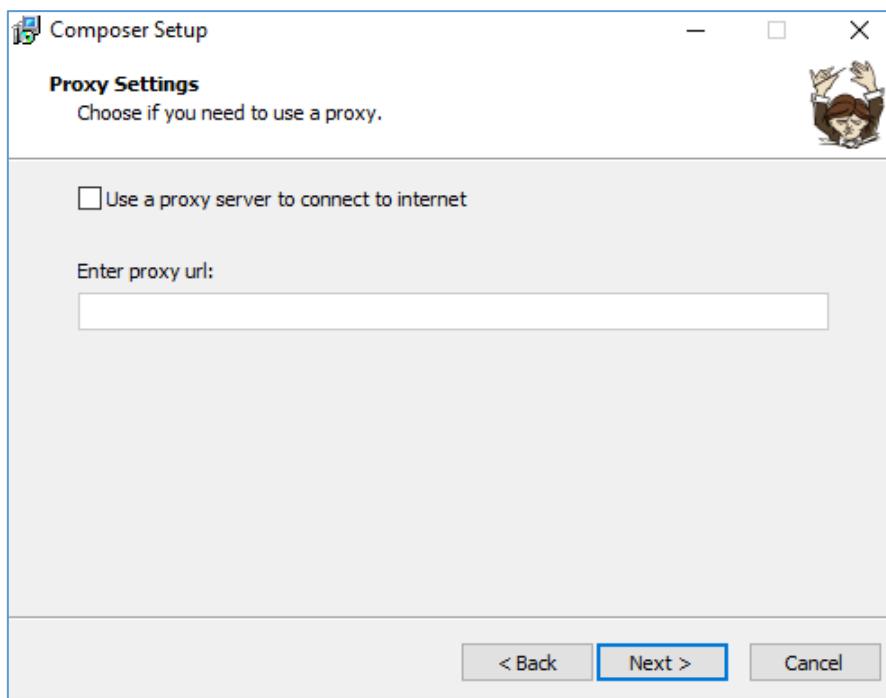
Une fois téléchargé, exécutez-le puis cliquez sur Next.



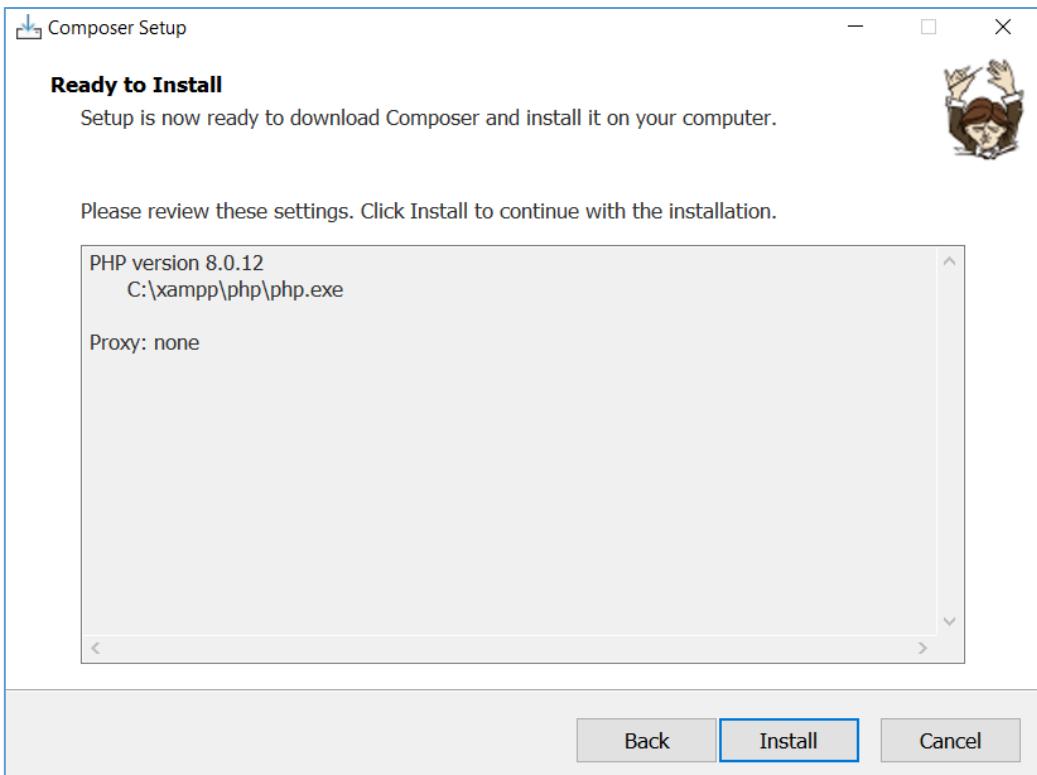
Maintenant, définissez le chemin PHP et cliquez sur Next :



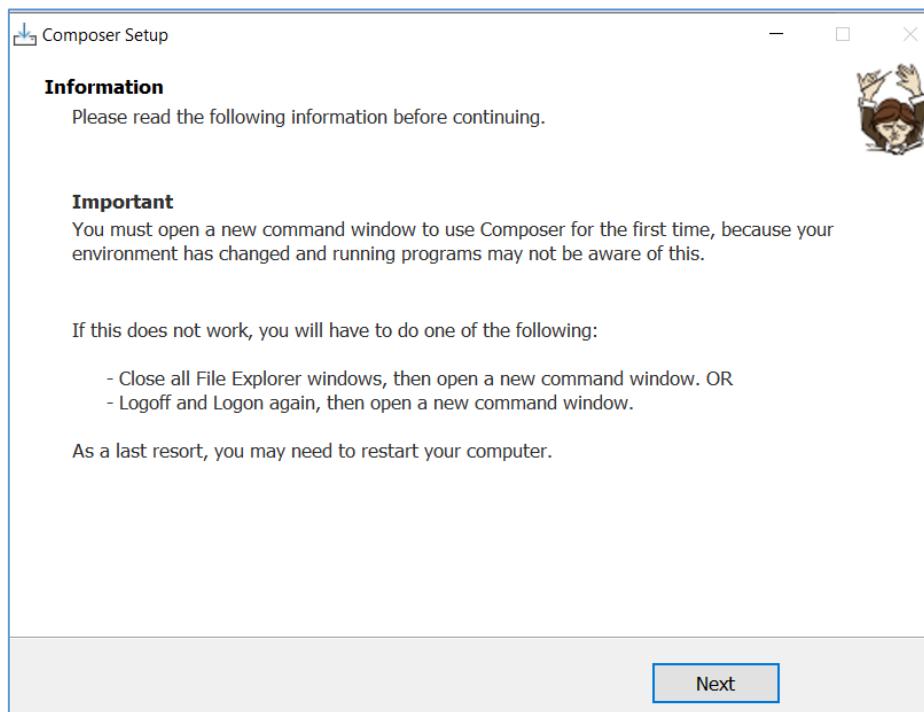
Si vous avez une URL proxy, entrez ici, mais si vous ne savez pas quelle valeur mettre, laissez-la vide et cliquez sur Next :



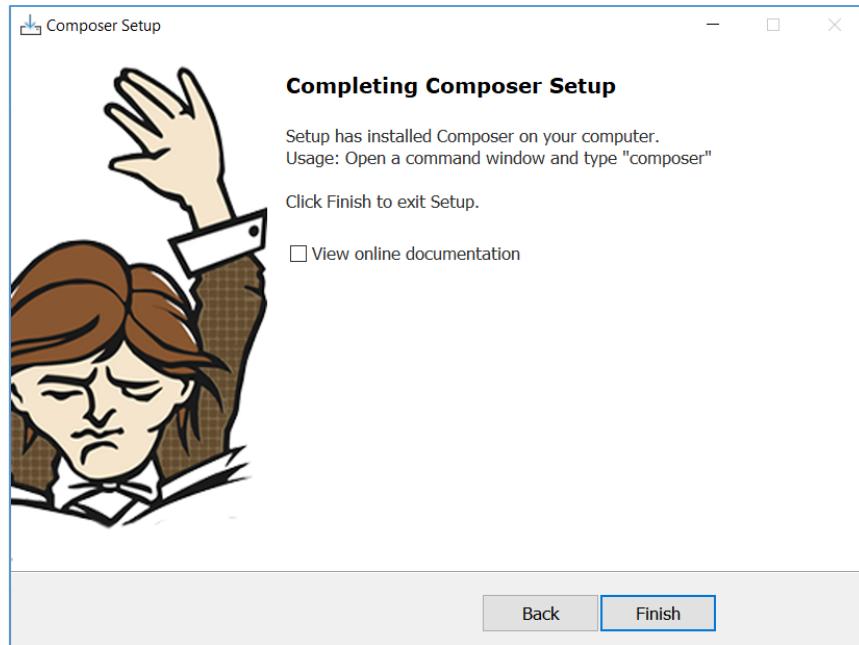
Maintenant, passez en revue l'assistant de configuration du composeur. Et cliquez sur install.



Cliquez sur Next.



Composer a été installé avec succès. Cliquez sur Finish.



Maintenant, ouvrez votre terminal et tapez la commande suivante à l'invite de commande :

Composer -v

Vous devriez normalement avoir le résultat suivant :

```
PS C:\Users\Anis> php -v
PHP 8.1.10 (cli) (built: Aug 30 2022 18:05:49) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.1.10, Copyright (c) Zend Technologies
PS C:\Users\Anis> composer -V

Composer version 2.5.1 2022-12-22 15:33:54

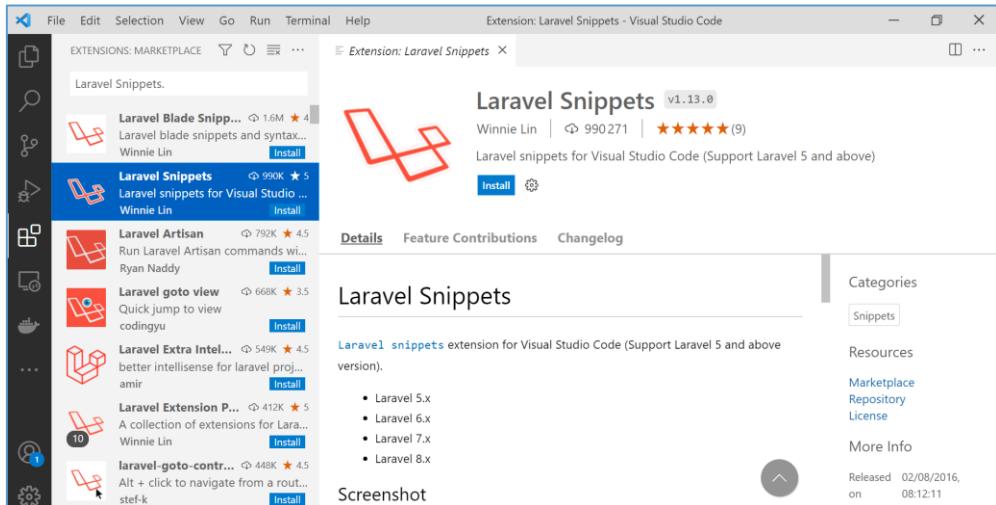
Usage:
  command [options] [arguments]

Options:
  -h, --help          Display help for the given command. When no command is given display
  command
  -q, --quiet         Do not output any message
  -V, --version       Display this application version
  --ansi|--no-ansi   Force (or disable --no-ansi) ANSI output
  -n, --no-interaction Do not ask any interactive question
  --profile          Display timing and memory usage information
  --no-plugins        Whether to disable plugins.
  --no-scripts        Skips the execution of all scripts defined in composer.json file
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache          Prevent use of the cache
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for
```

3. Éditeurs de code ou IDE PHP

Pour travailler efficacement, il faut utiliser un IDE ou un éditeur. Vous êtes bien évidemment libre de choisir votre environnement de travail. Néanmoins, dans ce tutoriel, nous allons utiliser Visual Studio Code.

VSCode peut intégrer plusieurs extensions qui aident l'écriture du code notamment Laravel Snippets.

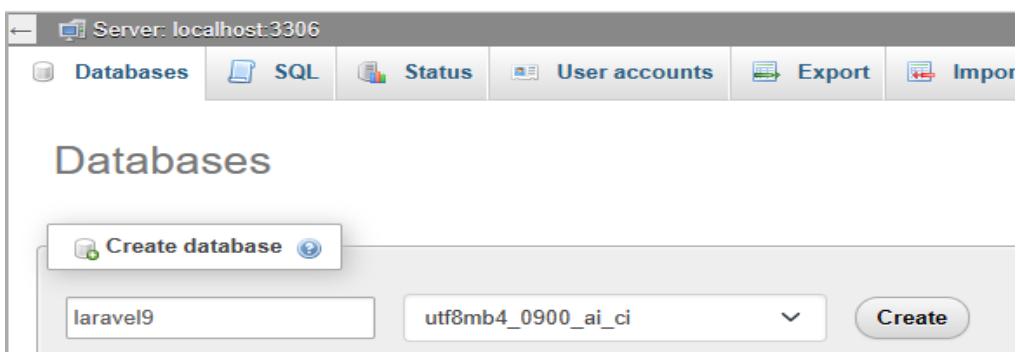


Ce site propose les meilleures extensions pour Laravel dans VSCode :

<https://devdojo.com/bobbyiliev/8-awesome-vs-code-extensions-for-laravel-developers>

IV. Créer une base de données

- Allez dans phpMyAdmin, cliquez sur créer un nouvel onglet.
- Nommez la base de données.
- Appuyez sur le bouton Créer.



V. Installer Laravel

Tapez la commande suivante dans votre fenêtre d'invite de commande pour créer un projet Laravel avec la dernière version de Laravel.

```
composer create-project --prefer-dist laravel/laravel projet1
```

composer create-project --prefer-dist laravel/laravel Project_name : cette commande installera Laravel et d'autres dépendances.

--prefer-dist et --prefer-source sont les deux options de composer.

--prefer-dist essaierait de télécharger et de décompresser les archives des dépendances à l'aide de GitHub ou d'une autre API lorsqu'elles sont disponibles. Il est utilisé pour un téléchargement plus rapide des dépendances dans la plupart des cas. Il ne télécharge pas tout l'historique VCS (systèmes de contrôle de version) des dépendances et il devrait être mieux mis en cache. Les archives sur GitHub peuvent également exclure certains fichiers dont vous n'avez pas besoin pour utiliser simplement la dépendance avec la directive d'exclusion.

--prefer-source essaiera de cloner et de conserver l'intégralité du référentiel VCS des dépendances lorsqu'il sera disponible. Ceci est utile lorsque vous souhaitez cloner les référentiels VCS d'origine dans votre fournisseur/dossier. Par exemple, vous voudrez peut-être travailler sur les dépendances - les modifier, les forks (fourches), soumettre des demandes de tirage, etc. tout en les utilisant dans le cadre d'un projet plus vaste qui les requiert en premier lieu.

```
Windows PowerShell
PS C:\Enseignement\Laravel> composer create-project --prefer-dist laravel/laravel projet1
Creating a "laravel/laravel" project at "./projet1"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v9.5.2)
- Installing laravel/laravel (v9.5.2): Extracting archive
Created project in C:\Enseignement\Laravel\projet1
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 106 installs, 0 updates, 0 removals
- Locking brick/math (0.10.2)
- Locking dflydev/dot-access-data (v3.0.2)
- Locking doctrine/inflector (2.0.6)
- Locking doctrine/instantiator (2.0.0)
- Locking doctrine/lexer (3.0.0)
- Locking dragonmantank/cron-expression (v3.3.2)
- Locking egulias/email-validator (4.0.1)
- Locking fakerphp/faker (v1.21.0)
- Locking filp/whoops (2.14.6)
- Locking fruitcake/php-cors (v1.2.0)
- Locking graham-campbell/result-type (v1.1.0)
- Locking guzzlehttp/guzzle (7.5.0)
- Locking guzzlehttp/promises (1.5.2)
- Locking guzzlehttp/psr7 (2.4.3)
- Locking hamcrest/hamcrest-php (v2.0.1)
- Locking laravel/framework (v9.50.2)
- Locking laravel/pint (v1.4.1)
- Locking laravel/sail (v1.19.0)
- Locking laravel/sanctum (v3.2.1)
- Locking laravel/serializable-closure (v1.3.0)
- Locking laravel/tinker (v2.8.0)

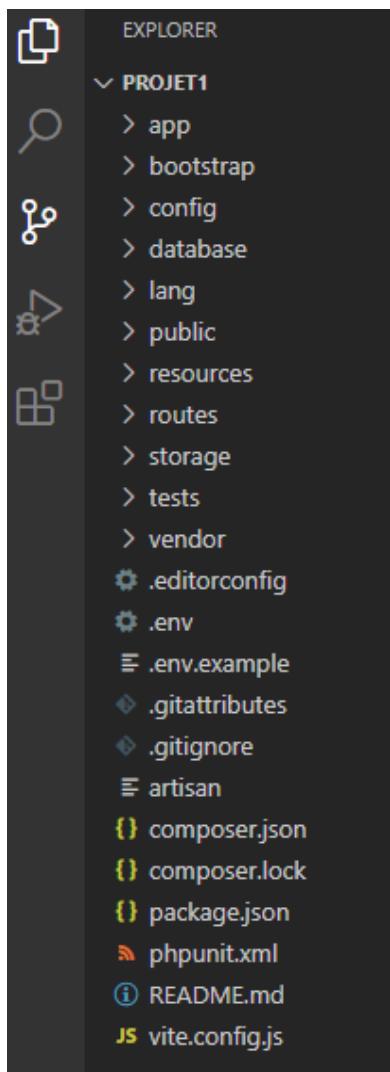
Installing spatie/backtrace (1.2.1). Extracting archive
- Installing spatie/flare-client-php (1.3.5): Extracting archive
- Installing spatie/ignition (1.4.3): Extracting archive
- Installing spatie/laravel-ignition (1.6.4): Extracting archive
63 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
[INFO] Discovering packages.

laravel/sail .....
laravel/sanctum .....
laravel/tinker .....
nesbot/carbon .....
nunomaduro/collision .....
nunomaduro/termwind .....
spatie/laravel-ignition .....

80 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
[INFO] No publishable resources for tag [laravel-assets].
No security vulnerability advisories found
> @php artisan key:generate --ansi
[INFO] Application key set successfully.
```

VI. Structure d'un projet Laravel

Une fois le projet créé, on se retrouve avec cette architecture :



1. Répertoires à la racine

- **app**

Ce répertoire contient l'intégralité du code source de notre projet. Il comprend les événements, les commandes, les exceptions, etc.

- **config**

Comme son nom l'indique, il stocke tous les fichiers de configuration de notre projet.

- **database**

Le répertoire database est l'endroit où nous mettons tous les fichiers de population et de migration. Ils déterminent la structure de la base de données.

- **public**

Ce répertoire contient le fichier index.php, qui est le point d'entrée de toutes les requêtes. Nous devons également placer tous les fichiers statiques (CSS et JS) dans ce répertoire qui seront générés à partir des fichiers CSS et JS du répertoire resources.

- ***routes***

routes contient toutes les déclarations d'URL pour notre projet. Par défaut, il y a quatre fichiers route : web.php, api.php, console.php et channels.php.

- ***resources***

Ce dossier stocke toutes les vues et les fichiers non compilés tels que LESS, SASS ou JavaScript.

[4. Répertoires du dossier app](#)

- ***Http/Controllers***

C'est ici que nous plaçons tous les contrôleurs de notre projet. Toute la logique permettant de traiter les demandes entrant dans votre application sera placée dans ce répertoire. Pour rappel, les contrôleurs sont les intermédiaires entre nos vues et nos modèles.

- ***Models***

Le dossier Models (venu avec Laravel 8) contient toutes vos classes de modèles Eloquent. L'ORM Eloquent inclus avec Laravel fournit une belle et simple implémentation ActiveRecord pour travailler avec votre base de données. Chaque table de votre base de données a un « modèle » correspondant qui est utilisé pour interagir avec cette table. Les modèles vous permettent d'interroger les données de vos tables, ainsi que d'insérer de nouveaux enregistrements dans la table.

[5. Configuration du projet](#)

Toutes les configurations d'environnement dans Laravel sont stockées dans le fichier .env dans le répertoire racine. Pour que notre projet fonctionne correctement, il y a quelques changements que nous devons faire.

VII. Mettre à jour le fichier .env

Dans le fichier .env il faut préciser la base de données précédemment créée. Cette étape explique comment établir une connexion à la base de données en ajoutant le nom de la base de données, le nom d'utilisateur et le mot de passe dans le fichier de configuration .env de votre projet.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel9
DB_USERNAME=root
DB_PASSWORD=
```

VIII. Démarrer le serveur

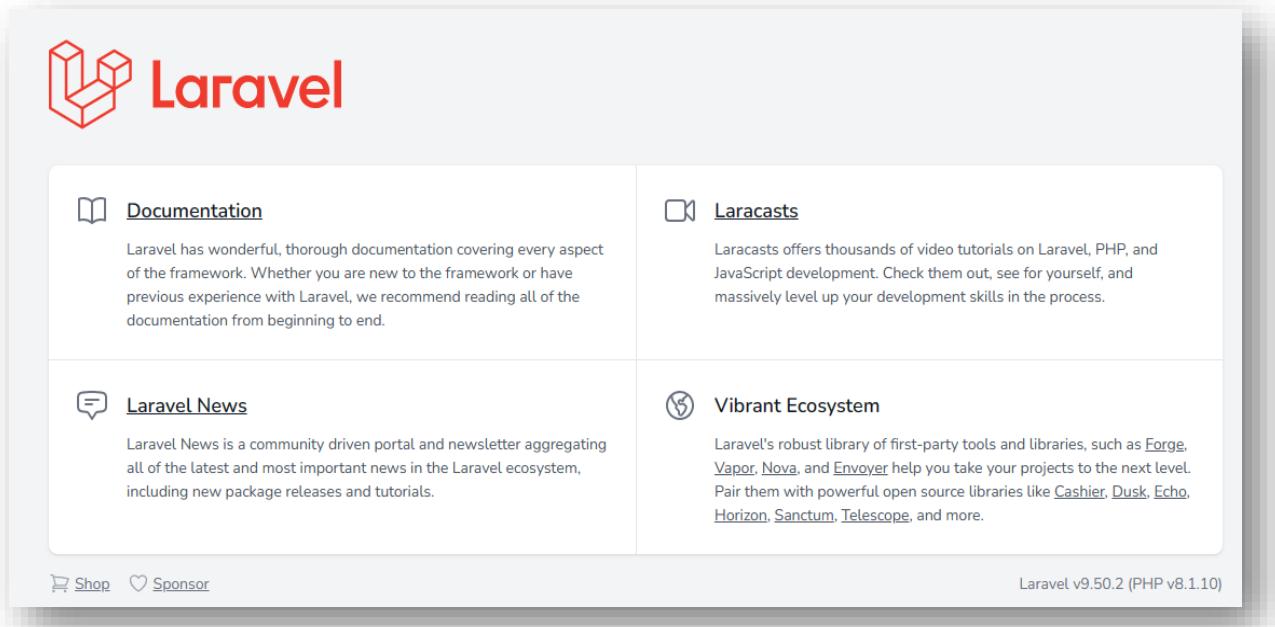
La commande qui démarre votre serveur de développement est la suivante :

```
php artisan serve
```

Laravel a sa propre interface de ligne de commande appelée artisan. Il est très utile pour effectuer une foule de tâches, dont la création d'un squelette pour un contrôleur. Pour l'utiliser, vous devez ouvrir une console puis vous placer dans le dossier de votre application.

```
PS C:\Enseignement\Laravel> cd projet1
PS C:\Enseignement\Laravel\projet1> php artisan serve
    INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

Dans le navigateur à l'adresse <http://localhost:8000/> on pourra visualiser la page de bienvenue de Laravel.



Quels sont les fichiers à l'origine de cette page ?

Pour cela dirigeons nous vers le fichier "web.php" situé dans le dossier « routes ».

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

Nous pouvons voir que la route '/' retourne une method view() avec comme paramètre 'welcome'.

Qu'est-ce-que tout cela veut dire ?

Cela veut dire que pour la route '/' on retourne une vue appelée « welcome ».

Qui se trouve plus précisément dans « resources/views/welcome.blade.php » (Blade étant le moteur de Template utilisé par Laravel).

Tutoriel 2 : Application CRUD avec Laravel & Blade

A la fin de ce tutoriel les objectifs atteints sont :

- Gérer les migrations
- Définir les modèles
- Maîtriser Eloquent
- Comprendre le mécanisme de routage avec Laravel
- Utiliser artisan (seeder, factories, ...)
- Créer et manipuler un Controller
- Lier un Controller à une Route
- Créer des vues avec Blade.

I. Introduction

L'objectif générale de ce tutoriel, est la création d'une application CRUD, tout en utilisant Laravel et son moteur de Template Blade. Pour cela, nous allons commencer par voir les migrations et les modèles. Ainsi que l'ORM Eloquent qui permet une représentation des tables sous forme d'objets pour simplifier les manipulations des enregistrements. Nous allons aussi découvrir l'outil Artisan qui est la boîte à outil du développeur pour Laravel. Nous verrons aussi le système de routage pour trier les requêtes. Et enfin les contrôleurs qui servent à traiter les requêtes dirigées par les routes et à fournir une réponse au client.

Indication :

On suppose que le projet est déjà créé et les configurations sont faites (voir tutoriel 1).

II. Migrations

1. C'est quoi les migrations ?

Les migrations sont comme le contrôle de version de votre base de données, permettant à votre équipe de modifier et de partager le schéma de base de données de l'application.

Les migrations sont généralement associées au générateur de schéma de Laravel pour créer le schéma de base de données de votre application.

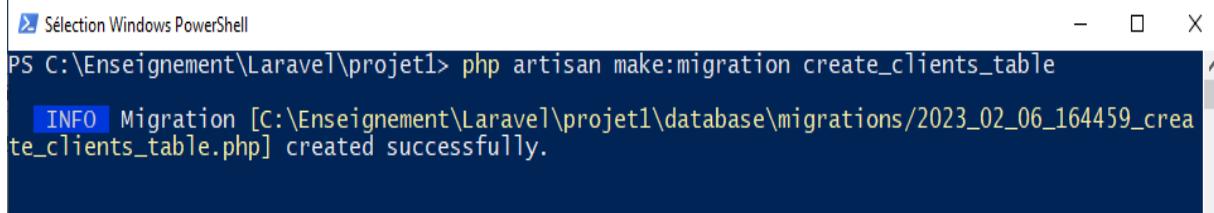
La migration est un processus de gestion de votre base de données en écrivant PHP plutôt que SQL. Il fournit également un moyen d'ajouter un contrôle de version à votre base de données.

La façade Laravel Schema fournit un support indépendant de la base de données pour la création et la manipulation de tables sur tous les systèmes de base de données pris en charge par Laravel.

2. Créez une migration

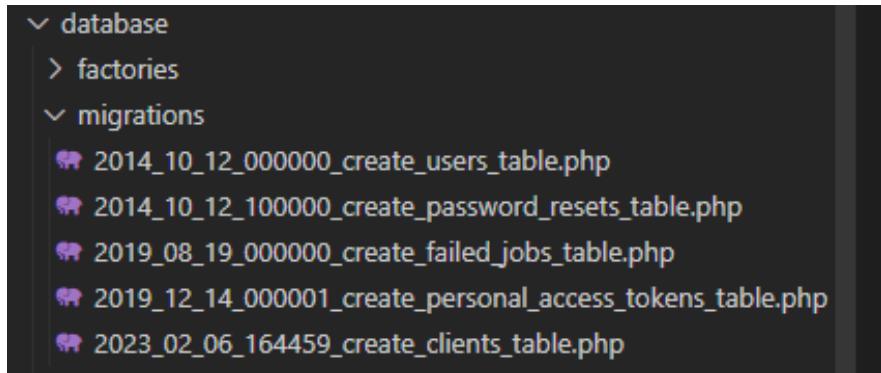
Pour créer une migration, utilisez la commande ***make:migration*** sur l'Artisan CLI :

```
php artisan make:migration create_clients_table
```



```
PS C:\Enseignement\Laravel\projet1> php artisan make:migration create_clients_table
INFO Migration [C:\Enseignement\Laravel\projet1\database\migrations\2023_02_06_164459_create_clients_table.php] created successfully.
```

La nouvelle migration sera placée dans votre répertoire database/migrations. Chaque nom de fichier de migration contient un horodatage, ce qui permet à Laravel de déterminer l'ordre des migrations.



Le contenu généré automatiquement est le suivant :

```
<?php

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

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

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
```

```

        Schema::dropIfExists('clients');
    }
}

```

On dispose dans cette classe de deux fonctions :

up : ici on a le code de création de la table et de ses colonnes

down : ici on a le code de suppression de la table

Dans la fonction up() , nous avons la méthode create de la façade Schema (schema builder) et le premier argument de cette fonction est le nom de la table à créer. Le deuxième argument est une fonction avec un objet Blueprint comme paramètre et pour définir la table (Un blueprint est simplement le plan d'un objet).

Dans la fonction down(), nous avons une méthode dropIfExists du générateur de schéma qui, lorsqu'elle est appelée, supprimera la table.

3. Personnaliser le contenu

Ajoutez le code suivant dans le fichier : database/migrations/create_clients_table.php

```

<?php

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

class CreateClientsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('clients', function (Blueprint $table) {
            $table->id();
            $table->string('npr');
            $table->text('adresse');
            $table->string('email');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
}

```

```
{  
    Schema::dropIfExists('clients');  
}  
}
```

Une migration permet de créer et de mettre à jour un schéma de base de données. Autrement dit, vous pouvez créer des tables, des colonnes dans ces tables, en supprimer, créer des index... Tout ce qui concerne la maintenance de vos tables peut être pris en charge par cet outil. Vous avez ainsi un suivi de vos modifications.

Eloquent suppose que chaque table possède une colonne de clé primaire appelée id. Vous pouvez définir une propriété \$primaryKey pour remplacer cette convention : public \$primaryKey = 'ncin' ;

De plus, Eloquent suppose que la clé primaire est une valeur entière incrémentée, ce qui signifie que, par défaut, la clé primaire sera automatiquement convertie en int. Si vous souhaitez utiliser une clé primaire non incrémentée ou non numérique, vous devez définir la propriété publique \$incrementing sur votre **modèle** sur false : public \$incrementing = false ;

Par défaut, Eloquent s'attend à ce que les colonnes created_at et updated_at existent sur vos tables. Si vous ne souhaitez pas que ces colonnes soient gérées automatiquement par Eloquent, définissez la propriété \$timestamps sur votre modèle sur false : public \$timestamps = false;

4. Exécution de migrations

Pour exécuter toutes vos migrations en cours, exécutez la commande migrate Artisan :

```
php artisan migrate
```

```
PS C:\Enseignement\Laravel\projet1> php artisan migrate  
[WARN] The database 'laravel19' does not exist on the 'mysql' connection.  
Would you like to create it? (yes/no) [no]  
y  
[INFO] Preparing database.  
Creating migration table ..... 55ms DONE  
[INFO] Running migrations.  
2014_10_12_000000_create_users_table ..... 75ms DONE  
2014_10_12_100000_create_password_resets_table ..... 68ms DONE  
2019_08_19_000000_create_failed_jobs_table ..... 51ms DONE  
2019_12_14_000001_create_personal_access_tokens_table ..... 74ms DONE  
2023_02_06_164459_create_clients_table ..... 32ms DONE  
PS C:\Enseignement\Laravel\projet1>
```

Attention :

- Le serveur de base de données MySQL doit être démarré.
- La base de données déjà créée et définie dans .env (voir tutoriel 1)

Le résultat de cette commande est la création des tables correspondantes dans la base de données.

Server: localhost:3306 » Database: laravel9

Table	Action	Rows	Type	Collation	Size	Overhead
clients	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
failed_jobs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
migrations	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
personal_access_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
users	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
6 tables	Sum	5	InnoDB	utf8mb4_0900_ai_ci	96.0 KiB	0 B

Et plus précisément les champs sont définis dans la table clients.

Server: localhost:3306 » Database: laravel9 » Table: clients

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint	UNSIGNED	No	<i>None</i>		AUTO_INCREMENT	Change Drop More	
2	npr	varchar(255)	utf8mb4_unicode_ci	No	<i>None</i>			Change Drop More	
3	adresse	text	utf8mb4_unicode_ci	No	<i>None</i>			Change Drop More	
4	email	varchar(255)	utf8mb4_unicode_ci	No	<i>None</i>			Change Drop More	
5	created_at	timestamp		Yes	<i>NULL</i>			Change Drop More	
6	updated_at	timestamp		Yes	<i>NULL</i>			Change Drop More	

Remarques :

- Pour annuler une migration on utilise la commande `rollback`. Cette commande annule le dernier "lot" de migrations, qui peut inclure plusieurs fichiers de migration.

```
php artisan migrate:rollback
```

Les méthodes `down` des migrations sont exécutées et les tables seront supprimées.

- La commande `fresh` supprime automatiquement les tables concernées.

```
php artisan migrate:fresh
```

- Pour annuler et relancer en une seule opération on utilise la commande **refresh**. La commande **migrate:refresh** annulera toutes vos migrations, puis exécutera la commande **migrate**. Cette commande recrée efficacement toute votre base de données.

```
php artisan migrate:refresh
```

- La commande **migrate:reset** annulera toutes les migrations de votre application :

```
php artisan migrate:reset
```

En tapant `php artisan` on pourra visualiser la liste des commandes :

<code>migrate</code>	
<code>migrate:fresh</code>	Drop all tables and re-run all migrations
<code>migrate:install</code>	Create the migration repository
<code>migrate:refresh</code>	Reset and re-run all migrations
<code>migrate:reset</code>	Rollback all database migrations
<code>migrate:rollback</code>	Rollback the last database migration
<code>migrate:status</code>	Show the status of each migration

III. Le modèle de données

Votre application Web Laravel pourrait communiquer directement avec les tables de la base de données. Cependant, il sera plus intéressant d'utiliser des modèles, c'est-à-dire une représentation objet de chacune des tables.

Lorsqu'on utilise une telle couche, placée entre le code et la base de données, on dira qu'on fait du mapping objet-relationnel, souvent référencé comme ORM (object-relational mapping). L'ORM livré avec Laravel s'appelle Eloquent.

1. Génération du modèle

Dès que votre base de données est modélisée (voir tutoriel 1), vous devez créer un modèle pour vos tables.

Le fichier dans lequel le modèle est défini sera généré à l'aide de la commande :

`php artisan make:model` suivie du nom du modèle à produire.

Pour suivre les standards de Laravel, le nom du modèle doit débuter par une majuscule et être au singulier.

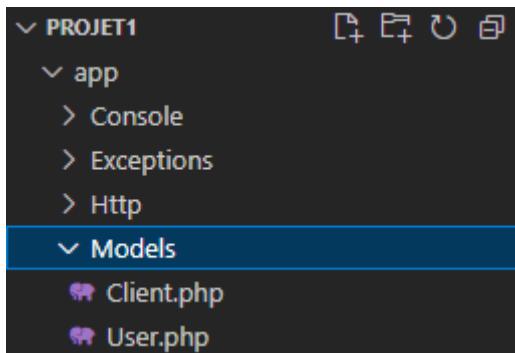
Exemple :

```
php artisan make:model Client
```

```
PS C:\Enseignement\Laravel\projet1> php artisan make:model Client
[INFO] Model [C:\Enseignement\Laravel\projet1\app\Models\Client.php] created successfully.
PS C:\Enseignement\Laravel\projet1>
```

Notez qu'il est rarement utile de créer un modèle pour les tables pivot (tables intermédiaires dans les relations de plusieurs à plusieurs) et ce, même si la table pivot contient des informations autres que les clés étrangères. Eloquent vous offrira d'autres alternatives pour accéder à ces informations.

La commande précédente créera un fichier nommé Client.php placé directement dans le dossier **app**, sous le dossier de votre projet.



2. Contenu du modèle :

Voici le contenu initial généré automatiquement du modèle :

```
<?php

namespace App\Models;

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

class Client extends Model
{
    use HasFactory;
}
```

Par convention le nom du Model est le singulier du nom de la table qui lui correspond dans la BD. Si le nom de la table est différent, il faut le spécifier dans le model : protected \$table = 'my_clients';

Selon la documentation officielle de Laravel, « les usines » (factories) sont des classes qui étendent la classe factory de Laravel et définissent une propriété de modèle et une méthode de définition. Pensez-y simplement de cette façon. Si vous avez déjà créé un modèle, comme le modèle Client (App \ Models \ Client), vous saurez que certains champs doivent être renseignés lorsque vous utilisez la méthode create ().

Dans vos tests, vous pouvez avoir des centaines d'endroits où vous devez générer un nouveau client. Si jamais vous changez votre code, vous devrez le changer à des centaines d'endroits différents (par exemple, en ajoutant des rôles à vos clients). Vous créez simplement un client de test afin de pouvoir exploiter une factory dans laquelle il créera ce client de test pour vous.

Laravel a déjà créé une factory User pour nous. Il est situé dans database / factories / UserFactory.php.

3. Définir les champs

Définissez les valeurs de la table Client dans le fichier app/Models/Client.php

```
<?php
```

```

namespace App\Models;

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

class Client extends Model
{
    use HasFactory;

    protected $fillable = [
        'npr', 'adresse', 'email'
    ];
}

```

Dans l'exemple ci-dessus, nous fournissons les champs de npr, d'adresse et email au modèle Client pour remplir la table des clients. Une factory définira simplement une manière standard pour nous de faire cela.

Le tableau \$fillable nous permet de créer et de mettre à jour le modèle en masse. Une fois que le tableau \$fillable existe, Laravel va accepter d'envoyer nos données à la base de données.

Remarque :

Une seule commande peut à la fois créer le modèle et le fichier de migration en une seule opération :

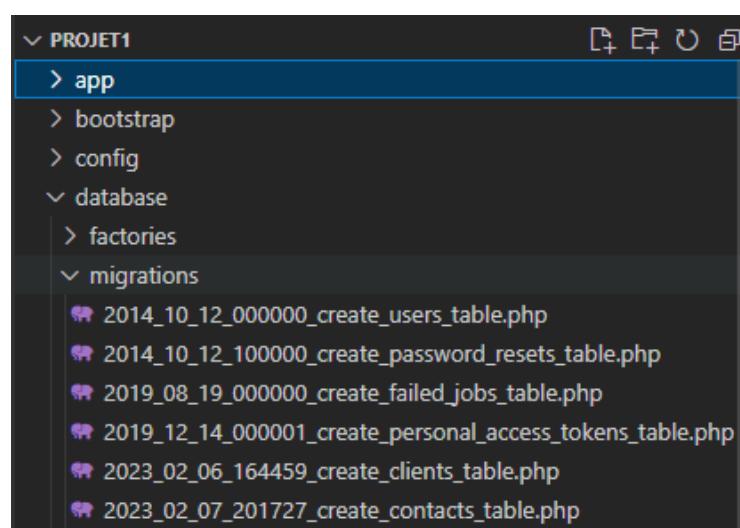
```
php artisan make:model Contact -m
```

ou bien :

```
php artisan make:model Contact --migration
```

Le nom du model peut être placé avant ou après l'option (-m ou --migration)

```
PS C:\Enseignement\Laravel\projet1> php artisan make:model Contact -m
[INFO] Model [C:\Enseignement\Laravel\projet1\app\Models>Contact.php] created successfully.
[INFO] Migration [C:\Enseignement\Laravel\projet1\database\migrations\2023_02_07_201727_create_contacts_table.php] created successfully.
```



IV. Seeders

1. Présentation

Laravel comprend une méthode pour remplir la base de données qui s'appelle seeding (population) pour le test. Toutes les classes seeds sont stockées dans le répertoire database/seeds.

Par convention, les noms des classes de seeding sont écrits dans ce format : ClientsTableSeeder.

2. Générer les fichiers de seeds

Pour générer un Seeder, il suffit d'exécuter la commande suivante :

```
php artisan make:seeder ClientsTableSeeder
```

```
PS C:\Enseignement\Laravel\projet1> php artisan make:seeder ClientsTableSeeder
[INFO] Seeder [C:\Enseignement\Laravel\projet1\database\seeders\ClientsTableSeeder.php] created successfully.
```

Le fichier est généré :

```
└── seeders
    ├── ClientsTableSeeder.php
    └── DatabaseSeeder.php
```

3. Contenu du fichier de seeds

Editer le fichier de seeds.

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class ClientsTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
    }
}
```

Une classe de Seeder ne contient qu'une seule méthode par défaut : run(), dans laquelle vous pouvez insérer des données dans votre BD.

Laravel intègre une bibliothèque très utile nommée **Faker**, qui permet de générer des fausses données à insérer dans la BD.

```
<?php
```

```
namespace Database\Seeders;

use Illuminate\Database\Seeder;

use Illuminate\Support\Facades\DB;

class ClientsTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $faker = \Faker\Factory::create();

        DB::table("clients")->insert([
            "npr" => $faker->name(),
            "adresse" => $faker->address,
            "email" => $faker->unique()->safeEmail,
        ]);
    }
}
```

4. Exécuter le Seeder

- Nous pouvons spécifier la classe qu'on veut effectuer le seed :

```
php artisan db:seed --class= ClientsTableSeeder
```

```
PS C:\Enseignement\Laravel\projet1> php artisan db:seed --class= ClientsTableSeeder
INFO Seeding database.
```

Dans la BD on voit la ligne ajoutée :

✓ Showing rows 0 - 0 (1 total, Query took 0.0007 seconds.)

```
SELECT * FROM `clients`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows:

Extra options

	id	npr	adresse	email	created_at	updated_at
<input type="checkbox"/>  Edit  Copy  Delete	1	Ethelyn West	287 Koepf Garden Suite 149 Simeonside, DE 48838-70...	alec75@example.com	NULL	NULL

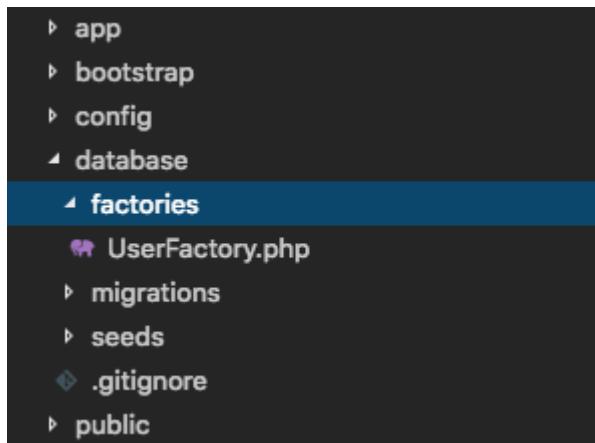
En effet, cette simple méthode montre très vite ses limites lorsque l'on veut complexifier nos enregistrements. C'est pourquoi les Factories vont nous être très utiles.

V. Les Factories

1. Présentation

Les Factories (“usines” en anglais) sont donc là pour nous permettre de créer des enregistrements en quantité et d'établir facilement diverses relations entre nos tables.

Vos Factories se situent dans le dossier /database/factories.



On peut voir que notre UserFactory est déjà créé par défaut. Nous allons donc créer le ClientFactory.

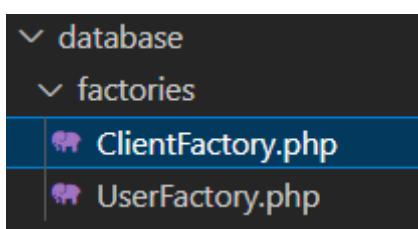
2. Générer le fichier factory

Pour générer un factory, il suffit d'exécuter la commande suivante :

```
php artisan make:factory ClientFactory --model=Client
```

```
PS C:\Enseignement\Laravel\projet1> php artisan make:factory ClientFactory --model=Client
[INFO] Factory [C:\Enseignement\Laravel\projet1\database\factories\ClientFactory.php] created successfully.
```

Le fichier est généré :



3. Contenu du fichier de factory

Editer le fichier de factory.

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
```

```

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Client>
 */
class ClientFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            //
        ];
    }
}

```

Définissons les valeurs de nos champs dans notre ClientFactory :

```

<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Client>
 */
class ClientFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition()
    {
        return [
            "npr" => fake()->name(),
            "adresse" => fake()->address(),
            "email" => fake()->unique()->safeEmail,
        ];
    }
}

```

```
}
```

Ici nous voulons simplement créer 20 Clients donc pas besoin de ClientTableSeeder, modifier la fichier **DatabaseSeeder** en suivant l'exemple du modèle User en commentaire.

```
<?php

namespace Database\Seeders;

// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        \App\Models\Client::factory(20)->create();
        // \App\Models\User::factory(10)->create();

        // \App\Models\User::factory()->create([
        //     'name' => 'Test User',
        //     'email' => 'test@example.com',
        // ]);
    }
}
```

Maintenant effacez à nouveau les enregistrements présents dans votre base de données et envoyez vos nouveaux enregistrements. Voici une commande plus rapide qui combine le migrate:fresh et le php artisan db:seed

```
php artisan migrate:fresh --seed
```

```

PS C:\Enseignement\Laravel\projet1> php artisan migrate:fresh --seed
Dropping all tables ..... 149ms DONE
INFO Preparing database.
Creating migration table ..... 54ms DONE
INFO Running migrations.
2014_10_12_000000_create_users_table ..... 52ms DONE
2014_10_12_100000_create_password_resets_table ..... 63ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 46ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 69ms DONE
2023_02_06_164459_create_clients_table ..... 26ms DONE
2023_02_07_201727_create_contacts_table ..... 24ms DONE
INFO Seeding database.

```

Et voilà, on vient d'ajouter 20 clients fake à notre base de données.

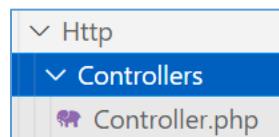
Table: clients								
			id	npr	adresse	email	created_at	updated_at
<input type="checkbox"/>				13	Ms. Jacklyn Wuckert	1116 Genoveva Island Suite 701 North Bartmouth, ME...	isidro85@example.net	2023-02-08 00:35:48
<input type="checkbox"/>				14	Agnes Paucek	5938 Walton Rapids Willberg, GA 46944-1101	casimer13@example.net	2023-02-08 00:35:48
<input type="checkbox"/>				15	Hollie Bergstrom	876 Clemmie Neck Suite 059 North Hilarioland, DE 3...	monserrate36@example.com	2023-02-08 00:35:48
<input type="checkbox"/>				16	Mr. Joe Kozey	2649 Koepf Square New Kaylahshire, OR 79330	melany63@example.com	2023-02-08 00:35:48
<input type="checkbox"/>				17	Alberta Jacobi MD	78519 Kuhn Mission New Amieview, DE 08055-9857	reynold.schowalter@example.com	2023-02-08 00:35:48
<input type="checkbox"/>				18	Dr. Amely Brown	310 Eleonore Camp Suite 414 Port Mallorymouth, MS ...	nking@example.org	2023-02-08 00:35:48
<input type="checkbox"/>				19	Keyon Ledner	92051 Antonetta Crest Apt. 431 Hillborough, IA 548...	christa.kertzmann@example.com	2023-02-08 00:35:48
<input type="checkbox"/>				20	Monique Wuckert	268 Ford Junction Shainaberg, NM 56985	cdaniel@example.net	2023-02-08 00:35:48

VI. Le Controller

1. Rôle

La tâche d'un contrôleur est de réceptionner une requête et de définir la réponse appropriée. Les Controllers permettent de regrouper les logiques de réponse aux requêtes http reliées dans la même classe.

On trouve les Controllers dans le dossier App\Http\Controllers.



2. Créer un Controller

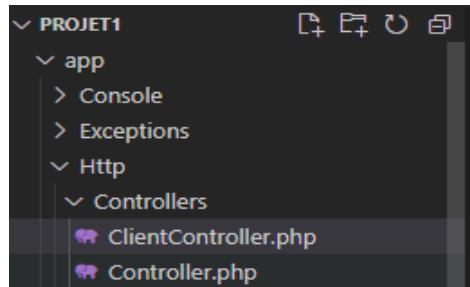
La création des Controller peut être faite manuellement, cependant, l'utilisation de la console intégrée de Laravel : Artisan s'avère une meilleure solution.

Pour créer un contrôleur nous allons utiliser Artisan. Dans la console entrez cette commande :

```
php artisan make:controller ClientController
```

```
PS C:\Enseignement\Laravel\projet1> php artisan make:controller ClientController
[INFO] Controller [C:\Enseignement\Laravel\projet1\app\Http\Controllers\ClientController.php] created successfully.
PS C:\Enseignement\Laravel\projet1>
```

Vous allez trouver le contrôleur créé sous le dossier Controllers/Http :



Le code généré automatiquement :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class ClientController extends Controller
{
    //
}
```

On trouve en premier l'espace de nom (App\Http\Controllers), le contrôleur hérite de la classe Controller qui se trouve dans le même dossier et qui permet de factoriser des actions communes à tous les contrôleurs.

3. Générer le Controller avec les méthodes

On peut créer des méthodes dans le Controller. Mais php artisan make:controller dispose d'options pour définir les méthodes d'opérations CRUD automatiquement.

Ces options sont --api et --resource

Avec --resource le contrôleur contiendra une méthode pour chacune des opérations de ressources disponibles – index(), create(), store(), show(), edit(), update(), destroy().

Avec --api utilisée comme --resource mais génère 5 méthodes : index(), store(), show(), update(), destroy(). Puisque les formulaires create/edit n'ont pas de besoin dans API.

Supprimez le contrôleur ClientController s'il est déjà créé puis exécutez cette commande :

```
php artisan make:controller ClientController --resource
```

On visualise le contenu du fichier pour voir les méthodes qui sont créées.

```
<?php

namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;

class ClientController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
```

```

/*
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    //
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    //
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    //
}

```

Laravel nous permet de travailler avec un objet qui représente la requête HTTP courante. Il s'agit d'un objet de type `\Illuminate\Http\Request`. Cet objet nous donnera accès à une foule d'informations, allant de l'URL de la page actuellement affichée jusqu'aux entêtes HTTP en passant par les cookies et les informations entrées dans un formulaire.

Dans une méthode d'action, pour travailler avec la variable `$request`, il faut demander à Laravel de l'injecter dans la méthode. Ceci sera fait simplement en ajoutant un paramètre de type `Request`.

Attention :

Pour pouvoir élaborer les opérations de CRUD sur le model il faut le déclarer dans le Controller :

```
use App\Models\Client;
```

4. La méthode index

Permet d'afficher tous les enregistrements de la table.

```
public function index()
{
    $clients = Client::all();
    return view('clients.index', compact('clients'));
// -> resources/views/clients/index.blade.php
}
```

Eloquent est le nom de l'ORM utilisé par Laravel. Celui-ci utilise une implémentation ActiveRecord qui est un design pattern permettant de lire les données d'une base de données. Il fonctionne en encapsulant les attributs d'une table ou d'une vue dans une Class. Les lignes de la table deviennent donc des « tuple » côté PHP

Client::all() vous permet de retourner un objet de la Class Collection et qu'il contient un tableau avec les différents objets de la Class Client.

L'une des utilités de la Collection est de permettre de ne faire qu'une requête sql, puis de rassembler les infos dans une collection. Ensuite lorsque nous voudrons manipuler telle ou telle donnée plus précisément nous n'aurons qu'à effectuer les actions sur notre collection sans avoir à faire une nouvelle requête sql.

5. La méthode create

Permet d'ouvrir la vue de création d'un nouveau client

```
public function create()
{
    return view('clients.create'); // ->
resources/views/clients/create.blade.php
}
```

6. La méthode store

Permet d'enregistrer un enregistrement dans la table et d'ouvrir la liste des clients

```
public function store(Request $request)
{
    $client = new Client([
        'npr' => $request->input('npr'),
        'adresse' => $request->input('adresse'),
        'email' => $request->input('email'),
    ]);
    $client->save();
    return redirect('/clients')->with('success', 'Client saved.');
// -> resources/views/clients/index.blade.php
}
```

Pour créer un nouvel enregistrement dans la base de données, créez une nouvelle instance de modèle, définissez des attributs sur le modèle, puis appelez la méthode save.

Nous affectons les paramètre npr, adresse et email de la requête HTTP entrante aux attributs correspondants de l'instance de modèle App\Models\Client. Lorsque nous appelons la méthode save, un enregistrement sera inséré dans la base de données. Les horodatages created_at et updated_at seront automatiquement définis lorsque la méthode save est appelée, il n'est donc pas nécessaire de les définir manuellement.

7. La méthode show

Permet d'afficher un seul client à travers la vue show

```
public function show($id)
{
    $client = Client::find($id);
    return view('clients.show', compact('client'));

}
```

On récupère un enregistrement unique en utilisant find. Au lieu de renvoyer une collection de modèles, cette méthode renvoie une seule instance du modèle.

8. La méthode edit

Permet d'ouvrir la vue edit avec comme argument le client à modifier.

```
public function edit($id)
{
    $client = Client::find($id);
    return view('clients.edit', compact('client'));
    // -> resources/views/clients/edit.blade.php
}
```

9. La méthode update

Permet de faire des mises à jour des données de la table et de retourner à la liste des clients.

```
public function update(Request $request, $id)
{
    $client = Client::find($id);
    $client->update($request->all());
    return redirect('/clients')->with('success', 'Client updated.');
    // -> resources/views/clients/index.blade.php
}
```

La méthode update attend un tableau de paires de colonnes et de valeurs représentant les colonnes à mettre à jour.

La méthode save peut également être utilisée pour mettre à jour des modèles qui existent déjà dans la base de données. Pour mettre à jour un modèle, vous devez le récupérer, définir les attributs que vous souhaitez mettre à jour, puis appeler la méthode save.

10. La méthode destroy

Après la création et la modification vient tout naturellement la suppression avec la méthode delete.

```
public function destroy($id)
{
    $client = Client::find($id);
    $client->delete();
    return redirect('/clients')->with('success', 'Client removed.');
    // -> resources/views/clients/index.blade.php
}
```

La suppression est définitive. Il existe aussi une possibilité de suppression provisoire (soft delete) mais il faut modifier le modèle en ajoutant un trait et une propriété.

Remarque :

On peut utiliser une seule commande pour créer model migrate et controller avec l'option -mcr qui est le raccourci de --migration --controller --resource

```
php artisan make:model Contact --migration --controller --resource
```

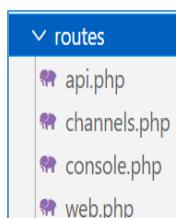
ou

```
php artisan make:model Contact --migration --controller --api
```

VII. Routing

1. Présentation

Toutes les Routes de Laravel sont définies dans le dossier « Routes ». Ces fichiers sont chargés automatiquement par le Framework depuis ce dossier.



Le fichier **web.php** contient les routes de l'interface Web de l'application. Les autres fichiers concernent des routes plus spécifiques comme pour les API avec le fichier **api.php** dont les routes sont affectées au groupe de middleware API et elles sont sans état. Ainsi que les routes pour les actions en ligne de commande avec le fichier **console.php**.

Il y a une commande pour connaître les routes prévues dans le code qui est :

```
php artisan route:list
```

Voici ce que ça donne avec une nouvelle installation :

```
PS C:\Enseignement\Laravel\projet1> php artisan route:list
GET|HEAD / ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
POST GET|HEAD _ignition/health-check ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST GET|HEAD _ignition/update-config ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ..... sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
Showing [6] routes
```

La commande `route:list` peut être utilisée pour afficher une liste de toutes les routes enregistrées pour l'application. Cette commande affiche le domaine, la méthode, l'URI, le nom, l'action et le middleware pour les routes qu'elle inclut dans la table générée.

2. Le Router

Le Router définit les routes pouvant répondre aux verbes http suivants :

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

3. Route Groups

Les groupes d'itinéraires (Route Groups) sont une fonctionnalité essentielle de Laravel, qui vous permet de regrouper tous les itinéraires.

Les groupes de routes sont utiles lorsque vous souhaitez appliquer les attributs à toutes les routes. Si vous utilisez des groupes de routes, vous n'avez pas besoin d'appliquer les attributs individuellement à chaque route ; cela évite les doublons.

Il vous permet de partager les attributs tels que le middleware ou les espaces de noms, sans définir ces attributs sur chaque route individuelle.

Ces attributs partagés peuvent être transmis dans un format de tableau en tant que premier paramètre à la méthode `Route::group()`.

```
Route::group( [ ] , callback);
```

Allez dans le fichier `routes/web.php` et modifier le comme suit :

```
<?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!
|
```

```

| */
use App\Http\Controllers\ClientController;

Route::resource('clients', ClientController::class);

```

Réexécuter la commande :

```
php artisan route:list
```

Et voir les changements :

```
PS C:\Enseignement\Laravel\projet> php artisan route:list
POST      _ignition/execute-solution ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST      _ignition/update-config .... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD clients ..... clients.index > ClientController@index
POST      clients ..... clients.store > ClientController@store
GET|HEAD clients/create ..... clients.create > ClientController@create
GET|HEAD clients/{client} ..... clients.show > ClientController@show
PUT|PATCH clients/{client} ..... clients.update > ClientController@update
DELETE   clients/{client} ..... clients.destroy > ClientController@destroy
GET|HEAD clients/{client}/edit ..... clients.edit > ClientController@edit
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
Showing [11] routes
```

VIII. Crédation des vues avec Blade

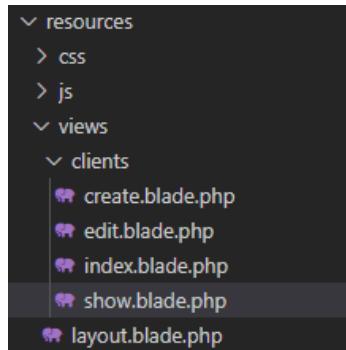
Blade est le moteur de Template utilisé par Laravel. Son but est de permettre d'utiliser du php sur notre vue mais d'une manière assez particulière. Pour créer un fichier qui utilise le moteur de template Blade il vous faut ajouter l'extension “.blade.php”. Comme nous l'avons vu dans la présentation de l'architecture de Laravel, les fichiers de vos vues se situent dans le dossier resources/views.

La première fonctionnalité la plus basique de Blade est l'affichage d'une simple variable comme nous l'avons vu dans le chapitre précédent. Exemple : Si je transmets à ma vue la variable \$maVariable = 'Hello World !' Dans ma vue {{ \$maVaribale }} affichera 'Hello World' .

Dans cette étape, nous devons créer uniquement les fichiers Blades. Nous devons donc principalement créer un fichier de mise en page, puis créer un nouveau dossier “clients”, puis créer des fichiers de Blade de l'application CRUD.

Donc, finalement, vous devez créer les fichiers de Blade ci-dessous :

- 1) layout.blade.php
- 2) index.blade.php
- 3) create.blade.php
- 4) edit.blade.php
- 5) show.blade.php



1. layout.blade.php

Créer ce fichier dans le répertoire « views ».

Ce template comporte la structure globale des pages et est déclaré comme parent par les autres vues : `@extends('template')`

```
<!-- layout.blade.php -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Projet1 Laravel 9 CRUD</title>
    <link href="{{ asset('css/app.css') }}" rel="stylesheet" type="text/css" />
    <link rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"
        integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
        crossorigin="anonymous">
</head>

<body>
    <div class="container">
        @yield('content')
    </div>
    <script src="{{ asset('js/app.js') }}" type="text/js"></script>
</body>
</html>
```

`@yield('content')` sera remplacé par le contenu spécifique à la vue qui hérite de layout

2. Index.blade.php

Créer ce fichier et les autres suivants sous le répertoire clients

C'est la vue qui va afficher la liste de tous les clients

```

<!-- index.blade.php -->

@extends('layout')

@section('content')

<style>
    .uper {
        margin-top: 40px;
    }
</style>



@if(session()->get('success'))
    <div class="alert alert-success">
        {{ session()->get('success') }}
    </div><br />
@endif



# Liste des clients



|                  |                   |                       |                     |                          |                          |                                                                                                                                                                                        |
|------------------|-------------------|-----------------------|---------------------|--------------------------|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ID               | Nom Prénom        | Adresse               | Email               | créé le                  | Mis à jour le            | Action                                                                                                                                                                                 |
| {{\$client->id}} | {{\$client->npr}} | {{\$client->adresse}} | {{\$client->email}} | {{\$client->created_at}} | {{\$client->updated_at}} | <a class="btn btn-success" href="{{ route('clients.show', \$client-&gt;id) }}">Détails</a> <a class="btn btn-primary" href="{{ route('clients.edit', \$client-&gt;id) }}">Modifier</a> |


```

```

        <form action="{{ route('clients.destroy', $client->id)}}"
method="post">
    @csrf
    @method( 'DELETE' )
    <button class="btn btn-danger"
type="submit">Supprimer</button>
</form>
</td>
</tr>
@endforeach
</tbody>
</table>
<div>
@endsection

```

3. Create.blade.php

Une qui permet d'afficher le formulaire d'ajout d'un nouveau client.

```

@extends('layout')

@section('content')
<style>
.uper {
    margin-top: 40px;
}
</style>

<div class="card uper">
    <div class="card-header">
        Ajouter un client
    </div>

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

```

```

<form method="post" action="{{ route('clients.store') }}">
    @csrf
    <div class="form-group">
        <label for="npr">Nom et prénom:</label>
        <input type="text" class="form-control" name="npr"/>
    </div>

    <div class="form-group">
        <label for="adresse">Adresse :</label>
        <input type="text" class="form-control" name="adresse"/>
    </div>

    <div class="form-group">
        <label for="email">Email :</label>
        <input type="text" class="form-control" name="email"/>
    </div>
    <button type="submit" class="btn btn-primary">Ajouter</button>
</form>
</div>
</div>
@endsection

```

4. Show.blade.php

Une vue qui permet d'afficher un seul client

```

@extends('layout')
@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="float-start">
                <h2> Détails client</h2>
            </div>
            <div class="float-end">
                <a class="btn btn-outline-primary" href="{{ route('clients.index') }}> Retour</a>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>ID</strong>
                {{ $client->id }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">

```

```

<div class="form-group">
    <strong>Nom et prénom</strong>
    {{ $client->npr }}
</div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
        <strong>Adresse</strong>
        {{ $client->adresse }}
    </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
        <strong>Email</strong>
        {{ $client->email }}
    </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
        <strong>Créé le</strong>
        {{ $client->created_at }}
    </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
        <strong>Modifié le</strong>
        {{ $client->updated_at }}
    </div>
</div>

</div>
@endsection

```

5. Edit.blade.php

Permettant d'afficher le formulaire de modification d'un client.

```

@extends('layout')
@section('content')
<style>
    .uper {
        margin-top: 40px;
    }
</style>

```

```

<div class="card uper">
    <div class="card-header">
        Modifier le client
    </div>

    <div class="card-body">

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

        <form method="post" action="{{ route('clients.update', $client->id ) }}>
            <div class="form-group">
                @csrf
                @method('PATCH')
                <label for="npr">Nom et prénom :</label>
                <input type="text" class="form-control" name="npr" value="{{ $client->npr }}"/>
            </div>

            <div class="form-group">
                <label for="adresse">Adresse :</label>
                <input type="text" class="form-control" name="adresse" value="{{ $client->adresse }}"/>
            </div>

            <div class="form-group">
                <label for="email">Email :</label>
                <input type="text" class="form-control" name="email" value="{{ $client->email }}"/>
            </div>
            <button type="submit" class="btn btn-primary">Modifier</button>
        </form>
    </div>
</div>
@endsection

```

Aperçu des vues

Taper l'adresse : <http://localhost:8000/clients/> pour consulter la liste

Et <http://localhost:8000/clients/create> pour l'ajout

Liste des clients

ID	Nom Prénom	Adresse	Email	créé le	Mis à jour le	Action
2	Sylvia Cummerata	2867 Vladimir Common North Pearlineville, MS 41290-8988	blick.brenda@example.org	2023-02- 08 00:35:48	2023-02- 08 00:35:48	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>
3	Prof. Victor Rempel DDS	48231 Armand Forge Apt. 877 Carrollport, WV 14152	swaniawski.eladio@example.com	2023-02- 08 00:35:48	2023-02- 08 00:35:48	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>
4	Denis Wiegand	3563 Weimann Loop Apt. 089 Framitown, NM 06417	brakus.kaylee@example.net	2023-02- 08 00:35:48	2023-02- 08 00:35:48	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>
5	Naomie Hoppe	35685 Wilderman Orchard Suite 781 Port Quintontown,	wernser@example.org	2023-02- 08 00:35:48	2023-02- 08 00:35:48	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>

Détails client

[Retour](#)

ID 2

Nom et prénom Sylvia Cummerata

Adresse 2867 Vladimir Common North Pearlineville, MS 41290-8988

Email blick.brenda@example.org

Créé le 2023-02-08 00:35:48

Modifié le 2023-02-08 00:35:48

Modifier le client

Nom et prénom :

Borchani Anis

Adresse :

Route Mahdia Km 2.5, Sfax. 3000

Email :

borchani_anis@yahoo.fr

Modifier

Client updated.

Liste des clients

ID	Nom Prénom	Adresse	Email	créé le	Mis à jour le	Action
2	Borchani Anis	Route Mahdia Km 2.5, Sfax. 3000	borchani_anis@yahoo.fr	2023-02- 08 00:35:48	2023-02- 10 20:51:33	Détails Modifier Supprimer
3	Prof. Victor Rempel DDS	48231 Armand Forge Apt. 877 Carrollport, WV 14152	swaniawski.eladio@example.com	2023-02- 08 00:35:48	2023-02- 08 00:35:48	Détails Modifier Supprimer

Client removed.

Liste des clients

ID	Nom Prénom	Adresse	Email	créé le	Mis à jour le	Action
2	Borchani Anis	Route Mahdia Km 2.5, Sfax. 3000	borchani_anis@yahoo.fr	2023-02- 08 00:35:48	2023-02- 10 20:51:33	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>
5	Naomie Hoppe	35685 Wilderman Orchard Suite 781 Port Quintontown, IL 93466	wernser@example.org	2023-02- 08 00:35:48	2023-02- 08 00:35:48	<button>Détails</button> <button>Modifier</button> <button>Supprimer</button>

Tutoriel 3 : Application CRUD REST API avec Laravel

A la fin de ce tutoriel les objectifs atteints sont :

- Gérer les migrations et les modèles
- Mettre en place le routage
- Développer les Controllers
- Comprendre les relations entre les modèles

I. Introduction

Dans ce tutoriel, nous allons voir comment créer un REST api CRUD application avec Laravel. En web développement ou programmation, le CRUD est une abréviation qui signifie (CREATE, READ, UPDATE et DELETE).

Pour les applications web, une API (Application Programming Interface) ou « Interface de Programmation d'Applications » en français, est un ensemble de définitions et de protocoles qui permettent à des applications de communiquer et de s'échanger mutuellement des données ou des services.

REST (Representational State Transfert) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer un service web.

On parle d'API REST ou REST API lorsqu'une API respecte les contraintes architecturales de REST.

Nous voulons voir dans ce tutoriel comment mettre en place une API REST qui retourne les données au format JSON dans un projet Laravel.

Pour mettre en place ce système, nous allons commencer par voir les prérequis pour l'API, ensuite le contrôleur de l'API, définir les routes de l'API puis compléter les méthodes du contrôleur.

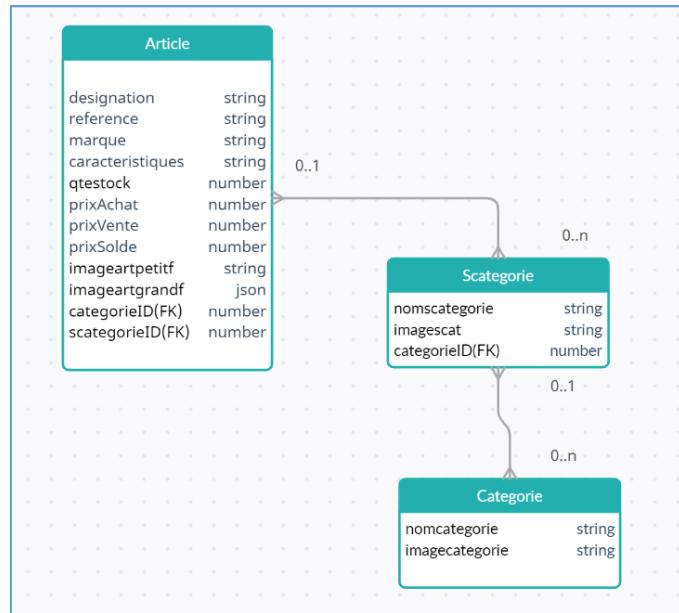
L'architecture de l'application à réaliser est la suivante :



Il s'agit de créer, lire, mettre à jour et supprimer l'API pour effectuer des opérations CRUD dans l'application Laravel qui aura comme interface utilisateur (UI) Vue JS.

D'abord on va préparer l'API et vérifier qu'elle est fonctionnelle pour qu'on puisse ensuite, dans un autre tutoriel, créer notre partie front end.

La base de données qu'on va opter pour ce tutoriel est la suivante :



Indication :

On suppose que le projet est déjà créé et les configurations sont faites (voir tutoriel 1).

II. Cas de la catégorie

1. Génération de catégorie

On peut créer un modèle, un contrôleur de ressources (lié au modèle) ainsi que la migration dans une seule commande :

```
php artisan make:model Categorie --migration --controller --api
```

The screenshot shows the Laravel IDE Helper extension's Explorer panel on the left, which lists the project structure under the 'CRUDAPI' folder. The 'Controllers' folder contains 'CategorieController.php' and 'Controller.php'. The right side of the interface shows the content of 'CategorieController.php' in a code editor. The code defines a controller that extends the 'Controller' class and includes methods for listing and storing categories.

```

EXPLORATEUR
...
CRUDAPI
    Exceptions
    Http
        Controllers
            CategorieController.php
            Controller.php
        Middleware
        Kernel.php
    Models
    Providers
    bootstrap
    config
    database
        factories
        migrations
            2014_10_12_000000_create_users_table.php
            2014_10_12_100000_create_password_reset_token_table.php
            2019_08_19_000000_create_failed_jobs_table.php
            2019_12_14_000001_create_personal_access_tokens_table.php
            2023_02_24_213355_create_categories_table.php
    ...
app > Http > Controllers > CategorieController.php
1  k?php
2
3  namespace App\Http\Controllers;
4
5  use App\Models\Categorie;
6  use Illuminate\Http\Request;
7
8  class CategorieController extends Controller
9  {
10
11     /**
12      * Display a listing of the resource.
13      */
14     public function index()
15     {
16         //
17     }
18
19     /**
20      * Store a newly created resource in storage.
21      */
22     public function store(Request $request)
23     {

```

2. Modifier la migration de catégorie

Sous le dossier database/migrations

```
public function up()
```

```

{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('nomcategorie');
        $table->string('imagecategorie');
        $table->timestamps();
    });
}

```

3. Modifier le model de catégorie

Dans Models/Categorie.php

```

class Categorie extends Model
{
    use HasFactory;
    protected $fillable = [
        'nomcategorie', 'imagecategorie'
    ];
}

```

4. Modifier le controller de catégorie

Dans Http/Controllers/CategorieController.php

```

<?php

namespace App\Http\Controllers;

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

class CategorieController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $categories = Categorie::all()->toArray();
        return array_reverse($categories);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
}

```

```
 */
public function store(Request $request)
{
    $categorie = new Categorie([
        'nomcategorie' => $request->input('nomcategorie'),
        'imagecategorie' => $request->input('imagecategorie')
    ]);
    $categorie->save();

    return response()->json('Catégorie créée !');
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $categorie = Categorie::find($id);
    return response()->json($categorie);
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $categorie = Categorie::find($id);
    $categorie->update($request->all());

    return response()->json('Catégorie MAJ !');
}

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    $categorie = Categorie::find($id);
    $categorie->delete();
```

```
        return response()->json('Catégorie supprimée !');
    }
}
```

5. Mettre en place les routes

Dans routes/api.php

```
use App\Http\Controllers\CategorieController;

Route::middleware('api')->group(function () {
    Route::resource('categories', CategorieController::class);
});
```

6. Exécuter la migration

Exécuter la commande :

```
php artisan migrate
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan migrate
[INFO] Preparing database.
Creating migration table ..... 57ms DONE
[INFO] Running migrations.

2014_10_12_000000_create_users_table ..... 60ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 63ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 50ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 76ms DONE
2023_02_24_213355_create_categories_table ..... 26ms DONE
```

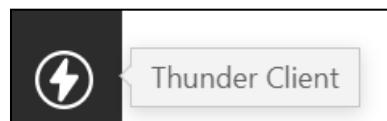
7. Exécuter et faire le test de l'api

Exécuter la commande :

```
php artisan serve
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan serve
[INFO] Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
2023-02-24 23:24:57 .....
2023-02-24 23:24:58 /favicon.ico .....
```

Puis on visualise le résultat de l'adresse url dans l'extension de Visual Studio Code : **Thunder Client**.



<http://localhost:8000/api/categories>

POST ajouter

POST ▼ http://localhost:8000/api/categories

Query Headers² Auth Body¹ Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

```
1  {
2    "nomcategorie": "smartphones",
3    "imagecategorie": "images/categories/smartphones.jpg"
4 }
```

Status: 200 OK Size: 34 Bytes Time: 276 ms

1 "Catégorie créée !"

Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

```
SELECT * FROM `categories`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

← ↑ →	▼ id	nomcategorie	imagecategorie	created_at	updated_at
<input type="checkbox"/>  Edit  Copy  Delete	1	smartphones	images/categories/smartphones.jpg	2023-02-24 22:35:15	2023-02-24 22:35:15

GET

GET ▾ http://localhost:8000/api/categories

Query Headers 2 Auth Body 1 Tests Pre Run

Query Parameters

parameter	value

Status: 200 OK Size: 180 Bytes Time: 299 ms

```

1  [
2    {
3      "id": 1,
4      "nomcategorie": "smartphones",
5      "imagecategorie": "images/categories/smartphones.jpg",
6      "created_at": "2023-02-24T22:35:15.00000Z",
7      "updated_at": "2023-02-24T22:35:15.00000Z"
8    }
9  ]

```

PUT mise à jour

PUT ▾ http://localhost:8000/api/categories/1

Query Headers 2 Auth **Body 1** Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

```

1  {
2    "nomcategorie": "phones",
3    "imagecategorie": "images/categories/smartphones.jpg"
4  }

```

Status: 200 OK Size: 22 Bytes Time: 280 ms

```

1  "Catégorie MAJ !"

```

Showing rows 0 - 0 (1 total, Query took 0.0007 seconds.)

SELECT * FROM `categories`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	id	nomcategorie	imagecategorie	created_at	updated_at
<input type="checkbox"/>	1	phones	images/categories/smartphones.jpg	2023-02-24 22:35:15	2023-02-24 22:47:15

DELETE

DELETE http://localhost:8000/api/categories/1

Query Headers 2 Auth Body 1 Tests Pre Run

Query Parameters

parameter

Status: 200 OK Size: 33 Bytes Time: 313 ms

1 "Catégorie supprimée !"

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0007 seconds.)

```
SELECT * FROM `categories`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

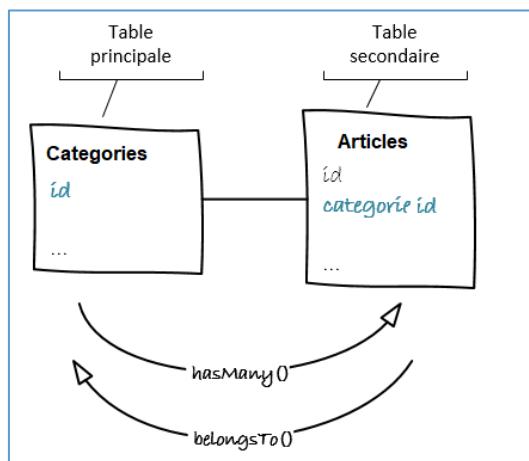
id	nomcategorie	imagecategorie	created_at	updated_at
----	--------------	----------------	------------	------------

III. Relation one to Many dans Laravel

Une relation d'un à plusieurs indique que chaque enregistrement de la table principale (celle contenant la clé primaire) peut avoir 0, 1 ou plusieurs enregistrements dans la table secondaire (celle contenant la clé étrangère). Inversement, chaque enregistrement de la table secondaire est relié à un et exactement un enregistrement dans la table principale.

La méthode `hasMany()` définit une relation de un à plusieurs dans le modèle de la table principale de la relation.

Par exemple, une catégorie peut avoir plusieurs articles. Dans cet exemple, la table `categories` contient la clé primaire de la relation (`id`) et la table `articles` contient la clé étrangère de la relation (`categorie_id`). La relation `hasMany()` sera donc définie dans le modèle de la table `categories`. Alors que la catégorie à laquelle appartient (`belongsTo()`) l'article.



IV. Cas de la sous catégorie

1. Génération de la sous catégorie

On peut créer un modèle, un contrôleur de ressources (lié au modèle) ainsi que la migration dans une seule commande :

```
php artisan make:model Scategorie --migration --controller --api
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan make:model Scategorie --migration --controller --api
[INFO] Model [C:\Enseignement\Laravel\CRUDAPI\app\Models\Scategorie.php] created successfully.
[INFO] Migration [C:\Enseignement\Laravel\CRUDAPI\database\migrations\2023_02_24_225929_create_scategories_table.php]
created successfully.
[INFO] Controller [C:\Enseignement\Laravel\CRUDAPI\app\Http\Controllers\ScategorieController.php] created successfull
```

2. Modifier la migration de la sous catégorie

Sous le dossier database/migrations

```
public function up()
{
    Schema::create('scategories', function (Blueprint $table) {
        $table->id();
        $table->string('nomscategorie');
        $table->string('imagescat');
        $table->unsignedBigInteger('categorieID');
        $table->foreign('categorieID')
            ->references('id')
            ->on('categories')
            ->onDelete('restrict')
            ->onUpdate('restrict');
        $table->timestamps();
    });
}
```

Dans la table scategories on déclare une clé étrangère (foreign) nommée categorieID qui référence (references) la colonne id dans la table (on) categories. En cas de suppression (onDelete) ou de modification (onUpdate) on a une restriction (restrict).

En mettant restrict on empêche la suppression d'une catégorie qui a des sous catégories. On doit donc commencer par supprimer ces sous catégories avant de le supprimer lui-même. On dit que la base assure l'intégrité référentielle. Elle n'acceptera pas non plus qu'on utilise pour categorieID une valeur qui n'existe pas dans la table categories.

Une autre possibilité est cascade à la place de restrict. Dans ce cas si vous supprimez une catégorie ça supprimera en cascade les sous catégories de cette catégorie.

C'est une option qui est rarement utilisée parce qu'elle peut s'avérer dangereuse, surtout dans une base comportant de multiples tables en relation. Mais c'est aussi une stratégie très efficace parce que c'est le moteur de la base de données qui se charge de gérer les enregistrements en relation, vous n'avez ainsi pas à vous en soucier au niveau du code.

3. Modifier le model de la sous catégorie

Dans Models/Scategorie.php

```

<?php

namespace App\Models;

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

class Scategorie extends Model
{
    use HasFactory;
    protected $fillable = [
        'nomscategorie', 'imagescat', 'categorieID'
    ];

    public function categories()
    {
        return $this->belongsTo(Categorie::class, "categorieID");
    }
}

```

On a défini une relation pour permettre à une sous catégorie d'accéder à sa catégorie parente. Pour définir l'inverse d'une relation hasMany, définissez une méthode de relation sur le modèle enfant qui appelle la méthode belongsTo.

4. Modifier le controller de la sous catégorie

Dans Http\Controllers\SCategorieController.php

```

<?php

namespace App\Http\Controllers;

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

class ScategorieController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $scategories = Scategorie::with('categories')->get()->toArray();
        return array_reverse($scategories);
    }
}

```

```
/*
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $scategorie = new Scategorie([
        'nomscategorie' => $request->input('nomscategorie'),
        'imagescat' => $request->input('imagescat'),
        'categorieID' => $request->input('categorieID'),
    ]);
    $scategorie->save();

    return response()->json('S/Categorie créée !');
}

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $scategorie = Scategorie::find($id);
    return response()->json($scategorie);
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $scategorie = Scategorie::find($id);
    $scategorie->update($request->all());

    return response()->json('S/Catégorie MAJ !');
}

/**
 * Remove the specified resource from storage.
 *
 * @param \App\Models\Scategorie $scategorie
 * @return \Illuminate\Http\Response
 */
```

```

        */
    public function destroy($id)
    {
        $scategorie = Scategorie::find($id);
        $scategorie->delete();

        return response()->json('Scategorie supprimée !');
    }

/**
 * Show specified resource.
 *
 * @param \App\Models\Scategorie $idcat
 * @return \Illuminate\Http\Response
 */
public function showSCategorieByCAT($idcat)
{
    $Scategorie= Scategorie::where('categorieID', $idcat)->with('categories')->get()->toArray();
    return response()->json($Scategorie);
}

}

```

En plus des méthodes CRUD, nous ajoutons une méthode qui permet d'avoir la liste des sous catégories pour une catégorie donnée (showSCategorieByCAT).

5. Mettre en place les routes

Dans routes/api.php ajouter :

```

use App\Http\Controllers\ScategorieController;

Route::middleware('api')->group(function () {
    Route::resource('scategories', ScategorieController::class);

});

Route::get('/scat/{idcat}', [ScategorieController::class, 'showSCategorieByCAT']);

```

Une route supplémentaire a été ajoutée qui permettra d'appeler la méthode showSCategorieByCAT du controller de sous catégories.

6. Modification du model de catégorie

Dans la classe de Models/Categories.php, la relationhasMany doit être définie comme suit dans la classe :

```

public function scategories()
{
    return $this->hasMany(Scategorie::class , "categorieID");
}

```

PuisquehasMany() peut retrouver plus d'un enregistrement, il est d'usage de mettre le ou les mots définissant la relation au pluriel.

Si vous ne spécifiez pas de manière explicite le nom de la table dans un modèle, Laravel le déduit à partir du nom du modèle en le mettant au pluriel et en mettant la première lettre en minuscule. Donc avec le modèle Article il en conclut que la table s'appelle articles par exemple.

Les deux méthodes mises en place (belongsTo dans Scategories et hasMany dans Categories) permettent de récupérer facilement un enregistrement lié.

7. Exécuter la migration

Exécuter la commande :

```
php artisan migrate
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan migrate
INFO  Running migrations.
2023_02_24_225929_create_scategories_table .....
```

8. Exécuter et faire le test de l'api

Exécuter la commande :

```
php artisan serve
```

<http://localhost:8000/api/scategories>

POST

POST http://localhost:8000/api/scategories

Query	Headers ²	Auth	Body ¹	Tests	Pre Run	
Json	Xml	Text	Form	Form-encode	Graphql	Binary

```

1 {
2     "nomscategorie": "accessoires",
3     "imagescat": "images/scategories/accessoires.jpg",
4     "categorieID": "2"
5 }
```

Status: 200 OK Size: 32 Bytes Time: 314 ms

1 "S/Categorie créée !"

POST ▼ http://localhost:8000/api/scategories

Query Headers 2 Auth Body 1 Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

```

1  {
2    "nomscategorie": "led",
3    "imagescat": "images/scategories/led.jpg",
4    "categorieID": "3"
5  }

```

Status: 200 OK Size: 32 Bytes Time: 294 ms

1 "S/Categorie créée !"

	id	nomscategorie	imagescat	categorieID	created_at	updated_at
<input type="checkbox"/>	1	accessoires	images/scategories/accessoires.jpg	2	2023-02-25 12:11:59	2023-02-25 12:11:59
<input type="checkbox"/>	2	supports	images/scategories/supports.jpg	2	2023-02-25 12:21:13	2023-02-25 12:21:13
<input type="checkbox"/>	3	led	images/scategories/led.jpg	3	2023-02-25 12:23:48	2023-02-25 12:23:48

PUT

PUT ▼ http://localhost:8000/api/scategories/2 Send

Query Headers 2 Auth Body 1 Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

Json Content		Format
<pre> 1 { 2 "id": 2, 3 "nomscategorie": "supports", 4 "imagescat": "images/scategories/supp.jpg", 5 "categorieID": 2 6 } </pre>		

Status: 200 OK Size: 25 Bytes Time: 277 ms

Response Headers 9 Cookies Results Docs

1 "S/Catégorie MAJ !"

En essayant de supprimer la catégorie. Test de la contrainte d'intégrité.

```

Status: 500 Internal Server Error Size: 937.13 KB Time: 883 ms

Response Headers 9 Cookies Results Docs { } ⌂
1 <!DOCTYPE html>
2 <html lang="en" class="auto">
3 <!--
4 Illuminate\Database\QueryException: SQLSTATE[23000]:
   Integrity constraint violation: 1451 Cannot delete or
   update a parent row: a foreign key constraint fails
   ('crudapi`.`scategories', CONSTRAINT
   `scategories_ibfk_1` FOREIGN KEY
   (`categorieID`) REFERENCES `categories` (`id`) ON
   DELETE RESTRICT ON UPDATE RESTRICT) (Connection: mysql
   , SQL: delete from `categories` where `id` = 4) in
   file C:\Enseignement\Laravel\CRUDAPI\vendor\laravel\framework\src\Illuminate\Database\Connection.php on line
   760
5
6 #0 C:\Enseignement\Laravel\CRUDAPI\vendor\laravel\framework\src\Illuminate\Database\Connection.php(720):
   Illuminate\Database\Connection->runQueryCallback
   (&#039;delete from `ca...&#039;, Array, Object(Closure

```

Supprimer la sous catégorie faisant référence :

```

Status: 200 OK Size: 29 Bytes Time: 287 ms

Response Headers 9 Cookies Results Docs { } ⌂
1 "Scategorie supprimée !"

```

Maintenant c'est possible de supprimer la catégorie :

```

Status: 200 OK Size: 33 Bytes Time: 303 ms

Response Headers 9 Cookies Results Docs { } ⌂
1 "Catégorie supprimée !"

```

Afficher tous les enregistrements :

```

Status: 200 OK Size: 759 Bytes Time: 329 ms

Response Headers 9 Cookies Results Docs { } ⌂
1 [
2   {
3     "id": 2,
4     "nomscategorie": "supports",
5     "imagescat": "images/scategories/supp.jpg",
6     "categorieID": 2,
7     "created_at": "2023-02-25T12:21:13.000000Z",
8     "updated_at": "2023-02-25T12:52:48.000000Z",
9     "categories": {
10       "id": 2,
11       "nomscategorie": "smartphones",
12       "imagecategorie": "images/categories/smartphones.jpg"
13     },
14     "created_at": "2023-02-25T12:02:18.000000Z",
15     "updated_at": "2023-02-25T12:02:18.000000Z"
16   },
17   {
18     "id": 1,
19     "nomscategorie": "accessoires",
20     "imagescat": "images/scategories/accessoires.jpg",
21     "categorieID": 2,
22     "created_at": "2023-02-25T12:11:59.000000Z",
23     "updated_at": "2023-02-25T12:11:59.000000Z",
24     "categories": {
25       "id": 2,
26       "nomscategorie": "smartphones",
27       "imagecategorie": "images/categories/smartphones.jpg"

```

Test de la requête qui affiche les sous catégories d'une catégorie donnée.

<http://localhost:8000/api/scat/3>

GET http://localhost:8000/api/scat/3 Send

Status: 200 OK Size: 2 Bytes Time: 307 ms

Response Headers Cookies Results Docs

1 []

<http://localhost:8000/api/scat/2>

GET http://localhost:8000/api/scat/2 Send

Status: 200 OK Size: 759 Bytes Time: 267 ms

Response Headers Cookies Results Docs

1 [

2 {

3 "id": 1,

4 "nomscategorie": "accessoires",

5 "imagescat": "images/scategories/accessoires.jpg",

6 "categorieID": 2,

7 "created_at": "2023-02-25T12:11:59.000000Z",

8 "updated_at": "2023-02-25T12:11:59.000000Z",

9 "categories": {

10 "id": 2,

11 "nomscategorie": "smartphones",

12 "imagecategorie": "images/categories/smartphones.jpg"

13 "created_at": "2023-02-25T12:02:18.000000Z",

14 "updated_at": "2023-02-25T12:02:18.000000Z"

15 },

16 },

17 {

18 "id": 2,

19 "nomscategorie": "supports",

20 "imagescat": "images/scategories/supp.jpg",

21 "categorieID": 2,

V. Cas de l'article

1. Génération de l'article

On peut créer un modèle, un contrôleur de ressources (lié au modèle) ainsi que la migration dans une seule commande :

```
php artisan make:model Article --migration --controller --api
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan make:model Article --migration --controller --api
[INFO] Model [C:\Enseignement\Laravel\CRUDAPI\app\Models\Article.php] created successfully.
[INFO] Migration [C:\Enseignement\Laravel\CRUDAPI\database\migrations\2023_02_25_131335_create_articles_table.php] created successfully.
[INFO] Controller [C:\Enseignement\Laravel\CRUDAPI\app\Http\Controllers\ArticleController.php] created successfully.
```

2. Modifier la migration de l'article

Sous le dossier database/migrations

```
public function up()
{
    Schema::create('articles', function (Blueprint $table) {
        $table->id();
        $table->string('caracteristiques');
        $table->string('designation');
        $table->string('marque');
```

```

        $table->string('reference');
        $table->string('qtestock');
        $table->string('prixAchat');
        $table->string('prixVente');
        $table->string('prixSolde');
        $table->string('imageartpetitf');
        $table->string('imageartgrandf');
        $table->unsignedBigInteger('categorieID');
        $table->foreign('categorieID')
            ->references('id')
            ->on('categories')
            ->onDelete('restrict')
            ->onUpdate('restrict');
        $table->unsignedBigInteger('scategorieID');
        $table->foreign('scategorieID')
            ->references('id')
            ->on('scategories')
            ->onDelete('restrict')
            ->onUpdate('restrict');
        $table->timestamps();
    });
}

```

Deux références, l'une à la catégorie et l'autre à la sous catégorie.

3. Modifier le model de l'article

Dans Models/Article.php

```

<?php

namespace App\Models;

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

class Article extends Model
{
    use HasFactory;

    protected $fillable = [
        'caracteristiques', 'designation', 'marque', 'reference', 'qtestock', 'prixAchat', 'prixVente', 'prixSolde', 'imageartpetitf', 'imageartgrandf', 'categorieID', 'scategorieID'
    ];

    public function categories()
    {
        return $this->belongsTo(Categorie::class, "categorieID");
    }
}

```

```

    }

    public function scategories()
    {
        return $this->belongsTo(Scategorie::class,"scategorieID");
    }
}

```

4. Modifier le controller de l'article

Dans Http\Controllers\ArticleController.php

```

<?php

namespace App\Http\Controllers;

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

class ArticleController extends Controller
{
    public function index()
    {
        $articles = Article::with('categories','scategories')->get()->toArray();
        return array_reverse($articles);
    }
    /**
     * Store a newly created resource in storage.
     *
     * @param \Illuminate\Http\Request $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $article = new Article([
            'caracteristiques' => $request->input('caracteristiques'),
            'designation' => $request->input('designation'),
            'marque' => $request->input('marque'),
            'reference' => $request->input('reference'),
            'qtestock' => $request->input('qtestock'),
            'prixAchat' => $request->input('prixAchat'),
            'prixVente' => $request->input('prixVente'),
            'prixSolde' => $request->input('prixSolde'),
            'imageartpetitf' => $request->input('imageartpetitf'),
            'imageartgrandf' => $request->input('imageartgrandf'),
            'categorieID' => $request->input('categorieID'),
            'scategorieID' => $request->input('scategorieID')
        ]);
    }
}

```

```
]);
$article->save();

return response()->json('Article créé !');
}

/**
 * Display the specified resource.
 *
 * @param \App\Models\Article $article
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $article= Article::find($id);
    return response()->json($article);
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param \App\Models\Article $article
 * @return \Illuminate\Http\Response
 */
public function update($id, Request $request)
{
    $article = Article::find($id);
    $article->update($request->all());

    return response()->json('Article MAJ !');

}

/**
 * Remove the specified resource from storage.
 *
 * @param \App\Models\Article $article
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    $article = Article::find($id);
    $article->delete();

    return response()->json('Article supprimé !');

}
}
```

5. Mettre en place les routes

Dans routes/api.php ajouter :

```
use App\Http\Controllers\ArticleController;

Route::middleware('api')->group(function () {
    Route::resource('articles', ArticleController::class);
});
```

6. Modification du model de catégorie

Dans la classe de Models/Categories.php, la relationhasMany doit être définies comme suit dans la classe :

```
public function article()
{
    return $this->hasMany(Article::class, "categorieID");
}
```

7. Modification du model de la sous catégorie

Dans la classe de Models/SubCategories.php, la relationhasMany doit être définies comme suit dans la classe :

```
public function article()
{
    return $this->hasMany(Article::class, "scategorieID");
}
```

8. Exécuter la migration

Exécuter la commande :

```
php artisan migrate
```

```
PS C:\Enseignement\Laravel\CRUDAPI> php artisan migrate
[INFO] Running migrations.
2023_02_25_131335_create_articles_table .....
```

9. Exécuter et faire le test de l'api

Exécuter la commande :

```
php artisan serve
```

<http://localhost:8000/api/articles>

POST

POST <http://localhost:8000/api/articles> Send

Status: 200 OK Size: 26 Bytes Time: 290 ms

Response Headers Cookies Results Docs

Query Headers² Auth Body¹ Tests Pre Run

Json XML Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
  "caracteristiques": "écouteur et micro",
  "designation": "écouteur",
  "marque": "X5",
  "reference": "E52364",
  "qtestock": "18",
  "prixAchat": "3.5",
  "prixVente": "5.6",
  "prixSolde": "4.1",
  "imageartpetitf": "images/articles/E52364p.jpg",
  "imageartgrandf": "images/articles/E52364g.jpg",
  "categorieID": 2,
  "scategorieID": 1
}
```

PUT

PUT <http://localhost:8000/api/articles/2> Send

Status: 200 OK Size: 15 Bytes Time: 276 ms

Response Headers⁹ Cookies Results Docs

Query Headers² Auth Body¹ Tests Pre Run

Json XML Text Form Form-encode GraphQL Binary

Json Content Format

```
1 {
  "caracteristiques": "écouteur et micro",
  "designation": "écouteur",
  "marque": "X5",
  "reference": "E52364",
  "qtestock": "11",
  "prixAchat": "3.5",
  "prixVente": "5.6",
  "prixSolde": "4.3",
  "imageartpetitf": "images/articles/E52364p.jpg",
  "imageartgrandf": "images/articles/E52364g.jpg",
  "categorieID": 2,
  "scategorieID": 1
}
```

DELETE

DELETE <http://localhost:8000/api/articles/1> Send

Status: 200 OK Size: 25 Bytes Time: 314 ms

Response Headers⁹ Cookies Results Docs

Query Headers² Auth Body¹ Tests Pre Run

Query Parameters

parameter	value
<input type="checkbox"/>	

Test de la contrainte d'intégrité de la catégorie :

POST <http://localhost:8000/api/articles> Send

Query Headers [2](#) Auth Body [1](#) Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

Json Content Format

```

1  {
2      "caracteristiques": "écouteur et micro",
3      "designation": "écouteur",
4      "marque": "X5",
5      "reference": "E52364",
6      "qtestock": "18",
7      "prixAchat": "3.5",
8      "prixVente": "5.6",
9      "prixSolde": "4.1",
10     "imageartpetitf": "images/articles/E52364p.jpg",
11     "imageartgrandf": "images/articles/E52364g.jpg",
12     "categorieID": 54,
13     "scategorieID": 1
14 }
```

Status: **500 Internal Server Error** Size: **949.87 KB** Time: **619 ms**

Response Headers [9](#) Cookies Results Docs { } ⚡

```

1  <!DOCTYPE html>
2  <html lang="en" class="auto">
3  <!--
4  Illuminate\Database\QueryException: SQLSTATE[23000]:
    Integrity constraint violation: 1452 Cannot add or
    update a child row: a foreign key constraint fails
    ('crudapi`.`articles', CONSTRAINT
    `articles_categorieid_foreign` FOREIGN KEY
    (`categorieID`) REFERENCES `categories` (`id`) ON
    DELETE RESTRICT ON UPDATE RESTRICT) (Connection: mysql
    , SQL: insert into `articles` ('caracteristiques',
    'designation', 'marque', 'reference', 'qtestock',
    'prixAchat', 'prixVente', 'prixSolde',
    'imageartpetitf', 'imageartgrandf', 'categorieID',
    'scategorieID', 'updated_at', 'created_at') values
    ('écouteur et micro', 'écouteur', 'X5', 'E52364', 18, 3.5, 5.6
    , 4.1, 'images/articles/E52364p.jpg', 'images/articles
    /E52364g.jpg', 54, 1, 2023-02-25 15:56:33, 2023-02-25
    15:56:33)) in file C
    :\Enseignement\Laravel\CRUDAPI\vendor\laravel\framework
    \src\Illuminate\Database\Connection.php on line 760
```

Test de la contrainte d'intégrité de la sous catégorie :

POST <http://localhost:8000/api/articles> Send

Query Headers [2](#) Auth Body [1](#) Tests Pre Run

Json Xml Text Form Form-encode Graphql Binary

Json Content Format

```

1  {
2      "caracteristiques": "écouteur et micro",
3      "designation": "écouteur",
4      "marque": "X5",
5      "reference": "E52364",
6      "qtestock": "18",
7      "prixAchat": "3.5",
8      "prixVente": "5.6",
9      "prixSolde": "4.1",
10     "imageartpetitf": "images/articles/E52364p.jpg",
11     "imageartgrandf": "images/articles/E52364g.jpg",
12     "categorieID": 2,
13     "scategorieID": 33
14 }
```

Status: **500 Internal Server Error** Size: **949.88 KB** Time: **619 ms**

Response Headers [9](#) Cookies Results Docs { } ⚡

```

1  <!DOCTYPE html>
2  <html lang="en" class="auto">
3  <!--
4  Illuminate\Database\QueryException: SQLSTATE[23000]:
    Integrity constraint violation: 1452 Cannot add or
    update a child row: a foreign key constraint fails
    ('crudapi`.`articles', CONSTRAINT
    `articles_scategorieid_foreign` FOREIGN KEY
    (`scategorieID`) REFERENCES `scategories` (`id`) ON
    DELETE RESTRICT ON UPDATE RESTRICT) (Connection: mysql
    , SQL: insert into `articles` ('caracteristiques',
    'designation', 'marque', 'reference', 'qtestock',
    'prixAchat', 'prixVente', 'prixSolde',
    'imageartpetitf', 'imageartgrandf', 'categorieID',
    'scategorieID', 'updated_at', 'created_at') values
    ('écouteur et micro', 'écouteur', 'X5', 'E52364', 18, 3.5, 5.6
    , 4.1, 'images/articles/E52364p.jpg', 'images/articles
    /E52364g.jpg', 2, 33, 2023-02-25 15:57:25, 2023-02-25
    15:57:25)) in file C
    :\Enseignement\Laravel\CRUDAPI\vendor\laravel\framework
    \src\Illuminate\Database\Connection.php on line 760
```

Liste de tous les articles.

GET <http://localhost:8000/api/articles> Send

Query Headers [2](#) Auth Body [1](#) Tests Pre Run

Query Parameters

<input type="checkbox"/>	parameter	value
--------------------------	-----------	-------

Status: **200 OK** Size: **804 Bytes** Time: **239 ms**

Response Headers [9](#) Cookies Results Docs { } ⚡

```

1  [
2  {
3      "id": 2,
4      "caracteristiques": "écouteur et micro",
5      "designation": "écouteur",
6      "marque": "X5",
7      "reference": "E52364",
8      "qtestock": "11",
9      "prixAchat": "3.5",
10     "prixVente": "5.6",
11     "prixSolde": "4.3",
12     "imageartpetitf": "images/articles/E52364p.jpg",
13     "imageartgrandf": "images/articles/E52364g.jpg",
14     "categorieID": 2,
15     "scategorieID": 1,
16     "created_at": "2023-02-25T15:48:40.000000Z",
17     "updated_at": "2023-02-25T15:49:57.000000Z",
18     "categories": {
19         "id": 2,
```