07 Jan 2016

USANDO REPOSITÓRIOS COM LARAVEL (PORTUGUÊS)

Imagine que você tem um model chamado Post. Ele interage com a tabela `Posts` e você o usa em vários métodos e ele está presente em praticamente toda a aplicação. Por enquanto, você o usa principalmente para procurar posts pelo ID, mas o seu chefe lhe disse que agora os posts deverão ser buscados pelo seu nome (slug), ou seja, `/posts/1` agora deve ser `/posts/nome-do-post`. Existem duas possibilidades: a sua aplicação não é grande o suficiente e você pode apenas procurar por todos os `Post::find($id)` e substituí-los por `Post::where('slug', $slug)->get()` ou algo do tipo. Tudo funciona perfeitamente! Mas e se a sua aplicação já for grande e você, por n motivos (testes, tempo, stress) não pode fazer isso? você ainda está procurando os posts pelo ID em todo canto do app. Mas... e se pudéssemos fazer uma camada entre a regra de negócio e a camada de dados (banco de dados, no nosso caso)? Iria evitar - entre outras coisas - código duplicado, problemas com testes, código descentralizado, etc. É mais ou menos esse o propósito de repositórios.

Vamos dizer que você quer mostrar um post. Geralmente, faríamos assim:

```
<?php
// app/Http/Controllers/PostsController.php
```

```
public method show($id) {
        $post = Post::find($id);
        return $post;
        // ou retornar uma view

?>
```

Não seria muito fácil manter esse tipo de código caso o app crescesse muito. Usando repositórios, podemos fazer algo do tipo:

```
<?php
// app/Repositories/PostRepository.php
namespace App\Repositories;

use App\Post;

class PostRepository
{
        protected $post;

        public function __construct(Post $post)
        {
                $this->post = $post;
        }

        public function find($id)
        {
                return $this->post->find($id);
        }

        public function findBy($att, $column)
        {
                return $this->post->where($att, $column)
        }
}
```

```
?>
```

Nós basicamente criamos uma camada que será usada entre o model e os controllers, de forma mais simples. Nós criamos métodos que buscam os posts. Para melhorar, poderíamos implementar uma interface, mas isso fica para outro post.

Agora, nós temos apenas **dois** métodos que nos permitem fazer várias coisas. Podemos achar um post pelo ID, slug, nome, título, conteúdo, qualquer coisa com apenas dois métodos. Nós precisaríamos de várias instâncias do model `Post` espalhadas pelo app, de forma desorganizada e descentralizada. Agora, temos tudo em uma só classe e vamos usá-la no `PostsController`.

```
<?php
// app/Http/Controllers/PostsController.php
namespace App\Http\Controllers;

use App\Repositories\PostRepository;

class PostsController
{
        protected $post;

        public function __construct(PostRepository $post)
        {
                $this->post = $post;
        }

        public function show($slug)
```

```
        {
                return $this->post->findBy('slug', $slug);
        }
}
?>
```

Dê uma olhada no que fizemos: primeiro, nós injetamos o nosso repositório, `PostRepository` no construtor e depois, no método `show` nós simplesmente usamos um método do repositório para achar posts em que a coluna slug da tabela posts tem o valor de `$slug`. O código está centralizado, fácil de testar e fácil de manter.

Espero que esse tutorial tenha ajudado alguém e mil desculas por quaisquer erros cometidos durante ele. :-)

Parte 2 do post: implementando interfaces e usando o container do Laravel

Share