TP N°6

Creation des Seeders et Factory

ISTA TINGHIR Module: M205

Realisée par: Groupe 6 le: 17 mai 2023 Déevelopper en Back-End

Introduction

Laravel est un framework d'applications Web qui fournit divers outils pour faciliter le développement d'applications Web. Dans cette présentation, nous aborderons les concepts de Seeder et Factory à Laravel. Seeders et Factories sont deux des outils proposés par Laravel qui sont utilisés pour remplir la base de données avec des données de test, ce qui facilite le développement et le test d'applications Web. Nous allons explorer ces concepts en détail et expliquer comment ils peuvent être utilisés efficacement pour améliorer le processus de développement d'une application Web avec Laravel.

1 . Creation de seeders

Seeder est un outil important dans Laravel qui permet aux développeurs d'insérer des données dans les tables de la base de données de manière programmatique. Seeder facilite le remplissage de la base de données avec des données de test, sans avoir à saisir manuellement les données dans les tables. Voici les étapes pour comprendre Seeder dans Laravel :

1. 1 Définition de Seeder :

Seeder est une fonctionnalité de Laravel qui permet aux développeurs de remplir la base de données de leur application avec des données de test. Il s'agit essentiellement d'une classe qui étend la classe Seeder fournie par Laravel et contient une méthode **`run()`** qui est appelée lorsque le seeder est exécuté.

1. 2 Création et exécution de Seeders :

Pour créer un Seeder dans Laravel, vous devez exécuter la commande :
`php artisan make:seeder SeederName`
dans le terminal. Cela créera une nouvelle classe Seeder dans le répertoire **`database/seeders`**.
Une fois que vous avez défini votre Seeder, vous pouvez l'exécuter en utilisant la commande
`php artisan db:seed`
Cela exécutera tous les seeders définis dans votre application.

En comprenant les étapes impliquées dans l'utilisation de Seeder dans Laravel, les développeurs peuvent utiliser efficacement cet outil pour rationaliser le processus de développement et de test de leur application. maintenant pour rendre l'idée plus claire, nous allons l'appliquer à la dimension optimale

Peremier exemple : (use Query Belder)

Supposons que nous ayons une table appelée `products` avec les colonnes `id`, `name`, `price` et `created_at`. Nous souhaitons remplir ce tableau avec des données de test. Voici comment nous pouvons créer une classe Seeder pour y parvenir :

1. Créez une nouvelle classe Seeder :

```
php artisan make:seeder ProductsTableSeeder
```

Open the ProductsTableSeeder.php file created in the database/seeders directory. In the run() method, define the data you want to seed:

```php
use Illuminate\Support\Facades\DB;
use Illuminate\Database\Seeder;

class ProductsTableSeeder extends Seeder
{
    public function run()
    {
        DB::table('products')->insert([
            [
                'name' => 'Product A',
                'price' => 10.99,
                'created_at' => now(),
            ],
            [
                'name' => 'Product B',
                'price' => 24.99,
                'created_at' => now(),
            ],
            [
                'name' => 'Product C',
                'price' => 5.99,
                'created_at' => now(),
            ],
        ]);
    }
}
```

Enfin, lancez le Seeder en utilisant la commande :

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

Cela exécutera la méthode `run()` définie dans la classe Seeder et insérera les données de test dans la table `products`.

Deuxième exemple : (use Eloquent)

```php
use App\Models\User;
use Illuminate\Database\Seeder;

class UsersTableSeeder extends Seeder
{
    public function run(): void
    {
        $users = [
            [
                'name' => 'John Doe',
                'email' => 'johndoe@example.com',
                'password' => bcrypt('secret'),
            ],
            [
                'name' => 'Jane Doe',
                'email' => 'janedoe@example.com',
                'password' => bcrypt('secret'),
            ],
            [
                'name' => 'Bob Smith',
                'email' => 'bobsmith@example.com',
                'password' => bcrypt('secret'),
            ],
        ];

        foreach ($users as $user) {
            User::create($user);
        }
    }
}
```

Troisième exemple : (use API)

```php
use App\Models\User;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Http;

class UsersTableSeeder extends Seeder
{
    public function run(): void
    {

        $response = Http::get('https://jsonplaceholder.typicode.com/users');
        $users = $response->json();

        foreach ($users as $user) {
            User::create([
                'name' => $user['name'],
                'email' => $user['email'],
                'password' => bcrypt('secret'),
            ]);
        }
    }
}
```

REMARQUE

Vous pouvez exécuter plusieurs Seeders avec une seule commande dans Laravel. Voici comment procéder :

1. Créez plusieurs classes Seeder, chacune avec sa propre méthode `run()` et ses propres données de test.
2. Ouvrez le fichier `DatabaseSeeder.php` situé dans le répertoire `database/seeders`.
3. Dans la méthode `run()` de `DatabaseSeeder`, appelez la méthode `call()` pour chaque classe Seeder que vous souhaitez exécuter :

```php
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run(): void
    {
        $this->call([
            UsersTableSeeder::class,
            ProductsTableSeeder::class
        ]);
    }
}
```

Dans cet exemple, nous appelons la méthode `call()` pour exécuter la méthode `run()` de deux classes Seeder : `UsersTableSeeder` et `ProductsTableSeeder`.

4. Enfin, exécutez la commande `php artisan db:seed` pour exécuter toutes les classes Seeder définies dans `DatabaseSeeder`.

2 . Factories:

2. 1 Définition de Factory :

Une factory est une classe dans Laravel qui permet de définir le modèle pour la création d'instances d'objets. Les factories sont utilisées pour générer des données fictives ou de test dans votre application Laravel. La classe factory de Laravel vous permet de définir les attributs par défaut de vos modèles et de générer plusieurs instances de modèles avec différentes valeurs pour leurs attributs. Les factories sont un outil puissant pour les tests et la génération de données fictives dans votre application Laravel.

2. 2 Création des Factories :

Pour créer une factory dans Laravel en utilisant la méthode factory, vous devez tout d'abord créer une nouvelle classe factory pour le modèle pour lequel vous souhaitez générer des données fictives. Vous pouvez le faire en utilisant la commande
`php artisan make:factory ModelnameFactory`
dans le terminal et en passant le nom du modèle en tant qu'argument. Cela créera une nouvelle classe Factory dans le répertoire `database/factories`.

Après avoir créé la classe factory pour un modèle spécifique, vous pouvez définir les valeurs par défaut des attributs en utilisant la méthode definition(). À l'intérieur de cette méthode, vous pouvez retourner un tableau associatif contenant les attributs et leurs valeurs par défaut. Par exemple (pour PostFactory):

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class PostsFactory extends Factory
{
    public function definition(): array
    {
        return [
            "title" => "title",
            "author" => 1,
            "content" => "lorem ipsum"
        ];
    }
}

```

2. 3 Generation des instances des Models:

Laravel propose deux méthodes pratiques pour générer des instances de modèle : **make** et **create**.
La méthode make permet de créer une instance de modèle et de la remplir avec des données simulées, mais ne la persiste pas en base de données.
La méthode create, quant à elle, permet de créer une instance de modèle, la remplir avec des données simulées, puis de la persister en base de données.

Pour utiliser ces méthodes, il suffit d'appeler la méthode statique **factory( )** sur la classe du modèle.

```php
// l'instance n'est pas sauvegardée en le base de données.
$post1 = Post::factory()->make();

// l'instance est sauvegardée en le base de données.
$post2 = Post::factory()->create();
```

Il est également possible de spécifier des valeurs personnalisées pour les attributs du modèle en utilisant un tableau associatif comme argument de la méthode **make( )** ou **create( )**. Par exemple :

```php
$personilizePost = Post::factory()->create([
    "title" => "title 1",
    "content" => "test"
]);
```

Il est également possible de générer plusieurs instances d'un modèle en une seule fois en utilisant la méthode **count( )** avec **make( )** ou **create( )**. Par exemple:

```php
/*
la méthode count() définit le nombre
d'instances de modèle Post à créer
*/
$posts = Post::factory()->count(10)->create();
```

2. 4 l'etat des factories:

Les états personnalisés sont une fonctionnalité utile pour définir des modèles avec des attributs spécifiques pour des cas d'utilisation particuliers. Pour créer un état personnalisé, vous pouvez déclarer une méthode dans la classe Factory du modèle correspondant. Dans cette méthode, vous pouvez utiliser la méthode **$this->state()** pour définir les attributs personnalisés que vous voulez appliquer à votre modèle. Le paramètre $attributes passé à la closure représente les attributs du modèle définis dans la méthode **definition()**. Vous pouvez utiliser l'état personnalisé en l'appelant sur votre factory lorsque vous créez une instance de modèle à l'aide de la méthode **create()** ou **make()**

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

class PostsFactory extends Factory
{
    public function private(): Factory
    {
        return $this->state(function (array $attributes){
            return [
                "state" => "private"
            ];
        });
    }
    public function definition(): array
    {
        return [
            "title" => "title",
            "author" => 1,
            "state" => "public",
            "content" => "lorem ipsum"
        ];
    }
}
```

vous pouvez créer une instance de modèle avec cet état en utilisant la syntaxe suivante:

```php
$privatePost = Post::factory()->private()->create();
```

2. 5 Les Relations entre des factories:

Les factories peuvent également être utilisées pour générer des modèles avec des relations entre eux.
on distingue entre deux relation "has" et "for", et chacun utilise une méthode du même nom `has()` et `for()`

a. La methode has():

La méthode has( ) de Laravel Factory permet de générer des modèles parents et enfants en une seule fois, en établissant une relation **has** entre eux. Cette méthode prend en argument une factory pour le modèle enfant, qui sera créée automatiquement et associée au modèle parent créé par la factory.

```
// cree un 'user' avec 3 'posts'
$user1 = User::factory()->has(Post::factory()->count(3))->create();

//methode magic
$user2 = User::factory()->hasPosts(3)->create();
```

a. La methode for():

La méthode for( ) permet de spécifier le modèle parent auquel un modèle créé par la factory appartient,Pour utiliser la méthode for et créer une relation parent-enfant entre deux modèles, vous pouvez utiliser la factory du modèle parent en passant une instance de cette dernière à la méthode for.

```
$posts1 = Post::factory()->count(3)->for(User::factory())->create();

$posts2 = Post::factory()->count(3)->for($user1)->create();

$posts3 = Post::factory()->count(3)->forUser()->create();
```

2. 6 faker biblio :

La bibliothèque Faker est un générateur de données aléatoires pour les tests unitaires. Elle est utilisée dans de nombreux frameworks et langages de programmation tels que PHP, Python, Ruby, etc. Cette bibliothèque permet de générer des données aléatoires pour divers types de champs tels que les noms, prénoms, adresses, adresses email, numéros de téléphone, dates, etc. Elle est utile pour créer des données fictives à utiliser dans les tests, afin de simuler des cas d'utilisation réels. Faker est facile à utiliser, et elle peut être personnalisée pour générer des données spécifiques à votre application. De plus, elle peut être utilisée pour générer des données de manière répétitive, ce qui est particulièrement utile lors de la création de jeux de données pour des tests d'acceptation.
Dans Laravel, la bibliothèque Faker est incluse par défaut pour générer des données aléatoires pour les usages de développement. Elle est utilisée en conjonction avec les factories pour créer des données cohérentes et variées pour les tests et les fausses données.

```php
public function definition()
{
    return [
        'name' => fake()->name(),
        'email' => fake()->unique()->safeEmail(),
        'email_verified_at' => now(),
        'password' => bcrypt('password'),
        'remember_token' => Str::random(10),
        'address' => fake()->address(),
        'phone' => fake()->phoneNumber(),
        'date_of_birth' => fake()
        ->dateTimeBetween('-60 years', '-18 years'),
        'gender' => fake()->randomElement(['male', 'female']),
        'bio' => fake()->paragraphs(3, true),
    ];
}
```

Les méthodes essentielles de Faker comprennent :

- name() : génère un nom complet aléatoire
- email() : génère une adresse e-mail aléatoire
- sentence() : génère une phrase aléatoire
- paragraph() : génère un paragraphe aléatoire
- numberBetween($min = 0, $max = 2147483647) : génère un nombre aléatoire compris entre $min et $max
- randomElement($array = array ('a','b','c')) : renvoie un élément aléatoire d'un tableau
- date($format = 'Y-m-d', $max = 'now') : génère une date aléatoire
- time($format = 'H:i:s', $max = 'now') : génère une heure aléatoire
- dateTime($format = 'Y-m-d H:i:s', $max = 'now') : génère une date et une heure aléatoires

conclusion :

En conclusion, les seeders et les factories sont deux outils très utiles pour le développement d'applications avec Laravel. Les seeders permettent de peupler la base de données avec des données de test ou de démonstration, tandis que les factories permettent de générer des instances de modèles avec des données aléatoires. En utilisant ces outils, les développeurs peuvent créer des applications plus efficacement en évitant les tâches répétitives et fastidieuses de création de données de test. De plus, la bibliothèque Faker fournit une multitude de méthodes pour générer des données aléatoires et uniques pour remplir les champs de la base de données. En somme, l'utilisation combinée de seeders, factories et Faker peut grandement faciliter le développement d'applications web avec Laravel.