Seja bem-vindo ao curso de Angularjs do Alura.

Quando desenvolvemos no server-side, organizamos nosso código em camadas para facilitar a manutenção, o reaproveitamento e a legibilidade de nosso código. É muito comum aplicarmos o modelo **MVC** (**M**odel, **V**iew, **C**ontroller), que consiste na separação de tarefas, facilitando assim a reescrita de alguma parte e a manutenção do código.

Porém, não é raro o mesmo desenvolvedor deixar de lado essas práticas quando codifica no client-side. Mesmo aqueles que procuram organizar melhor seu código acabam criando soluções caseiras que nem sempre são bem documentadas.

Tendo como base este cenário, frameworks MVC client-side foram criados. Entre eles temos o [Backbone](http://backbonejs.org/), [Ember](http://emberjs.com/), [Knockout](http://knockoutjs.com/), [CanJS](http://canjs.com/), [Batman](http://batmanjs.org/), entre outros.

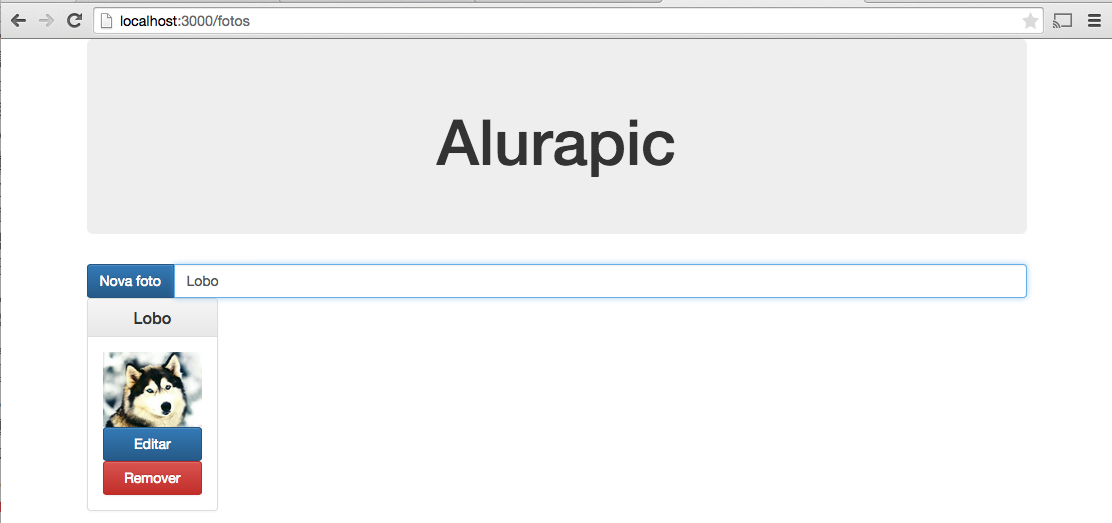
**Angular, o framework MVC da Google**

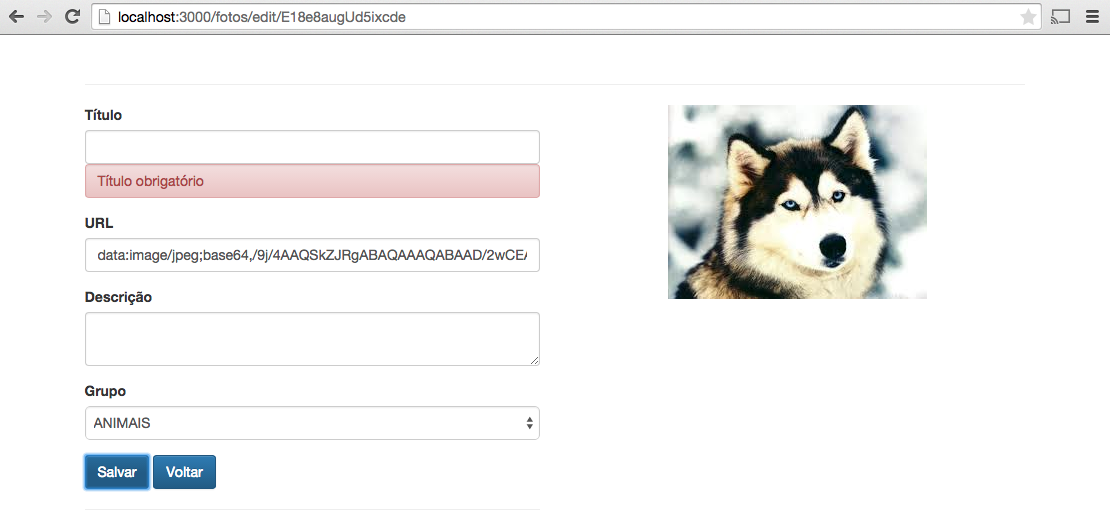
Um framework MVC no lado do cliente que tem ganhado muita atenção da comunidade é o Angular. Criado como um projeto interno da Google e liberado para o público em 2009, ele tem como foco a criação de **Single Page Applications** (SPA’s). Este tipo de aplicação não recarrega a página durante seu uso, dando uma experiência mais fluída para o usuário. Não se preocupe se você é novo para este tipo de aplicação, você terá a oportunidade de entender melhor seus conceitos ao longo do treinamento.

**Conhecendo um pouco da nossa aplicação, o Alurapic**

Durante o treinamento construiremos a aplicação **Alurapic**, um sistema simples de gerenciamento de imagens, permitindo que o usuário busque por aquelas que seguem determinado critério. Mas não se engane: o domínio do problema, apesar de simples, será suficiente para empregarmos grande parte do "arsenal" que o Angular nos fornece, inclusive toda aplicação funcionará em cima de um servidor web já preparado.







**Instalando e configurando toda infra necessária**

Agora que você já conhece um pouco sobre a aplicação que construiremos, saiba que alguns recursos do Angular dependem de um servidor web para funcionarem, em nosso caso, um servidor local. A boa notícia é que já disponibilizamos um para você, livrando-o dos seus detalhes de configuração. Inclusive ele fará persistência de dados sem que você tenha que instalar um banco de dados específico para isso. Porém, para tudo funcionar, você precisa ter instalado o Node.js.

O arquivo do projeto, o tutorial de instalação do Node.js e as instruções de como levantar o servidor estão no [primeiro exercício deste capítulo](https://cursos.alura.com.br/course/angularjs-mvc/task/9014). Esta é uma boa hora de fazê-lo antes de continuar. Vamos assumir a partir deste ponto que você realizou este primeiro exercício.

**Por onde começar?**

Angular é um framework que roda no lado do cliente, sendo assim, como qualquer outro script, deve ser importado na página que desejamos eleger como principal da aplicação, em nosso caso, a página já existente, index.html, que está salva dentro da pasta alurapic/public (não sabe que pasta é essa? Você fez o [primeiro exercício](https://cursos.alura.com.br/course/angularjs-mvc/section/2/task/3) do capítulo conforme solicitado?). Primeiramente vamos dar uma olhada em sua estrutura:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br">

<head>

<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">

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

</div> <!-- fim container -->

</body>

</html>

Repare que a página já importa o CSS do [Bootstrap](http://getbootstrap.com/). Se você ainda não o conhece, fique sabendo que ele permite aplicar um visual profissional em nosso projeto, com zero de esforço, apenas adicionando classes declaradas em seu arquivo CSS. Que classes são essas? Muita calma nessa hora! Elas serão introduzidas à medida que formos evoluindo nossa aplicação, mas já podemos adiantar que a classe container centraliza todo conteúdo da página e a text-center centraliza um elemento do tipo block, em nosso caso, a tag h1.

Agora que você já entendeu o papel do Bootstrap em nosso projeto, já podemos continuar. Todas as páginas, bibliotecas, scripts e qualquer outro arquivo dentro da pasta alurapic/public serão acessíveis através do seu navegador, inclusive já temos dentro da pasta alurapic/public/js/lib todos os arquivos do Angular que importaremos à medida que formos precisando. Vamos importar o script angular.min.js, o *core* do Angular dentro da tag head:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

</div> <!-- fim container -->

</body>

</html>

Pronto, mas ainda não escrevemos um código que utiliza o Angular! Para fazermos isso, primeiro precisamos criar um módulo.

**Criando o alicerce da nossa aplicação, o módulo principal**

A história é a seguinte: o framework nos ajuda a separar nosso código em pequenos grupos de código que podem ser combinados e reaproveitados quando necessário, esses grupos são chamados de **módulos**. Uma aplicação pode ter um, dez ou até mesmo mais de cinquenta módulos, tudo dependerá da complexidade da aplicação. Porém, há sempre um módulo que é o primeiro a ser inicializado assim que sua página é carregada pela primeira vez, inclusive ele também é o responsável pelo carregamento de outros módulos de que sua aplicação precisa para funcionar. É este que criaremos agora!

Vamos criar o arquivo public/js/main.js e mesmo sem escrever qualquer linha de código, vamos importá-lo sem demora na página index.html, abaixo da importação do core do Angular, para não corrermos o risco de esquecermos de importá-lo:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

</div> <!-- fim container -->

</body>

</html>

Pronto, e agora? Como criamos nosso módulo? Bem, o Angular disponibiliza para nós no escopo global o objeto angular (nada criativo, não?) que possui uma série de funções que nos permite interagir com o framework, entre elas a função **module** responsável pela criação de módulos.

Você deve estar pensando "Ok, entendi que é através desse objeto global que eu crio módulos e um monte de coisas do framework, mas escopo global não é algo ruim?". Preocupação justa, mas não se preocupe! Em uma aplicação bem-feita em Angular, este é o único objeto disponível globalmente, todo restante fica confinado dentro dos módulos do Angular! Com o tempo isso ficará ainda mais claro para você.

Criando o módulo principal da aplicação:

// public/js/main.js

angular.module('alurapic', []);

Acabamos de criar nosso primeiro módulo. Perceba que a função module recebe dois parâmetros. O primeiro é o nome do módulo que desejamos criar e o segundo é uma array com todos os módulos de que nosso módulo depende. Como não avançamos ainda com o projeto, não temos nenhuma dependência ainda, porém, você **não deve omitir**este parâmetro. Mais tarde você entenderá o que acontece quando ele é omitido (suspense!).

**Ensinando um truque novo para o navegador**

Tudo certo, criamos nosso módulo, mas como o Angular saberá que deve carregá-lo? Será que importar o script public/js/main.js é suficiente? Com certeza não.

Precisamos indicar em nossa página qual será o escopo de atuação do Angular, isto é, se ele gerenciará a página inteira ou apenas parte dela. Isso é importante, porque outro framework MVC pode estar sendo utilizado (algo raro, porém pode acontecer) e não queremos que o Angular bagunce seu trabalho.

Como apenas utilizaremos o Angular, gerenciaremos a página inteira, isto, a tag html e todos seus elementos filhos! Tudo bem, mas como faremos essa associação? Através do atributo **ng-app**:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

</div> <!-- fim container -->

</body>

</html>

Repare que o atributo ng-app tem como valor o nome do nosso módulo e não poderia ser diferente. Quando nossa página é carregada pela primeira vez o Angular encontrará esse atributo e carregará o módulo alurapic, tudo isso automaticamente, sem termos que nos preocupar em carregá-lo! É claro que se você esquecer de importar o arquivo main.js o Angular não será capaz de carregar o módulo, certo?

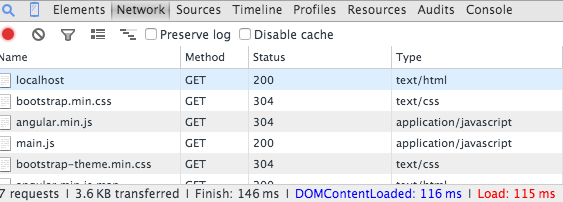
**Ah, então isso é uma diretiva?**

Agora, só uma coisa antes de continuarmos: o atributo ng-app existe no mundo HTML? Com certeza não, ele não faz parte da especificação da W3C. O "atributo" ng-app é na verdade uma **diretiva** do Angular.

Diretivas ampliam o vocabulário do navegador, ensinando-o novos truques, inclusive aprenderemos a criar nossas próprias diretivas ao longo do treinamento! Nesse caso a diretiva ng-app fornece a capacidade de nossa página carregar/iniciar o módulo principal da aplicação. Aliás, não vamos mais usar o termo página, usaremos **view**, termo mais correto quando estamos no universo do Angular!

**Nossa primeira página dinâmica**

Legal, agora, com o servidor iniciado (não lembra como iniciá-lo? Veja o [primeiro exercício](https://cursos.alura.com.br/course/angularjs-mvc/task/9014) do capítulo) vamos acessar o endereço http://localhost:3000 e verificar o resultado. Como esperado, nada impressionante, pois apenas preparamos a infraestrutura mínima de uma aplicação em Angular, que nada faz. No máximo, podemos ver através do console do navegador (eu uso Chrome, e você?) todos os arquivos carregados:



Bom, vamos começar a dar um colorido para nossa view index.html. Vamos adicionar uma foto, você pode escolher a URL de uma específica, não precisa ser igual a minha. Na tag img, utilizaremos as classes img-responsive center-block para que ela escale corretamente nos mais diversos dispositivos:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

</head>

<body>

<div class="container">

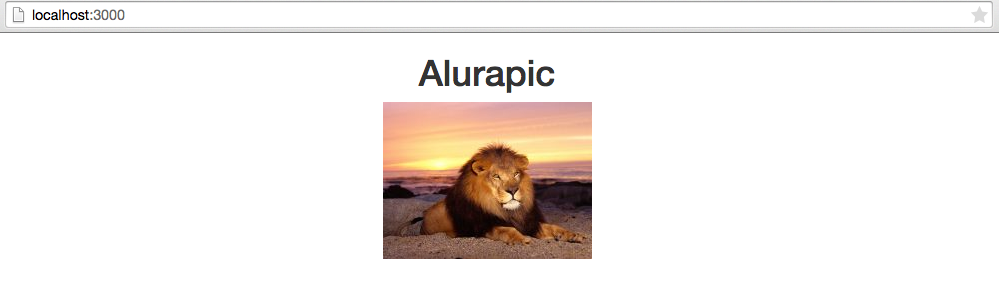
<h1 class="text-center">Alurapic</h1>

<img class="img-responsive center-block" src="http://www.fundosanimais.com/Minis/leoes.jpg" alt="Leão">

</div><!-- fim container -->

</body>

</html>



Bravo! Mas isso não impressiona, além do mais, se tivéssemos 100 imagens teríamos que repetir a tag img 100 vezes! Em nossa aplicação aprenderemos a cadastrar informações de imagens e a partir desses dados cadastrados montaremos dinamicamente uma lista de imagens! Só que antes de pensar em integração com o back-end, precisamos primeiro entender como o Angular fornece dados para a nossa view e como ela se constrói a partir desses dados.

Vamos realizar uma pequena mudança na tag img, alterando os atributos src e alt:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</div><!-- fim container -->

</body>

</html>

A primeira coisa é entender que **abrimos lacunas em nossa view index.html** através da sintaxe {{ }}. Se temos uma view que agora possui lacunas, podemos chamá-la de **template**. Quando alguém envia um memorando para outra, raramente ela começa do zero, ou seja, ela adota um template, toda uma estrutura já pronta e seu único trabalho é preencher essas lacunas que variam de acordo com a situação. Isso se aplica no mundo Angular! O que acontece se visualizarmos nossa página agora? **Nenhuma imagem será exibida e nenhum erro ocorrerá!** Aliás, o termo lacuna é muito genérico, no mundo Angular usamos **Angular Expression (AE)**. Todo {{ }} que encontrarmos chamaremos de AE. Combinado?

Quando temos um template que precisa de algum dado através de uma AE e não encontra, simplesmente aquela expressão fica em branco. Agora, a pergunta que não quer calar: quem, no modelo MVC é o responsável em disponibilizar dados para a views? Se você respondeu **controller** acertou!

**Nosso primeiro controller**

Precisamos criar um controller que disponibilize para a view o dado que ela precisa, no caso um objeto que contenha como chave o título e o endereço de uma foto, por exemplo { titulo: 'Leão', url : 'http://www.fundosanimais.com/Minis/leoes.jpg'}.

Lembra do nosso módulo principal da aplicação? Podemos criar um ou mais controllers diretamente nele, porém é uma boa prática declarar cada controller em arquivos separados, mesmo que eles façam parte do módulo alurapic. Vamos criar dentro de public/js/controllers/fotos-controller.js.

Criando nosso controller:

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

angular.module('alurapic').controller('FotosController', function() {

// definição do controller aqui

});

Veja que chamamos novamente angular.module, só que dessa vez sem passar o segundo parâmetro, o array vazio. Quando fazemos isso, indicamos que queremos acessar o módulo e não criar um novo. Faz todo sentido, se queremos que nosso controller faça parte do módulo alurapic. Em seguida encadeamos uma chamada à função **controller**que recebe dois parâmetros. O primeiro é o nome do controller que estamos criando na convenção *PascalCase*, o segundo uma função que define o controller.

Sabemos que o controller deve fornecer o objeto foto para a view e que esse objeto deve conter as chaves titulo e url. Faremos isso agora!

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

angular.module('alurapic').controller('FotosController', function() {

var foto = {

titulo : 'Leão',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

};

});

Excelente, agora precisamos importar o novo arquivo js que acabamos de criar em nossa view principal index.html:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body>

<div class="container">

<h1 class="text-center">Alurapic</h1>

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</div><!-- fim container -->

</body>

</html>

Será que isso é suficiente? Não, precisamos indicar dentro da nossa view index.htmlqual fragmento será gerenciado pelo nosso controller. Angular permite associarmos diferentes controllers para diferentes partes de nossa view, uma maneira de separar responsabilidades. Porém, neste exemplo, queremos que o controller gerencie a tag body e todos os seus elementos filhos e fazemos isso através da diretiva **ng-controller**, que deve ter como valor o nome exato do controller que criamos:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<h1 class="text-center">Alurapic</h1>

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</div><!-- fim container -->

</body>

</html>

Será que funciona? Visualizando no navegador, a imagem não é exibida e também não temos uma mensagem de erro. Por quê? O objeto foto não foi criado dentro do nosso controller? Sim, mas como ele foi declarado com var dentro de uma função, ele possui escopo privado. Ah, então para resolvermos vamos remover var, fazendo com que ele caia no escopo global? Nem pensar! Porém, o Angular disponibiliza uma ponte de ligação entre o controller e a view chamada $scope e tudo que for "pendurado" neste objeto será enxergado pela view. Mas como teremos acesso a esse objeto tão especial dentro do mundo Angular? Pedindo! Como? Recebendo-o na função que declara o controller:

angular.module('alurapic').controller('FotosController', function($scope) {

});

Angular encontra na declaração do controller $scope e sabe que tem que criar um para nós. Se tivéssemos escrito o nome do parâmetro de outra maneira, o framework não o criaria. Ou seja, Angular sabe o que deve buscar de sua infraestrutura de acordo com o nome do parâmetro que recebemos em nosso controller. Agora que já temos acesso à $scope, a ponte de ligação do controller com a view, podemos pendurar os dados da foto como sua propriedade.

angular.module('alurapic').controller('FotosController', function($scope) {

$scope.foto = {

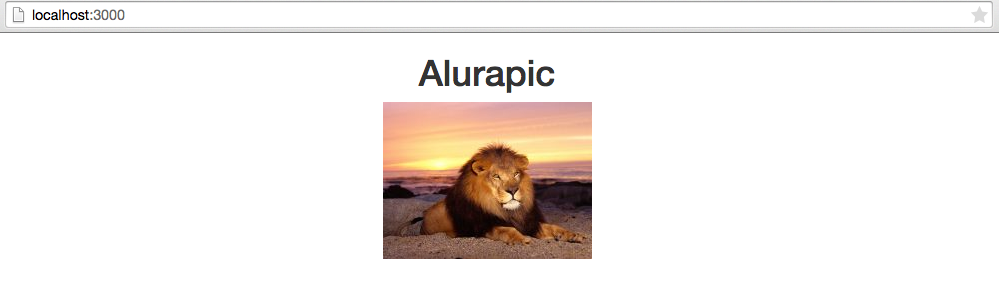
titulo : 'Leão',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

};

});

Lembre-se que tudo pendurado em $scope será acessível em nossa view, em nosso caso , através da angular expression. Duvida? Só testar e verificar o resultado.



**Ah, isso é data binding?**

Angular possui um termo apropriado para associação de um dado disponibilizado por um controller para a view: **data binding** (associação/ligação de dados). Qualquer alteração no dado do controller dispara uma atualização da view sem que o desenvolvedor tenha que se preocupar ou intervir no processo.

Excelente! Conseguimos um resultado semelhante ao que tínhamos antes, com a diferença de que agora a AE (Angular Expression) de nossa view foi processada com os dados fornecidos por FotosController. Pode parecer pouco, mas isso abre a porteira para que possamos avançar ainda mais no framework da Google.

**O que aprendemos neste capítulo?**

* o papel do modelo MVC
* o objeto global angular
* importação do Angular e criação do módulo principal da aplicação
* as diretivas ng-app e ng-controller
* Angular Expression (AE)
* o conceito de template
* criação de um controller
* o conceito de data binding

# Preparando o ambiente

Nosso foco é aprender Angular, porém alguns recursos do framework necessitam de um servidor web rodando localmente em sua máquina. Para que você não perca o foco do Angular e não caia em questões de infraestrutura que dizem respeito a um servidor web, disponibilizamos o projeto [alurapic (baixe agora)](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/01-alurapic.zip) com tudo necessário para subir um servidor web localmente, inclusive com os arquivos do angular já baixados.

**É importante destacar que o uso do projeto inicial alurapic não é opcional**, pois ele já possui registrado todos os endpoints que serão consumidos pela nossa aplicação em Angular. Além disso, para que o servidor funcione, é necessário ter o Node.js instalado em sua máquina.

O [Node.js](https://nodejs.org/) é um ambiente JavaScript multiplataforma disponível para Linux, Mac e Windows. Para instalá-lo, siga as instruções abaixo referentes a sua plataforma (versão 4.X):

## Linux (Ubuntu)

No Ubuntu, através do terminal (permissão de administrador necessária) execute o comando abaixo:

sudo apt-get install -y nodejs

**ATENÇÃO:** em algumas distribuições Linux, pode haver um conflito de nomes quando o Node é instalado pelo apt-get. Neste caso específico, no lugar do binário ser node, ele passa a se chamar **nodejs**. Isso gera problemas, pois a instrução **npm start** não funcionará, pois ela procura o binário node e não nodejs. Para resolver, use a seguinte instrução no terminal para subir o servidor:

nodejs server

É uma pena haver essa discrepância, mas fica aqui essa dica!

## Windows

Baixe o instalador clicando no grande botão install diretamente da página do [Node.js](https://nodejs.org/). Durante a instalação, você apenas clicará botões para continuar o assistente. Não troque a pasta padrão do Node.js durante a instalação a não ser que você saiba exatamente o que está fazendo.

## MAC

O [homebrew](http://brew.sh/) é a maneira mais recomendada para instalar o Node.js em sua máquina, através do comando:

brew update

brew install node

Não usa homebrew? Sem problema, baixe o instalador clicando no grande botão install diretamente da página do [Node.js](https://nodejs.org/).

## Rodando o servidor

Depois do Node.js ter sido instalado, **dentro da pasta do projeto alurapic** que você descompactou anteriormente, busque todas as dependências do projeto através do seu terminal (prompt de comando, no caso do Windows) favorito com o comando.

npm install

**ATENÇÃO USUÁRIOS DE WINDOWS**: se por acaso mensagens de erro forem exibidas, procure pelo texto **npm ERR! self signed certificate**. Se ele existir, isso indica um problema no certificado do seu roteador (proxy). Não se preocupe, basta rodar o comando no terminal **npm set strict-ssl false** que resolvera este problema.

Em menos de um minuto, todas as dependências para rodar o servidor terão sido baixadas. Para subi-lo utilizamos o comando:

npm start

Repare que seu terminal ficará aguardando indefinidamente, sinal de que o servidor está escutando. Agora é só abrir o navegador no endereço http://localhost:3000. Uma página de boas-vindas será exibida.

## Curiosidade

O projeto Alurapic utiliza [Express](http://expressjs.com/) para criar endpoints REST e o [NeDB](https://github.com/louischatriot/nedb), um banco de dados totalmente feito em Node.js e que não necessita qualquer configuração especial.

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/02-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

**Melhorando a apresentação da nossa página**

Bom, antes de avançarmos, vamos melhorar rapidamente o visual da nossa aplicação. Primeiro, vamos colocar seu título em destaque movendo a tag h1 para dentro de uma div, com a classe jumbotron:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</div><!-- fim container -->

</body>

</html>

Em seguida, vamos colocar nossa figura dentro de um painel do Bootstrap:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

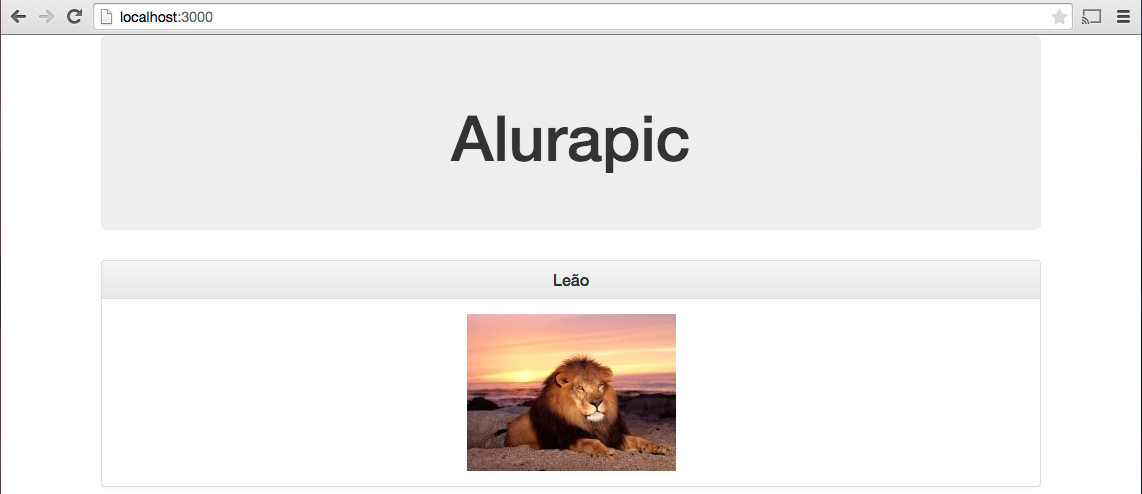
</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

</div><!-- fim container -->

</body>

</html>



Que tal agora adicionarmos mais fotos? Para simplificar, vamos copiar e colar o painel que já criamos!

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

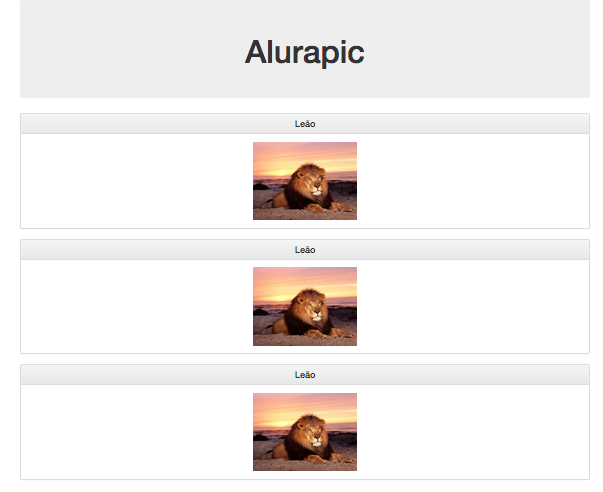
</div><!-- fim panel panel-default -->

</div><!-- fim container -->

</body>

</html>

O resultado, como esperado, é a repetição da imagem do leonino:



**Repetição de marcação dá trabalho!**

E se agora quisermos uma foto diferente para cada painel? Sabemos que, dentro de um controller, é por intermédio de $scope que disponibilizamos dados para a view. Que tal criarmos foto2 e foto3? Só para simplificar, vamos deixar a mesma url para eles, apenas mudaremos seus títulos:

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

angular.module('alurapic').controller('FotosController', function($scope) {

$scope.foto = {

titulo: 'Leão',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

};

$scope.foto2 = {

titulo: 'Leão2',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

};

$scope.foto3 = {

titulo: 'Leão3',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

};

});

Precisamos alterar a AE dos novos painéis para apontar para os dados corretos:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto2.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto2.url}}">

</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto3.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto3.url}}">

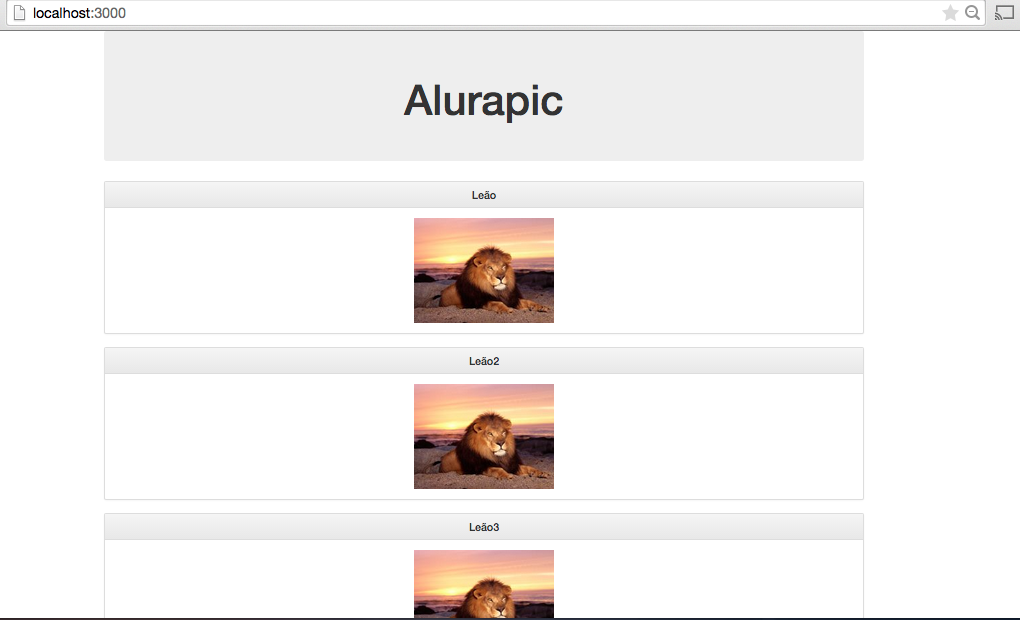
</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

</div><!-- fim container -->

</body>

</html>



Legal, cada foto possui um nome, porém essa solução deixa a desejar. Perceba que estamos repetindo a mesma marcação do painel do Bootstrap três vezes, mas o que muda é apenas o valor da AE. Segundo, para cada nova foto teremos que adicionar mais uma propriedade em $scope e copiar e colocar um novo painel.

Primeiro, vamos resolver o problema das múltiplas propriedades em $scope. No mundo JavaScript, quando queremos representar uma lista utilizamos um array:

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

angular.module('alurapic').controller('FotosController', function($scope) {

$scope.fotos = [

{

titulo: 'Leão',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

},

{

titulo: 'Leão2',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

},

{

titulo: 'Leão3',

url : 'http://www.fundosanimais.com/Minis/leoes.jpg'

}

];

});

Hum, interessante. Se precisarmos adicionar mais uma foto, basta adicioná-la dentro da lista. Porém, isso não resolve a renderização da nossa view. Se visualizarmos novamente, três painéis vazios serão exibidos, porque a view não foi capaz de avaliar a AE de cada um deles.

**Repetir? É com o Angular mesmo!**

Não seria interessante se o Angular fosse inteligente o suficiente para **repetir** o HTML do painel para cada objeto foto de nossa lista? Puxa vida, teríamos apenas um painel, o que evitaria a repetição, inclusive, para cada nova foto que entrar na lista, um novo painel seria criado! Já aprendemos as diretivas ng-app e a ng-controller, consegue ver um padrão? Diretivas do Angular começam com ng-! Então, como é repetir em inglês? *Repeat*! Então, se eu quero uma diretiva do Angular que repita a criação de uma marcação HTML para uma determinada lista temos a diretiva **ng-repeat**:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="panel panel-default" ng-repeat="foto in fotos">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

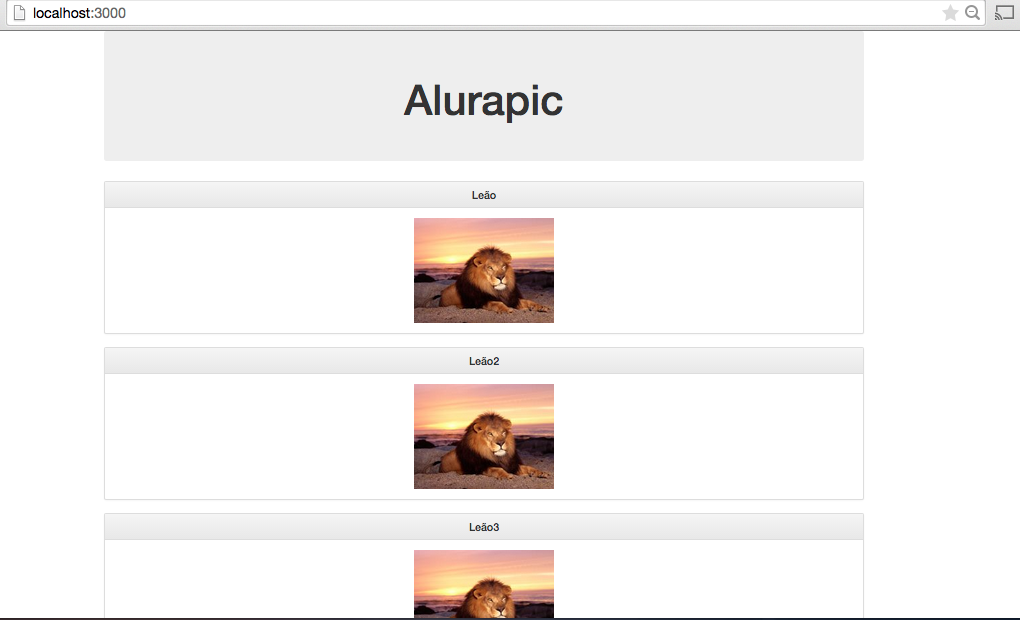
</div><!-- fim panel panel-default -->

</div><!-- fim container -->

</body>

</html>

Ao recarregarmos nossa página, as três imagens são exibidas, cada uma com seu título, com a diferença que temos um código HTML mais enxuto!



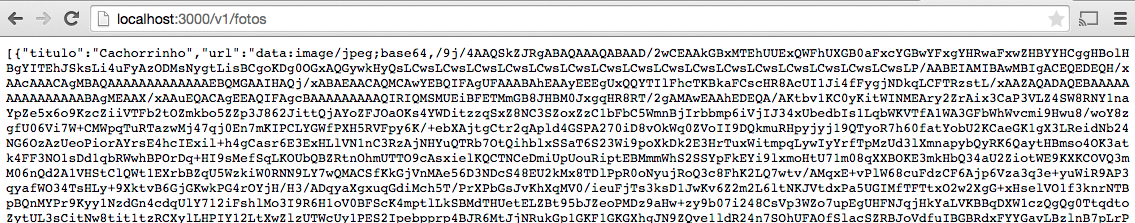
Na diretiva ng-repeat indicamos qual lista estamos varrendo, em nosso caso, fotos, que está no $scope de FotosController. Mas como ter acesso a cada contato da lista? Para isso, damos um apelido para cada item da lista, em nosso caso, escolhemos o apelido foto. Como temos três fotos em nossa lista, a diretiva ng-repeat será repetida três vezes e através da AE podemos ter acesso aos dados da foto, por exemplo, com {{foto.titulo}} acessamos seu título.

O bonito seria se a lista de fotos viesse de um servidor web. É tão bonito que faremos isso agora, chega de ficar digitando dados fixos!

**Chega de dados fixos! Vamos buscá-los do servidor!**

É extremamente comum uma aplicação feita em Angular consumir dados de um servidor através de uma API REST, que retorna esses dados na estrutura JSON. Nosso servidor já possui uma série de *endpoints* que utilizaremos ao longo do treinamento, porém há um deles que nos interessa. Que tal abrir em seu navegador para ver o resultado?

http://localhost:3000/v1/fotos



Não se assuste, mas o que o servidor retornou para você foi uma lista de fotos, porém, na chave url, não há o endereço de uma foto, mas o *DataUri* de uma foto. Fizemos isso para termos a garantia de que a foto sempre existirá, o que poderia não acontecer caso tivéssemos preenchido com o endereço de uma foto da web qualquer. Pois bem, a ideia é acessarmos essa lista através da nossa aplicação para em seguida jogá-la em $scope.fotos.

Para que possamos acessar os dados do nosso servidor precisaremos realizar requisições Ajax. Lembre-se que este tipo de requisição é assíncrono, isto é, que não bloqueia o uso da aplicação enquanto é executado. Se você vem do mundo jQuery já deve ter usado $.ajax ou uma de suas especializações, porém, o Angular possui seu próprio serviço para executar este tipo de requisição, o **$http**.

Da mesma maneira que pedimos ao Angular a criação de um $scope podemos pedir o $http recebendo-o como parâmetro na função que define FotosController. Vamos aproveitar e deixar a lista de fotos vazia:

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

});

Esse sistema de "pedirmos" ao Angular o que precisamos é chamado de injeção de dependências. Nós gritamos "eu preciso de um $http!" e o framework se vira para nos entregar um. Outra coisa é que podemos inverter a ordem dos parâmetros da função que ainda assim o Angular saberá injetá-los. É por isso que o nome dos parâmetros são importantes. Se digitarmos http no lugar de $http, Angular não saberá que precisamos desse serviço e seu valor será undefined.

Pois bem, temos $http. E agora? Como realizamos uma requisição assíncrona para o endpoint v1/fotos? Como queremos obter dados, usamos a função get:

// não entra em nenhum lugar, apenas ilustrativo

$http.get('/v1/fotos');

Então, o retorno da função $http.get é a nossa lista de fotos do servidor? Não, não é! Toda requisição assíncrona é incerta, não sabemos quanto tempo ela demorará para ser executada e se realmente será bem sucedida.

**Angular faz promessas?**

O que $http.get nos retorna é uma **promessa** de que ele buscará os dados. Sabemos que se essa promessa for cumprida, teremos os dados, caso contrário, ficaremos a ver navios. Tecnicamente falando, o que $http.get retorna é uma **promise**:

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

var promise = $http.get('/v1/fotos');

});

Quando essa promessa for cumprida, daí (then) podemos ter acesso aos dados retornados:

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

var promise = $http.get('/v1/fotos');

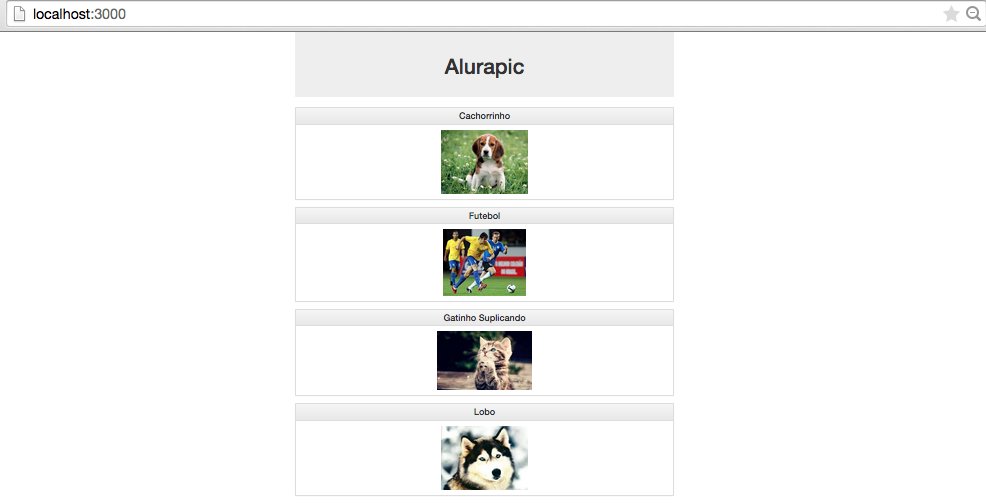
promise.then(function(retorno) {

$scope.fotos = retorno.data;

});

});

Veja que a função then recebe como parâmetro uma função passada por nós que será chamada apenas quando a requisição for concluída com sucesso, ou melhor, quando nossa promessa for resolvida. Na documentação do Angular, está escrito que é nela que temos acesso aos dados retornados pelo servidor, em nosso caso, escolhemos o nome retorno, mas poderia ser qualquer um. Porém, é de retorno.data que efetivamente temos nossa lista de fotos.



E se alguma coisa der errada? Por exemplo, a URL não existir ou o servidor cair? Podemos encadear uma chamada à função .catch que nos fornecerá um objeto com informações do erro que ocorreu:

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

var promise = $http.get('/v1/fotos');

promise.then(function(retorno) {

$scope.fotos = retorno.data;

})

.catch(function(erro) {

console.log(erro)

});

});

Por enquanto só vamos exibir no console do navegador a mensagem, mais tarde aprenderemos a exibir mensagens amigáveis para o usuário.

Podemos ainda omitir a declaração da variável promise:

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$http.get('/v1/fotos')

.then(function(retorno) {

$scope.fotos = retorno.data;

})

.catch(function(erro) {

console.log(erro);

});

});

Ou ainda usarmos success e error. A diferença é que com success não precisamos fazer retorno.data:

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$http.get('/v1/fotos')

.success(function(retorno) {

console.log(retorno);

$scope.fotos = retorno; // não precisa fazer retorno.data

})

.error(function(erro) {

console.log(erro);

});

});

Ainda neste treinamento aprenderemos a cadastrar novas fotos, editar, inclusive apagar aquelas que não nos interessam. Porém, como podemos perceber, a quantidade de fotos aumentou e pode ficar ainda maior. Vamos ajustar nosso layout colocando um grid responsivo do Bootstrap:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="row">

<div class="panel panel-default col-md-2" ng-repeat="foto in fotos">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

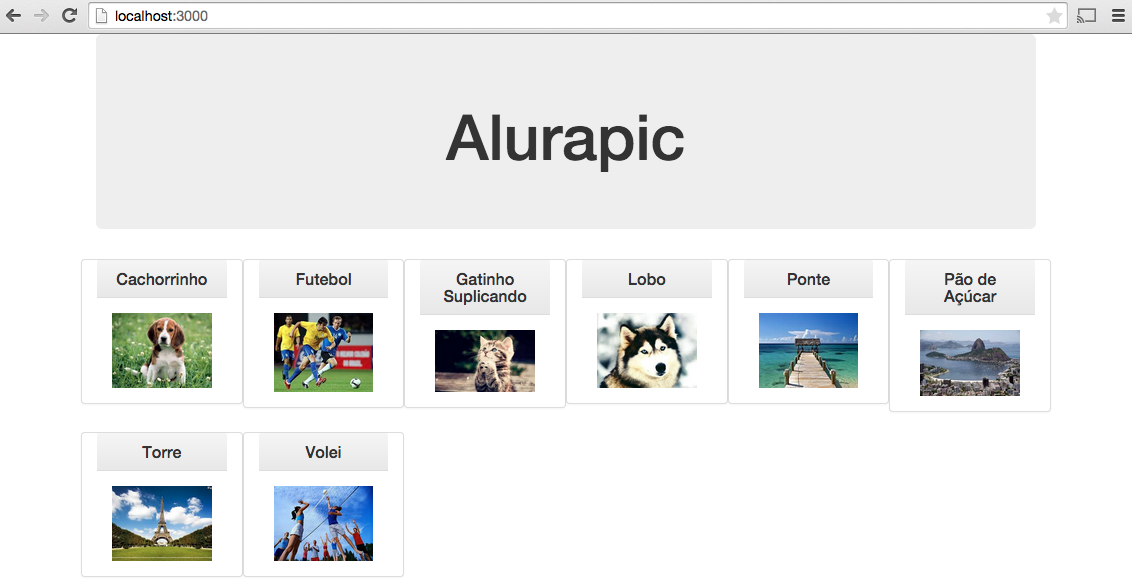
</div><!-- fim panel panel-default -->

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>



**O que aprendemos neste capítulo?**

* a diretiva ng-repeat
* injeção de dependências baseada no nome de parâmetro
* o serviço $http
* o conceito de promise
* comunicação com o back-end

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/03-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Progredimos com nossa aplicação, porém a marcação HTML da página index.html está um pouco confusa, pelo menos para mim. São muitas divs e classes, tudo porque estamos usando o Bootstrap. Ganhamos de um lado, mas perdemos em termos de legibilidade. Por exemplo, a marcação de um painel é assim:

<!-- exemplo, não entra em nenhum lugar -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">Leão</h3>

</div>

<div>

// conteúdo do painel

</div>

</div>

**Confinando a complexidade do painel em uma diretiva**

O mundo seria melhor se pudéssemos esconder a complexidade da marcação que vimos utilizando algo como um componente:

<!-- exemplo, não entra em nenhum lugar -->

<meu-painel titulo="Leão">

// conteúdo do Painel

</meu-painel>

Mas a tag meu-painel é um componente que não existe no mundo HTML. Porém, lembram das diretivas do Angular? Elas nada mais são do que componentes reutilizáveis e que podem existir não apenas como atributos, mas como tags também. Será que podemos criar uma diretiva customizada do Angular que esconda a complexidade do painel do Bootstrap? Sim e é isso que faremos agora para deixar nossa marcação mais elegante, inclusive poderemos aproveitá-la em outros lugares da nossa aplicação.

O primeiro passo é criar um módulo em separado que conterá nossas diretivas. Vamos criá-lo em:

public/js/directives/minhas-diretivas.js

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

Criamos um novo módulo. Antes de mais nada, precisamos importar seu script em index.html, e inclusive adicioná-lo como dependência do módulo principal da nossa aplicação.

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="row">

<div class="panel panel-default col-md-2" ng-repeat="foto in fotos">

<div class="panel-heading">

<h3 class="panel-title text-center">{{foto.titulo}}</h3>

</div>

<div class="panel-body">

<img class="img-responsive center-block" src="{{foto.url}}">

</div><!-- fim panel-body -->

</div><!-- fim panel panel-default -->

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas']);

Pronto, agora criamos nossa primeira diretiva através da função **directive**. Ela recebe como primeiro parâmetro o nome da diretiva em *camelCase* e como segundo uma função que deve retornar um **directive definition object (DDO)**:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

return ddo;

});

Veja que estamos retornando um objeto ainda sem qualquer configuração. Vamos começar restringindo a forma de uso da nossa diretiva. Como assim? Uma diretiva em Angular pode ser usada como **E**lemento, **A**tributo ou **C**omentário (esta última muito incomum). Vamos estipular que nossa diretiva pode ser usada tanto como atributo ou como elemento, adicionando em nosso DDO a propriedade **restrict** com valor "AE":

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

return ddo;

});

Nossa diretiva ainda não está pronta, mas ela pode ser usada como elemento, assim:

<!-- exemplo, não entra em nenhum lugar -->

<meu-painel></meu-painel>

Ou como atributo, neste caso, usamos uma div adicionando a diretiva:

<!-- exemplo, não entra em nenhum lugar -->

<div meu-painel></div>

Uma coisa importante que pode ter passado sem você perceber. O nome da nossa diretiva está em *camelCase*, porém na marcação HTML estamos usando hífen. Este é um padrão do Angular que não podemos deixar de seguir, caso contrário nossa diretiva não funcionará.

Muito bem, qual é o próximo passo? Bem, podemos ter várias diretivas meu-painelnuma mesma página, mas cada uma com seu próprio título. Algo assim:

<!-- exemplo, não entra em nenhum lugar -->

<meu-painel titulo="Leão"></meu-painel>

<meu-painel titulo="Zebra"></meu-painel>

<meu-painel titulo="Girafa"></meu-painel>

**Isolando nossa diretiva**

Para que cada diretiva tenha seu próprio título, cada uma precisará ter um escopo isolado, que existe independente do contexto na qual está incluída. Além de permitir que cada uma tenha seus próprios dados, podemos reutilizar a diretiva em qualquer lugar sem que ela bagunce o escopo pai no qual está inserida!

Precisamos capturar o título passado pela diretiva para dentro de seu escopo isolado e fazemos isso adicionando em nosso DDO a propriedade **scope**:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

ddo.scope = {

titulo: '@titulo'

};

return ddo;

});

Criamos a propriedade titulo no escopo isolado da diretiva, porém o valor desta propriedade é curioso. Veja que nele temos @titulo, a sintaxe @ indica que estamos **copiando o valor como string** do atributo titulo adicionando na diretiva em nossa marcação. Porém, quando o nome do atributo na diretiva na marcação é igual ao nome da propriedade que guardará o seu valor, podemos deixar apenas @:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

ddo.scope = {

titulo: '@'

};

return ddo;

});

Agora que restringimos a forma de uso da diretiva e criamos seu escopo isolado, precisamos definir a marcação HTML que será utilizada por ela. Fazemos isso através da propriedade **template**:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

ddo.scope = {

titulo: '@'

};

ddo.template =

'<div class="panel panel-default">'

+ ' <div class="panel-heading">'

+ ' <h3 class="panel-title text-center">{{titulo}}</h3> '

+ ' </div>'

+ ' <div class="panel-body">'

+ ' </div>'

+ '</div>'

return ddo;

});

Veja que no próprio template estamos usando a Angular Expression {{titulo}} para exibir o titulo passado como parâmetro para a diretiva. Será que funciona? Primeiro, vamos alterar index.html para fazer uso da nossa diretiva:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="row">

<meu-painel ng-repeat="foto in fotos" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

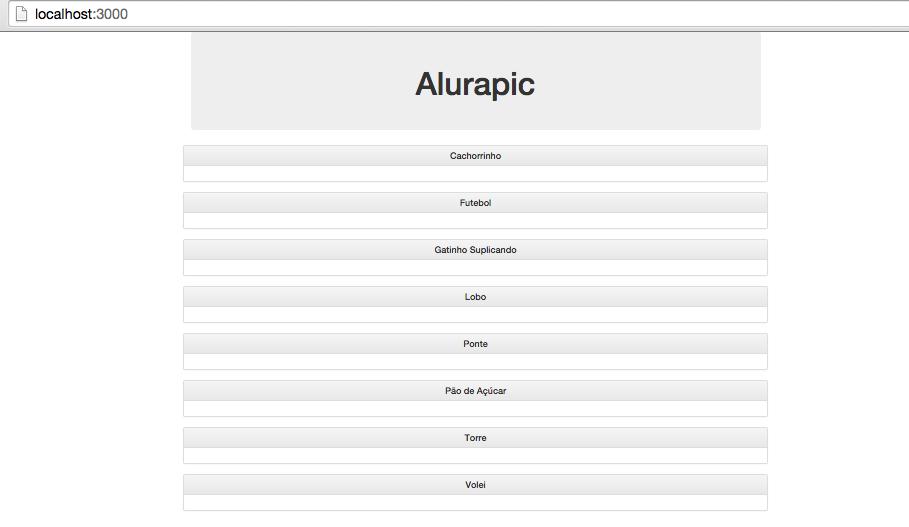
</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

Veja que adicionamos em nossa diretiva também a diretiva ng-repeat, porque queremos repetir nossa diretiva de acordo com a quantidade de fotos. E o resultado final fica:



**Resultado não esperado? Entendendo um pouco mais de diretivas**

Ops! Funcionou parcialmente! Veja que o painel é montado e o título exibido, mas parece que nossa diretiva ignorou todo o seu conteúdo. Isso acontece porque o Angular altera aquele fragmento do DOM substituindo por um novo que representa a marcação da nossa diretiva, nesse processo, perdemos todos os seus elementos filhos. Para que o Angular preserve o conteúdo original da diretiva, precisamos usar o mecanismo de transclusão.

Apesar do nome intimidador, não se preocupe. Para ativarmos a transclusão basta adicionar no DDO a propriedade transclude com valor true e na marcação da diretiva adicionar a diretiva ng-transclude no elemento que deve preservar seus elementos filhos:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

ddo.transclude = true;

ddo.scope = {

titulo: '@'

};

ddo.template =

'<div class="panel panel-default">'

+ ' <div class="panel-heading">'

+ ' <h3 class="panel-title text-center">{{titulo}}</h3> '

+ ' </div>'

+ ' <div class="panel-body" ng-transclude>'

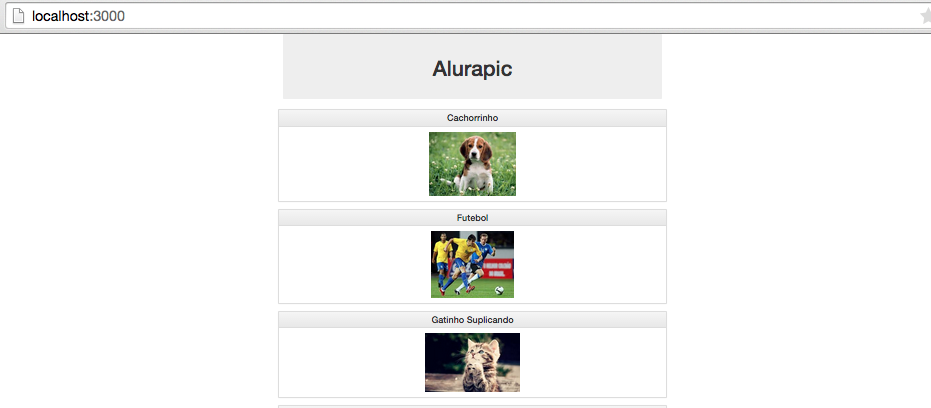
+ ' </div>'

+ '</div>'

return ddo;

});

Agora sim! Vejamos o resultado:



**Separar ainda é a melhor coisa: HTML de um lado e diretiva do outro**

Criamos nossa primeira diretiva! Porém, ficar concatenando HTML dentro da propriedade template não é algo gostoso de se fazer. Que tal colocarmos a marcação dentro de um arquivo HTML, local onde ele pertence? Para isso, vamos criar o arquivo public/js/directives/meu-painel.html com a marcação:

<!-- public/js/directives/meu-painel.html -->

<div class="panel panel-default">

<div class="panel-heading">

<h3 class="panel-title text-center">{{titulo}}</h3>

</div>

<div class="panel-body" ng-transclude>

</div>

</div>

E agora, trocamos em nossa diretiva a propriedade template por **templateUrl**apontando para o arquivo HTML com a marcação da diretiva:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

var ddo = {};

ddo.restrict = "AE";

ddo.transclude = true;

ddo.scope = {

titulo: '@'

};

ddo.templateUrl = 'js/directives/meu-painel.html';

return ddo;

});

O resultado tem que continuar o mesmo, pois alteramos apenas a maneira pela qual organizamos nosso código.

**O que aprendemos neste capítulo?**

* diretivas são componentes reutilizáveis
* a criar nossas própria diretivas
* diretivas reutilizáveis devem possuir escopo isolado
* mecanismo de transclusão

Oi Aluno,

No próximo video tem uma pequena falha onde o instrutor não mostra uma parte do código.

Para tudo realmente funcionar é preciso adicionar a seguinte linha no arquivo main.js:

angular.module('alurapic', ['minhasDiretivas','ngAnimate']);

Essa linha importar o módulo ngAnimate necessário para trabalhar com animações CSS.

Pedimos desculpas pela falha e desejamos uma boa aula.

Equipe Alura

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/04-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Errata: No vídeo tem uma pequena falha onde o instrutor não mostra como importar o módulo ngAnimate. Segue a linha de código que deve ser adicionado no arquivo main.js:

angular.module('alurapic', ['minhasDiretivas','ngAnimate']);

**A lista pode crescer, e agora?**

Muito bem, evoluímos ainda mais nossa aplicação sob o ponto de vista interno, porém podemos melhorar a experiência do usuário. Por enquanto temos poucas fotos, mas eu tenho certeza que assim que aprendermos a cadastrar novas fotos e agrupá-las, o número será muito, muito maior! Então, que tal já implementarmos um mecanismo de procura que permita o usuário exibir apenas fotos de acordo com algum critério?

**Que tal filtrarmos nossa lista?**

Primeiro, vamos adicionar o campo de entrada da pesquisa. Vamos aproveitar a adicionar mais uma linha usando o sistema de GRID do Bootstrap, inclusive utilizaremos algumas classes especiais para estilizar elementos de entrada:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2" ng-repeat="foto in fotos" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

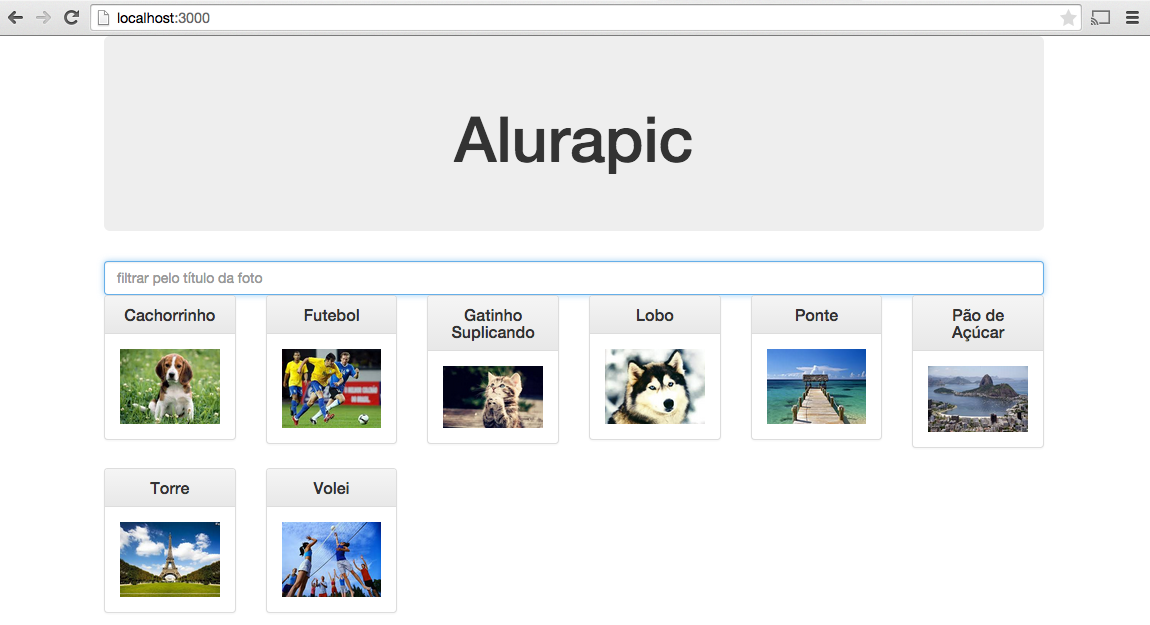
</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>



A ideia é simples: a medida que formos digitando, queremos exibir apenas as fotos que contenham em qualquer uma de suas propriedades o texto procurado.

Vamos adicionar em FotosController uma propriedade que guardará o texto digitado pelo usuário:

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$scope.filtro = '';

$http.get('/v1/fotos')

.success(function(retorno) {

$scope.fotos = retorno;

})

.error(function(erro) {

console.log(erro)

});

});

**Ah, então esse é o two-way data binding?**

Já sabemos acessar qualquer propriedade de $scope através de uma angular expression (AE), porém temos um problema: toda AE é somente leitura, isto é, não é capaz de atualizar $scope, que é justamente o que precisamos. É com base no que o usuário digitar que elaboraremos nossa estratégia de busca. Não queremos um data bindingunidirecional, queremos um bidirecional, aquele que é capaz de ler de $scope, inclusive atualizar seu valor de acordo com a entrada do usuário.

Felizmente o Angular suporta two-way data binding. A diferença é que não usamos AE, mas a diretiva **ng-model**. Vamos adicioná-la no input que recebe o filtro do usuário:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2" ng-repeat="foto in fotos" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

**Eu não sabia que podemos filtrar com ng-repeat!**

E agora? Bem, toda vez que o usuário digitar neste campo, a propriedade $scope.filtroserá atualizada! Excelente, mas como utilizaremos o valor corrente de $scope.filtropara filtrar a lista de fotos? A diretiva ng-repeat aceita receber um filtro através da propriedade filter, que deve ser adicionada imediatamente após um pipe |, sendo assim, ela ficará assim: **ng-repeat="foto in fotos | filter: filtro"**

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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">

<script src="js/lib/angular.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

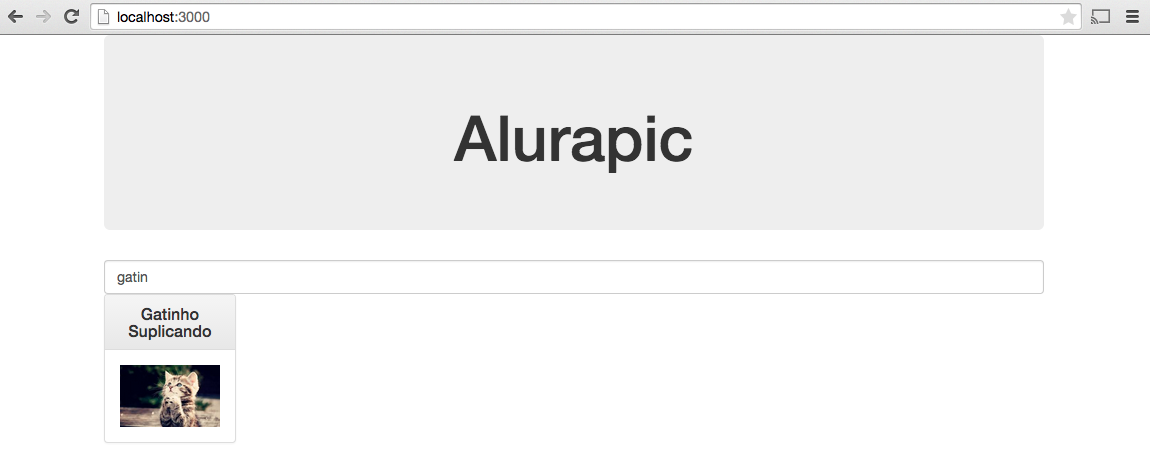
</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>



Inacreditável! Com pouquíssimo esforço, filtramos nossa lista, porém vale ressaltar que o filtro é aplicado em todas as propriedade do objeto foto de nossa lista, sendo assim, podemos encontrar uma foto pelo seu título, descrição, etc.

**Deixando o usuário ainda mais feliz animando nossa lista**

Você há de concordar que melhoramos bastante a aplicação, porém a aplicação do filtro é abrupta, instantânea, imediata! Para deixar o usuário feliz, que tal se animássemos a aplicação do filtro? Podemos fazer isso de várias maneiras, por exemplo, aplicado um efeito fadeout nas fotos que não atenderem nosso filtro.

Normalmente, esta é uma tarefa do mundo CSS, sendo assim, precisamos criar uma classe com o efeito desejado e aplicar essa classe via JavaScript para os elementos que deixarem nossa lista. Vamos criá-la, mas dentro do arquivo public/css/efeitos.css:

/\* public/css/efeitos.css\*/

.fade {

-moz-transform: scale(0.1);

-webkit-transform: scale(0.1);

-ms-transform: scale(0.1);

-o-transform: scale(0.1);

transform: scale(0.1);

}

Agora, vamos importar o CSS em index.html e aplicar a classe fade em nosso painel com a diretiva ng-repeat:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2 fade" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

E testar logo em seguida:



Hum, não funcionou como esperado. Os elementos já começam invisíveis. A ideia é aplicar a classe painel-animado apenas no elemento que **sair (leave)** da lista, isto é, para o elemento que não atender o critério do nosso filtro. E agora?

A equipe do Angular criou o módulo **ngAnimate**, que ataca justamente este problema. Antes de entrarmos nos detalhes de seu funcionamento, vamos importar seu script logo após o script *core* do Angular e adicioná-lo como dependência no módulo principal da aplicação

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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>

<!-- importando o módulo ngAnimate -->

<script src="js/lib/angular-animate.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<!-- adicionando a classe painel-animado -->

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

Agora, alterando a main.js:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate']);

Pronto. Quando usamos o módulo ngAnimate, algumas diretivas do Angular passam a adicionar ou remover classes automaticamente sem a nossa ciência. São classes definidas pelo Angular, algo que lembra muito as pseudo classes do CSS3. Por exemplo, quando usamos a diretiva ng-repeat e um elemento sai da lista ele ganha a classe ng-leave, e quando está para sair ng-leave-active. Existem outras classes e nem todas as diretivas suportam as mesmas classes, sendo necessário recorrer à documentação do Angular. Sendo assim, podemos resolver nosso problema aumentado a especificidade do nosso seletor .painel-animado:

/\* public/css/efeitos.css\*/

.painel-animado.ng-leave-active {

-moz-transform: scale(0.1);

-webkit-transform: scale(0.1);

-ms-transform: scale(0.1);

-o-transform: scale(0.1);

transform: scale(0.1);

}

Testando mais uma vez, nossa lista é exibida mas nenhuma animação é efetuada quando digitamos algo no filtro, isto porque não usamos o poderoso transition do CSS. Por mais que estejamos aprendendo Angular, trabalhar com o módulo ngAnimate requer que você conheça os poderosos recursos do CSS3, a única coisa que o Angular faz é aplicar ou remover determinadas classes. Para efetuar a transição fazemos:

/\* public/css/efeitos.css\*/

/\* novidade! Ativa a transição no elemento. Agora, quando o transform for aplicado apenas para os elementos que estão saindo da lista, a escala será animada!\*/

.painel-animado {

-moz-transition:transform 0.8s;

-webkit-transition:transform 0.8s;

-ms-transition:transform 0.8s;

-o-transition:transform 0.8s;

transition:transform 0.8s;

}

.painel-animado.ng-leave-active {

-moz-transform: scale(0.1);

-webkit-transform: scale(0.1);

-ms-transform: scale(0.1);

-o-transform: scale(0.1);

transform: scale(0.1);

}

CUIDADO: não é para dar um espaço entre as classes .painel-animado e .ng-leave-active , caso contrário o seletor se tornará hierárquico e o efeito não vai funcionar. Os dois devem estar grudados! Se quiser saber mais sobre esse tipo de seletor o Alura possui treinamentos de CSS.

Maravilha! Agora nossa transição funciona perfeitamente, ou quase perfeita (para perceber, é necessário ver o vídeo, não é possível através da foto ilustrar o problema a seguir)!

**O atraso pode ser a melhor opção!**

**IMPORTANTE:** Foi cortado o trecho do vídeo onde é explicado a necessidade de importar o ngAnimate no index.html, antes de continuar faça o import. usando:<script src="js/lib/angular-animate.min.js"></script>

Repare que nossa lista é filtrada a cada tecla que pressionamos e os elementos que não condizem com nosso filtro são animados e retirados da lista, mas a coisa é feita tão instantaneamente que o efeito deixa um pouco a desejar. Para resolvermos isso, precisamos realizar um pequeno atraso (delay) na entrada do usuário. Por exemplo, a cada dígito do usuário, vamos aguardar meio segundo (500ms) para que o framework atualize o valor de $scope.filtro e filtre nossa lista. Fazemos isso através da diretiva **ng-model-options**:

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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>

<!-- importando o módulo ngAnimate -->

<script src="js/lib/angular-animate.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body ng-controller="FotosController">

<div class="container">

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade, a row com o campo de busca -->

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro" ng-model-options="{ debounce: 500 }">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

</div><!-- fim row -->

</div><!-- fim container -->

</body>

</html>

Agora sim! Há um pequeno atraso na atualização do valor de $scope.filtro, por conseguinte um atraso nos elementos que deixaram a lista, deixando nossa animação mais bacana.

Melhoramos ainda mais a experiência do usuário e com certeza, só temos a ganhar com isso!

**O que aprendemos neste capítulo?**

* a diretiva ng-model e two-way data binding
* aplicação de filtro na diretiva ng-repeat
* ng-model-options e postergação do two-way data binding
* animações com o módulo ngAnimate
* animações requerem conhecimento sólido de CSS3

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/05-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

**Desmistificando Single Page Applications**

Agora que a nossa aplicação ficou mais atraente, permitiremos que o usuário cadastre e edite fotos. Nada mais justo do que criarmos uma página de cadastro exclusiva. Porém, o Angular tem como foco a criação de Single Page Applications (Aplicação de única página), aquele tipo de aplicação que não recarrega durante uso. Então, será que a solução é colocar o HTML da página de cadastro e de todas as outras dentro de index.html? Com certeza não, já demos um duro para tornar nossa página mais fácil de manter. Então que diabos é esse de "não recarrega a página durante seu uso"?

Então, realmente a página index.html não é recarregada, é a mesma durante todo o ciclo de vida da aplicação. Porém, é através de URL's especiais chamadas de **rotas** que o Angular busca a página desejada, inclusive ele já faz o trabalho de atualizar index.htmlautomaticamente para nós com o conteúdo da página. É como se index.html fosse um template com uma única lacuna que recebe o conteúdo de outras páginas, mas uma por vez! Essas páginas não devem conter as tags head e body, pois serão inseridas dentro da página principal da aplicação que já os tem. É por isso que são chamadas de **páginas parciais** , ou *partial pages*, no inglês.

**Nossas primeiras views parciais**

Antes de avançarmos, vamos criar a parcial: public/partials/principal.html. Ela conterá a cópia da marcação que lista nossas fotos lá em index.html:

<!-- public/partials/principal.html -->

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="row">

<div class="col-md-12">

<form>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro" ng-model-options="{ debounce: 500 }">

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<img class="img-responsive center-block" src="{{foto.url}}" alt="{{foto.titulo}}">

</meu-painel>

</div>

Então, em index.html vamos substituir esta marcação pela diretiva **ng-view**. Vamos aproveitar e **remover a diretiva ng-controller da tag body**:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/directives/minhas-diretivas.js"></script>

</head>

<body>

<div class="container">

<!-- opa! abri uma mega lacuna! -->

<ng-view></ng-view>

</div><!-- fim container -->

</body>

</html>

**O módulo ngRoute e configurações de rotas**

Você não precisa ser um vidente para saber que, assim que abrirmos o endereço http://localhost:3000, veremos uma página em branco, porém já sabe que precisamos configurar as tais **rotas** do Angular.

Rotas são configuradas com auxílio do módulo **ngRoute**. Como sempre, isso envolve a importação de um script e adição do módulo como dependência do módulo principal da aplicação:

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="pt-br" ng-app="alurapic">

<head>

<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>

<!-- Importando o script do módulo ngRoute -->

<script src="js/lib/angular-route.min.js"></script>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-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>

Agora, em main.js:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute']);

Pronto, agora que temos encaixadas as peças do quebra-cabeças, vamos às configurações! Bem, poderíamos até criar um módulo exclusivo com as configurações das rotas da aplicação, mas não é incomum essa configuração ser feita diretamente no módulo principal da aplicação através do serviço **$routeProvider**, que obtemos através do sistema de injeção de dependências do Angular, dentro da função **config**:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider) {

});

Usamos $routeProvider da seguinte maneira. Quando (**when**) o usuário acessar determinada rota, enviaremos para ele uma view parcial (**templateUrl**) e também indicamos qual é o controlador (**controller**) para aquela parcial. Definir o controller nos dá flexibilidade de usar uma mesma parcial com controllers diferentes:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider) {

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

});

Se abrirmos o endereço http://localhost:3000 nada será exibido, inclusive se tentarmos acessar http://localhost:3000/fotos. O Angular precisa de alguma maneira saber que o endereço que estamos tentando acessar não é um endereço do nosso servidor, mas uma de suas rotas. Fazemos isso adicionando um # (hash) antes da rota:

http://localhost:3000/#/fotos

**Rotas ainda mais elegantes com Html5Mode**

Agora sim! Nossa view parcial principal.html é exibida! Mas um olhar atento revela que essa URL é um tanto estranha, pela presença do #, mas esse é um subterfúgio muito utilizado para que seja possível favoritar em seu navegador a parcial, mas, ainda assim, quando alguém abrir o endereço, quem será carregada é a view principal index.html, que buscará a parcial através do sistema de rotas do Angular. Com o advento do HTML5 foi criada a History API, que permite conseguirmos o mesmo resultado, mas sem o uso do #. O Angular suporta a History API e podemos ativá-la através do serviço **$locationProvider**, que também podemos pedir através do sistema de injeção de dependências:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider, $locationProvider) {

$locationProvider.html5Mode(true);

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

});

Além dessa configuração, precisamos adicionar a tag <base href="/">:

<!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/main.js"></script>

<script src="js/controllers/fotos-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>

Agora, podemos remover o hash e acessar a URL diretamente:

http://localhost:3000/fotos

Outro ponto importante é que se o seu navegador não suportar este modo, automaticamente o Angular adotará a estratégia com # (hash).

**ATENÇÃO**: para usar html5Mode seu servidor precisa estar preparado. Esta configuração está fora do escopo deste treinamento e você deve consultar a documentação do seu servidor favorito.

Excelente, mas o que acontece se agora acessarmos um endereço que não existe? Nada será exibido, porém podemos indicar uma rota alternativa caso o endereço acessado pelo usuário não exista:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider, $locationProvider) {

$locationProvider.html5Mode(true);

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

$routeProvider.otherwise({redirectTo: '/fotos'});

});

Fantástico! Agora vamos criar a parcial do cadastro de fotos, apenas com um título e sua respectiva rota:

<!-- public/partials/foto.html -->

<h1>Cadastro</h1>

E agora vamos a rota /fotos/new, porém ainda sem definirmos um controller:

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider, $locationProvider) {

$locationProvider.html5Mode(true);

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

// foto no singular!

$routeProvider.when('/fotos/new', {

templateUrl: 'partials/foto.html'

});

$routeProvider.otherwise({redirectTo: '/fotos'});

});

Agora, basta acessarmos:

http://localhost:3000/fotos/new

Agora sim! Temos duas parciais com responsabilidades diferentes.

**O que aprendemos neste capítulo?**

* criação de parciais
* a diretiva ng-view
* o módulo ngRoute
* $routeProvider e configurações de rota
* rota padrão
* hash
* $LocationProvider e html5Mode

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/06-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Não podemos postergar ainda mais, precisamos concluir o cadastro de novas fotos! Sabemos que a rota do Angular localhost:3000/fotos/new exibirá a view, mas não é nada elegante pedir que o usuário digite esse endereço toda vez quando for cadastrar uma foto, não?

## Navegando entre views

Podemos melhorar sua experiência adicionando um link que, ao ser clicado na parcial principal.html, chamará nossa rota que exibirá nossa tela de cadastro.

Vamos editar public/partials/principal.html para adicionarmos um link que terá um visual de botão graças ao Bootstrap:

<!-- public/partials/principal.html -->

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<div class="row">

<div class="col-md-12">

<form>

<!-- Novidade! -->

<div class="input-group">

<span class="input-group-btn">

<a href="/fotos/new" class="btn btn-primary" type="button">

Nova foto

</a>

</span>

<input class="form-control" placeholder="filtrar pelo título da foto" ng-model="filtro" ng-model-options="{ debounce: 500 }">

</div>

<!-- fim novidade! -->

</form>

</div> <!-- fim col-md-12 -->

</div> <!-- fim row -->

<div class="row">

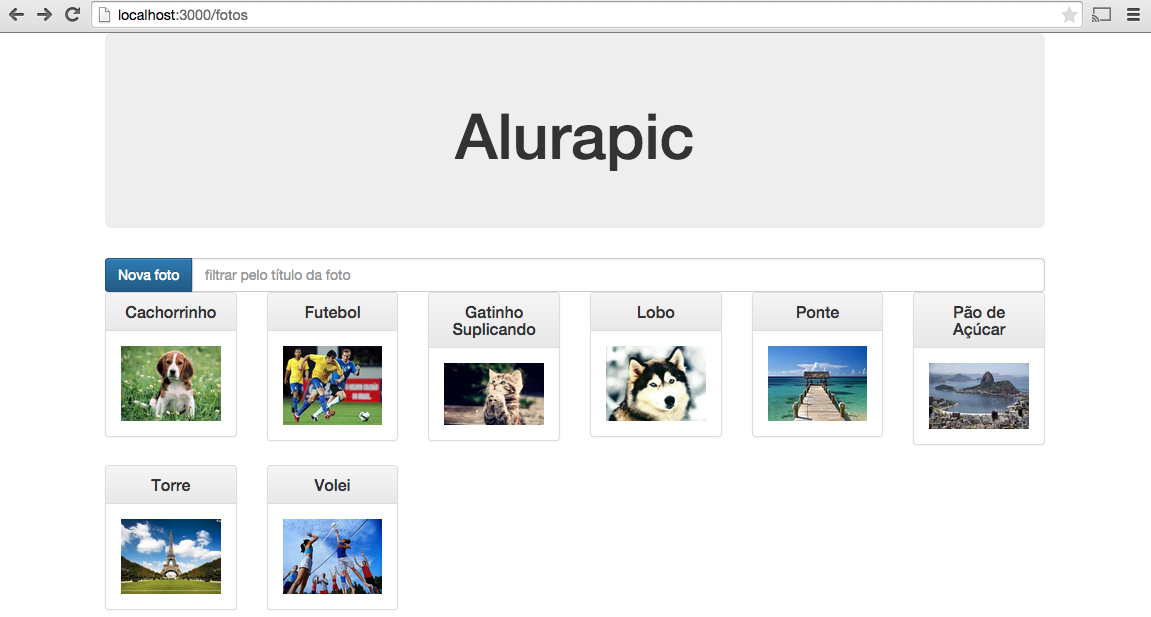
<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

</meu-painel>

</div>



Agrupamos nosso botão e nosso campo de pesquisa dentro de um input-group do Bootstrap, inclusive nosso botão decora o campo de pesquisa com a classe input-group-btn.

Repare também que nosso botão, na verdade um link, aponta para o endereço /fotos/new, justamente a rota que já temos registrada. Agora que já conseguimos navegar entre as parciais principal.html e foto.html, podemos atacar a marcação desta última.

## Nossa primeira view de cadastro

Vamos criar nosso formulário de cadastro, mas ainda sem nos preocuparmos com expressões do Angular, inclusive já vamos adicionar os botões "salvar" e "voltar":

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>TITULO DA FOTO AQUI</h1>

</div>

<form name="formulario" class="row">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control">

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control">

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Excelente, já podemos verificar o resultado!



Queremos agora que cada input do nosso formulário alimente as propriedades de um objeto foto sem qualquer informação. Vamos adicionar a já conhecida diretiva ng-model para cada um dos inputs. No título da página, usaremos uma AE:

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form name="formulario" class="row">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control" ng-model="foto.titulo">

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control" ng-model="foto.url">

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Quando recarregamos nossa página, o título some e o formulário ainda é exibido. Lembre-se que uma AE não avaliada não resulta em erro, mas apenas na ausência de valor no local onde é utilizada. A mesma coisa acontece com a diretiva ng-model.

Agora que já temos o "esqueleto" da nossa tela de cadastro, precisamos de um controller que nos dê suporte para a operação de cadastro. Vamos criar o arquivo public/js/controllers/foto-controller.js e definir o controller **FotoController** (no singular), importando-o logo em seguida na view principal da aplicação public/index.html:

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

angular.module('alurapic')

.controller('FotoController', function($scope) {

$scope.foto = {};

});

<!-- 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>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<!-- importando o novo controller -->

<script src="js/controllers/foto-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>

Veja que não importamos o novo script como último script. Por uma questão de organização apenas, ele foi importado imediatamente após o controller já existente.

Vamos voltar nossa atenção para FotoController. Veja que $scope disponibiliza a propriedade foto, um objeto JavaScript, porém sem qualquer propriedade. Não se preocupe, como estamos usando a diretiva ng-model, a propriedade indicada será criada automaticamente no objeto, sendo assim, se usamos ng-model="foto.titulo", o Angular criará automaticamente em $scope.foto a propriedade titulo, inclusive atribuindo o valor digitado pelo usuário.

## Preparando o terreno para cadastrarmos novas fotos

O que precisamos agora é implementar o botão salvar. O que ele deve fazer? Submeter o formulário, claro, mas precisamos acessar $scope.foto no momento da submissão para que possamos enviar os dados assincronamente através do serviço $http. JavaScript possui o evento **submit** justamente para isso.

O evento submit é disparado quando um formulário é submetido e nele podemos adicionar uma função que permite a execução de um código arbitrário que pode cancelar sua submissão caso haja algum problema, como o de um campo que não foi preenchido. Mas estamos usando Angular, e agora? Como interagir com a interface de eventos do JavaScript?

O Angular suporta a interface de eventos do JavaScript através de diretivas. Por exemplo, se quisermos o evento click, usamos a diretiva ng-click, o evento mouseouver, a diretiva ng-mouseover e assim por diante. Sendo assim, para lidarmos com o evento submit disparado pelo formulário adicionamos a diretiva **ng-submit** diretamente na tag form:

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<form name="formulario" class="row" ng-submit="submeter()">

<!-- código posterior omitido -->

Note que o valor da diretiva ng-submit chama uma função que deve ser definida na propriedade $scope.submeter. Vamos adicioná-la em nosso controller, porém exibiremos apenas os dados da foto no console do navegador:

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

angular.module('alurapic')

.controller('FotoController', function($scope) {

$scope.foto = {};

$scope.submeter = function() {

console.log($scope.foto);

};

});

Se testarmos nosso código, nada acontecerá, por quê? O motivo é simples: não associamos FotoController à view parcial foto.html. Lembram onde realizamos essa associação? Na configuração de rotas! Vamos editar o arquivo public/js/main.js para adicionarmos a chave controller que faltava para a foto /fotos/new:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider, $locationProvider) {

$locationProvider.html5Mode(true);

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

// adicionando a propriedade controller que faltava.

$routeProvider.when('/fotos/new', {

templateUrl: 'partials/foto.html',

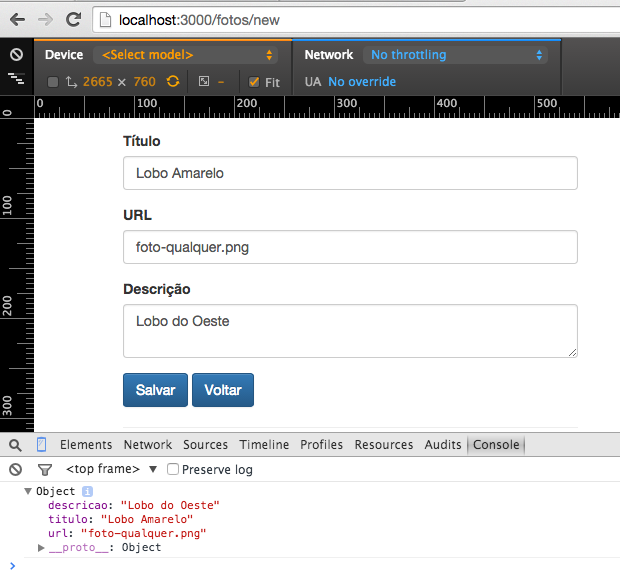
controller: 'FotoController'

});

$routeProvider.otherwise({redirectTo: '/fotos'});

});

Agora já podemos recarregar nossa página, clicar no botão nova foto, preencher alguma informação e clicar no botão salvar. Para vermos os dados da foto, precisamos abrir o console do navegador com F12 (CMD + SHIFT + C, no MAC):



Funciona! Agora só nos resta enviar os dados capturados para uma rota **back-end**especializada nesta tarefa, usando o serviço $http. Porém, não é incomum validarmos os dados do usuário verificando a obrigatoriedade de algum campo ou aplicando alguma regra mais específica de validação. Em nossa aplicação não será diferente e faremos isso através do Angular.

Quando queremos que o Angular tome conta da validação do formulário para nós, precisamos abdicar do sistema de validação do HTML5. Apesar de extremamente funcional, ele não se integra perfeitamente com o Angular e não é tão flexível quanto este último.

Para desabilitar a validação do HTML5, adicionamos o atributo **novalidate** na tag form:

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<!-- código posterior omitido -->

Pronto, agora vamos tornar todos os campos do nosso formulário obrigatórios, exceto o campo de descrição. Fazemos isso adicionando o mesmo atributo que é usado no HTML5, o atributo **required**:

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Desligamos a validação do HTML, mas se deixarmos o título em branco e clicarmos em salvar, já teríamos que receber uma mensagem e não recebemos. Diferente do HTML, que já existe uma mensagem por padrão, o Angular precisa que você defina essa mensagem. A vantagem é que temos a flexibilidade de exibir mensagens de validação da forma que desejarmos.

Vamos adicionar, imediatamente após o campo título do nosso formulário, uma tag span com as classes form-control alert-danger

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required>

<!-- novidade -->

<span class="form-control alert-danger">

Título obrigatório

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Ainda não funciona conforme esperado, porque se recarregarmos a página a mensagem de erro será exibida. Para que funcione, sua exibição deve ser condicional. Algo do tipo se o campo titulo é inválido, exiba a tag span. O Angular possui a diretiva ng-show, que permite a exibição condicional de elementos da tela. Quando seu valor é true, o elemento no qual a diretiva está aplicada é exibida, caso contrário não é exibido.

A questão toda é: quem fornecerá o valor da diretiva ng-show? A resposta mora em um objeto criado implicitamente que representa nosso formulário. Qual o nome deste objeto? Seu nome é o valor do atributo name do formulário, em nosso caso, **formulario**. É através dele que temos acesso a todos os campos do formulário, contanto que cada um deles também tenham definido um valor para o atributo name. Sendo assim, podemos fazer para o campo título:

// apenas exemplo, não entra em nenhum lugar por enquanto

ng-show = "formulario.titulo.$error.required"

Acessamos formulario.titulo.$error, que nos dá acesso à interface de erros do Angular. Como queremos saber o status da validação required, usamos formulario.titulo.$error.required:

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required>

<!-- novidade -->

<span ng-show = "formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

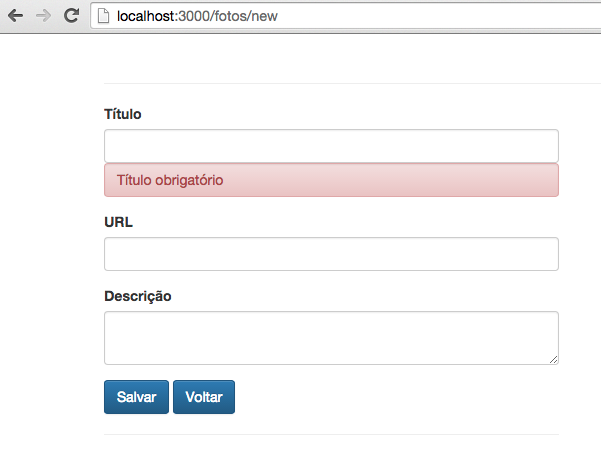
</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>



Hum, nosso formulário já exibe a mensagem de validação assim que é recarregado. Está certo? Depende do que desejamos. Se queremos exibir mensagens de erro sempre que um campo do formulário estiver errado, está certo. Porém, se quisermos validar os campos apenas quando o formulário for submetido, não. Aliás, vamos alterar o valor da diretiva ng-show para exibir a mensagem de erro de validação apenas quando o formulário for submetido. Basta adicionarmos mais uma condição, a formulario.$submitted que retorna verdadeiro apenas se o formulário foi submetido.

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<span ng-show = "formulario.$submitted && formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

<!-- código posterior omitido -->

Veja que agora ng-show só exibirá o elemento se as duas condições forem verdadeiras: o formulário for submetido e o campo inválido. Agora, ao recarregar a página, nosso formulário só será validado quando salvarmos o formulário, ação que disparará sua submissão.

Vamos deixar agora campo URL obrigatório e também preparar a mensagem de erro:

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required>

<span ng-show = "formulario.$submitted && formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

<!-- novidade -->

<span ng-show = "formulario.$submitted && formulario.url.$error.required"

class="form-control alert-danger">

URL obrigatória

</span>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Excelente, mas o Angular permite fazer mais do que simplesmente considerar a obrigatoriedade de um campo. Existem diretivas específicas para validação. Por exemplo, vamos estipular que o campo título não pode passar de 20 caracteres através da diretiva **ng-maxlength**.

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required

ng-maxlength="20">

<span ng-show = "formulario.$submitted && formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

<span ng-show="formulario.$submitted && formulario.titulo.$error.maxlength" class="form-control alert-danger">

No máximo 20 caracteres!

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

<span ng-show = "formulario.$submitted && formulario.url.$error.required"

class="form-control alert-danger">

URL obrigatória

</span>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto></minha-foto>

</div>

</form>

Vamos aproveitar e fazer o data binding da URL da foto com nossa diretiva minha-foto:

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<div class="col-md-6">

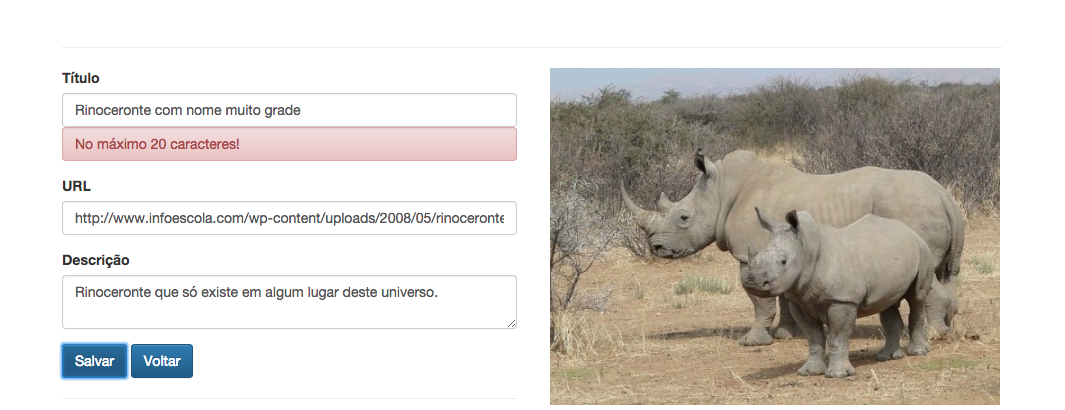
<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

</div>

</form>

Agora vamos testar nossa validação. Vamos aproveitar e digitar uma URL válida, isso fará com que nossa diretiva minha-foto exibe a foto instantaneamente:



Perfeito! Agora já podemos alterar a função $scope.submeter e utilizar o serviço $httppara gravar nosso produto. Como já dizemos, pedimos $http ao serviço de injeção de dependências do Angular. Como queremos enviar os dados, usamos $http.post, que recebe dois parâmetros. O primeiro é a URL do nosso server, /v1/fotos, e como segundo os dados que serão enviados, no caso, $scope.fotos. O restante é igual já fizemos:

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

angular.module('alurapic')

.controller('FotoController', function($scope, $http) {

$scope.foto = {};

$scope.submeter = function() {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

console.log('Foto adicionada com sucesso');

})

.error(function(erro) {

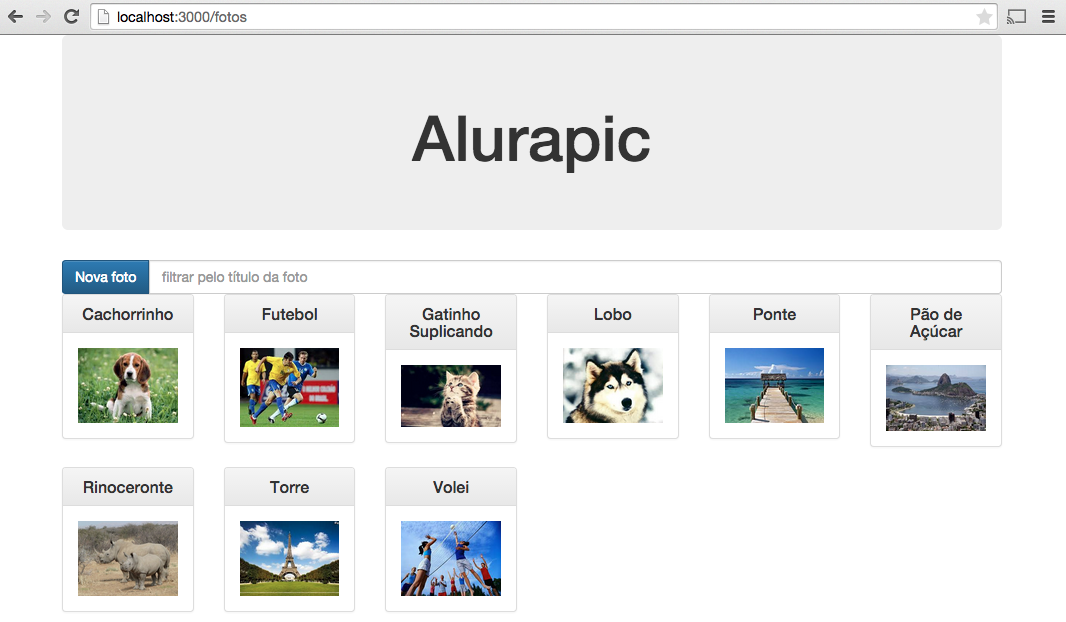
console.log('Não foi possível cadastrar a foto');

})

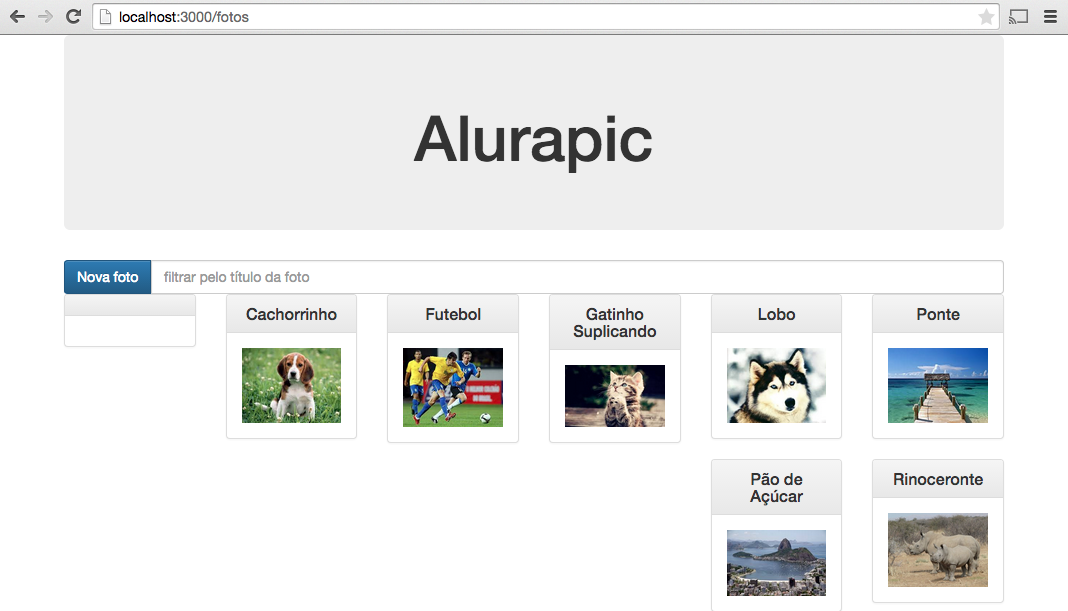
};

});

Já podemos testar. Quando digitarmos dados válidos e clicamos em salvar, nosso formulário continua preenchido e nem sequer recebemos uma mensagem de sucesso! Bom, atacaremos isso em breve, mas se as informações foram enviadas e salvas, basta clicarmos no botão voltar e verificarmos se nossa foto aparece na lista, e realmente aparece!



Muito perfeito! Mas se clicarmos no botão salvar com dados inválidos? Por exemplo, com o tamanho do título sendo maior do que 20 caracteres e deixando o campo URL em branco? O resultado será este:



"Oh my god!", os dados inválidos do nosso formulário foram submetidos e uma foto cadastrada indevidamente. Mas por que o nome ficou em branco? Isso porque dados consideramos inválidos pelo Angular não são aplicados do formulário para o atributo relacionado em $scope. Então, no momento do envio dos dados, $scope.foto.tituloficou vazio.

Para resolver isso, basta consultarmos em nosso $scope o status do formulário com a sintaxe **$scope.formulario.$valid**. Isso mesmo, através de $scope podemos acessar nosso formulário através do seu name e perguntarmos se ele é válido consultando a propriedade $valid.

Alterando nosso FotoController:

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

angular.module('alurapic')

.controller('FotoController', function($scope, $http) {

$scope.foto = {};

$scope.submeter = function() {

if ($scope.formulario.$valid) {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

console.log('Foto adicionada com sucesso');

})

.error(function(erro) {

console.log('Não foi possível cadastra a foto');

})

}

};

});

Agora, nossa lógica de envio das informações só será executada caso o formulário seja válido. Antes de testarmos, podemos melhorar ainda mais a experiência do usuário habilitando a exibição do botão salvar apenas se o formulário estiver válido.

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<button type="submit" class="btn btn-primary" ng-disabled="formulario.$invalid">

Salvar

</button>

<!-- código posterior omitido -->

A cada interação do usuário com nosso formulário, a diretiva ng-disabled consultará o status do formulário para saber se ele é invalido, caso seja, o botão ficará desabilitado:



Agora, para deixar ainda melhor nosso formulário, vamos exibir uma mensagem de fracasso ou sucesso para indicar o status da operação com o servidor. Inclusive vamos limpar os dados do formulário quando a operação for bem sucedida. Vamos adicionar um parágrafo que consultará $scope.mensagem. O parágrafo só será exibido se exibir alguma mensagem:

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<!-- novidade! Aqui será exibida mensagens para o usuário -->

<p ng-show="mensagem.length" class="alert alert-info">{{mensagem}}</p>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required

ng-maxlength="20">

<span ng-show = "formulario.$submitted && formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

<span ng-show="formulario.$submitted && formulario.titulo.$error.maxlength" class="form-control alert-danger">

No máximo 20 caracteres!

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

<span ng-show = "formulario.$submitted && formulario.url.$error.required"

class="form-control alert-danger">

URL obrigatória

</span>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<button type="submit" class="btn btn-primary" ng-disabled="formulario.$invalid">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

</div>

</form>

Agora, alterando nosso controller para popular a $scope.mensagem:

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

angular.module('alurapic')

.controller('FotoController', function($scope, $http) {

$scope.foto = {};

$scope.mensagem = '';

$scope.submeter = function() {

if ($scope.formulario.$valid) {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível cadastrar a foto';

})

}

};

});

Muito bem, agora basta cadastrarmos uma nova foto e verificar a exibição da nossa mensagem:



Quase lá! Precisamos limpar o formulário quando a mensagem for adicionada com sucesso. Basta atribuirmos um objeto vazio à $scope.foto:

angular.module('alurapic')

.controller('FotoController', function($scope, $http) {

$scope.foto = {};

$scope.mensagem = '';

$scope.submeter = function() {

if ($scope.formulario.$valid) {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.foto = {};

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível cadastrar a foto';

})

}

};

});

Agora, para cada foto cadastrada com sucesso, o formulário será limpo automaticamente, tudo por causa do data binding do Angular.

**O que aprendemos neste capítulo?**

* navegação entre views
* a diretiva ng-submit e a interface de eventos do Angular
* executar uma ação no controller através de ações do usuário
* validação de formulário com Angular
* envio de dados para o servidor com $http.post
* mensagens de aviso para o usuário

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/07-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

**Temos dados inconsistentes, precisamos removê-los!**

Quem viu e quem vê nossa aplicação! O único problema é que temos dados inconsistentes cadastrados que precisam ser removidos, aliás, a funcionalidade de remoção é fundamental para o usuário, assim como poder alterar fotos já cadastradas. Vamos atacar primeiro a remoção, mas já aproveitaremos para adicionar o botão de edição.

Vamos adicionar um link para o botão editar e um botão para o botão remover dentro da diretiva meu-painel, abaixo da diretiva minha-foto:

<!-- public/partials/principal.html -->

<!-- código anterior omitido -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

<a class="btn btn-primary btn-block" href="">Editar</a>

<button class="btn btn-danger btn-block" >Remover</button>

</meu-painel>

</div>

Usamos um link para edição porque uma navegação será gerada e usamos botão para remoção porque a ação de remover manterá o usuário na mesma página.

Sabemos que o Angular interage com a interface de eventos do JavaScript através de diretivas. Vamos adicionar a diretiva ng-click no botão "remover" que chamará a função remover em FotosController:

<!-- public/partials/principal.html -->

<!-- código anterior omitido -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

<a class="btn btn-primary btn-block" href="">Editar</a>

<button class="btn btn-danger btn-block" ng-click="remover()">Remover</button>

</meu-painel>

</div>

Agora, vamos implementar em FotosController a função remover:

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$scope.filtro = '';

$http.get('/v1/fotos')

.success(function(retorno) {

$scope.fotos = retorno;

})

.error(function(erro) {

console.log(erro)

});

$scope.remover = function() {

// como saber qual foto será removida?

};

});

Sabemos que o click do botão remover chamará nossa função, mas qual foto deverá ser removida? Aquela que nossa função receber como parâmetro! Voltando para a view principal.htmL, passaremos foto como parâmetro de remover:

<!-- public/partials/principal.html -->

<!-- código anterior omitido -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

<a class="btn btn-primary btn-block" href="">Editar</a>

<!-- novidade aqui! -->

<button class="btn btn-danger btn-block" ng-click="remover(foto)">Remover</button>

</meu-painel>

</div>

Lembre-se que a diretiva ng-repeat constrói nosso template, repetindo o elemento no qual foi adicionada para a quantidade de elementos da lista, e que podemos dar um apelido para termos acesso ao elemento, em nosso caso, usamos foto. Sendo assim, quando o usuário clicar em remover, nossa função receberá o objeto fotocorrespondente! Maravilha! Agora basta alterarmos nosso controller e utilizarmos $http.delete para apagarmos nosso foto no servidor. Já temos uma rota no servidor criada para tal tarefa com a seguinte estrutura:

/v1/fotos/IDdaFotoQueDesejamosApagar

Caso você trabalhasse em outra empresa, o endereço poderia ser diferente, o importante aqui é saber que o desenvolvedor front-end que não cria seu próprio back-end deve estar em sintonia com a equipe de back-end e solicitar os pontos de acesso. Ninguém fez curso "mãe Diná" para saber que endereços são esses!

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$scope.filtro = '';

$http.get('/v1/fotos')

.success(function(retorno) {

$scope.fotos = retorno;

})

.error(function(erro) {

console.log(erro)

});

$scope.remover = function(foto) {

$http.delete('/v1/fotos/' + foto.\_id)

.success(function() {

console.log('Foto ' + foto.titulo + ' removida com sucesso!');

})

.error(function(erro) {

console.log('Não foi possível apagar a foto ' + foto.titulo);

});

};

});

Antes de testarmos, que tal exibirmos também em principal.html uma mensagem de sucesso ou erro de acordo com o resultado da operação? Simples, da mesma maneira que fizemos em foto.html:

<!-- public/partials/principal.html -->

<div class="jumbotron">

<h1 class="text-center">Alurapic</h1>

</div>

<!-- novidade aqui! -->

<p ng-show="mensagem.length" class="alert alert-info">

{{mensagem}}

</p>

<!-- código posterior omitido -->

Agora, vamos alterar nosso controller para exibir as mensagens no lugar de imprimi-las no console. Porém, quando temos uma mensagem de erro, esta sim, exibimos no console:

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$scope.filtro = '';

$scope.mensagem = '';

$http.get('/v1/fotos')

.success(function(retorno) {

$scope.fotos = retorno;

})

.error(function(erro) {

console.log(erro)

});

$scope.remover = function(foto) {

$http.delete('/v1/fotos/' + foto.\_id)

.success(function() {

$scope.mensagem = 'Foto ' + foto.titulo + ' removida com sucesso!';

})

.error(function(erro) {

console.log(erro);

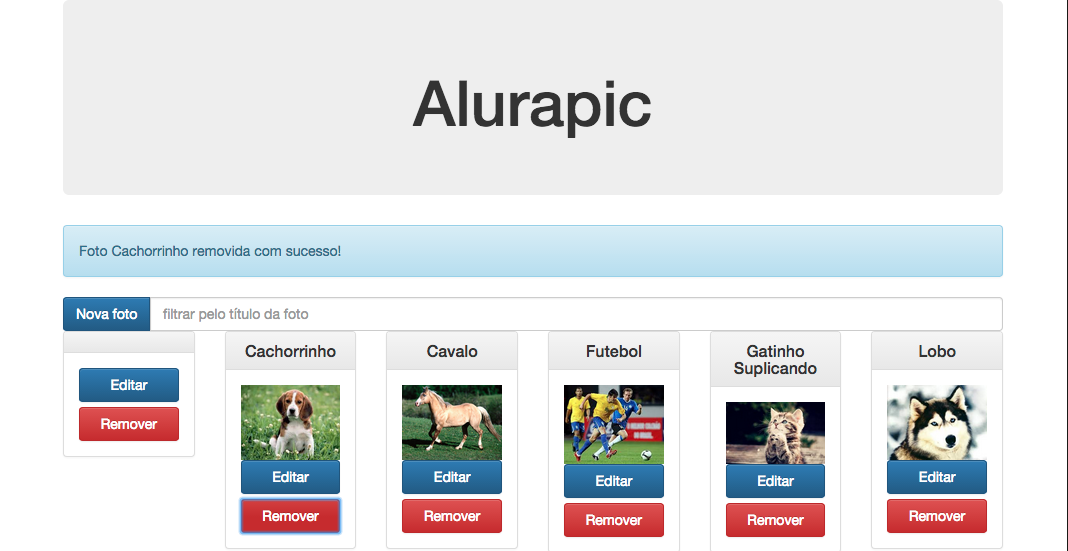
$scope.mensagem = 'Não foi possível apagar a foto ' + foto.titulo;

});

};

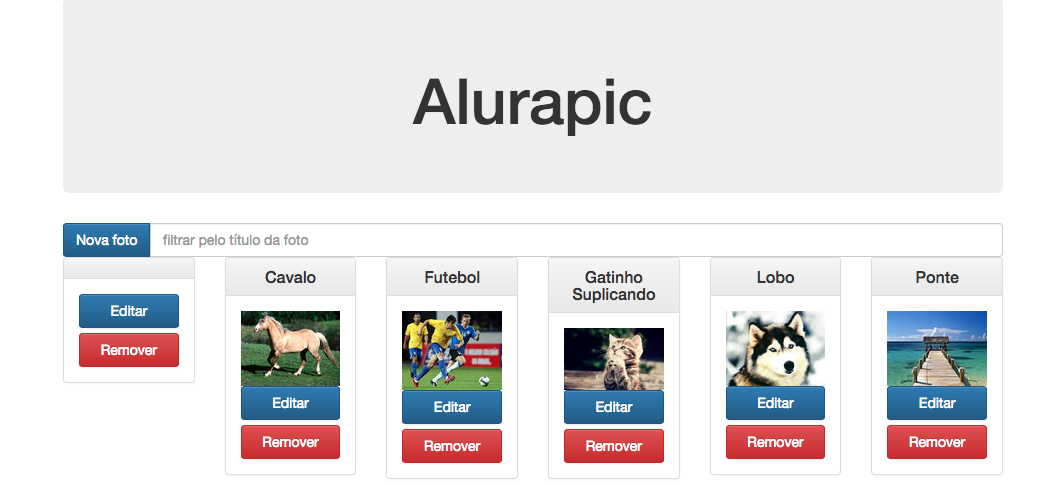
});

E agora? O que falta? Testar!



**Não removeu? Não se preocupe, a solução será performática**

Opa! Recebemos uma mensagem de sucesso dizendo que a foto com o título "Cachorrinho" foi removida, porém ela continua sendo exibida. E se recarregarmos a página?



O problema é que quando removemos a foto do servidor, $scope.fotos ainda contém nossa foto. Podemos resolver isso facilmente solicitando novamente a lista de fotos do servidor, porém estaríamos realizando uma requisição extra e evitar o número de requisições é sempre uma boa ideia, ainda mais se o usuário estiver numa rede móvel de alta latência.

Que tal removermos a foto da lista quando a operação de remoção for bem sucedida? Evitaríamos assim a requisição extra. Como $scope.fotos nada mais é do que uma array, podemos usar a tão conhecida função splice para remover a foto:

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

angular.module('alurapic').controller('FotosController', function($scope, $http) {

$scope.fotos = [];

$scope.filtro = '';

$scope.mensagem = '';

$http.get('/v1/fotos')

.success(function(retorno) {

$scope.fotos = retorno;

})

.error(function(erro) {

console.log(erro)

});

$scope.remover = function(foto) {

$http.delete('/v1/fotos/' + foto.\_id)

.success(function() {

var indiceDaFoto = $scope.fotos.indexOf(foto);

$scope.fotos.splice(indiceDaFoto, 1);

$scope.mensagem = 'Foto ' + foto.titulo + ' removida com sucesso!';

})

.error(function(erro) {

console.log(erro);

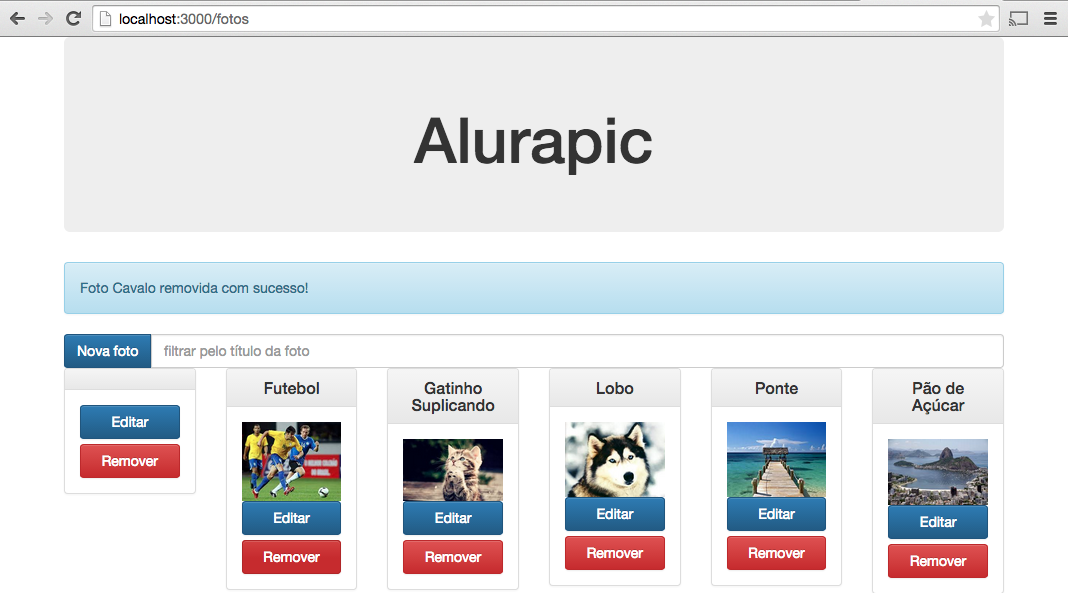
$scope.mensagem = 'Não foi possível apagar a foto ' + foto.titulo;

});

};

});

Agora, vamos remover o cavalo:



Excelente, funcionou, inclusive a remoção foi animada (ngAnimate)! Agora podemos remover os dados inválidos que cadastramos durante nossos testes. Não estranhe se receber a mensagem foto undefined foi removida, porque existem fotos cadastradas sem título! Agora, precisamos atacar a edição fotos.

**Quero editar! Como chegar até minha foto?**

A estratégia de edição será a seguinte: quando o usuário clicar no botão editar, iremos para a parcial foto.html, porém enviaremos o ID da foto como parâmetro. Em FotoController, capturaremos este ID. Quando o ID for passado, buscaremos a foto através deste ID atualizando o $scope.foto com seus dados.

O primeiro passo é registrarmos uma nova rota no Angular que saiba lidar com o ID da foto:

// public/js/main.js

angular.module('alurapic', ['minhasDiretivas', 'ngAnimate', 'ngRoute'])

.config(function($routeProvider, $locationProvider) {

$locationProvider.html5Mode(true);

$routeProvider.when('/fotos', {

templateUrl: 'partials/principal.html',

controller: 'FotosController'

});

$routeProvider.when('/fotos/new', {

templateUrl: 'partials/foto.html',

controller: 'FotoController'

});

// novidade aqui! Nova rota!

$routeProvider.when('/fotos/edit/:fotoId', {

templateUrl: 'partials/foto.html',

controller: 'FotoController'

});

$routeProvider.otherwise({redirectTo: '/fotos'});

});

Repare que registramos nossa nova rota da seguinte maneira:

'/fotos/edit/:fotoId'

No final da rota, usamos o curinga **:fotoId** que serve para duas coisas: indicar que a rota pode aceitar qualquer valor na posição do curinga e para indicar como teremos acesso ao parâmetro passado em nossos controllers. Veja também que estamos utilizando o mesmo controller e mesma view.

**Criando URLs dinamicamente**

Agora que já temos nossa rota cadastrada, vamos alterar o botão editar da nossa view parcial principal.html. Precisamos construir um endereço diferente para cada foto, levando em consideração seu ID. Isso é fácil, basta usarmos uma AE:

<!-- public/partials/principal.html -->

<!-- código anterior omitido -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

<!-- novidade aqui! -->

<a class="btn btn-primary btn-block" href="/fotos/edit/{{foto.\_id}}">

Editar

</a>

<button class="btn btn-danger btn-block"

ng-click="remover(foto)">

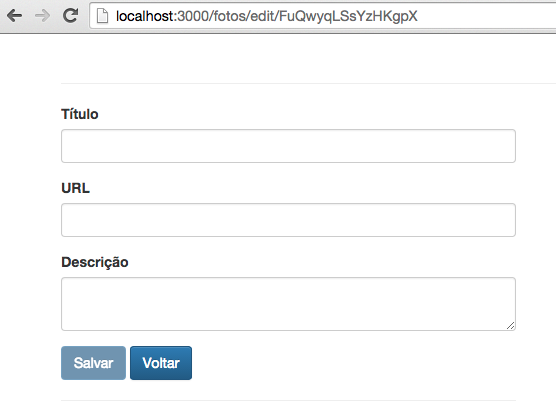
Remover

</button>

</meu-painel>

</div>

Depois de recarregarmos a página para que nossas alterações surtam efeito, quando clicamos em uma das fotos somos jogados para a view parcial foto.html, mas com a diferença de que o ID da foto é exibido como parte da URL:



**Achei! mas onde estão os dados da foto?**

Excelente, mas ainda precisamos ter acesso ao ID da foto em FotoController para que possamos buscá-la em nosso servidor. Existe um serviço especializado do Angular que nos fornecerá este parâmetro, o **$routeParams**. Como qualquer serviço no Angular, ele é recebido como parâmetro na função que define nosso controller:

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

angular.module('alurapic')

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

// código do controller omitido

});

Como utilizar o serviço? Basta sabermos o nome do curinga que usamos em nossa rota e utilizá-lo como propriedade de $routeParams:

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

angular.module('alurapic')

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

$scope.foto = {};

$scope.mensagem = '';

if($routeParams.fotoId) {

// busca a foto no servidor

}

$scope.submeter = function() {

if ($scope.formulario.$valid) {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.foto = {};

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível cadastrar a foto';

})

}

};

});

Se o parâmetro foi passado, buscamos a foto no servidor através de $http.get. O importante é saber que receberemos na função passada para success a foto que buscamos. Mais uma vez o programador front-end precisa conhecer os endereços e os dados retornados, caso ele não tenha sido o responsável pelo back-end:

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

angular.module('alurapic')

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

$scope.foto = {};

$scope.mensagem = '';

if($routeParams.fotoId) {

$http.get('/v1/fotos/' + $routeParams.fotoId)

.success(function(foto) {

$scope.foto = foto;

})

.error(function(erro) {

console.log(erro);

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

});

}

$scope.submeter = function() {

if ($scope.formulario.$valid) {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.foto = {};

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

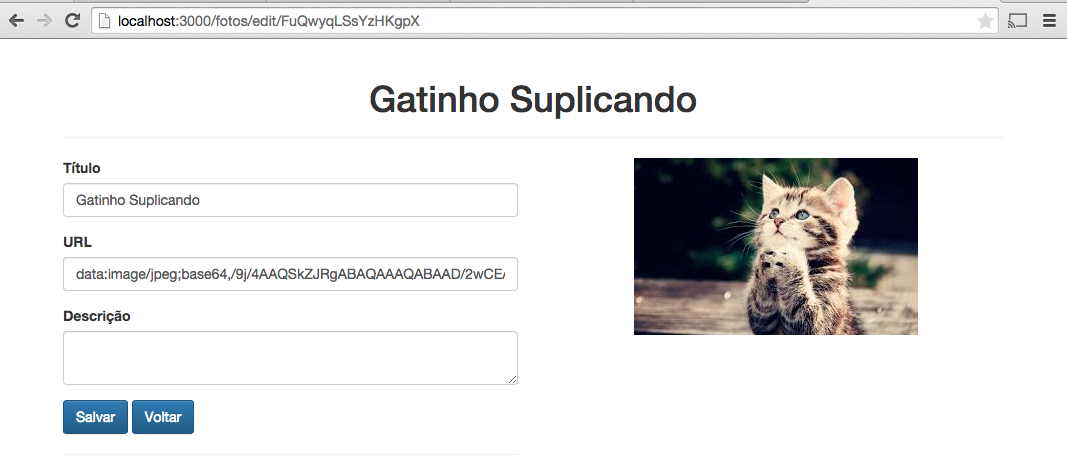
$scope.mensagem = 'Não foi possível cadastrar a foto';

})

}

};

});



Ótimo! Agora, precisamos alterar a lógica de $scope.submeter para que consiga tomar uma ação diferente para inclusão e alteração de fotos. Na alteração, normalmente se usa o verbo HTTP PUT, é por isso que utilizaremos $http.put:

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

angular.module('alurapic')

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

$scope.foto = {};

$scope.mensagem = '';

if($routeParams.fotoId) {

$http.get('/v1/fotos/' + $routeParams.fotoId)

.success(function(foto) {

$scope.foto = foto;

})

.error(function(erro) {

console.log(erro);

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

});

}

$scope.submeter = function() {

if ($scope.formulario.$valid) {

if($routeParams.fotoId) {

$http.put('/v1/fotos/' + $scope.foto.\_id, $scope.foto)

.success(function() {

$scope.mensagem = 'Foto alterada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível alterar';

});

} else {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.foto = {};

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível cadastrar a foto';

})

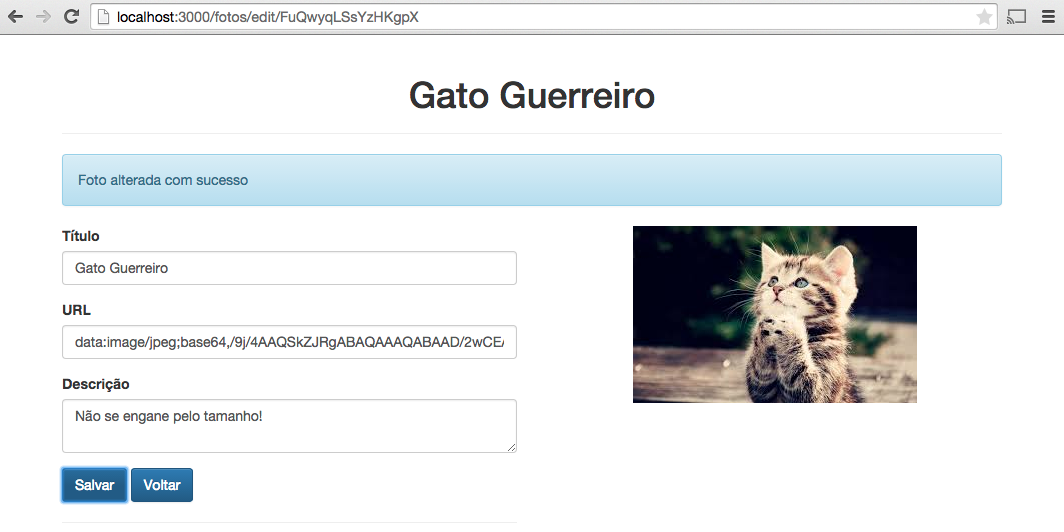
}

}

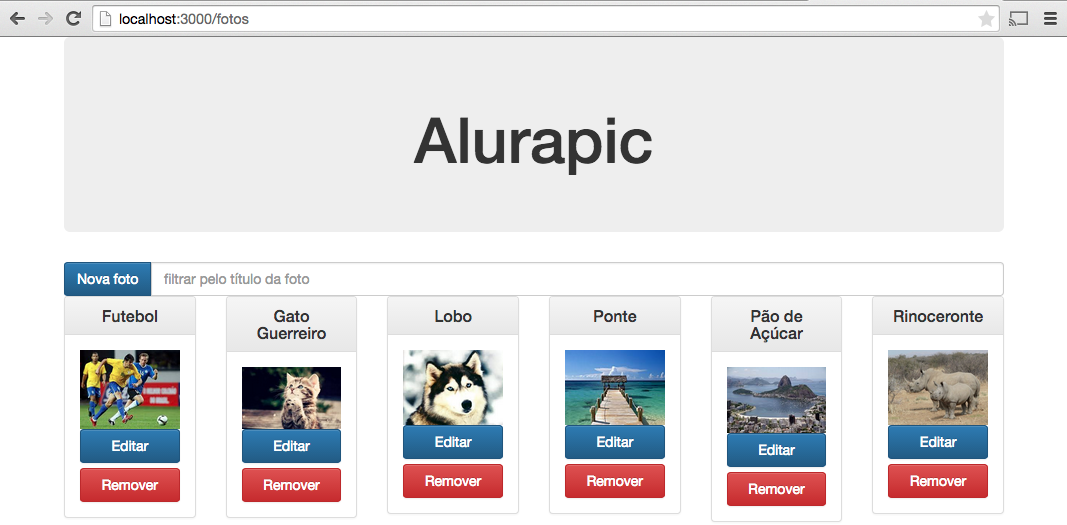
};

});

Agora só nos resta testar. Vamos alterar o nome da foto e colocar uma descrição qualquer. Quando salvamos, recebemos a mensagem de sucesso:



E agora? Se clicarmos em "voltar', nossa parcial principal.html buscará as fotos novamente no servidor, inclusive a que alteramos:



Podemos ainda querer exibir o nome o título da foto que foi alterado. Para isso, basta realizar uma simples concatenação. Nosso código fica assim:

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

angular.module('alurapic')

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

$scope.foto = {};

$scope.mensagem = '';

if($routeParams.fotoId) {

$http.get('/v1/fotos/' + $routeParams.fotoId)

.success(function(foto) {

$scope.foto = foto;

})

.error(function(erro) {

console.log(erro);

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

});

}

$scope.submeter = function() {

if ($scope.formulario.$valid) {

if($routeParams.fotoId) {

$http.put('/v1/fotos/' + $scope.foto.\_id, $scope.foto)

.success(function() {

$scope.mensagem = 'Foto ' + $scope.foto.titulo + ' foi alterada';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível alterar a foto ' + $scope.foto.titulo;

});

} else {

$http.post('/v1/fotos', $scope.foto)

.success(function() {

$scope.foto = {};

$scope.mensagem = 'Foto cadastrada com sucesso';

})

.error(function(erro) {

console.log(erro);

$scope.mensagem = 'Não foi possível cadastrar a foto';

})

}

}

};

});

**O que aprendemos neste capítulo?**

* a diretiva ng-click
* deleção de recurso com $http.delete
* o truque com a função splice para evitar uma nova requisição
* rotas do Angular com curingas (parametrizadas)
* $routeParams e acesso a parâmetros
* alteração de recurso com $http.put
* único formulário para inclusão e alteração

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/08-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Já conseguimos adicionar, alterar e remover fotos, mas há uma informação que também precisamos cadastrar: cada foto pode pertencer a 3 grupos: esportes, lugares e animais. Uma maneira de forçarmos o usuário a escolher um desses grupos é usar uma combobox que, ao ser clicado, exibe todas opções possíveis. No mundo HTML usamos a tag select, onde cada uma das opções é representada pela tag option.

**Nossa primeira combobox!**

No lugar de deixarmos as opções fixas no HTML, montaremos nossa combobox dinamicamente pelo Angular com os dados retornados pelo endereço http://localhost:3000/v1/grupos. Já sabemos obter dados do nosso servidor com o serviço $http.get.

Em primeiro lugar, vamos adicionar a marcação da nossa combobox em foto.html:

<!-- public/partials/foto.html -->

<div class="page-header text-center">

<h1>{{foto.titulo}}</h1>

</div>

<p ng-show="mensagem.length" class="alert alert-info">{{mensagem}}</p>

<form novalidate name="formulario" class="row" ng-submit="submeter()">

<div class="col-md-6">

<div class="form-group">

<label>Título</label>

<input name="titulo" class="form-control"

ng-model="foto.titulo" required

ng-maxlength="20">

<span ng-show = "formulario.$submitted && formulario.titulo.$error.required"

class="form-control alert-danger">

Título obrigatório

</span>

<span ng-show="formulario.$submitted && formulario.titulo.$error.maxlength" class="form-control alert-danger">

No máximo 20 caracteres!

</span>

</div>

<div class="form-group">

<label>URL</label>

<input name="url" class="form-control"

ng-model="foto.url" required>

<span ng-show = "formulario.$submitted && formulario.url.$error.required"

class="form-control alert-danger">

URL obrigatória

</span>

</div>

<div class="form-group">

<label>Descrição</label>

<textarea name="descricao" class="form-control" ng-model="foto.descricao">

</textarea>

</div>

<!-- novidade aqui! -->

<div class="form-group">

<label>Grupo</label>

<select name="grupo" class="form-control" required>

<option value="">Escolha um grupo</option>

</select>

<span ng-show="formulario.$submitted && formulario.grupo.$error.required" class="form-control alert-danger">

Grupo obrigatório

</span>

</div>

<button type="submit" class="btn btn-primary" ng-disabled="formulario.$invalid">

Salvar

</button>

<a href="/" class="btn btn-primary">Voltar</a>

<hr>

</div>

<div class="col-md-6">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

</div>

</form>

Agora, vamos criar um controller que fornecerá única e exclusivamente nossa lista de grupos. Vamos criar o arquivo public/js/controllers/grupos-controller.js:

// public/js/controllers/grupos-controller.js

angular.module('alurapic')

.controller('GruposController', function($scope, $http) {

$scope.grupos = [];

$http.get('/v1/grupos')

.success(function(grupos) {

$scope.grupos = grupos;

})

.error(function(erro) {

console.log(erro);

});

});

Não podemos deixar de importá-lo em index.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>

<script src="js/main.js"></script>

<script src="js/controllers/fotos-controller.js"></script>

<script src="js/controllers/foto-controller.js"></script>

<!-- novo aqui! -->

<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>

Prontinho! Mas como iremos associar GruposController com a tag select? Simples, através da já conhecida diretiva ng-controller. Aproveitaremos também para associar a tag com $scope.foto.grupo através da diretiva ng-model.

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<div class="form-group">

<label>Grupo</label>

<select name="grupo"

ng-model="foto.grupo" class="form-control" required

ng-controller="GruposController">

<option value="">Escolha um grupo</option>

</select>

<span ng-show="formulario.$submitted && formulario.grupo.$error.required" class="form-control alert-danger">

Grupo obrigatório

</span>

</div>

<!-- código posterior omitido -->

Por mais que foto.html seja gerenciada por FotoController, definido no cadastro de rotas do Angular, nossa tag select será gerenciada por GruposController. Fantástico!

**Mais uma diretiva do angular para nos ajudar! A ng-options**

Agora, vamos montar nossas opções através da diretiva *ng-options*:

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<div class="form-group">

<label>Grupo</label>

<select name="grupo"

ng-model="foto.grupo" class="form-control" required

ng-controller="GruposController"

ng-options="grupo.\_id as grupo.nome for grupo in grupos">

<option value="">Escolha um grupo</option>

</select>

<span ng-show="formulario.$submitted && formulario.grupo.$error.required" class="form-control alert-danger">

Grupo obrigatório

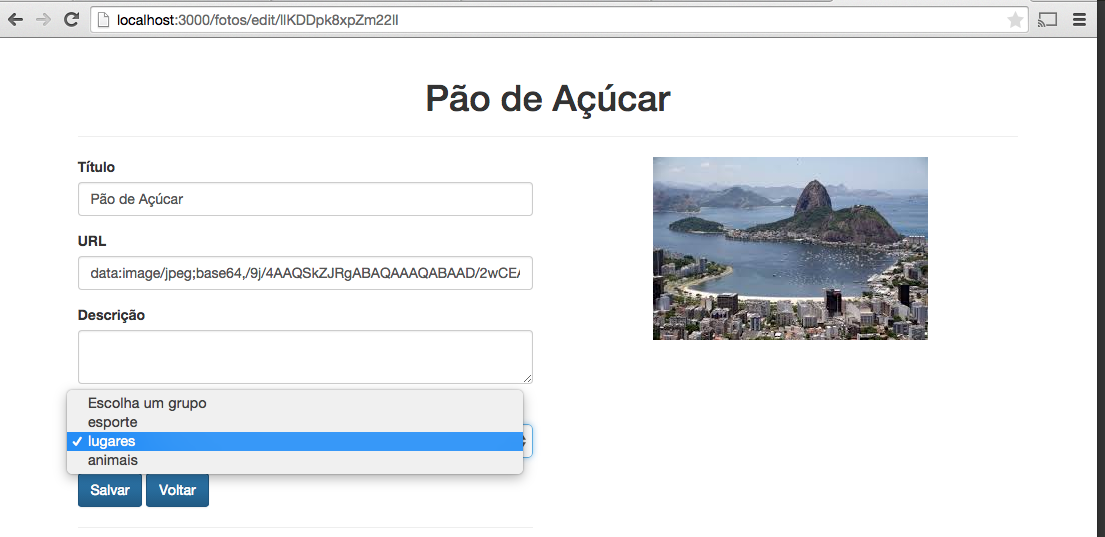
</span>

</div>

<!-- código posterior omitido -->

Adicionamos a diretiva **ng-options**, que possui comportamento parecido com ng-repeat, porém a sintaxe "grupo.\_id as grupo.nome" indica que o valor do elemento será o ID do grupo e o que será exibido para seleção será seu nome. O restante "for grupo in grupos" percorrerá a lista de grupos disponibilizada no escopo do controller, construindo cada item de nossa lista.

Já podemos visualizar o resultado:



Melhor que isso só "dois disso"!

**Filtros que transformam!**

Podemos alterar o grupo, salvar, listar e alterar novamente e verificarmos que o grupo é modificado realmente. Mas podemos ainda melhorar um detalhe: veja que quem cadastrou lá no servidor o nome dos grupos utilizou caixa baixa. Para melhorar a experiência do usuário, seria interessante colocar o nome dos grupos em caixa alta. Será que precisaremos incomodar o fera do back-end para que retorne os dados em caixa alta? Não, o Angular consegue dar conta disso através de filtros. Utilizamos o filtro filter em ng-repeat, agora podemos utilizar o filtro uppercase para colocar o nome do grupo em caixa alta. Vamos alterar a diretiva ng-options:

<!-- public/partials/foto.html -->

<!-- código anterior omitido -->

<div class="form-group">

<label>Grupo</label>

<select name="grupo"

ng-model="foto.grupo" class="form-control" required

ng-controller="GruposController"

ng-options="grupo.\_id as (grupo.nome | uppercase) for grupo in grupos">

<option value="">Escolha um grupo</option>

</select>

<span ng-show="formulario.$submitted && formulario.grupo.$error.required" class="form-control alert-danger">

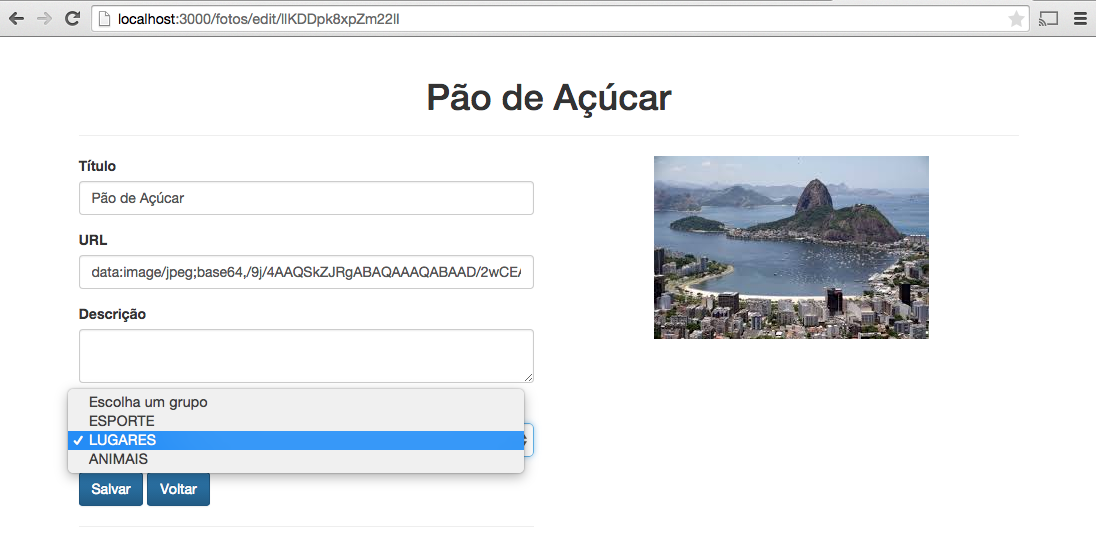
Grupo obrigatório

</span>

</div>

<!-- código posterior omitido -->

Verificando o resultado:



Terminamos, mas vamos fechar esta aula com mais uma diretiva. Topa?

**Facilitando mais uma vez nossa vida com diretivas**

Vamos criar a diretiva meu-botao-perigo. Na verdade, ela é um atalho para a seguinte estrutura que usamos em principal.html:

<button class="btn btn-danger btn-block" ng-click="acao()">{{nome}}</button>

Fica claro que ela terá dois parâmetros: a função que desejamos executar e o nome do botão. Também vamos restringir seu uso à tag:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

// código omitido

})

.directive('minhaFoto', function() {

// código omitido

})

.directive('meuBotaoPerigo', function() {

var ddo = {};

ddo.restrict = "E";

ddo.scope = {

nome: '@',

acao : '@'

}

ddo.template = '<button class="btn btn-danger btn-block" ng-click="acao()">{{nome}}</button>';

return ddo;

});

Excelente, até agora nenhuma novidade. Vamos agora substituir o botão Remover em principal.html pela nossa diretiva:

<!-- public/partials/principal.html -->

<!-- código anterior omitido -->

<div class="row">

<meu-painel class="col-md-2 painel-animado" ng-repeat="foto in fotos | filter: filtro" titulo="{{foto.titulo}}">

<minha-foto url="{{foto.url}}" titulo="{{foto.titulo}}">

</minha-foto>

<a class="btn btn-primary btn-block" href="/fotos/edit/{{foto.\_id}}">

Editar

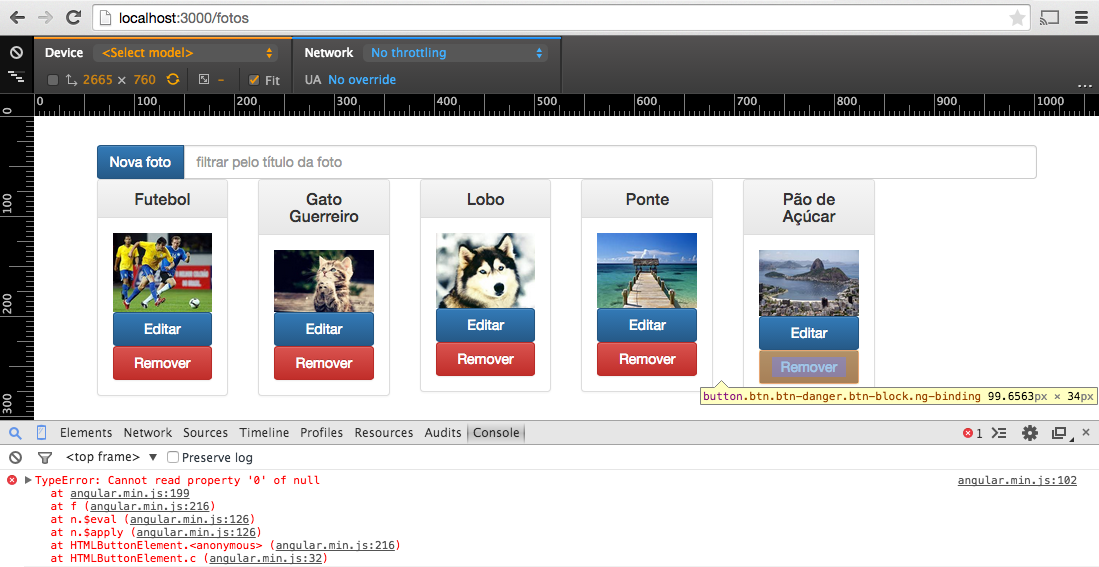
</a>

<meu-botao-perigo nome="Remover" acao="remover(foto)"></meu-botao-perigo>

</meu-painel>

</div>

Será que funciona? Recarregando nossa página o botão é exibido, ufa! Porém, quando clicamos nada acontece. Se abrirmos o console do navegador vemos uma mensagem de erro:



Este problema acontece porque no escopo isolado da nossa diretiva, usamos @ para a propriedade acao. Esse modificador realiza uma cópia do valor passado para a diretiva, guardando-a no escopo isolado como string. Isso não funciona porque o valor de acao é uma expressão que deve ser avaliada no contexto do controller. A diretiva de nada sabe do seu resultado, mas deve ser capaz de executar a **referência** passada como parâmetro. Nem tudo está perdido, pois Angular possui o modificador **&**, que permite fazer binding para uma referência. Alterando nossa diretiva:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

// código omitido

})

.directive('minhaFoto', function() {

// código omitido

})

.directive('meuBotaoPerigo', function() {

var ddo = {};

ddo.restrict = "E";

ddo.scope = {

nome: '@',

acao : '&'

}

ddo.template = '<button class="btn btn-danger btn-block" ng-click="acao()">{{nome}}</button>';

return ddo;

});

Agora sim! Nossa diretiva funciona como esperado!

**O que aprendemos neste capítulo?**

* criar uma combobox com Angular
* o papel da diretiva ng-options
* transformação com filtros (ex. uppercase)
* a diferença entre @ e & em diretivas

**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 ngResource em nossa view index.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 $resourcenão 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 passamos null, 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 módulo meusServicos, 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 dependência ngResource 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 por recursoFoto, 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

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 editar public/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 chamar recursoFoto.update. Na função de sucesso de recursoFoto é que passamos o que desejamos para a função resolve 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

Começando deste ponto? Você pode fazer o [DOWNLOAD](https://s3.amazonaws.com/caelum-online-public/angular-1/stages/11-alurapic.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

**Pedidos de melhorias sempre acontecem**

É bem provável que nossos usuários estejam satisfeitos, mas não é incomum a solicitação de melhorias depois da aplicação pronta e podemos antecipar uma delas! Percebam que após a inclusão ou alteração de uma foto o botão salvar ganha foco. Isso faz todo sentido, porque ele foi o último elemento da página que sofreu interação. Mas se o mesmo usuário quiser alterar uma nova foto em seguida? Com certeza ele terá que clicar no botão voltar para pesquisar a foto na página principal.

Podemos melhorar a experiência do usuário fazendo com que o botão voltar ganhe o foco logo após a operação de inclusão ou alteração ser concluída. Essa pequena mudança permitirá que a tecla ENTER retorne o usuário para a página principal rapidamente. Mas como faremos isso?

**Fazendo as pazes com DOM, mas cada um no seu quadrado!**

Que tal realizarmos manipulação de DOM dentro do nosso controller, por exemplo, usando jQuery? Definitivamente não! Se fizermos isso será muito complicado testar nosso controller com nosso framework de testes favorito, justamente pela dependência do DOM.

Angular permite manipulação de DOM não em nossos controllers, mas em nossas diretivas! Como diretivas possuem escopo isolado, elas podem ser testadas separadamente, sem impactar no código dos controllers. Já sabemos criar o esqueleto de uma diretiva, aliás chamaremos nossa nova diretiva de **meuFocus**.

Editando public/js/directives/minhas-diretivas.js:

// public/js/directives/minhas-diretivas.js

angular.module('minhasDiretivas', [])

.directive('meuPainel', function() {

// código omitido

})

.directive('minhaFoto', function() {

// código omitido

})

.directive('meuBotaoPerigo', function() {

// código omitido

})

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

ddo.scope = {};

return ddo;

});

Bom, criamos o esqueleto da nossa diretiva, mas como será sua API? Bom, ela será usada como atributo no botão. Mesmo sem terminarmos nossa diretiva, vamos adicioná-la no botão voltar:

<!-- public/partials/foto.html -->

<!-- HTML anterior omitido -->

<a href="/" meu-focus focado="focado" class="btn btn-primary">Voltar</a>

<!-- HTML posterior omitido -->

Além da nossa diretiva, adicionamos o atributo focado, que faz parte da API da nossa diretiva. Ele deve apontar para a propriedade $scope.focado do controller. Se for true, o elemento ganha o foco, se for false, ele deixa de ganhar. Sendo assim, o papel de FotoController será atribuir o valor true toda vez que cadastrarmos uma nova foto. Já vamos realizar essa alteração:

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

// código anterior omitido

$scope.submeter = function() {

if ($scope.formulario.$valid) {

cadastroDeFotos.cadastrar($scope.foto)

.then(function(dados) {

$scope.mensagem = dados.mensagem;

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

// novidade aqui!

$scope.focado = true;

})

.catch(function(erro) {

$scope.mensagem = erro.mensagem;

});

}

};

**Diretivas, @&= e seus segredos**

Um ponto importante: depois do nosso botão ganhar o foco, $scope.focado deve receber o valor false, caso contrário nosso botão ficará com foco eternamente! Atribuir este valor false será papel da diretiva. É por esta razão que não podemos usar @ e nem & em seu escopo isolado, modificadores que já usamos em outras diretivas. Usaremos =, que criará uma relação bidirecional, isto é, FotoController e a diretiva meuFocus trabalharão com a mesma referência para $scope.focado. Se focado mudar na diretiva, mudará no controller, se mudar no controller, mudará na diretiva:

// public/js/directives/minhas-diretivas.js

// código anterior omitido

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

ddo.scope = {

focado : '='

};

return ddo;

});

Muito bem, mas isso não resolve o seguinte problema: precisamos ter acesso ao elemento do DOM para que possamos focá-lo em nossa página. Então, toda diretiva processada pelo Angular passa por duas fases: **compile** e **link**. O retorno da fase compile sempre devolve uma função de link, inclusive é apenas nesta função que podemos atribuir observadores que são executados sempre que o valor da propriedade observada mudar, recurso que precisamos. Nela, também temos acesso ao escopo isolado da diretiva e ao elemento do DOM no qual a diretiva foi adicionada.

// public/js/directives/minhas-diretivas.js

// código anterior omitido

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

ddo.scope = {

focado : '='

};

ddo.link = function(scope, element) {

};

return ddo;

});

Excelente, mas não confunda scope da diretiva com $scope. O primeiro é o escopo isolado da diretiva, o segundo é escopo de um controller, tudo bem?

**Conhecendo o oráculo do data binding: $watcher!**

Um ponto que precisamos resolver: só podemos executar o código da diretiva quando focado mudar, ação que será inicialmente empreendida pelo nosso controller. Nossa diretiva só saberá que ela mudou se adicionarmos um observador, na linguagem do Angular, um **watcher**:

// public/js/directives/minhas-diretivas.js

// código anterior omitido

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

ddo.scope = {

focado : '='

};

ddo.link = function(scope, element) {

scope.$watch('focado', function() {

// executado toda vez que o valor mudar

if (scope.focado) {

// se mudou e é verdadeiro, o elemento deve ganhar o foco

}

});

};

return ddo;

});

O scope.$watch recebe dois parâmetros. O primeiro indica qual propriedade do escopo privado da diretiva desejamos observar por mudanças e o segundo a função que será executada sempre que a propriedade mudar. Se scope.focado for true, focaremos nosso elemento:

// public/js/directives/minhas-diretivas.js

// código anterior omitido

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

ddo.scope = {

focado : '='

};

ddo.link = function(scope, element) {

scope.$watch('focado', function() {

if (scope.focado) {

element[0].focus();

scope.focado = false;

}

});

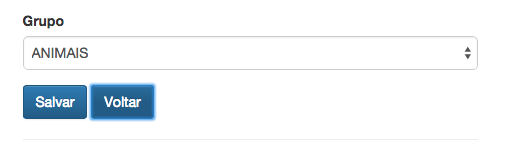
};

return ddo;

});

O element é um elemento DOM, porém encapsulado pelo jqLite. O que é isso? É uma API de manipulação de DOM usada pelo Angular que tenta seguir o padrão da API Sizzle, mesmo padrão usando pelo jQuery, jbMobi ou Zepto. O problema é que o jqLite não possui a função .focus() que o jQuery possui. Neste caso, quando usamos element[0] significa que estamos acessando diretamente o elemento do DOM encapsulado pelo jqLite, sendo assim, podemos chamar a função focus() diretamente no elemento.

Será que funciona? Vamos selecionar uma foto qualquer e clicar em salvar:



Funciona perfeito! O mais bacana desse exemplo é que ele nos mostra de perto como funciona o mecanismo de data binding do Angular.

**Consultar oráculos sai caro, há alternativa?**

Conhecemos o $watcher, o "oráculo" do data binding do Angular, mas não podemos abusar dele em nossa diretivas, porque há um custo computacional nisso, o que pode tornar a atualização de nossa view bem lenta.

Não é o nosso caso, mas sempre que formos criar uma diretiva devemos nos perguntar se há outra forma de conseguirmos a funcionalidade sem o uso de watchers.

**Menor custo com barramento de eventos**

Em nossa diretiva, no lugar de usarmos watchers, trabalharemos com o barramento de eventos (*event bus*) do Angular.

Nosso primeiro passo será editar FotoController e substituir $scope.focado pelo disparo do evento **fotoCadastrada**, através de **$scope.$broadcast**:

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

// código anterior omitido

$scope.submeter = function() {

if ($scope.formulario.$valid) {

cadastroDeFotos.cadastrar($scope.foto)

.then(function(dados) {

$scope.mensagem = dados.mensagem;

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

// novidade aqui!

$scope.$broadcast('fotoCadastrada');

})

.catch(function(erro) {

$scope.mensagem = erro.mensagem;

});

}

};

O nome do evento poderia ser qualquer um, mas deixamos claro que é um evento disparado quando a foto é incluída ou alterada. O $broadcast desce na hierarquia de elementos no escopo do controller, diferente do seu irmão $emit, que sobe na hierarquia.

Agora, vamos mudar nossa diretiva no que diz respeito à sua API. Primeiro, vamos alterar foto.html:

<!-- public/partials/foto.html -->

<!-- HTML anterior omitido -->

<a href="/" meu-focus class="btn btn-primary">Voltar</a>

<!-- HTML posterior omitido -->

Note que agora temos apenas a diretiva meu-focus sem qualquer outro atributo. Simples assim? Vamos alterar nossa diretiva:

// public/js/directives/minhas-diretivas.js

// código anterior omitido

.directive('meuFocus', function() {

var ddo = {};

ddo.restrict = "A";

// não tem mais scope

ddo.link = function(scope, element) {

scope.$on('fotoCadastrada', function() {

element[0].focus();

});

};

return ddo;

});

As mudanças são notáveis: primeiro, não precisamos configurar um escopo isolado. Em seguida, usamos scope.$on para ouvir o evento fotoCadastrada que será disparado pelo nosso controller. Quando o evento for disparado, manipulamos o DOM focando nosso botão. O resultado deve continuar o mesmo: nosso botão voltar ganhará o foco.

Interessante, mas uma pergunta que não quer calar: se alguém esquecer de disparar o evento no controller? E se outras diretivas também estiverem interessadas no evento fotoCadastrada? Parece que um bom local para dispararmos esse evento é dentro do nosso serviço cadastroDeFotos. Não haverá acoplamento do produtor do evento e de quem o consumirá, algo interessante sob o ponto de vista da manutenção do nosso código.

O problema é que dentro de um serviço não temos acesso ao $scope, só controllers têm. Porém, podemos injetar dentro do nosso serviço o $rootScope, escopo pai de todos os escopos para a partir dele dispararmos nosso evento.

Primeiro, vamos remover o disparo do evento em FotoController:

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

// código anterior omitido

$scope.submeter = function() {

if ($scope.formulario.$valid) {

cadastroDeFotos.cadastrar($scope.foto)

.then(function(dados) {

$scope.mensagem = dados.mensagem;

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

// o broadcasting foi removido

})

.catch(function(erro) {

$scope.mensagem = erro.mensagem;

});

}

};

Agora, vamos editar public/js/services/meus-servicos. No serviço cadastroDeFotos, injetaremos $rootScope e a partir dele dispararemos o evento fotoCadastrada quando a foto for incluída ou alterada. Como ele é o pai de todos os escopos dos controllers, os $scope também serão afetados pelo disparo do evento:

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

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

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

'update' : {

method: 'PUT'

}

});

})

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

// novidade

var evento = 'fotoCadastrada';

var service = {};

service.cadastrar = function(foto) {

return $q(function(resolve, reject) {

if(foto.\_id) {

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

// novidade

$rootScope.$broadcast(evento);

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() {

// novidade

$rootScope.$broadcast(evento);

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;

});

Show! Agora é cair dentro dos exercícios!

**O que aprendemos neste capítulo?**

* manipulação de DOM em diretivas
* observar mudanças em propriedades através de $watchers
* o modificador =
* barramento de eventos do Angular

O treinamento contribuiu com uma visão geral sobre diretivas, porém não deixamos de aprender bastante coisa, inclusive truques. Pode ser que tenha passado na sua cabeça a seguinte pergunta: como criar diretivas que busquem dados do servidor e utilizem esses dados? Pergunta justa, não? Muito bem, se você fez todos os exercícios até agora, merece esse bônus, vamos lá!

Vamos criar uma diretiva chamada meusTitulos. Essa diretiva buscará fotos do servidor e montará uma lista com apenas os títulos dessas fotos. Vamos alterar public/js/directives/minhas-diretivas.js:

angular.module('minhasDiretivas', [])

// diretivas anteriores omitidas

.directive('meusTitulos', function() {

var ddo = {};

ddo.restrict = 'E';

ddo.template = '<ul><li ng-repeat="titulo in titulos">{{titulo}}</li></ul>';

return ddo;

});

Até aqui, nenhuma novidade. Precisamos agora elaborar o código que busca as fotos do servidor. Para isso, precisaremos de recursoFoto, mas como? Sabemos que ele é um artefato injetável em controllers em serviços, mas em diretivas? A solução mora na propriedade controller do nosso ddo:

angular.module('minhasDiretivas', [])

// diretivas anteriores comentadas

.directive('meusTitulos', function() {

var ddo = {};

ddo.restrict = 'E';

ddo.template = '<ul><li ng-repeat="titulo in titulos">{{titulo}}</li></ul>';

ddo.controller = function($scope, recursoFoto) {

};

return ddo;

});

A propriedade controller permite passarmos uma função que permite termos acesso aos injetáveis do Angular, como $scope e recursoFoto. Há outros elementos exclusivos que não abordaremos aqui. Você deve estar se perguntando: ok, você me convenceu, mas como recursoFoto foi injetado se não temos o módulo meusServicos como dependência de minhasDiretivas? Resposta elementar caro aluno: nosso módulo principal da aplicação já carrega o módulo meusServicos, inclusive o módulo minhasDiretivas, por isso recursoFoto é injetável. Porém, fica mais bonito declarar explicitamente essa dependência em nosso módulo, sem efeito colateral algum.

Agora, basta buscarmos nossas fotos e adicionarmos o resultado em $scope.titulos. Veja que acessamos esta propriedade através da diretiva ng-repeat do nosso template:

// explicitei a dependência do módulo `meusServicos`

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

// diretivas anteriores comentadas

.directive('meusTitulos', function() {

var ddo = {};

ddo.restrict = 'E';

ddo.template = '<ul><li ng-repeat="titulo in titulos">{{titulo}}</li></ul>';

ddo.controller = function($scope, recursoFoto) {

recursoFoto.query(function(fotos) {

$scope.titulos = fotos; // ainda não é isso que queremos!

});

};

return ddo;

});

Espere um pouco, $scope.titulos está recebendo a lista de fotos, não queremos isso! Queremos é uma lista de títulos. Que tal um pouquinho de JavaScript do "bem" para nos ajudar na tarefa de criar uma nova lista a partir de outra? Vamos usar a função .map:

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

// diretivas anteriores comentadas

.directive('meusTitulos', function() {

var ddo = {};

ddo.restrict = 'E';

ddo.template = '<ul><li ng-repeat="titulo in titulos">{{titulo}}</li></ul>';

ddo.controller = function($scope, recursoFoto) {

recursoFoto.query(function(fotos) {

$scope.titulos = fotos.map(function(foto) {

return foto.titulo;

});

});

};

return ddo;

});

A função map itera sobre nossa lista fornecendo acesso ao elemento da iteração no seu parâmetro. Poderia ser qualquer nome, mas nada mais justo chamarmos de foto, já que estamos iterando sobre uma lista de fotos. Para cada foto retornamos seu titulo, isto é, no final da iteração teremos uma nova lista, mas de títulos apenas.

Muito bem, agora é só utilizarmos nossa diretiva. Para não bagunçar nosso projeto, vamos adicioná-la como último elemento da parcial `principal.html', assim:

<meus-titulos></meus-titulos>

Recarregue a página principal da aplicação e veja o resultado. Uma simples lista com os títulos de nossas fotos deve ser exibida no final da página. Imagine as possibilidades que essa abordagem abre! Como diz o ditado, "grandes poderes trazem grandes responsabilidades".

Sucesso e bom estudo!

É extremamente comum a minificação de scripts para reduzir o tamanho dos arquivos e por conseguinte diminuir o uso de banda por parte do cliente, ainda mais se ele estiver em uma rede móvel como a 3G.

O problema é que o processo de minificação altera o nome dos parâmetros das funções. Não há problema algum nisso, contanto que o novo nome seja trocado em todos os lugares em que é usado, porém o sistema de injeção de dependências do Angular é baseado no nome dos parâmetros. A conclusão disso é que nada mais funcionará no Angular após a minificação, já que os parâmetros das funções serão trocados por outros nomes aleatórios e menores que não tem nada a ver.

Para solucionar este problema, o Angular possui o annotation system, um sistema de anotação que permite dizer o que deve ser injetado para cada parâmetro do controller, mesmo que seu nome seja trocado. Veja a solução:

Este controller :

angular.module('alurapic')

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

// código omitido

});

Vira:

angular.module('alurapic')

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

// código omitido

}]);

Veja que o segundo parâmetro do controller é um array que recebe primeiro todos os artefatos que o controller do Angular receberá e por último a função que define o controller. O processo de minificação jamais tocará nos dados do array e o Angular segue a convenção que o primeiro parâmetro do array será injetado como primeiro parâmetro da função do controller. Se o nome do parâmetro da função do controller muda, tudo continuará funcionando.

A mesma coisa pode ser feita com diretivas e serviços.

Devido ao grande volume de JavaScript carregado nas nossas páginas, as vezes nossa performance e uso de banda ficam prejudicados ao usar um framework js para Single Page Applications (SPA's). Daí a necessidade de usar técnicas de compressão de arquivos como citado no exercício. O pessoal mais ligado à performance web geralmente critica o uso dessas ferramentas, mas algumas técnicas para minimizar esse inconveniente vêm sendo cada vez mais usadas.