**FACULDADE DE TECNOLOGIA DE TAUBATÉ**

**Everton de Oliveira Santos**

**Lucas Eduardo Castilho de Campos**

##### Padrão MVC no Laravel

**Taubaté**

**2018**

**Everton de Oliveira Santos**

**Lucas Eduardo Castilho de Campos**

##### Padrão MVC no Laravel

Trabalho apresentado ao professor da disciplina de Programação para WEB do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Centro Estadual de Educação Tecnológica Paula Souza para a obtenção da avaliação do 2º bimestre do 2º semestre de 2018.

**Professor  
Prof. Me Fernando Salles Claro**

**Taubaté**

**2018**

**SUMÁRIO**

[**Desenvolvimento** 4](#_Toc522697153)

**Desenvolvimento**

**1.0 MVC**

MVC é nada mais que um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário(view), a camada de manipulação dos dados(model) e a camada de controle(controller).

* 1. **Model**

Sempre que você pensar em manipulação de dados, pense em model. Ele é responsável pela leitura e escrita de dados, e de suas validações.

* 1. **View**

Simples: a camada de interação com o usuário. Ela apenas faz a exibição dos dados, sendo ela por meio de um html ou xml.

* 1. **Controller**

O responsável por receber todas as requisições do usuário. Seus métodos chamados actions são responsáveis por uma página, controlando qual model usar e qual view será mostrado ao usuário.

O diálogo das camadas: View: Fala Controller ! O usuário acabou de pedir para acessar o Facebook ! Pega os dados de login dele ai. Controller: Blz. Já te mando a resposta. Ai model, meu parceiro, toma esses dados de login e verifica se ele loga. Model: Os dados são válidos. Mandando a resposta de login. Controller. View, o usuário informou os dados corretos. Vou mandar pra vc os dados dele e você carrega a página de perfil. View: Mostrando ao usuário…

**2.0 Criação do Model no framework Laravel**

O Eloquent ORM incluído com o Laravel oferece uma implementação de ActiveRecord bela e simples para trabalhar com seu banco de dados. Cada tabela de banco de dados tem um "Modelo" correspondente que é usado para interagir com essa tabela. Os modelos permitem consultar dados em suas tabelas, bem como inserir novos registros na tabela. (link Documentação) Agora que já dei uma pincelada sobre a teoria vamos para um pouco de pratica.

Criando um Model:

Para criamos um model basta executar o seguinte comando:

php artisan make:model Cliente

Esse comando irá criar o Model Cliente, simples e fácil agora caso queira que já seja criado a migration basta adicionar o parâmetro "-m":

php artisan make:model Cliente -m

Com isso será criado uma migration dentro de database/migrations "2017\_08\_30\_224705\_create\_cliente\_table.php" com algumas colunas já configuradas (id e timestamp).

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateTsTable extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

Schema::create('clientes', function (Blueprint $table) {

$table->increments('id');

$table->timestamps();

});

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

Schema::dropIfExists('clientes');

}

}

Agora é só adicionar as suas colunas!

Sempre que criar um novo Model crie ele no seu singular por questão de padrão, e também porque por padrão o Model se relaciona ao nome do modelo em plural, exemplo: o Model Cliente se relaciona com a tabela clientes por padrão.

Mas eu não tinha falado que não é necessário refazer o meu banco ou tabelas para que funcione no Laravel? Agora sim vamos a parte que queria compartilhar com vocês!

Quando rodamos o php artisan make:model Cliente, é criado o seguinte modelo dentro de app:

namespace App;

use Illuminate\Database\Eloquent\Model;

class Cliente extends Model

{

//

}

**3.0 Criação de View no framework Laravel**

As visualizações contêm o HTML exibido por seu aplicativo e separam sua lógica de controlador / aplicativo de sua lógica de apresentação. As visualizações são armazenadas no diretório. Uma visão simples pode ser algo como isto:resources/views

<!-- View stored in resources/views/greeting.blade.php -->

<html>

<body>

<h1>Hello, {{ $name }}</h1>

</body>

</html>

Como esta visão é armazenada em , podemos devolvê-la usando o ajudante global da seguinte forma:resources/views/greeting.blade.phpview

Route::get('/', function () {

return view('greeting', ['name' => 'James']);

});

Como você pode ver, o primeiro argumento passado ao viewajudante corresponde ao nome do arquivo de visão no diretório. O segundo argumento é uma matriz de dados que deve ser disponibilizada para a exibição. Nesse caso, estamos passando a variável, que é exibida na visualização usando a sintaxe do Blade .resources/viewsname

Obviamente, as visualizações também podem ser aninhadas em subdiretórios do diretório. A notação "Dot" pode ser usada para referenciar vistas aninhadas. Por exemplo, se sua exibição estiver armazenada em , você poderá fazer referência a ela da seguinte forma:resources/viewsresources/views/admin/profile.blade.php

return view('admin.profile', $data);

Determinando se uma visão existe

Se você precisar determinar se uma exibição existe, você pode usar a Viewfachada. O existsmétodo retornará truese a visualização existir:

use Illuminate\Support\Facades\View;

if (View::exists('emails.customer')) {

//

}

Criando a primeira vista disponível

Usando o firstmétodo, você pode criar a primeira exibição que existe em uma determinada matriz de exibições. Isso é útil se seu aplicativo ou pacote permitir que as visualizações sejam personalizadas ou sobrescritas:

return view()->first(['custom.admin', 'admin'], $data);

Claro, você também pode chamar este método através da View fachada :

use Illuminate\Support\Facades\View;

return View::first(['custom.admin', 'admin'], $data);

**4.0 Criação de Controllers no framework Laravel**

Em vez de definir toda a sua lógica de manipulação de solicitações como Closures em arquivos de rota, você pode organizar esse comportamento usando classes de Controlador. Os controladores podem agrupar a lógica de manipulação de solicitações relacionadas em uma única classe. Controladores são armazenados no diretório.app/Http/Controllers

Controladores Básicos

Definindo Controladores

Abaixo está um exemplo de uma classe de controlador básica. Observe que o controlador estende a classe do controlador de base incluída no Laravel. A classe base fornece alguns métodos de conveniência, como o middlewaremétodo, que podem ser usados ??para conectar o middleware às ações do controlador:

<?php

namespace App\Http\Controllers;

use App\User;

use App\Http\Controllers\Controller;

class UserController extends Controller

{

/\*\*

\* Show the profile for the given user.

\*

\* @param int $id

\* @return View

\*/

public function show($id)

{

return view('user.profile', ['user' => User::findOrFail($id)]);

}

}

Você pode definir uma rota para esta ação do controlador da seguinte forma:

Route::get('user/{id}', 'UserController@show');

Agora, quando uma solicitação corresponde ao URI de rota especificado, o showmétodo na UserControllerclasse será executado. Naturalmente, os parâmetros da rota também serão passados ?para o método.

Os controladores não precisam estender uma classe base. No entanto, você não terá acesso a recursos de conveniência, como o middleware, validatee dispatchmétodos.

Controladores e espaços para nome

É muito importante observar que não precisamos especificar o namespace do controlador completo ao definir a rota do controlador. Como os RouteServiceProviderarquivos da sua rota são carregados em um grupo de rotas que contém o namespace, apenas especificamos a parte do nome da classe que vem após a parte do namespace.App\Http\Controllers

Se você optar por aninhar seus controladores mais profundamente no diretório, use o nome de classe específico relativo ao namespace raiz. Então, se a sua classe de controlador completa é , você deve registrar as rotas para o controlador da seguinte forma:App\Http\ControllersApp\Http\ControllersApp\Http\Controllers\Photos\AdminController

Route::get('foo', 'Photos\AdminController@method');

Controladores de ação única

Se você quiser definir um controlador que manipule apenas uma única ação, poderá colocar um único \_\_invokemétodo no controlador:

<?php

namespace App\Http\Controllers;

use App\User;

use App\Http\Controllers\Controller;

class ShowProfile extends Controller

{

/\*\*

\* Show the profile for the given user.

\*

\* @param int $id

\* @return View

\*/

public function \_\_invoke($id)

{

return view('user.profile', ['user' => User::findOrFail($id)]);

}

}

Ao registrar rotas para controladores de ação única, você não precisa especificar um método:

Route::get('user/{id}', 'ShowProfile');

Você pode gerar um controlador invokable usando a --invokableopção do comando Artisan:make:controller

php artisan make:controller ShowProfile --invokable

Middleware do Controlador

O middleware pode ser atribuído às rotas do controlador em seus arquivos de rota:

Route::get('profile', 'UserController@show')->middleware('auth');

No entanto, é mais conveniente especificar o middleware no construtor do seu controlador. Usando o middlewaremétodo do construtor do seu controlador, você pode facilmente atribuir o middleware à ação do controlador. Você pode até mesmo restringir o middleware apenas a determinados métodos na classe do controlador:

class UserController extends Controller

{

/\*\*

\* Instantiate a new controller instance.

\*

\* @return void

\*/

public function \_\_construct()

{

$this->middleware('auth');

$this->middleware('log')->only('index');

$this->middleware('subscribed')->except('store');

}

}

Os controladores também permitem que você registre o middleware usando um Closure. Isso fornece uma maneira conveniente de definir um middleware para um único controlador sem definir uma classe de middleware inteira:

$this->middleware(function ($request, $next) {

// ...

return $next($request);

});

Você pode atribuir um middleware a um subconjunto de ações do controlador; no entanto, isso pode indicar que seu controlador está crescendo muito. Em vez disso, considere dividir seu controlador em vários controladores menores.

Controladores de recursos

O roteamento de recursos do Laravel atribui as rotas "CRUD" típicas a um controlador com uma única linha de código. Por exemplo, você pode criar um controlador que manipule todas as solicitações HTTP para "fotos" armazenadas pelo seu aplicativo. Usando o comando Artisan, podemos criar rapidamente esse controlador:make:controller

php artisan make:controller PhotoController --resource

Este comando irá gerar um controlador em . O controlador conterá um método para cada uma das operações de recursos disponíveis.app/Http/Controllers/PhotoController.php

Em seguida, você pode registrar uma rota com recursos para o controlador:

Route::resource('photos', 'PhotoController');

Essa declaração de rota única cria várias rotas para manipular uma variedade de ações no recurso. O controlador gerado já terá métodos stubbed para cada uma dessas ações, incluindo notas informando sobre os verbos HTTP e URIs que eles manipulam.

Você pode registrar muitos controladores de recursos de uma só vez passando um array para o resourcesmétodo:

Route::resources([

'photos' => 'PhotoController',

'posts' => 'PostController'

]);

Especificando o modelo de recurso: Se você estiver usando a vinculação de modelo de rota e desejar que os métodos do controlador de recursos identifiquem uma instância de modelo, use a --modelopção ao gerar o controlador:

php artisan make:controller PhotoController --resource --model=Photo

Métodos de formulário de falsificação Como os formulários HTML não podem fazer PUT, PATCHou DELETEsolicitações, você precisará adicionar um \_methodcampo oculto para falsificar esses verbos HTTP. A @methoddiretiva Blade pode criar este campo para você:

<form action="/foo/bar" method="POST">

@method('PUT')

</form>