



Victor Hugo Rocha

Follow

Desenvolvedor Web e escrevo umas paradas.

May 8 · 4 min read

## Criando um CRUD básico com Laravel 5.4



Olá, DevZ.

Nesse post eu vou partir do princípio de que você já sabe começar o seu projeto em Laravel e está querendo dar os seus primeiros passos com desenvolvimento web. Algum tempo atrás eu escrevi dois textos um sobre Migrations e outro sobre Faker & Factory que podem complementar o conteúdo dessa publicação.

O Laravel é um framework MVC, **Model-View-Controller**, que é um dos vários padrões de design de software, temos também o MVW, MVVM mas vamos nos focar no padrão MVC que possuem como

conhecimento sólido desse *design patterns* é um bom começo para utilizar qualquer ferramenta.

. . .

Vamos começar pela migration. Como você já sabe ela é como um controle de versão do seu banco de dados e permite que você adicione tabelas e colunas no seu banco sem muita dificuldade. Para criar a sua migration você deve rodar o comando *artisan* abaixo.

```
php artisan make:migration create_nametable_table
```

```
1  <?php
2
3  use Illuminate\Support\Facades\Schema;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Database\Migrations\Migration;
6
7  class CreateProductsTable extends Migration
8  {
9      public function up()
10     {
11         Schema::create('products', function (Blueprint $table) {
12             $table->increments('id');
13             $table->string('name', 30);
14             $table->text('description')->nullable();
15             $table->string('quantity');
16             $table->decimal('price', 5, 2);
17             $table->timestamps();
```

```
php artisan make:migration create_products_table
```

Na function **up** eu estou criando uma tabela *products* com os campos id, name, description, quantity e price. Eu explico o que são os campos *timestamps* e *softDeletes* [aqui](#). A função **down** dropa a tabela criada anteriormente. É interessante notar que eu posso definir coisas simples desde dizer que o campo description pode ser nulo até criar chaves primárias de maneira fácil como mostra a [documentação](#).

Agora vamos falar da Model. O Laravel possui integrado o Eloquent ORM que é uma forma mais bonita e abstrata de interagir com o seu banco de dados. Cada tabela de banco de dados tem um *Model* correspondente que é usado para interagir com essa tabela. Para criar a sua Model basta rodar o seguinte comando no terminal. Se você passar a Flag -m é criada uma migration referente a Model.

```
php artisan make:model NameModel
php artisan make:model NameModel -m
```

```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Product extends Model
8  {
9      protected $fillable = ['name', 'description', 'quantity'];
```

No meu caso a minha model Product é bem simples e possui três variáveis protegidas: *fillable*, *guarded* e *table*. A variável fillable define quais os campos que podem ser inseridos pelo usuário do sistema no Banco, o campo guarded protege os campos de inserções. Ele impede que alguém insira dados em alguns campos da nossa tabela.

Mesmo que não seja possível realizar essas inserções através do Front-end é importante que de alguma forma protejamos esses campos de possíveis vulnerabilidades que possam ser explorados por usuários mal intencionados. Agora que definimos na Model qual campos podem ser preenchidos pelo usuário podemos trabalhar na nossa **CRUD**.

Se você gostou do Eloquent você também pode usá-lo em um projeto que não seja Laravel. Você pode encontrar mais informações sobre isso na [documentação](#).

```
php artisan make:controller NameController
```

```

1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Http\Requests\ProductRequest;
6  use App\Product;
7
8  class ProductController extends Controller
9  {
10     public function index()
11     {
12         $products = Product::orderBy('created_at', 'desc')-
13         return view('products.index', ['products' => $produc
14     }
15
16     public function create()
17     {
18         return view('products.create');
19     }
20
21     public function store(ProductRequest $request)
22     {
23         $product = new Product;
24         $product->name = $request->name;
25         $product->description = $request->description;
26         $product->quantity = $request->quantity;
27         $product->price = $request->price;
28         $product->save();
29         return redirect()->route('products.index')->with('m
30     }
31
32     public function show($id)
33     {
34         //
35     }
36
37     public function edit($id)
38     {
39         $product = Product::findOrFail($id);

```

Dentro da Controller você pode ver que criamos as funções referentes

ao sistema CRUD de produtos. Cada função dentro do método de cada

```

1  <?php
2  ...
3
4  Route::group(['prefix' => 'admin', 'middleware' => 'auth'],
5      Route::get('/', 'AdminController@getIndex');
6      Route::get('projetos', 'ProjetosController@getIndex');
7      Route::get('projetos/inserir', 'ProjetosController@getIndex');
8      Route::post('projetos/inserir', 'ProjetosController@pos
9      Route::get('projetos/editar/{id}', 'ProjetosController@
10     Route::post('projetos/editar/{id}', 'ProjetosController

```

Nas versões anteriores do Laravel (4.2.\*) você precisava criar uma rota para cada função da sua CRUD. Nela você passava como segundo parâmetro a sua Controller e o método correspondente aquela rota. Como você pode perceber também é possível passar um middleware para a Rota, nesse exemplo apenas os usuários autenticados tem acesso a essa rota.

Se você encontrar algum problema em que você precise tratar uma rota específica você ainda pode fazer dessa maneira mas na versão 5.4 e superiores há um facilitador do Laravel, você pode fazer as mesmas 9 linhas do código anterior em apenas uma.

Com essa declaração de rota única você deixa para que o Laravel crie todas as rotas para a variedade de ações do *ProductController*, *dessa maneira, diferente do exemplo anterior, ele define os verbs PUT/PATCH para Update e DELETE para Delete da sua CRUD de Produtos.*

Você também pode fazer isso direto do terminal caso crie o controller dessa maneira. Note que além de definir que aquele é um Resource Controller dessa maneira você também indica qual o Model correspondente aquele Controller.

```

php artisan make:controller NameController --resource --
model=NameModel

```

1. Telegram Bot—Como desenvolver seu Bot com Javascript e Heroku
2. Docker—Dockerhub, pull e push nas suas imagens
3. TDD—Test Driven Development

See ya!

