

Laravel



# Présentation du cours

## Déroulement des séances :

- 18 Heures de cours séparées en 2,5 jours
  - Vendredi 26/01/2024 (8h30 à 11h30 et 13h00 à 16h00)
  - Mardi 6/02/2024 (14h30 à 17h30)
  - Vendredi 7/02/2024 (8h30 à 11h30 et 13h00 à 17h30)
- Création d'une API Rest avec Laravel

## Notes :

- Un projet (coef 1)
- Une interrogation à la dernière séance (coef 2)



# Aujourd'hui

## Matin:

- Présentation globale de Laravel
- Base de données avec Laravel
  - Migrations
  - Models
  - Eloquent (ORM)
  - Factories
  - Seeders

## Après-midi :

- Routing
- Controller

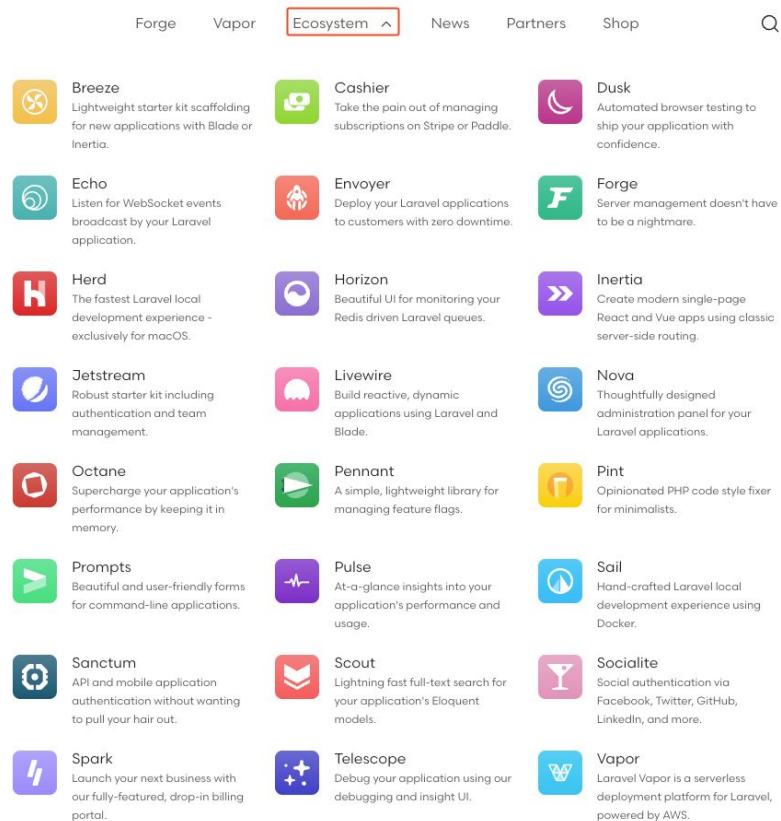
# Laravel c'est quoi ?

- Framework PHP open source
- Développé en programmation orientée objet
- Architecture MVC (Modèle Vue Contrôleur)
- Première release : Juin 2011
- Sources hébergées sur GitHub (75,8k stars)
- Distribué sous licence MIT
- Créateur Taylor Otwell



# Pourquoi choisir Laravel en 2024 ?

- Maturité et stabilité (création en 2011)
- Communauté et ressources
- Écosystème riche
- Rapidité de développement



The screenshot displays the 'Ecosystem' section of the Laravel website, which is highlighted with a red border in the original image. The navigation bar at the top includes links for 'Forge', 'Vapor', 'Ecosystem' (selected), 'News', 'Partners', and 'Shop', along with a search icon. The main content area is a grid of 24 items, each featuring a colored icon, a title, and a brief description:

- Breeze**: Lightweight starter kit scaffolding for new applications with Blade or Inertia.
- Cashier**: Take the pain out of managing subscriptions on Stripe or Paddle.
- Dusk**: Automated browser testing to ship your application with confidence.
- Echo**: Listen for WebSocket events broadcast by your Laravel application.
- Envoyer**: Deploy your Laravel applications to customers with zero downtime.
- Forge**: Server management doesn't have to be a nightmare.
- Herd**: The fastest Laravel local development experience - exclusively for macOS.
- Horizon**: Beautiful UI for monitoring your Redis driven Laravel queues.
- Inertia**: Create modern single-page React and Vue apps using classic server-side routing.
- Jetstream**: Robust starter kit including authentication and team management.
- Livewire**: Build reactive, dynamic applications using Laravel and Blade.
- Nova**: Thoughtfully designed administration panel for your Laravel applications.
- Octane**: Supercharge your application's performance by keeping it in memory.
- Pennant**: A simple, lightweight library for managing feature flags.
- Pint**: Opinionated PHP code style fixer for minimalists.
- Prompts**: Beautiful and user-friendly forms for command-line applications.
- Pulse**: At-a-glance insights into your application's performance and usage.
- Sail**: Hand-crafted Laravel local development experience using Docker.
- Sanctum**: API and mobile application authentication without wanting to pull your hair out.
- Scout**: Lightning fast full-text search for your application's Eloquent models.
- Socialite**: Social authentication via Facebook, Twitter, GitHub, LinkedIn, and more.
- Spark**: Launch your next business with our fully-featured, drop-in billing portal.
- Telescope**: Debug your application using our debugging and insight UI.
- Vapor**: Laravel Vapor is a serverless deployment platform for Laravel, powered by AWS.



## Config nécessaire pour installer Laravel

- Une version de PHP entre 8.1 et 8.3 (pour Laravel 10)
- Composer (<https://getcomposer.org/>)

Version	PHP (*)	Release	Bug Fixes Until	Security Fixes Until
8	7.3 - 8.1	September 8th, 2020	July 26th, 2022	January 24th, 2023
9	8.0 - 8.2	February 8th, 2022	August 8th, 2023	February 6th, 2024
10	8.1 - 8.3	February 14th, 2023	August 6th, 2024	February 4th, 2025
11	8.2 - 8.3	Q1 2024	August 5th, 2025	February 3rd, 2026

Lien de la doc : <https://laravel.com/docs/10.x/releases#support-policy>



# Créer un nouveau projet Laravel

- 1) Avec la commande “create-projet” de Composer :

```
composer create-project laravel/laravel nom-du-projet
```

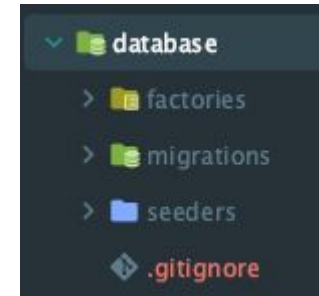
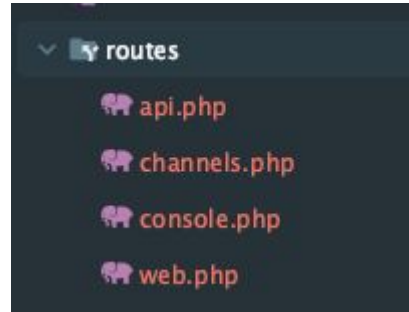
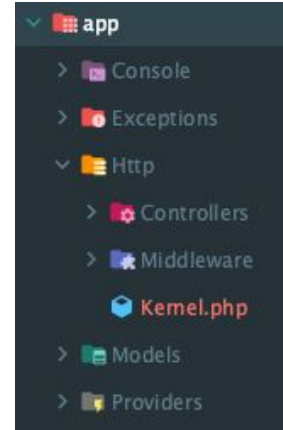
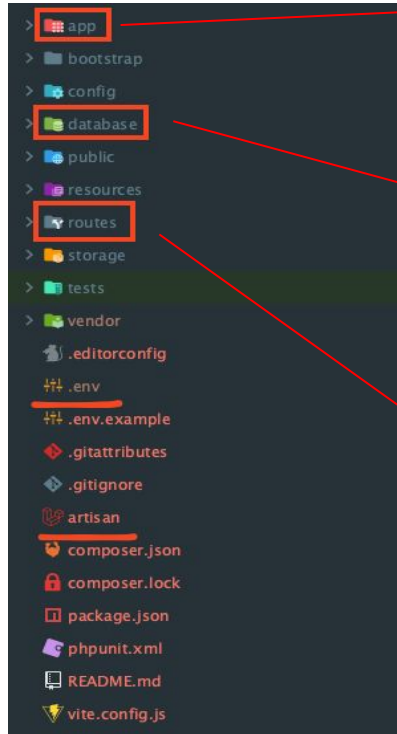
- 2) En installant le programme d'installation de Laravel via Composer :

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

```
laravel new nom-du-projet
```

Lien de la doc : <https://laravel.com/docs/10.x/installation#creating-a-laravel-project>

# Structure d'un projet Laravel





# Artisan



- Artisan est un CLI (Command Line Interface) intégré à Laravel.
- Il offre un certain nombre de commandes qui vous permettra de gagner du temps lors du développement de votre application.
- Pour consulter la liste complète des commandes, vous pouvez utiliser la commande 'list'.

```
php artisan list
Laravel Framework 10.41.0

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no command is g
  -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
      --env[=ENV]        The environment the command should run under
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more details, 3 for all
```

```
Available commands:
  about            Display basic information about your application
  clear-compiled    Remove the compiled class file
  completion        Dump the shell completion script
  db               Start a new database CLI session
  docs             Access the Laravel documentation
  down             Put the application into maintenance / demo mode
  env              Display the current framework environment
  help             Display help for a command
  inspire          Display an inspiring quote
```

Lien de la doc : <https://laravel.com/docs/10.x/artisan>



## Lancer son projet

Pour lancer son projet vous pouvez utiliser la commande “php artisan serve”. Cette commande démarre un serveur de développement, sur le port 8000 (si il est libre sinon il incrément de 1 jusqu’à trouver un port libre ).

```
php artisan serve
```

```
INFO Server running on [http://127.0.0.1:8000].
```

```
Press Ctrl+C to stop the server
```

Lien de la doc : <https://laravel.com/docs/10.x/installation#creating-a-laravel-project>



# Base de données



# Base de données

- 1) .env
- 2) Migration
- 3) Models
- 4) ORM/ Eloquent
- 5) Factories
- 6) Seeder

Lien de la doc : <https://laravel.com/docs/10.x/migrations>



# Fichier d'environnement

- Ne doit jamais être versionné (de base dans le .gitignore)
- Toujours avoir un “.env.example” avec les mêmes clés que le .env
- Lors de la récupération d'un projet -> `cp .env.example .env`

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

Lien de la doc :

<https://laravel.com/docs/10.x/configuration#environment-variable-types>

<https://laravel.com/docs/10.x/database#configuration>



# Migration

Les migrations dans Laravel permettent de réaliser des changements sur la structure de la base de données de notre application.

Pour réaliser une migration :

- Création du fichier de migration
- Définition de la structure
- Exécution de la migration

Lien de la doc : <https://laravel.com/docs/10.x/migrations>



# 1. Création du fichier de migration

Pour créer une migration :

```
~  
php artisan make:migration create_users_table
```

Cette commande va créer un fichier.

[votre\_projet]/database/migrations/2024\_01\_23\_073242\_create\_users\_table.php

Lien de la doc : <https://laravel.com/docs/10.x/migrations#generating-migrations>

## 2. Définition de la structure

### # Available Column Types

The schema builder blueprint offers a variety of methods that correspond to the different types of columns you can add to your database tables. Each of the available methods are listed in the table below:

<a href="#">bigIncrements</a>	<a href="#">jsonb</a>	<a href="#">string</a>
<a href="#">bigInteger</a>	<a href="#">lineString</a>	<a href="#">text</a>
<a href="#">binary</a>	<a href="#">longText</a>	<a href="#">timeTz</a>
<a href="#">boolean</a>	<a href="#">macAddress</a>	<a href="#">time</a>
<a href="#">char</a>	<a href="#">mediumIncrements</a>	<a href="#">timestampTz</a>
<a href="#">dateTimeTz</a>	<a href="#">mediumInteger</a>	<a href="#">timestamp</a>
<a href="#">dateTime</a>	<a href="#">mediumText</a>	<a href="#">timestampsTz</a>
<a href="#">date</a>	<a href="#">morphs</a>	<a href="#">timestamps</a>
<a href="#">decimal</a>	<a href="#">multiLineString</a>	<a href="#">tinyIncrements</a>
<a href="#">double</a>	<a href="#">multiPoint</a>	<a href="#">tinyInteger</a>
<a href="#">enum</a>	<a href="#">multiPolygon</a>	<a href="#">tinyText</a>
<a href="#">float</a>	<a href="#">nullableMorphs</a>	<a href="#">unsignedBigInteger</a>
<a href="#">foreignId</a>	<a href="#">nullableTimestamps</a>	<a href="#">unsignedDecimal</a>
<a href="#">foreignIdFor</a>	<a href="#">nullableUuidMorphs</a>	<a href="#">unsignedInteger</a>
<a href="#">foreignUuid</a>	<a href="#">nullableUuidMorphs</a>	<a href="#">unsignedMediumInteger</a>
<a href="#">foreignUuid</a>	<a href="#">point</a>	<a href="#">unsignedSmallInteger</a>
<a href="#">geometryCollection</a>	<a href="#">polygon</a>	<a href="#">unsignedTinyInteger</a>
<a href="#">geometry</a>	<a href="#">rememberToken</a>	<a href="#">uuidMorphs</a>
<a href="#">id</a>	<a href="#">set</a>	<a href="#">uuidMorphs</a>
<a href="#">increments</a>	<a href="#">smallIncrements</a>	<a href="#">uuid</a>
<a href="#">integer</a>	<a href="#">smallInteger</a>	<a href="#">uuid</a>
<a href="#">ipAddress</a>	<a href="#">softDeletesTz</a>	<a href="#">year</a>
<a href="#">json</a>	<a href="#">softDeletes</a>	

```
<?php
```

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

```
return new class extends Migration
{
```

```
    /**
```

```
     * Run the migrations.
```

```
    */
```

```
    public function up(): void
```

```
    {
```

```
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }
```

```
    /**
```

```
     * Reverse the migrations.
```

```
    */
```

```
    public function down(): void
```

```
    {
```

```
        Schema::drop('flights');
```

```
    }
```

```
};
```

Lien de la doc : <https://laravel.com/docs/10.x/migrations#available-column-types>





## 2. Exécution de la migration

Pour exécuter une migration il suffit d'appeler la commande :

```
php artisan migrate
```

```
INFO Preparing database.
```

```
Creating migration table ..... 16ms DONE
```

```
INFO Running migrations.
```

```
2014_10_12_000000_create_users_table ..... 19ms DONE  
2014_10_12_100000_create_password_reset_tokens_table ..... 5ms DONE  
2019_08_19_000000_create_failed_jobs_table ..... 6ms DONE  
2019_12_14_000001_create_personal_access_tokens_table ..... 13ms DONE
```


Lien de la doc : <https://laravel.com/docs/10.x/migrations#running-migrations>



# Models

Un modèle/model dans Laravel est une classe qui représente une table dans votre base de données.

Chaque instance d'un modèle correspond à une ligne spécifique dans cette table



```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Message extends Model
{
}
```



```
$messages = Message::query()
    ->where('title','=','first message')
    ->get();

foreach ($messages as $message) {
    echo $message->title; //first message
}
```



# Models

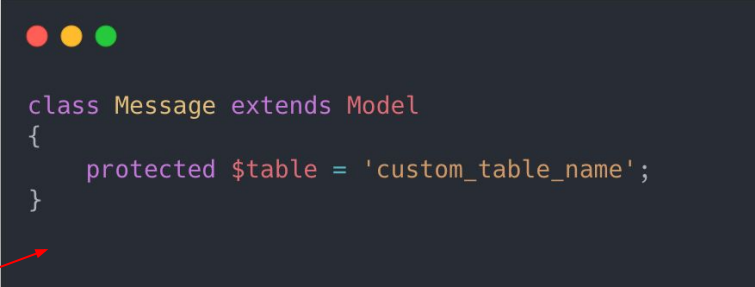
Comment fait-il pour récupérer le nom de la table ?

Réponse :


- Regarde s'il a un attribut `$table` dans le model
- Utilise le nom de la classe et la met au pluriel puis en snake case (voir méthode `getTable`)

Exemple :

- User -> table "users"
- Agency -> table "agencies"



```
class Message extends Model
{
    protected $table = 'custom_table_name';
}
```



```
Get the table associated with the model.
Returns: string

public function getTable()
{
    return $this->table ?? Str::snake(Str::pluralStudly(class_basename($this)));
}
```





# Models

Comment fait-il pour avoir des attributs sur mon model ?

Réponse :

- Il utilise les méthodes magique de PHP ( `__get` et `__set` )

```
class Message extends Model
{
    ?
}
```

```
$messages = Message::query()
    ->where('title','=','first message')
    ->get();

foreach ($messages as $message) {
    echo $message->title; //first message
}
```



# Models - exemple “method magic”

```
class Model
{
    private array $data;

    /**
     * @param array $data
     */
    public function __construct(array $data)
    {
        $this->data = $data;
    }

    public function __get(string $name)
    {
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        }
        return null;
    }

    public function __set(string $name, $value): void
    {
        $this->data[$name] = $value;
    }
}
```



```
$model = new Model([
    'firstname' => 'John',
    'lastname' => 'Doe',
]);

var_dump($model->firstname); // John
var_dump($model->hello); // null
$model->hello = 'Hello world';
var_dump($model->hello); // Hello world
```



# Eloquent - ORM

Eloquent est l'ORM (Object-Relational Mapping) inclus par défaut avec Laravel.

Un ORM est un moyen de mapper les tables de bases de données aux classes de votre application, permettant une interaction plus intuitive avec la base de données en utilisant des concepts orientés objet.

- ◉ MariaDB 10.10+ ([Version Policy](#))
- ◉ MySQL 5.7+ ([Version Policy](#))
- ◉ PostgreSQL 11.0+ ([Version Policy](#))
- ◉ SQLite 3.8.8+
- ◉ SQL Server 2017+ ([Version Policy](#))

# Eloquent - Récupération de la donnée

```
/** @var Collection<Message> $messages */
$messages = Message::all();

$messages = Message::query()->find( id: 2);

/** @var Collection<Message> $messages */
$messages = Message::query()
    ->where( column: 'title', operator: '=', value: 'hello world')
    ->get();

/** @var ?Message $messages */
$message = Message::query()
    ->where( column: 'title', operator: '=', value: 'hello world')
    ->first();

/** @var Collection<Message> $messages */
$messages = Message::query()
    ->where( column: 'title', operator: '=', value: 'hello world')
    ->orderBy( column: 'created_at', direction: 'desc')
    ->get();
```

# Eloquent - query() ou pas query() 🤔

```
$flights = Flight::where('active', 1)
    ->orderBy('name')
    ->take(10)
    ->get();
```

Source: laravel.com

- query() renvoie un QueryBuilder, ce qui améliore la compréhension de l'IDE
- Utiliser directement Model::where passe par les méthode magic de PHP

```
$messages = Message::query()
    ->where( column: 'title', operator: '=', value: 'hello world')
    ->get();
```

Source: Arthur

Handle dynamic static method calls into the model.

Parameters: string \$method  
array \$parameters  
Returns: mixed

```
public static function __callStatic($method, $parameters)
{
    return (new static)->$method(...$parameters);
}
```

```
public function __call($method, $parameters)
{
    if (in_array($method, ['increment', 'decrement', 'incrementQuietly', 'decrementQuietly'])) {
        return $this->$method(...$parameters);
    }

    if ($resolver = $this->relationResolver(class: static::class, $method)) {
        return $resolver($this);
    }

    if (Str::startsWith($method, 'through') &&
        method_exists($this, $relationMethod = Str::of($method)->after('search: through')->toString())) {
        return $this->through($relationMethod);
    }

    return $this->forwardCallTo($this->newQuery(), $method, $parameters);
}
```

```
protected function forwardCallTo($object, $method, $parameters)
{
    try {
        return $object->{$method}(...$parameters);
    } catch (Error|BadMethodCallException $e) {
        $pattern = '~^Call to undefined method (?P<class>[^\:]+\:)(?P<method>[^\(\)]+\(\)\$~';

        if (! preg_match($pattern, $e->getMessage(), &$matches)) {
            throw $e;
        }

        if ($matches['class'] != get_class($object) ||
            $matches['method'] != $method) {
            throw $e;
        }

        static::throwBadMethodCallException($method);
    }
}
```





## Eloquent - Création

```
$message = new Message();  
$message->title = 'hello world';  
$message->text = 'lorem ipsum ...';  
$message->save();
```

```
Message::create( [  
    'title' => 'hello world',  
    'text' => 'lorem ipsum ...',  
]);
```

# Eloquent - Fillable / Guarded



Illuminate\Database\Eloquent\MassAssignmentException

**Add [title] to fillable property to allow mass assignment on [App\Models\Post].**

- Fillable : j'autorise seulement ceux dans le tableau
- Guarded : j'autorise tous les champs sauf ceux dans le tableau

```
/**
 * @property string $title
 * @property string $text
 */
26 usages
class Message extends Model
{
    no usages
    protected $fillable = [
        'title',
    ];

    // ou

    no usages
    protected $guarded = [
        'text'
    ];
}
```

Lien de la doc :



## Eloquent - Modification

```
/** @var Message $message */  
$message = Message::query()->find( id: 1);  
$message->title = 'hello world';  
$message->update();
```

```
Message::query()  
    ->find( id: 1)  
    ->update([  
        'title' => 'hello world',  
        'text' => 'lorem ipsum ...',  
    ]);
```



# Eloquent - Tinker

```
composer require laravel/tinker
```

```
php artisan tinker
```

```
Psy Shell v0.12.0 (PHP 8.3.0 - cli) by Justin Hileman
```

```
> App\Models\Message::query()->first();
```

```
= App\Models\Message {#5717
```

```
  id: 1,
```

```
  title: "Non sit velit facilis molestias quibusdam in sunt.",
```

```
  text: "Laudantium ea tempore molestiae maiores nihil rerum. Voluptatem
```

```
  created_at: "2024-01-24 10:53:35",
```

```
  updated_at: "2024-01-24 10:53:35",
```

```
}
```

```
> █
```

Lien de la doc :



## Eloquent - Suppression

```
$message = Message::query()->find( id: 1 );  
$message->delete();
```



# Factories

Une "factory" est une classe qui permet de créer des instances de modèles Eloquent avec des données de test.

Ces données sont généralement générées de manière aléatoire et servent à remplir la base de données avec des données fictives.

```
php artisan make:factory MessageFactory
```

```
INFO Factory [database/factories/MessageFactory.php] created successfully.
```

# Factories

Name	Type
id	bigint unsigned
title	varchar(255)
text	varchar(255)
created_at	timestamp
updated_at	timestamp

```
public function definition(): array
{
    return [
        'title' => $this->faker->sentence,
        'text' => $this->faker->paragraph,
    ];
}
```

```
<?php

namespace Database\Factories;

use App\Models\Message;
use Illuminate\Database\Eloquent\Factories\Factory;

/**
 * @extends Factory<Message>
 */
no usages
class MessageFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    no usages
    public function definition(): array
    {
        return [
            //
        ];
    }
}
```

# Factories

Attention : ne pas oublier d'ajouter le trait `HasFactory` sur son model.

```
<?php

namespace App\Models;

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

1 usage
class Message extends Model
{
    use HasFactory;
}
```

```
<?php

namespace Illuminate\Database\Eloquent\Factories;

trait HasFactory
{
    Get a new factory instance for the model.
    Parameters: array|callable|int|null $count
    array|callable $state
    Returns: \Illuminate\Database\Eloquent\Factories\Factory<static>

    public static function factory($count = null, $state = [])
    {
        $factory = static::newFactory() ?: Factory::factoryForModel(get_called_class());

        return $factory
            ->count( count: is_numeric($count) ? $count : null)
            ->state( state: is_callable($count) || is_array($count) ? $count : $state);
    }

    Create a new factory instance for the model.
    Returns: \Illuminate\Database\Eloquent\Factories\Factory<static>

    protected static function newFactory()
    {
        //
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-factories>





## Executer une Factory

```
$message = Message::factory()->make();  
  
$messages = Message::factory()  
    ->count( count: 10)  
    ->make();
```



# Seeder

Un seeder est une classe qui permet d'insérer des données dans la base de données.  
Ces données peuvent être statiques (prédéfinies) ou dynamiques (générées via des factories par exemple).

```
php artisan make:seeder MessageSeeder
```

```
INFO Seeder [database/seeder/MessageSeeder.php] created successfully.
```

Lien de la doc : <https://laravel.com/docs/10.x/seeding>



# Ecrire le Seeder

Ne pas oublier de changer le “make”  
par “create” pour faire l’insertion en  
base

Lien de la doc : <https://laravel.com/docs/10.x/seeding>

```
<?php

namespace Database\Seeders;

use App\Models\Message;
use Illuminate\Database\Seeder;

2 usages
class MessageSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        Message::factory()
            ->count( count: 10)
            ->create();
    }
}
```

## Executer un / des Seeder(s)

- Utilisez `php artisan db:seed --class=NomDuSeeder` pour exécuter un seeder spécifique.
- Utilisez `php artisan db:seed` pour exécuter tous les seeders.
- Vous pouvez aussi exécuter un Seeder depuis un autre Seeder

```
php artisan db:seed  
  
INFO Seeding database.
```

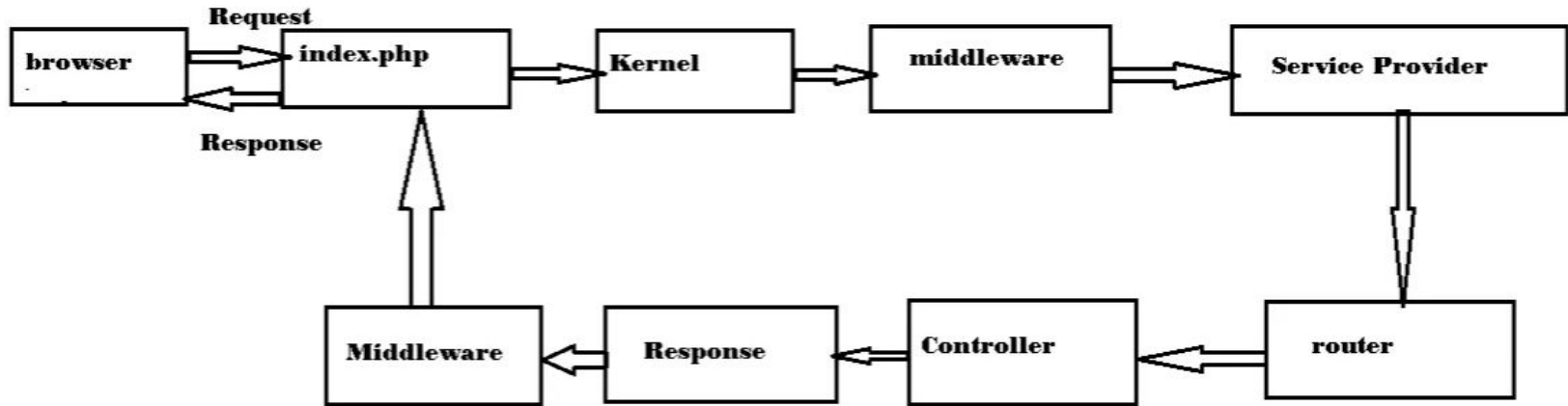
```
php artisan db:seed --class=MessageSeeder  
  
INFO Seeding database.
```

```
class DatabaseSeeder extends Seeder  
{  
    /**  
     * Seed the application's database.  
     */  
    public function run(): void  
    {  
        $this->call( class: MessageSeeder::class);  
        // ou  
        $this->call([  
            MessageSeeder::class,  
            UserSeeder::class,  
        ]);  
    }  
}
```



# Routing / Controller

## Cycle de vie d'une requête (version simple)





# Routing

- C'est le processus de mappage des URLs aux contrôleurs et fonctions spécifiques dans une application.
- Les routes dans Laravel sont définies dans les fichiers situés dans le dossier “/routes”.

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Lien de la doc :



## Routing - méthode HTTP

HTTP définit un ensemble de méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée. (source [developer.mozilla.org](https://developer.mozilla.org))

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

source : <https://developer.mozilla.org/fr/docs/Web/HTTP/Methods>



# Routing - Paramètre dans l'url

- Utiliser la class Request pour accéder aux données
- Ne pas utiliser les superglobals \$\_GET et \$\_POST ...

```
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::get( uri: '/my-name', function (Request $request) {

    $name = $request->get( key: 'name');

    return response()->json([
        'name' => $name
    ]);
});
```

/my-name?name="john"



Lien de la doc :



## Routing - Paramètre dynamique dans l'url

/my-name/john



```
Route::get( uri: '/my-name/{name}', function (Request $request,string $name) {  
    return response()->json([  
        'name' => $name  
    ]);  
});
```



# Controller

Les Controllers permettent d'éviter de concentrer toute la logique de traitement des requêtes dans des fonctions anonymes (dans vos fichiers de route) .

Les controllers servent à rassembler les traitements de requêtes qui sont reliés entre eux au sein d'une même classe.

Par exemple, une classe UserController serait responsable de la gestion de toutes les requêtes concernant les utilisateurs, incluant leurs affichages, créations, mises à jour et suppressions.

Par défaut, ces classes de controllers se trouvent dans le dossier `app/Http/Controllers`."



# Controller

Pour créer un Controller :

```
php artisan make:controller MessageController
```

```
INFO Controller [app/Http/Controllers/MessageController.php] created successfully.
```

```
php artisan make:controller MessageController --api
```

```
INFO Controller [app/Http/Controllers/MessageController.php] created successfully.
```

Lien de la doc :



# Controller

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

no usages
class MessageController extends Controller
{
    //
}
```

```
class MessageController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    no usages
    public function index()
    {}


    /**
     * Store a newly created resource in storage.
     */
    no usages
    public function store(Request $request)
    {}

    /**
     * Display the specified resource.
     */
    no usages
    public function show(string $id)
    {}

    /**
     * Update the specified resource in storage.
     */
    no usages
    public function update(Request $request, string $id)
    {}

    /**
     * Remove the specified resource from storage.
     */
    no usages
    public function destroy(string $id)
    {}
}
```

Avec -- api



Lien de la doc :

# Controller

Pour relier une route à un Controller vous pouvez utiliser cette syntaxe dans vos routes :

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

Ne pas oublier d'inclure le namespace du Controller

Nom du Controller

Nom de la méthode du Controller

Lien de la doc :

# Controller / Route - Convention de nommage



Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Lien de la doc :



# API



# API



Une API (application programming interface) est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

Réutilisabilité : Les API permettent la réutilisation de services et de fonctionnalités, réduisant ainsi le temps et le coût de développement des logiciels.

Interopérabilité : Elles facilitent l'interopérabilité entre différents systèmes et logiciels, même ceux écrits dans différents langages de programmation.

Sécurité : En agissant comme un point de contrôle, les API peuvent gérer de manière sécurisée l'accès aux données ou aux fonctionnalités d'une application, en s'assurant que seules les demandes autorisées sont traitées.



# API Resource

# Controller

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

no usages
class MessageController extends Controller
{
    //
}
```

```
class MessageController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    no usages
    public function index()
    {}

    /**
     * Store a newly created resource in storage.
     */
    no usages
    public function store(Request $request)
    {}

    /**
     * Display the specified resource.
     */
    no usages
    public function show(string $id)
    {}

    /**
     * Update the specified resource in storage.
     */
    no usages
    public function update(Request $request, string $id)
    {}

    /**
     * Remove the specified resource from storage.
     */
    no usages
    public function destroy(string $id)
    {}
}
```

Avec -- api

Lien de la doc :

# Resource



Une Resource dans Laravel est une classe qui encapsule la transformation de vos modèles Eloquent (ou toute autre donnée) en un format JSON structuré.

Elle sert à personnaliser la sortie JSON de vos données, en vous permettant de choisir quelles informations inclure, comment les formater et même d'y ajouter des données calculées sans modifier la structure de vos bases de données.

## Avantages :

- Permet de contrôler précisément ce qui est inclus dans la réponse JSON, utile si vous ne voulez pas exposer toutes les données de votre modèle.
- Centralise la logique de présentation de vos données, rendant votre code plus propre et plus facile à maintenir.

Lien de la doc :

# Resource - Création



Pour créer une Resource, vous pouvez utiliser la commande Artisan suivante :

```
php artisan make:resource UserResource
```

```
INFO Resource [app/Http/Resources/UserResource.php] created successfully.
```

Cette commande génère une classe dans le dossier app/Http/Resources.

# Resource - Création



```
namespace App\Http\Resources;

use App\Models\Message;
use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

no usages
class MessageResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @return array<string, mixed>
     */
    no usages
    public function toArray(Request $request): array
    {
        /** @var Message $message */
        $message = $this->resource;

        return [
            'title' => $message->title,
            'text' => $message->text,
        ];
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-resources#concept-overview>

# Resource - Exemple de création dans un Controller

```
public function index()
{
    /** @var Collection<Message> $messages */
    $messages = Message::all();

    return MessageResource::collection($messages);
}

no usages

public function show(string $id)
{
    /** @var Message $message */
    $message = Message::query()->findOrFail($id);

    return MessageResource::make($message);
}
```

## Resource - Exemple de réponse



```
{  
  "data": {  
    "title": "Consequatur voluptatum ipsam accusamus dolores.",  
    "text": "Rerum vel est. Et doloribus quisquam doloremque voluptatem."  
  }  
}
```





# Form Request

# Form Request



Les "Form Request" permettent d'encapsuler la logique de validation des formulaires dans des classes dédiées.

Cela permet non seulement de garder les contrôleurs propres et concentrés sur leur rôle principal, mais offre également une approche plus structurée et réutilisable pour gérer la validation des données.

# Form Request - Création



Pour créer une FormRequest, vous pouvez utiliser la commande Artisan suivante :

```
php artisan make:request MessageRequest
```

```
INFO Request [app/Http/Requests/MessageRequest.php] created successfully.
```

Cette commande génère une classe dans le dossier app/Http/Requests.

# Form Request - Création

authorize détermine si l'utilisateur actuel est autorisé à effectuer l'action.

En retournant false, ce bloc indique que, par défaut, aucun utilisateur n'est autorisé à procéder avec cette requête, ce qui signifie que la requête sera automatiquement rejetée avant même que la validation du formulaire ne soit effectuée.

```
class MessageRequest extends FormRequest
{
    no usages
    public function authorize(): bool
    {
        return false;
    }

    no usages
    public function rules(): array
    {
        return [
            //
        ];
    }
}
```

Définit un ensemble de règles de validation pour les données soumises via le formulaire.

Chaque clé dans le tableau retournée par cette méthode correspond à un champ de formulaire, et chaque valeur associée définit les règles de validation pour ce champ spécifique.

# Form Request - Exemple



```
public function rules(): array
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

# Form Request - Règles de validations

## # Available Validation Rules

<a href="#">Accepted</a>	<a href="#">Exclude If</a>	<a href="#">Not Regex</a>	<a href="#">Date Format</a>	<a href="#">Lowercase</a>	<a href="#">Required Without</a>
<a href="#">Accepted If</a>	<a href="#">Exclude Unless</a>	<a href="#">Nullable</a>	<a href="#">Decimal</a>	<a href="#">MAC Address</a>	<a href="#">Required Without All</a>
<a href="#">Active URL</a>	<a href="#">Exclude With</a>	<a href="#">Numeric</a>	<a href="#">Declined</a>	<a href="#">Max</a>	<a href="#">Required Array Keys</a>
<a href="#">After (Date)</a>	<a href="#">Exclude Without</a>	<a href="#">Present</a>	<a href="#">Declined If</a>	<a href="#">Max Digits</a>	<a href="#">Same</a>
<a href="#">After Or Equal (Date)</a>	<a href="#">Exists (Database)</a>	<a href="#">Present If</a>	<a href="#">Different</a>	<a href="#">MIME Types</a>	<a href="#">Size</a>
<a href="#">Alpha</a>	<a href="#">Extensions</a>	<a href="#">Present Unless</a>	<a href="#">Digits</a>	<a href="#">MIME Type By File Extens...</a>	<a href="#">Sometimes</a>
<a href="#">Alpha Dash</a>	<a href="#">File</a>	<a href="#">Present With</a>	<a href="#">Digits Between</a>	<a href="#">Min</a>	<a href="#">Starts With</a>
<a href="#">Alpha Numeric</a>	<a href="#">Filled</a>	<a href="#">Present With All</a>	<a href="#">Dimensions (Image Files)</a>	<a href="#">Min Digits</a>	<a href="#">String</a>
<a href="#">Array</a>	<a href="#">Greater Than</a>	<a href="#">Prohibited</a>	<a href="#">Distinct</a>	<a href="#">Missing</a>	<a href="#">Timezone</a>
<a href="#">Ascii</a>	<a href="#">Greater Than Or Equal</a>	<a href="#">Prohibited If</a>	<a href="#">Doesnt Start With</a>	<a href="#">Missing If</a>	<a href="#">Unique (Database)</a>
<a href="#">Bail</a>	<a href="#">Hex Color</a>	<a href="#">Prohibited Unless</a>	<a href="#">Doesnt End With</a>	<a href="#">Missing Unless</a>	<a href="#">Uppercase</a>
<a href="#">Before (Date)</a>	<a href="#">Image (File)</a>	<a href="#">Prohibits</a>	<a href="#">Email</a>	<a href="#">Missing With</a>	<a href="#">URL</a>
<a href="#">Before Or Equal (Date)</a>	<a href="#">In</a>	<a href="#">Regular Expression</a>	<a href="#">Ends With</a>	<a href="#">Missing With All</a>	<a href="#">ULID</a>
<a href="#">Between</a>	<a href="#">In Array</a>	<a href="#">Required</a>	<a href="#">Enum</a>	<a href="#">Multiple Of</a>	<a href="#">UUID</a>
<a href="#">Boolean</a>	<a href="#">Integer</a>	<a href="#">Required If</a>	<a href="#">Exclude</a>	<a href="#">Not In</a>	
<a href="#">Confirmed</a>	<a href="#">IP Address</a>	<a href="#">Required If Accepted</a>			
<a href="#">Current Password</a>	<a href="#">JSON</a>	<a href="#">Required Unless</a>			
<a href="#">Date</a>	<a href="#">Less Than</a>	<a href="#">Required With</a>			
<a href="#">Date Equals</a>	<a href="#">Less Than Or Equal</a>	<a href="#">Required With All</a>			

Lien de la doc : <https://laravel.com/docs/10.x/validation#available-validation-rules>

# Form Request - Utilisation / Récupération de la donnée

```
/**
 * Store a new blog post.
 */
public function store(StorePostRequest $request): RedirectResponse
{
    // The incoming request is valid...

    // Retrieve the validated input data...
    $validated = $request->validated();

    // Retrieve a portion of the validated input data...
    $validated = $request->safe()->only(['name', 'email']);
    $validated = $request->safe()->except(['name', 'email']);

    // Store the blog post...

    return redirect('/posts!');
}
```

Lien de la doc : <https://laravel.com/docs/10.x/validation#form-request-validation>

# Form Request - Utilisation / Récupération de la donnée

```
public function store(MessageRequest $request)
{
    $message = Message::query()->create($request->validated());
    return MessageResource::make($message);
}
```



# Form Request - Retourner les erreurs avec du Json

```
class ArticleRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     */
    no usages
    public function authorize(): bool
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array<string, string>
     */
    no usages
    public function rules(): array
    {
        return [
            //
        ];
    }

    no usages
    protected function failedValidation(Validator $validator)
    {
        $response = response()->json([
            'message' => 'Validation errors',
            'data' => $validator->errors()
        ], status: 422);
        throw new HttpResponseException($response);
    }
}
```

```
protected function failedValidation(Validator $validator)
{
    $response = response()->json([
        'message' => 'Validation errors',
        'data' => $validator->errors()
    ], status: 422);
    throw new HttpResponseException($response);
}
```

Lien de la doc : <https://laravel.com/docs/10.x/validation#form-request-validation>



# **Relation entre les models**

# Relation entre les models



Une relation définit comment une entité se rapporte à une autre.

Par exemple, dans une boutique en ligne un client peut passer plusieurs commandes, établissant ainsi une relation "One to Many" entre le client et ses commandes.

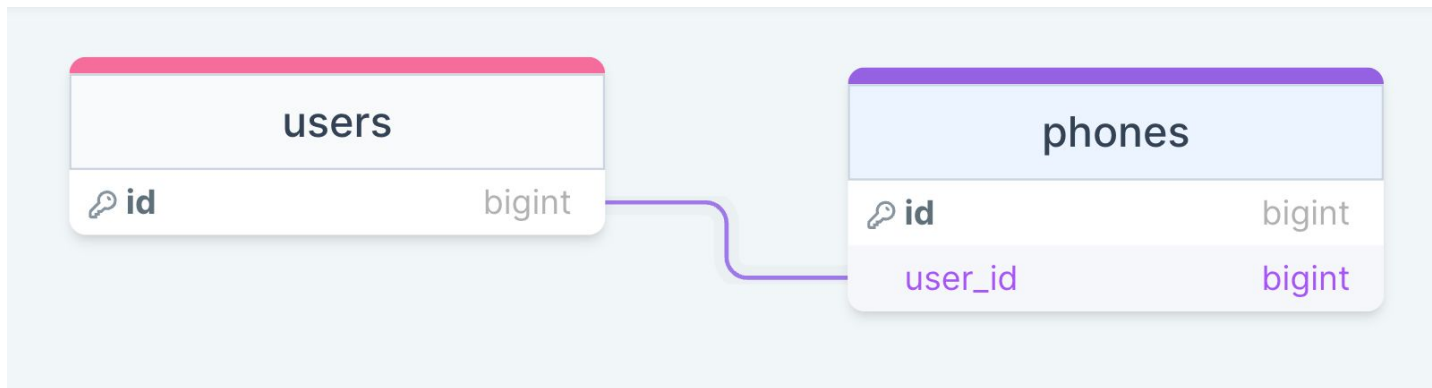
## Types de relations :

**One-to-One** : Chaque enregistrement dans une table correspond à un seul enregistrement dans une autre table. Exemple : chaque utilisateur a un profil unique.

**One-to-Many** : Un seul enregistrement dans une table est associé à plusieurs enregistrements dans une autre table. Exemple : un article de blog avec des commentaires.

**Many-to-Many** : Des enregistrements dans une table sont liés à plusieurs enregistrements dans une autre table, nécessitant souvent une table de jointure. Exemple : des étudiants suivant plusieurs cours.

# Relation - One to One



# Relation - One to One



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasOne;

class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone(): HasOne
    {
        return $this->hasOne(Phone::class);
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-relationships#defining-relationships>

# Relation - One to One



```
$phone = User::find(1)->phone;
```

# Relation - One to One



```
<?php

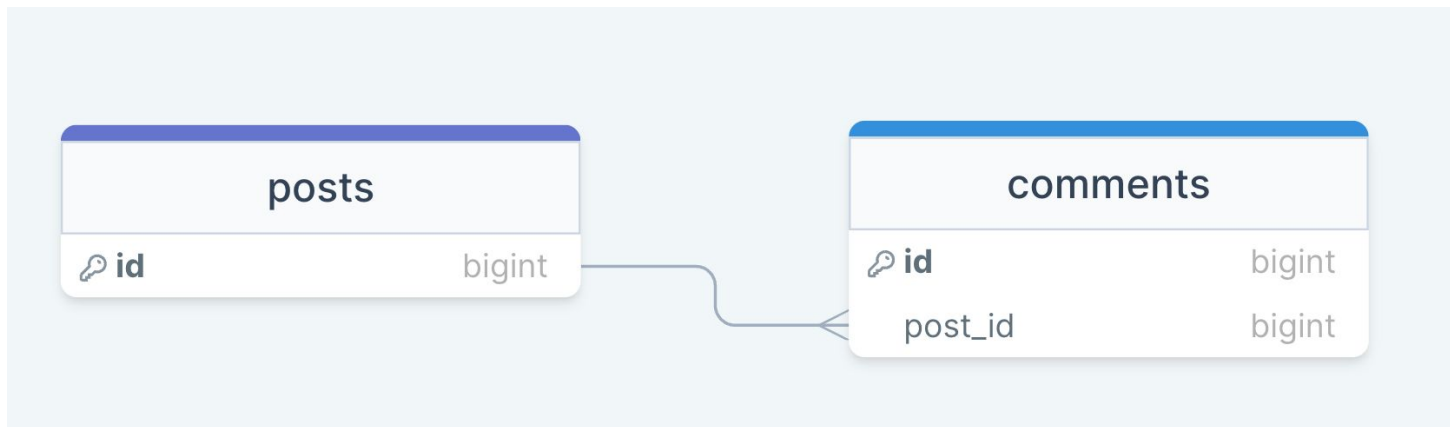
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-relationships#defining-relationships>

# Relation - One to Many





# Relation - One to Many



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Post extends Model
{
    /**
     * Get the comments for the blog post.
     */
    public function comments(): HasMany
    {
        return $this->hasMany(Comment::class);
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-relationships#defining-relationships>

# Relation - One to Many

```
use App\Models\Post;

$comments = Post::find(1)->comments;

foreach ($comments as $comment) {
    // ...
}
```

# Relation - One to Many



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Comment extends Model
{
    /**
     * Get the post that owns the comment.
     */
    public function post(): BelongsTo
    {
        return $this->belongsTo(Post::class);
    }
}
```

Lien de la doc : <https://laravel.com/docs/10.x/eloquent-relationships#defining-relationships>

# Relation - Eager Loading



L'eager loading est un concept dans Laravel qui permet de charger à l'avance les relations de modèles. Cela signifie que plutôt que de charger les données relationnelles sur demande (ce qui est connu sous le nom de "lazy loading"), Laravel récupère les relations spécifiées en même temps que le modèle principal, en utilisant un nombre minimal de requêtes.

## Pourquoi ?

Sans eager loading, chaque fois que vous accédez à une relation d'un modèle, Laravel exécute une requête supplémentaire de base de données pour cette relation. Ceci est connu comme le problème N+1 : pour N modèles chargés, vous pourriez finir par exécuter N+1 requêtes (1 pour les modèles principaux, plus N pour leurs relations). L'eager loading résout ce problème en pré-chargeant toutes les relations nécessaires en utilisant un nombre réduit de requêtes.

# Relation - Eager Loading

Sans Eager Loading

```
$books = Book::all();

foreach ($books as $book) {
    echo $book->author->name;
}
```

Avec Eager Loading

```
$books = Book::with('author')->get();

foreach ($books as $book) {
    echo $book->author->name;
}
```

```
select * from books
```

```
select * from authors where id in (1, 2, 3, 4, 5, ...)
```