01

Tornando nossa aplicação mais real

Transcrição

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:

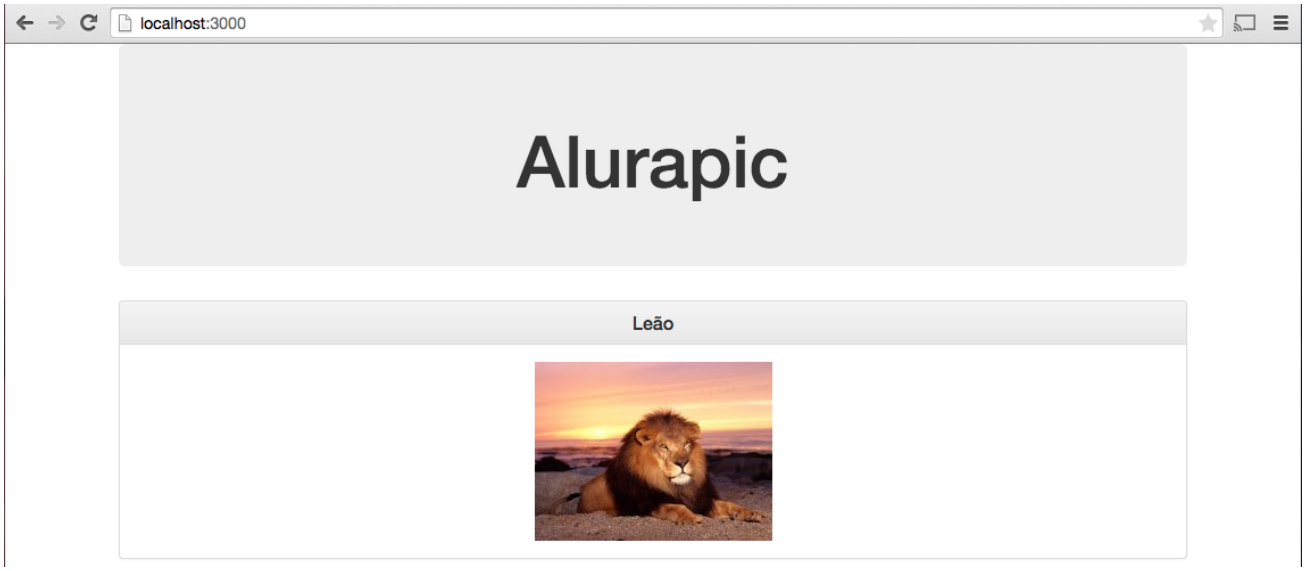
```
<!-- public/index.html -->
<!DOCTYPE html>
<html lang="pt-br" ng-app="alurapic">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
        <title>Alurapic</title>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <link rel="stylesheet" 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.titu
                </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">
```

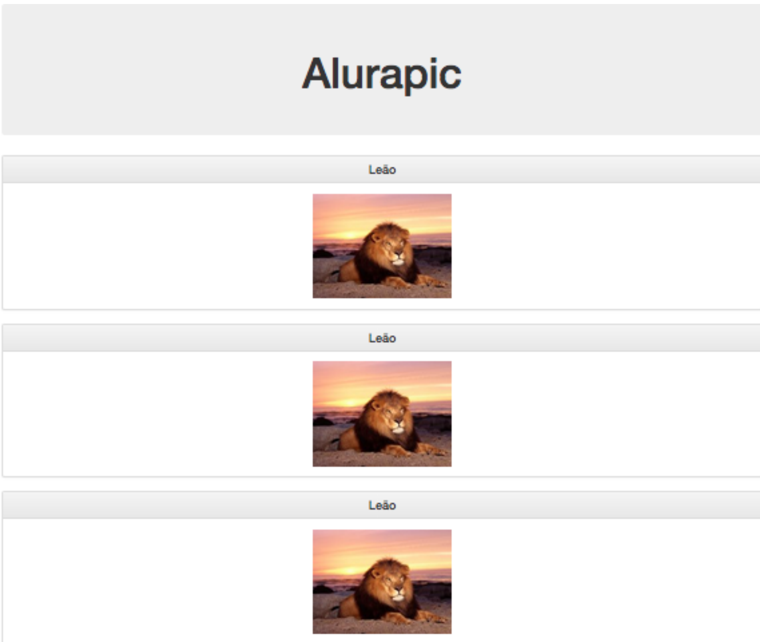
```
                <h1 class="text-center">Alurapic</h1>
            </div>
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h3 class="panel-title text-center">{{foto.titulo}}</h3>
                </div>
                <div class="panel-body">
                    <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:

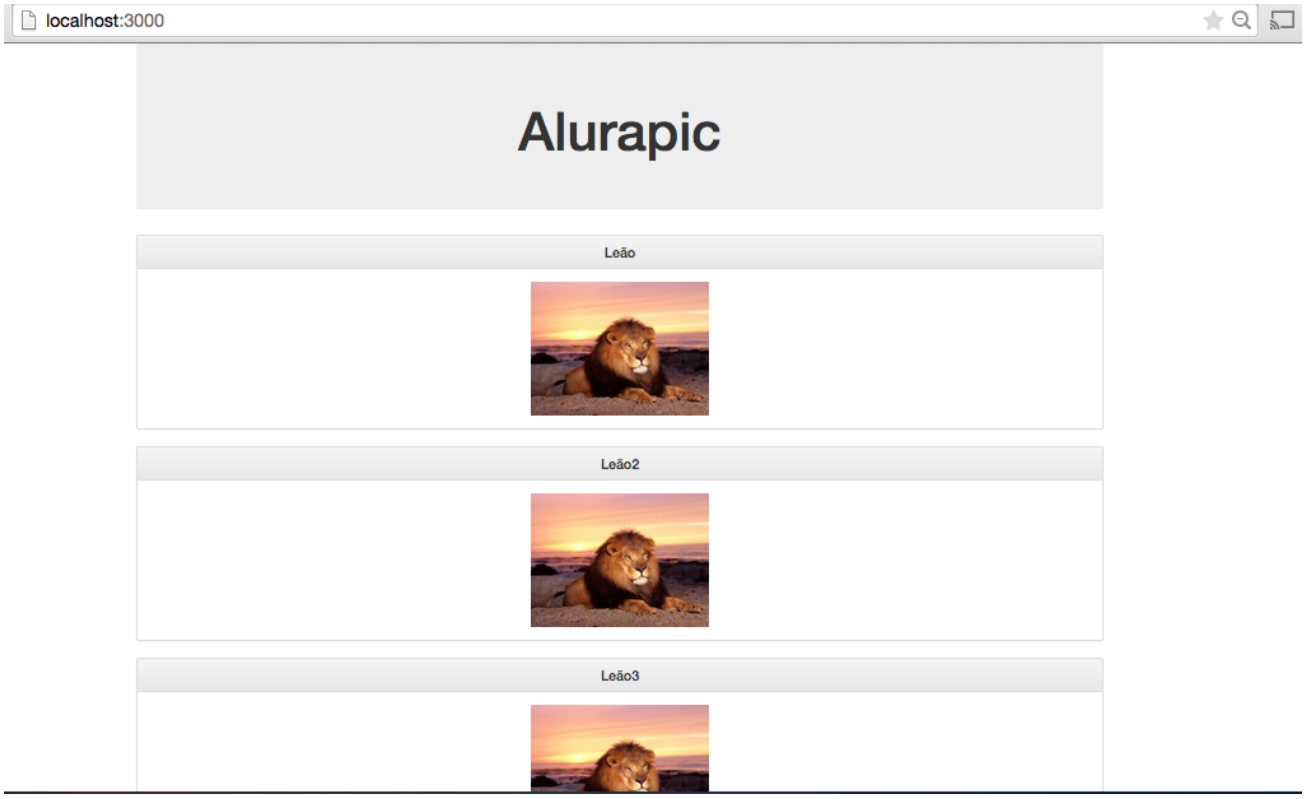
```
<!-- public/index.html -->
<!DOCTYPE html>
<html lang="pt-br" ng-app="alurapic">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
        <title>Alurapic</title>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <link rel="stylesheet" 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.

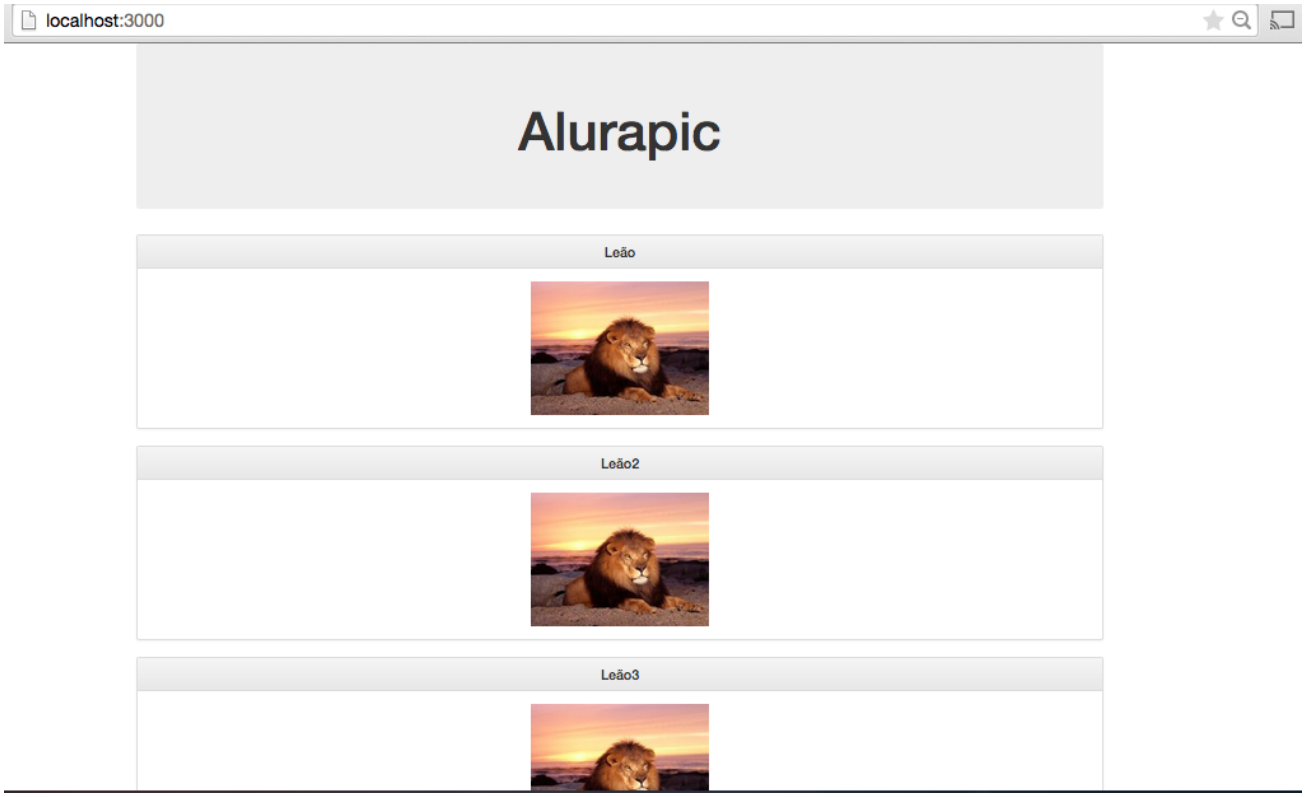
Repetir? É com o Angular mesmo!

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');
});
```

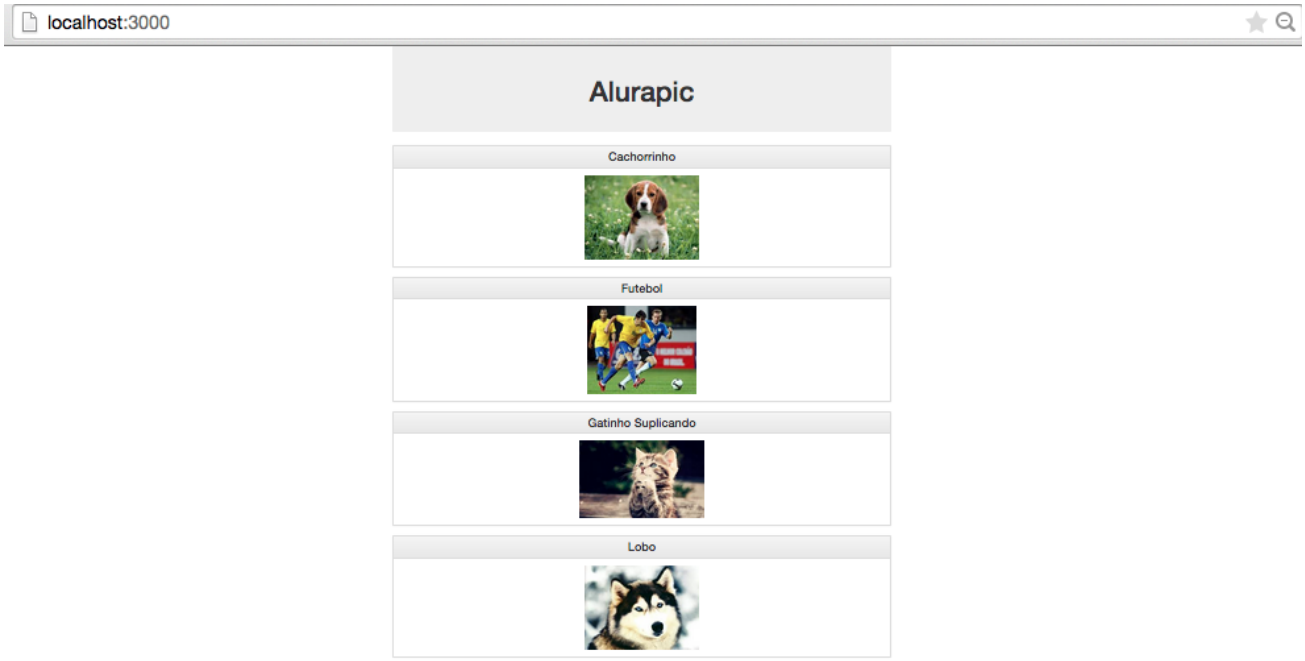
Quando essa promessa for cumprida, daí (then) podemos ter acesso aos dados retornados:

```
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);
    });

});
```

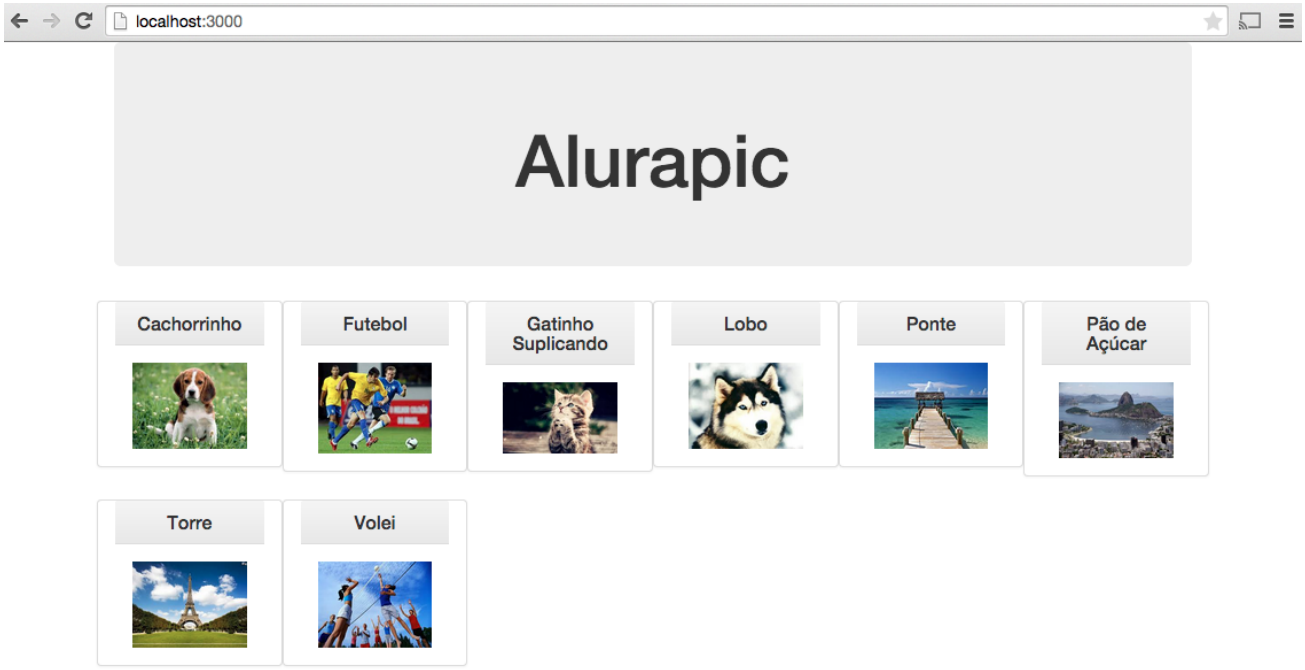
Ainda neste treinamento aprenderemos a cadastrar novas fotos, editar, inclusive apagar aquelas que não nos interessam. Porém, como podemos perceber, a quantidade de fotos aumentou e pode ficar ainda maior. Vamos ajustar nosso layout colocando um grid responsivo do Bootstrap:

```
<!-- 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