Artisan Commands

Sidrit Trandafili

@sidis405

@strandafili

- Sviluppatore PHP 15+ anni
- Laravel da 4 anni
- Back-end
  - php
  - java
  - node
  - python
- Front-end
  - VueJS
  - Vanilla JS
  - TailwindCSS
  - Sass
- Devops + Sysops da 13 anni

Artisan Commands

Artisan Commands

“ Tranne i comandi base già presenti tramite artisan, Laravel ci dà la possibilità di estendere la nostra applicazione tramite la creazione di comandi custom, per svolgere compiti da noi definiti. Questi comandi saranno poi accessibili sempre tramite artisan.

Artisan Commands

Esempio: creeremo un comando artisan per generare dati tramite le nostre model factories.

Dopo aver creato un modello User e uno Post, insieme alle migrations, creiamo due factories nel seguente modo

```php
<?php // database/factories/UserFactory.php
use Faker\Generator as Faker;
$factory->define(App\User::class, function (Faker $faker) {
    return [
        'name' => $faker->name,
        'email' => $faker->unique()->safeEmail,
        'password' => bcrypt('secret'),
        'remember_token' => str_random(10),
    ];
});
```

```php
<?php //database/factories/PostFactory.php
use Faker\Generator as Faker;
$factory->define(App\Post::class, function (Faker $faker) {
    return [
        'title' => $faker->sentence,
        'preview' => $faker->paragraph(2),
        'body' => $faker->paragraph(34),
        'user_id' => function () {
            return factory(App\User::class)->create()->id;
        }
    ];
```

Artisan Commands

Creiamo adesso il nostro comando

```
php artisan make:command PopulateDB // app/Console/Commands/PopulateDB.php
```

```
[...]
class PopulateDB extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';
[...]
}
```

Dentro il file creato, facciamo attenzione a due proprietà

$signature e $description sono i due dati che verranno stampati nel listato di comandi artisan.

Il primo definisce il modo in cui il comando verrà richiamato, il secondo è una breve descrizione

Artisan Commands

Modifichiamoli nel seguente modo

```
[...]
class PopulateDB extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'blog:populate';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Create fake users and posts';
[...]
}
```

Per poter essere riconosciuti da artisan, i nostri comandi custom devono essere registrati in app/Console/Kernel.php, nell'array $commands

```
protected $commands = [
        \App\Console\Commands\PopulateDB::class
    ];
```

Artisan Commands

Proviamo a listare i comandi artisan

```
php artisan
```

e vedremo quello appena creato da noi

```
blog
  blog:populate          Create fake users and posts
```

Piano d'azione: vogliamo ricevere il numero di utenti fake da creare, se creare posts o meno per questi utenti e se sì, quanti.
In questo modo il nostro comando dovrebbe essere invocato nel seguente modo

```
php artisan blog:populate 5 --posts=10
```

Dobbiamo dunque modificare la signature del nostro comando

```
protected $signature = 'blog:populate {users} {--posts=0}';
```

Artisan Commands

```
protected $signature = 'blog:populate {users} {--posts=0}';
```

{users} - in questo modo vengono definiti gli argomenti di un comando. Per definire un argomento opzionale gli si appende un '?' alla fine {users?}

Questi argomenti sono accessibili tramite $this->argument('users')

{--posts=0} - in questo modo vengono definite le opzioni di un comando. Questo input è opzionale e spesso gli viene associato un valore di default

Artisan Commands

Alla fine possiamo implementare la nostra funzione di seeding

```
public function handle()
    {
        $this->line("creating users " . $this->argument('users'));
        $this->line("creating posts " . $this->option('posts'));

        $users = factory(\App\User::class, intval($this->argument('users')))->create();

        if ($this->option('posts')) {
            foreach ($users as $user) {
                $this->line("Creating {$this->option('posts')} for user {$user->name}");
                $posts = factory(\App\Post::class, intval($this->option('posts')))
                    ->create(['user_id' => $user->id]);

                foreach ($posts as $post) {
                    $this->line("Created post: {$post->title}");
                }
            }
        }
    }
```