Laravel CRUD генератор з нуля

23.05.2019 Cinex Веб-розробка 0

ПОШУК

ПОШУК ...

ПОЗНАЧКИ

APACHE AUTORUN BASH BITCOIN BLOCKCHAIN COMPOSER CRYPTOCURRENCY CSS DATABASE DJANGO DOCKER ELOQUENT ENGLISH ENUM GAMES GIMP GIT HTML JAVA JAVASCRIPT JQUERY LARAVEL MERCURIAL PARSING PHOTOSHOP PHP PHPSTORM

Давайте подивимося, як ми можемо створити власний генератор CRUD в Laravel, щоб спростити нашу роботу.

Як правило, вам знадобиться генератор, як тільки ви визначитеся з архітектурою контролерів, щоб в майбутньому їх не переписували.

Шаг 1 - створення проекту

Створіть новий проект Laravel:

```Shell
composer create-project laravel/laravel Contest
```

Шаг 2 - налаштування

Підключіть додаток laravel до бази даних і запустіть сервер.

Шаг 3 - створення команди

Тут ми починаємо працювати над командою ремісника(artisan) для генератора CRUD.

Створіть команду генератора CRUD:

```Shell
php artisan make:command CrudGenerator
```

Якщо зараз ви запустите:

```Shell
php artisan
```

Вам має бути доступна нова команда:

PLAYONLINUX PLUGINS PROBLEM SOLVING PYTHON REGEXP SEO SOFTWARE ENGINEERING SPRING SQL TELEGRAM TIPS AND TRICKS TIPS AND TRICKS UNIT TESTING WORDPRESS ВІРШІ ШПАРГАЛКИ

ОСТАННІ НОТАТКИ

20 Laravel Eloquent порад і трюків
15.12.2019

Як зробити скріншот сайту по URL на PHP
04.08.2019

Docker Cheat Sheet
05.07.2019

Гарячі клавіші Ubuntu Linux
01.07.2019

Git happens! 6 типових помилок Git і як їх виправити
22.06.2019

НЕДАВНІ КОМЕНТАРІ

```
command:name          Command description
```

Звичайно, команда ще не налаштована, і тому ви бачите ім'я і опис за замовчуванням.

Перед тим, як ми почнемо працювати над командою, нам знадобляться заглушки або креслення, то що ви віддаєте перевагу.

Шаг 4 - заглушки

Скопіюйте наступні заглушки в resources/stubs.

Каталогу заглушок не існує, тому обов'язково створіть його. Також назвіть файли з заголовками нижче. Controller.stub, Model.stub і Request.stub.

Як ви можете бачити нижче, у заглушок є наповнювачі, які ми збираємося пізніше замінити реальними даними.

Controller.stub

```php
<?php

namespace App\Http\Controllers;

use App\Http\Requests\{{modelName}}Request;
use App\{{modelName}};

class {{modelName}}Controller extends Controller
{
    public function index()
    {
        ${{modelNamePluralLowerCase}} = {{modelName}}::latest()->get();

        return response()->json(${{modelNamePluralLowerCase}});
    }

    public function store({{modelName}}Request $request)
```

```
    {
        ${{modelNameSingularLowerCase}} = {{modelName}}::create($request->all());

        return response()->json(${{modelNameSingularLowerCase}}, 201);
    }

    public function show($id)
    {
        ${{modelNameSingularLowerCase}} = {{modelName}}::findOrFail($id);

        return response()->json(${{modelNameSingularLowerCase}});
    }

    public function update({{modelName}}Request $request, $id)
    {
        ${{modelNameSingularLowerCase}} = {{modelName}}::findOrFail($id);
        ${{modelNameSingularLowerCase}}->update($request->all());

        return response()->json(${{modelNameSingularLowerCase}}, 200);
    }

    public function destroy($id)
    {
        {{modelName}}::destroy($id);

        return response()->json(null, 204);
    }
}
```

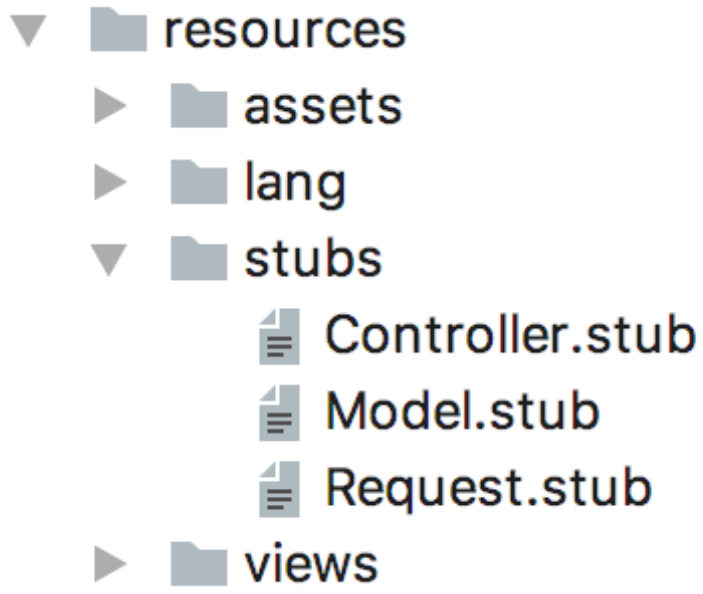
Model.stub

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class {{modelName}} extends Model
{
    protected $guarded = ['id'];
}
```

Request.stub

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class {{modelName}}Request extends FormRequest
{
    public function authorize()
    {
        return true;
    }

    public function rules()
    {
        return [];
    }
}
```

Вот как должна выглядеть ваша директория ресурсов:



Шаг 5 - робота на генераторі

Давайте тепер попрацюємо над командою консолі artisan, яку ми створили на кроці 3.

Відкрийте **CrudGenerator.php**. Ви можете знайти файл в app/Console/Commands.

Змініть підпис і опис наступним чином:

```php
protected $signature = 'crud:generator
    {name : Class (singular) for example User}';

protected $description = 'Create CRUD operations';
```

Опис досить простий.

Що стосується підпису, ми даємо йому ім'я і приймаємо аргумент, який ми називаємо **ім'ям.**

Ви можете викликати команду в такий спосіб:

```php
php artisan crud:generator Car
```

Круто, тепер давайте створимо функцію, щоб отримати заглушку, яка нам потрібна.

```php
protected function getStub($type)
{
    return file_get_contents(resource_path("stubs/$type.stub"));
}
```

getStub просто бере тип, по якому відбувається пошук і повертає вміст файлу-заглушки.

Далі, давайте подивимося, як ми можемо створити модель, використовуючи заглушку всередині resources/stubs.

```php
protected function model($name)
{
    $modelTemplate = str_replace(
        ['{{modelName}}'],
        [$name],
        $this->getStub('Model')
    );

    file_put_contents(app_path("/{$name}.php"), $modelTemplate);
}
```

Як бачите, функція моделі просто бере ім'я, яке ми передаємо через команду.

Подивіться на $modelTemplate. Ми використовуємо str_replace, щоб замінити наповнювачі всередині файлу Model.stub нашими бажаними значеннями.

По суті, в разі файлу Model.stub ми замінюємо **{{modelName}}** на **$name**. Пам'ятайте, що ім'я в прикладі вище, це Car.

Ви можете відкрити Model.stub і виконати математику, де б ви не побачили {{modelName}}, замініть його на Car.

file_put_contents просто створить новий файл, в якому ми знову будемо використовувати **$name**, тому файл буде називатися Car.php, і ми передамо в цей файл контент, отриманий з **$modelTemplate**, який є вмістом файла Model.stub, але з замінениими наповнювачами.

Те ж саме відбувається для контролера і запиту. Таким чином, я просто вставлю вміст для кожної функції тут.

```php
protected function controller($name)
{
    $controllerTemplate = str_replace(
        [
            '{{modelName}}',
            '{{modelNamePluralLowerCase}}',
            '{{modelNameSingularLowerCase}}'
        ],
        [
            $name,
            strtolower(str_plural($name)),
            strtolower($name)
        ],
        $this->getStub('Controller')
    );

    file_put_contents(app_path("/Http/Controllers/{$name}Controller.php"), $controllerTe
}

protected function request($name)
{
    $requestTemplate = str_replace(
        ['{{modelName}}'],
        [$name],
        $this->getStub('Request')
    );

    if(!file_exists($path = app_path('/Http/Requests')))
        mkdir($path, 0777, true);
```

```
30
31      file_put_contents(app_path("/Http/Requests/{$name}Request.php"), $requestTemplate);
32  }
```

Відмінно, до сих пір ми дивилися на наші допоміжні функції.

Тепер усередині handle() ми просто викликаємо їх.

```php
1  public function handle()
2  {
3      $name = $this->argument('name');
4
5      $this->controller($name);
6      $this->model($name);
7      $this->request($name);
8
9      File::append(base_path('routes/api.php'), 'Route::resource(\'' . str_plural(strtolow
10 }
```

Очевидно, що нам потрібно отримати введення, який користувач передав, використовуючи команду, яка в нашому випадку завжди буде Car, ми викликаємо функції, які ми пояснили вище, і в кінці ми створюємо ресурс маршруту і додаємо його в api.php файл.

Шаг 6 - запуск команди

Давайте спробуємо зараз, коли у нас все налаштовано.

Відкрийте термінал і запустіть:

```shell
1  php artisan crud:generator Car
```

Тепер у вас повинні бути доступні модель **Car.php**, контролер **CarController.php** і запит **CarRequest.php**.

Як бачите, команда НЕ створить міграцію для вас. Оскільки це завдання середнього рівня, я повинен все спростити, тому давайте швидко створимо міграцію з таблиці автомобілів.

```shell
php artisan make:migration create_cars_table --create=cars
```

Змініть метод **up()** на:

```php
public function up()
{
    Schema::create('cars', function (Blueprint $table) {
        $table->increments('id');
        $table->string('model');
        $table->integer('year');
        $table->timestamps();
    });
}
```

і запустіть ваші міграції.

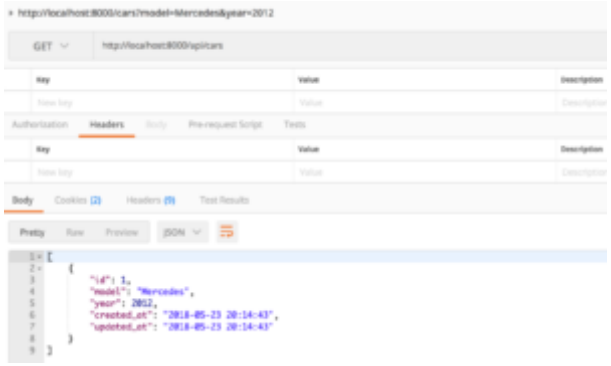
```shell
php artisan migrate
```

Крок 7 - тестування команди

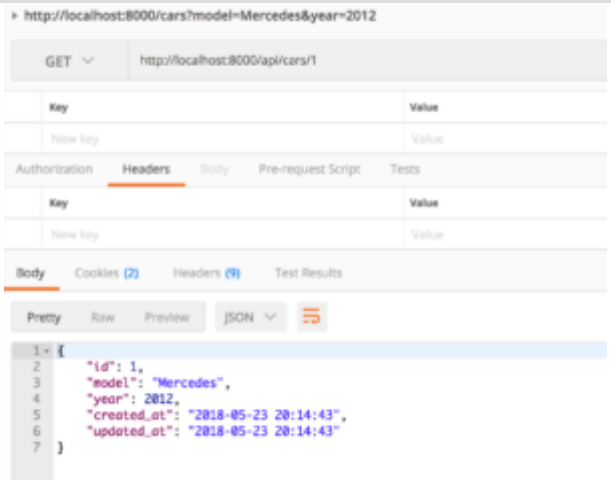
На цьому етапі я збираюся використовувати **Postman** для перевірки кінцевих точок.

Store

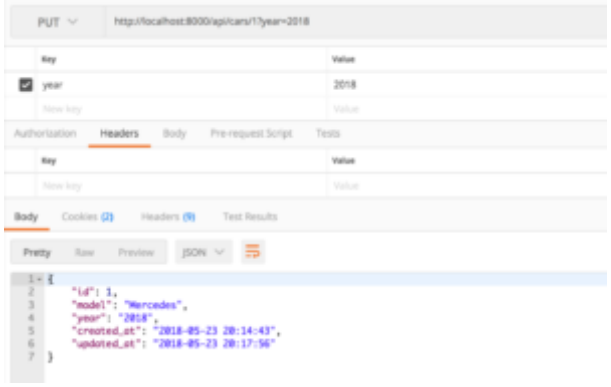
Index



Find



Update



Destroy

Висновок

Тепер у вас є генератор CRUD, готовий для вашого проекту. Якщо ви не дивіться на зовнішній вигляд заглушок, ви можете піти вперед і змінити їх.

В принципі, вам більше не потрібно витрачати 10-15 хвилин на створення простих операцій CRUD, ви можете отримати все в 1 простий команді. Наскільки це круто?

Оскільки я отримав багато задоволення під час написання цього посту, я збираюся зробити його повним пакетом з купою функцій в майбутньому! 🙂

Залишайтеся з нами і щасливого кодування!

Джерело

LARAVEL SOFTWARE ENGINEERING

« ПОПЕРЕДНІЙ
Як видалити Apache2 в Ubuntu або Debian

ДАЛІ »
Як написати команду artisan в Laravel, щоб зареєструвати супер адміністратора

ЗАЛИШТЕ ПЕРШИЙ КОМЕНТАР

Залишити коментар

Вашу адресу електронної пошти не буде опубліковано

Коментувати

Ім'я*

Email *

Збережіть моє ім'я, електронну пошту у цьому веб-переглядачі наступного разу, коли я коментуватиму.

ОПУБЛІКУВАТИ КОМЕНТАР