**Melhorando a experiência, agora do desenvolvedor!**

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/10-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

**Nosso controller faz coisas demais!**

Vamos atacar ainda mais a aplicação sob o ponto de vista do desenvolvedor, ele merece também uma boa experiência! Nosso controller, em teoria, só deveria se preocupar em atualizar nossa view, mas, se olharmos atentamente, vemos a lógica que testa se estamos alterando ou incluindo uma foto, para em seguida decidir qual operação chamar em recursoFoto. Se não tomarmos cuidado, nosso controller crescerá e se tornará um "bicho" complicado de se manter.

Vamos isolar a responsabilidade de cadastrar fotos em um novo serviço que receberá apenas uma foto como parâmetro e que decidirá que operação realizar. Se por acaso algum outro controller também precisar incluir ou alterar uma foto, ele poderá utilizar o mesmo serviço. Aliás, é através da criação de serviços que compartilhamos códigos entre controllers.

Em nosso controller, queremos usar uma sintaxe parecida com:

// não entra em nenhum lugar, apenas ilustrativo

cadastroDeFotos.cadastrar($scope.foto)

.then(function(dados) {

$scope.mensagem = dados.mensagem;

// limpa o formulário se for inclusão

if($scope.inclusao) $scope.foto = {};

})

.catch(function(dados) {

$scope.mensagem = dados.mensagem;

});

Muito mais clean do que antes, inclusive cadastroDeFotos segue o padrão promise, utilizado por outros serviços do Angular.

**Criando mais um serviço, agora com o promise pattern**

O primeiro passo é criarmos um novo serviço para o módulo meusServicos. Vamos editarpublic/js/services/meus-servicos.js para encadearmos mais uma chamada para a função .factory:

angular.module('meusServicos', ['ngResource'])

.factory('recursoFoto', function($resource) {

// código omitido

})

.factory("cadastroDeFotos", function(recursoFoto) {

var service = {};

service.cadastrar = function(foto) {

};

return service;

});

Não temos nenhuma novidade até agora: recebemos recursoFoto através do sistema de injeção de dependência do Angular. Faz todo sentido, já que nosso novo serviço precisará interagir com o back-end. Não custa lembrar que toda função factory deve retornar um objeto, nunca uma função. É por isso que criamos o objeto service. Logo em seguida, adicionamos ao objeto a função cadastrar, que recebe como parâmetro uma foto. A grande jogada é que a função cadastrar deve retornar uma promise e o Angular possui um serviço especializado na criação desse padrão, o **$q**. Ele também é recebido através de injeção:

angular.module('meusServicos', ['ngResource'])

.factory('recursoFoto', function($resource) {

// código omitido

})

.factory("cadastroDeFotos", function(recursoFoto, $q) {

var service = {};

service.cadastrar = function(foto) {

};

return service;

});

Antes de implementarmos a função cadastrar, vamos retornar logo de cara uma promise com auxílio de $q.

angular.module('meusServicos', ['ngResource'])

.factory('recursoFoto', function($resource) {

return $resource('/v1/fotos/:fotoId', null, {

'update' : {

method: 'PUT'

}

});

})

.factory("cadastroDeFotos", function(recursoFoto, $q) {

var service = {};

service.cadastrar = function(foto) {

return $q(function(resolve, reject) {

});

};

return service;

});

O serviço $q recebe em seu construtor uma função com dois parâmetros: resolve e reject. Ambos são funções, sendo que a primeira recebe como valor os dados que desejamos acessar chamando a função then de nossa promise, e para a segunda passamos qualquer informação de erro que temos acesso através da função catch.

Caso a foto tenha ID preenchido, é porque estamos alterando o recurso e, por conseguinte, devemos chamarrecursoFoto.update. Na função de sucesso de recursoFoto é que passamos o que desejamos para a funçãoresolve da nossa promise, em nosso caso, passamos um objeto com duas propriedades: uma que guarda a mensagem que desejamos enviar como resposta e um boolean para indicar que não foi uma inclusão, e sim uma alteração:

angular.module('meusServicos', ['ngResource'])

.factory('recursoFoto', function($resource) {

// código omitido

})

.factory("cadastroDeFotos", function(recursoFoto, $q) {

var service = {};

service.cadastrar = function(foto) {

return $q(function(resolve, reject) {

if(foto.\_id) {

recursoFoto.update({fotoId: foto.\_id}, foto, function() {

resolve({

mensagem: 'Foto ' + foto.titulo + ' atualizada com sucesso',

inclusao: false

});

}, function(erro) {

console.log(erro);

reject({

mensagem: 'Não foi possível atualizar a foto ' + foto.titulo

});

});

} else {

// vamos implementar já já, espere!

}

});

};

return service;

});

Agora, é só concluir o else:

angular.module('meusServicos', ['ngResource'])

.factory('recursoFoto', function($resource) {

return $resource('/v1/fotos/:fotoId', null, {

'update' : {

method: 'PUT'

}

});

})

.factory("cadastroDeFotos", function(recursoFoto, $q) {

var service = {};

service.cadastrar = function(foto) {

return $q(function(resolve, reject) {

if(foto.\_id) {

recursoFoto.update({fotoId: foto.\_id}, foto, function() {

resolve({

mensagem: 'Foto ' + foto.titulo + ' atualizada com sucesso',

inclusao: false

});

}, function(erro) {

console.log(erro);

reject({

mensagem: 'Não foi possível atualizar a foto ' + foto.titulo

});

});

} else {

recursoFoto.save(foto, function() {

resolve({

mensagem: 'Foto ' + foto.titulo + ' incluída com sucesso',

inclusao: true

});

}, function(erro) {

console.log(erro);

reject({

mensagem: 'Não foi possível incluir a foto ' + foto.titulo

});

});

}

});

};

return service;

});

Perceba que tanto na inclusão quanto alteração, quando um erro acontece, logamos essa informação e enviamos uma mensagem de alto nível para o usuário, que inclusive pode ser consumida pelo controller para que seja exibida.

**Botando nosso serviço para trabalhar**

Agora que temos nosso serviço pronto, vamos alterar FotoController para que faça uso dele. Só não esqueça de injetar nosso serviço cadastroDeFotos:

// public/js/controller/foto-controller.js

angular.module('alurapic')

.controller('FotoController', function($scope, recursoFoto, $routeParams, cadastroDeFotos) {

$scope.foto = {};

$scope.mensagem = '';

if($routeParams.fotoId) {

recursoFoto.get({fotoId: $routeParams.fotoId}, function(foto) {

$scope.foto = foto;

}, function(erro) {

$scope.mensagem = 'Não foi possível obter a foto'

});

}

$scope.submeter = function() {

if ($scope.formulario.$valid) {

cadastroDeFotos.cadastrar($scope.foto)

.then(function(dados) {

$scope.mensagem = dados.mensagem;

if (dados.inclusao) $scope.foto = {};

})

.catch(function(erro) {

$scope.mensagem = erro.mensagem;

});

}

};

});

Veja como $scope.submeter ficou mais simples e legível! Agora é praticar com os exercícios!

**O que aprendemos neste capítulo?**

* organizar lógicas reutilizáveis em serviços
* criar serviços que retornam promises