# **E se o nosso back-end mudasse?**

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/09-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

## **O problema de URL's espalhadas**

Nossa aplicação é funcional, não podemos negar! Mas o que aconteceria se os endereços dos nossos serviços mudassem? Teríamos que alterar em diversos lugares da aplicação! Outro ponto é que alguns endereços nasceram da concatenação de strings, algo muito sujeito a erro.

Esses problemas aparecem porque $http não é capaz de esconder parte da complexidade que é lidar com nossos endpoints REST.

## **Interagindo com o servidor em alto nível**

Porém, há um módulo especializado que pode ser aplicado quando usamos este padrão, pelo menos na definição de nomes de URL's, o **ngResource**. A ideia é substituirmos $http por este serviço para tornar nossa aplicação mais enxuta e fácil de manter.

O primeiro passo para usarmos $resource é importarmos o script do módulo ngResoure em nossa viewindex.html.

<!-- public/index.html -->  
<!DOCTYPE html>  
<html lang="pt-br" ng-app="alurapic">  
 <head>  
 <base href="/">  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width">  
 <title>Alurapic</title>  
 <link rel="stylesheet" href="css/bootstrap.min.css">  
 <link rel="stylesheet" href="css/bootstrap-theme.min.css">  
 <link rel="stylesheet" href="css/efeitos.css">  
 <script src="js/lib/angular.min.js"></script>  
 <script src="js/lib/angular-animate.min.js"></script>  
 <script src="js/lib/angular-route.min.js"></script>  
  
 <!-- novo aqui! -->  
 <script src="js/lib/angular-resource.min.js"></script>  
  
 <script src="js/main.js"></script>  
 <script src="js/controllers/fotos-controller.js"></script>   
 <script src="js/controllers/foto-controller.js"></script>  
 <script src="js/controllers/grupos-controller.js"></script>  
 <script src="js/directives/minhas-diretivas.js"></script>  
 </head>  
 <body>  
 <div class="container">  
 <ng-view></ng-view>  
 </div><!-- fim container -->  
 </body>  
</html>

O segundo, não menos importante, é declararmos ngResource como dependência do módulo principal da aplicação:

// public/js/main.js  
  
angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute', 'ngResource'])  
 .config(function($routeProvider, $locationProvider) {  
  
 // código omitido   
 });

Vamos começar a modificar FotosController, substituindo na injeção de dependências $http por$resource e criando um novo recurso:

// public/js/controllers/fotos-controller.js  
  
angular.module('alurapic').controller('FotosController', function($scope, $resource) {  
  
 // novidade aqui! Criando o recurso  
 var recursoFoto = $resource('/v1/fotos/:fotoId');  
  
 $scope.fotos = [];  
 $scope.filtro = '';  
 $scope.mensagem = '';  
  
 // código posterior omitido   
});

Percebam que passamos como parâmetro para $resource a URL de um endpoint REST, porém com um curinga idêntico ao usado em uma das nossas rotas do Angular. Esse curinga é importante, porque permitirá que o serviço monte por debaixo dos panos as URL's de acesso ao recurso. O retorno é um recurso devidamente configurado.

Agora, vamos substituir $http que busca as fotos do servidor por uma chamada à **recursoFoto.query** que recebe dois parâmetros. O primeiro é uma função que será chamada quando a requisição for bem sucedida, o segundo apenas chamado quando houver um erro. Fazendo uma analogia com $http, o primeiro equivale à função success e o segundo error:

// public/js/controllers/fotos-controller.js  
  
angular.module('alurapic').controller('FotosController', function($scope, $resource) {  
  
 var recursoFoto = $resource('/v1/fotos/:fotoId');  
 $scope.fotos = [];  
 $scope.filtro = '';  
 $scope.mensagem = '';  
  
 // novidade aqui! Saiu $http.get!  
 recursoFoto.query(function(fotos) {  
 $scope.fotos = fotos;  
 }, function(erro) {  
 console.log(erro);  
 });  
  
 $scope.remover = function(foto) {  
 // código omitido  
 };  
});

Agora, vamos substituir a chamada de $http.delete pela chamada de recursoFoto.delete. Precisamos estar atentos, porque o primeiro parâmetro da função é um objeto cujas chaves devem corresponder ao curinga que usamos na definição da URL do recurso. No caso, passamos {fotoId: foto.\_id}, ou seja, o valor da chave é o ID da foto que desejamos apagar. Os outros dois parâmetros são as funções executadas quando a requisição for bem sucedida e quando houver algum erro respectivamente:

// public/js/controllers/fotos-controller.js  
  
angular.module('alurapic').controller('FotosController', function($scope, $resource) {  
  
 var recursoFoto = $resource('/v1/fotos/:fotoId');  
 $scope.fotos = [];  
 $scope.filtro = '';  
 $scope.mensagem = '';  
  
 recursoFoto.query(function(fotos) {  
 $scope.fotos = fotos;  
 }, function(erro) {  
 console.log(erro);  
 });  
  
 $scope.remover = function(foto) {  
  
 // novidade aqui!  
 recursoFoto.delete({fotoId: foto.\_id}, function() {  
 var indiceDaFoto = $scope.fotos.indexOf(foto);  
 $scope.fotos.splice(indiceDaFoto, 1);  
 $scope.mensagem = 'Foto ' + foto.titulo + ' removida com sucesso!';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível apagar a foto ' + foto.titulo;  
 });  
 };   
});

Recarregando nossa página, nossa lista continua sendo exibida como antes, porém utilizamos um recurso menos verboso e especializado para consumir dados do servidor. Não acabamos ainda, FotoController, responsável pelo cadastro, também precisa ser modificado para utilizar $resource. Vamos trocar a injeção de$http por $resource e migrar apenas a parte que busca a foto pelo seu ID:

// public/js/controllers/foto-controller.js  
  
angular.module('alurapic')  
 .controller('FotoController', function($scope, $resource, $routeParams) {  
  
 // novidade!  
 var recursoFoto = $resource('/v1/fotos/:fotoId');  
 $scope.foto = {};  
 $scope.mensagem = '';  
  
 // migrando esta parte do código  
 if($routeParams.fotoId) {  
 recursoFoto.get({fotoId: $routeParams.fotoId}, function(foto) {  
 $scope.foto = foto;   
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível obter a foto'  
 });  
 }  
  
 $scope.submeter = function() {  
  
 // código omitido  
 };  
  
 });

Veja que recursoFoto.get também recebe como parâmetro uma chave com o mesmo nome do curinga usado na definição do recurso, assim como fizemos quando usamos recursoFoto.delete. Ótimo, agora precisamos migrar todo o código dentro da $scope.submeter, mas primeiro vamos migrar apenas a condição que cadastra novos fotos, deixaremos a atualização por último.

// public/js/controllers/foto-controller.js  
  
angular.module('alurapic')  
 .controller('FotoController', function($scope, $resource, $routeParams) {  
  
 var recursoFoto = $resource('/v1/fotos/:fotoId');  
 $scope.foto = {};  
 $scope.mensagem = '';  
  
 if($routeParams.fotoId) {  
 recursoFoto.get({fotoId: $routeParams.fotoId}, function(foto) {  
 $scope.foto = foto;   
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível obter a foto'  
 });  
 }  
  
 $scope.submeter = function() {  
  
 if ($scope.formulario.$valid) {  
 if($routeParams.fotoId) {  
  
 // código omitido, ainda não tocamos nele!  
  
 } else {  
  
 // novidade aqui! Alterando apenas a inclusão!  
  
 recursoFoto.save($scope.foto, function() {  
 $scope.foto = {};  
 $scope.mensagem = 'Foto cadastrada com sucesso';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível cadastrar a foto';  
 });  
 }  
 }  
 };  
 });

Usamos recursoFoto.save, que gera por debaixo dos panos uma requisição do tipo POST para o recurso/v1/fotos. Ela recebe como primeiro parâmetro os dados que desejamos enviar, o restante não é novidade.

## **Alto nível, mas sem poder alterar recursos? Há solução!**

Testando, fica evidente que está funcionando. Para terminar, precisamos alterar o código que utiliza$http.put, contudo temos um sério problema: O serviço $resource não dá suporte ao verbo PUT, importante, pois é através desse verbo HTTP que nosso servidor saberá distinguir entre inclusão (POST) e alteração (PUT) de recurso. E agora, vamos voltar com a injeção de $http e utilizá-lo apenas para as alterações? Não é necessário, se $resource não suporta o verbo PUT, podemos criá-lo!

Vamos criar a função update em nosso recurso durante sua criação:

// public/js/controllers/foto-controller.js  
  
angular.module('alurapic')  
 .controller('FotoController', function($scope, $resource, $routeParams) {  
  
 // novidade aqui! Alteramos a criação de recursoFoto!  
 var recursoFoto = $resource('/v1/fotos/:fotoId', null, {  
 'update' : {   
 method: 'PUT'  
 }  
 });  
  
 $scope.foto = {};  
 $scope.mensagem = '';  
  
 // código posterior omitido  
 });

O serviço $resource recebe mais dois parâmetros além da URL com o curinga. O primeiro, que passamosnull, é utilizado quando queremos enviar os parâmetros através de query string, isto é, a URL é construída utilizando ?parametro=valor, algo que não usaremos. O segundo, um objeto com todas as novas ações que desejamos adicionar ao nosso recurso. Em nosso caso, adicionamos a ação update, que possui como parâmetro um objeto que indica qual método será utilizado, em nosso caso PUT.

Agora, podemos terminar de migrar a parte de alteração:

// public/js/controllers/foto-controller.js  
  
angular.module('alurapic')  
 .controller('FotoController', function($scope, $resource, $routeParams) {  
  
 var recursoFoto = $resource('/v1/fotos/:fotoId', null, {  
 'update' : {   
 method: 'PUT'  
 }  
 });  
  
 $scope.foto = {};  
 $scope.mensagem = '';  
  
 if($routeParams.fotoId) {  
 recursoFoto.get({fotoId: $routeParams.fotoId}, function(foto) {  
 $scope.foto = foto;   
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível obter a foto'  
 });  
 }  
  
 $scope.submeter = function() {  
  
 if ($scope.formulario.$valid) {  
  
 if($routeParams.fotoId) {  
  
 // Novidade aqui! Usando nosso update!  
  
 recursoFoto.update({fotoId: $scope.foto.\_id},   
 $scope.foto, function() {  
 $scope.mensagem = 'Foto alterada com sucesso';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível alterar';  
 });  
  
 } else {  
 recursoFoto.save($scope.foto, function() {  
 $scope.foto = {};  
 $scope.mensagem = 'Foto cadastrada com sucesso';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível cadastrar a foto';  
 });  
 }  
 }  
 };  
  
 });

Basta recarregarmos nossa página e testarmos alterando qualquer foto. Excelente, temos um código mais flexível, bem, nem tanto. O problema é que teremos que redefinir a função update em todos os controllers que quiserem a função. Além disso, temos outro problema parecido com o original: se a URL do recurso mudar? Precisaríamos mudar em todos os controllers que fizerem uso do recurso.

Para resolver problemas como esse, o Angular permite a criação de serviços que podem ser injetados, como qualquer recurso do Angular. Podemos esconder a complexidade do nosso $resource criando um serviço que o retorne já configurado.

Primeiro, vamos criar o arquivo public/js/services/meus-servicos.js. Criaremos o módulomeusServicos, que terá como dependência o módulo ngResource.

// public/js/servicos/meus-servicos.js  
  
angular.module('meusServicos', ['ngResource'])  
 .factory('recursoFoto', function($resource) {  
  
 return $resource('/v1/fotos/:fotoId', null, {  
 'update' : {   
 method: 'PUT'  
 }  
 });  
 });

É através da função factory que criamos um serviço passando seu nome e uma função que deve retornar um objeto. Em nosso caso, estamos devolvendo $resource já configurado.

Agora, precisamos importar nosso script em nossa página index.html e substituir a dependênciangResource por meusServicos, no módulo principal da aplicação main.js:

<!DOCTYPE html>  
<html lang="pt-br" ng-app="alurapic">  
 <head>  
 <base href="/">  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width">  
 <title>Alurapic</title>  
 <link rel="stylesheet" href="css/bootstrap.min.css">  
 <link rel="stylesheet" href="css/bootstrap-theme.min.css">  
 <link rel="stylesheet" href="css/efeitos.css">  
 <script src="js/lib/angular.min.js"></script>  
 <script src="js/lib/angular-animate.min.js"></script>  
 <script src="js/lib/angular-route.min.js"></script>  
 <script src="js/lib/angular-resource.min.js"></script>  
 <script src="js/main.js"></script>  
 <script src="js/controllers/fotos-controller.js"></script>   
 <script src="js/controllers/foto-controller.js"></script>  
 <script src="js/controllers/grupos-controller.js"></script>  
 <script src="js/directives/minhas-diretivas.js"></script>  
  
 <!-- novidade aqui! -->  
  
 <script src="js/services/meus-servicos.js"></script>  
 </head>  
 <body>  
 <div class="container">  
 <ng-view></ng-view>  
 </div><!-- fim container -->  
 </body>  
</html>

Agora main.js:

// public/js/main.js  
  
angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute', 'meusServicos'])  
 .config(function($routeProvider, $locationProvider) {  
  
 // código omitido  
  
 });

Agora, tanto em FotoController quanto em FotosController trocaremos a injeção de $resource porrecursoFoto, apagando também a linha onde criamos recursoFoto, já que ele agora é recebido via injeção de dependência:

// public/js/controllers/foto-controller.js  
  
angular.module('alurapic')  
 .controller('FotoController', function($scope, recursoFoto, $routeParams) {  
  
 $scope.foto = {};  
 $scope.mensagem = '';  
  
 if($routeParams.fotoId) {  
 recursoFoto.get({fotoId: $routeParams.fotoId}, function(foto) {  
 $scope.foto = foto;   
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível obter a foto'  
 });  
 }  
  
 $scope.submeter = function() {  
  
 if ($scope.formulario.$valid) {  
  
 if($routeParams.fotoId) {  
  
 recursoFoto.update({fotoId: $scope.foto.\_id},   
 $scope.foto, function() {  
 $scope.mensagem = 'Foto alterada com sucesso';  
 }, function() {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível alterar';  
 });  
  
 } else {  
 recursoFoto.save($scope.foto, function() {  
 $scope.foto = {};  
 $scope.mensagem = 'Foto cadastrada com sucesso';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível cadastrar a foto';  
 });  
 }  
 }  
 };  
  
 });

Agora em public/js/controllers/fotos-controller.js.

// public/js/controllers/fotos-controller.js  
  
angular.module('alurapic').controller('FotosController', function($scope, recursoFoto) {  
  
 $scope.fotos = [];  
 $scope.filtro = '';  
 $scope.mensagem = '';  
  
  
 recursoFoto.query(function(fotos) {  
 $scope.fotos = fotos;  
 }, function(erro) {  
 console.log(erro);  
 });  
  
 $scope.remover = function(foto) {  
  
 recursoFoto.delete({fotoId: foto.\_id}, function() {  
 var indiceDaFoto = $scope.fotos.indexOf(foto);  
 $scope.fotos.splice(indiceDaFoto, 1);  
 $scope.mensagem = 'Foto ' + foto.titulo + ' removida com sucesso!';  
 }, function(erro) {  
 console.log(erro);  
 $scope.mensagem = 'Não foi possível apagar a foto ' + foto.titulo;  
 });  
 };  
});

Agora é só verificar se a aplicação continua funcionando. Caso algum erro aconteça, basta fazer um checklist de todos os passos que executamos nesta alteração.

## **O que aprendemos neste capítulo?**

* o problema de URL's espalhadas
* $resource para consumir endpoints REST
* criar nossa própria função em $resource
* criar nosso próprio serviço