Índice

1. ANGULARJS

1.1. Introdução

Este framework é mantido pelo Google e possui algumas particularidades interessantes, que o fazem um framework muito poderoso. Uma dessas particularidades é que ele funciona como uma extensão ao documento HTML, adicionando novos parâmetros e interagindo de forma dinâmica com vários elementos. Ou seja, com AngularJS podemos adicionar novos atributos no html para conseguir adicionar funcionalidades extras, sem a necessidade de programar em javascript. AngularJS é quase uma linguagem declarativa, ou seja, você usa novos parâmetros na linguagem html para alterar o comportamento padrão do html. Estes parâmetros (ou propriedades) são chamados de diretivas, na qual iremos conhecer cada uma ao longo do curso.

Além disso, é fornecido também um conjunto de funcionalidades que tornam o desenvolvimento web muito mais fácil e empolgante, tais como o DataBinding, templates, fácil uso do Ajax, controllers e muito mais. Todas essas funcionalidades serão abordadas ao longo desta obra

1.2. Preparando o ambiente

É preciso muito pouco para começar a aprender AngularJS. Em um nível mais básico, você precisa de um editor de textos e de um navegador web. Mas para otimizar o nosso desenvolvimento, temos algumas ferramentas que nos ajudam bastantes.

1.2.1. Editor de código

Existem vários editores de código, mas aqui no curso recomendaremos o Visual Studio Code (https://code.visualstudio.com/), ele possui facilidades que ajudam bastante no dia a dia de um desenvolvedor.

1.2.2. NodeJS

Com a evolução do javascript nos últimos anos, outra tecnologia ganhou destaque no desenvolvimento web, que é o **Node.js**, no qual iremos chamar simplesmente de node. Node é uma plataforma para executar javascript no lado do servidor, construída sobre o motor Javascript do Google Chrome.

https://nodejs.org/pt-br/

1.2.3. Instalação AngularJS

Para trabalhar com o AngularJS a única coisa que você precisará fazer é importar o seu arquivo JavaScript para o seu projeto, seja através do download direto ou do uso da interface npm.

Trabalharemos com a versão 1.7.8 que é a mais atual do AngularJS.

https://angularjs.org/ - Neste link encontraremos o AngularJS para download, bem como sua documentação.

1.3. Conhecendo o AngularJS

Agora que temos o básico em funcionamento, vamos aprender as principais regras do AngularJS. Através delas será possível realizar diferentes tarefas que irão tornar o desenvolvimento web muito mais simples e prazeroso.

1.3.1. DataBind

Uma das principais vantagens do AngularJS é o seu DataBind. Este termo é compreendido como uma forma de ligar automaticamente uma variável qualquer a uma outra. Geralmente, o DataBind é usado para ligar uma variável do JavaScript (ou um objeto) a algum elemento do documento HTML.

No exemplo a seguir, estamos usando o AngularJS para ligar uma caixa de texto (o elemento input do html) à um cabeçalho.

databind.html

```
<html ng-app>
<meta charset="utf-8" />

<head>
   <title>DataBind</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
</head>
<body>
    Nome: <input type="text" ng-model="nome" />
    <hr/>
    <h1>Olá {{nome}}!</h1>
</body>

</html>
```

Além da propriedade ***ng-app*** (linha 1), utilizamos para DataBind a propriedade ***ng-model***, para informar que este elemento estará ligado a uma variável do AngularJS, através do variável ***nome***, na linha 9. Isso significa que qualquer alteração na caixa de texto irá atualizar o valor da variável. Na linha 11, temos a chamada à variável através do comando ***{{nome}}***, que imprime o valor da variável. Como o DataBind é dinâmico, ao mesmo tempo que algo é escrito na caixa de texto, o seu referido bind é realizado, atualizando instantaneamente o seu valor. Bind também pode ser realizado em objetos, mas antes de começar a aumentar a complexidade do código, vamos criar um controller para melhorar a organização do código.

Normalização

O AngularJS normaliza a tag de um elemento e o nome do atributo para determinar quais elementos correspondem a quais diretivas. Normalmente, nos referimos às diretivas pelo nome normalizado de camelCase que diferencia maiúsculas de minúsculas (por exemplo, ngModel). No entanto, como o HTML não faz distinção entre maiúsculas e minúsculas, nos referimos às diretivas no DOM em minúsculas, geralmente usando atributos delimitados por traços nos elementos DOM (por exemplo, ng-model).

O processo de normalização é o seguinte:

Retire x- e data- da frente do elemento / atributos e substitua por **“:”**, **“-“** ou por **“_”.**

No exemplo abaixo, todos correspondem à diretiva ngBind:

```
<div>
    Olá <input ng-model='nome2'>
    <span ng-bind="nome2"></span> <br/>
    <span ng:bind="nome2"></span> <br/>
    <span ng_bind="nome2"></span> <br/>
    <span data-ng-bind="nome2"></span> <br/>
    <span x-ng-bind="nome2"></span> <br/>
</div>
```

Boas Práticas: *Prefira usar o formato delimitado por traços (por exemplo, ng-bind para ngBind). Se você quiser usar uma ferramenta de validação de HTML, poderá usar a versão com prefixo data- (por exemplo, data-ng-bind para ngBind). Os outros formulários mostrados acima são aceitos por motivos herdados, mas recomendamos que você os evite.*

1.3.2. Controladores

Um controller é, na maioria das vezes, um arquivo JavaScript que contém funcionalidades pertinentes à alguma parte do documento HTML. Não existe uma regra para o controller, como por exemplo ter um controller por arquivo HTML, mas sim uma forma de sintetizar as regras de negócio (funções javascript) em um lugar separado ao documento HTML.

controller.html

```
<html ng-app="app">
<meta charset="utf-8" />

<head>
   <title>DataBind - Controller</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
   <script src="exemplo.controller.js"></script>
</head>

<body ng-controller="exemploController">
    Nome: <input type="text" ng-model="nome" />
    <hr/>
    <h1>Olá {{nome}}!</h1>
    <button ng-click="contarCliques()">
        Clique aqui
    </button> Você clicou no botão "Clique aqui" {{quantideCliques}} vezes
</body>

</html>
```

exemplo-controller.js

```
angular
    .module('app', [])
    .controller('ExemploController', exemploController);

function exemploController($scope) {

    $scope.nome = 'Felipe Augusto';
    $scope.quantideCliques = 0;

    /**
     * @description Adiciona 1 à variável quantidadeCliques
     */
    $scope.contarCliques = function() {

        $scope.quantideCliques = $scope.quantideCliques + 1;
    };
}
```

No controller, criamos a variável ***quantidadeCliques*** e também o método ***contarCliques***, que manipula a variável, de forma que o seu valor é refletido automaticamente na view (html) através do dataBind.

1.3.3. Loop

Outra característica do AngularJS é utilizar templates para que se possa adicionar conteúdo dinâmico. Um loop é sempre realizado através da propriedade ***ng-repeat*** e obedece a uma variável

que geralmente é um array de dados. O exemplo a seguir ilustra este processo, utilizando a tag ***li*** para exibir uma lista qualquer.

loop.html

```
<html ng-app="app">

<head>
   title>Loop</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
   <script src="exemplo.controller.js"></script>
</head>

<body ng-controller="ExemploController">
    <ul>
        <li ng-repeat="fruta in frutas">{{fruta}}</li>
    </ul>
</body>

</html>
```

exemplo-controller.js

```
angular
    .module('app', [])
    .controller('ExemploController', exemploController);

function exemploController($scope) {

    $scope.frutas = ['banana', 'maçã', 'laranja'];
}
```

1.3.4. Formulários

Existem diversas características que um formulário contém, tais como validação, mensagens de erro, formato dos campos, entre outros. Neste caso, usamos o AngularJS de diferentes formas, e usamos vários parâmetros **ng** para controlar todo o processo. O exemplo a seguir exibe apenas algumas dessas propriedades, para que você possa entender como o processo funciona, mas durante o curso iremos verificar todos os detalhes necessários para construir um formulário por completo.

formulário.html

```
<html ng-app="app">
<meta charset="utf-8" />

<head>
   <title>Formulário</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
   <script src="exemplo.controller.js"></script>
</head>

<body ng-controller="ExemploController">
    <form name="meuFormulario">
        <input type="text"
            ng-model="nomeFormulario" name="nomeFormulario" value="Seu nome é..." required/>
        <button ng-disabled="meuFormulario.$invalid" ng-click="salvar()" />Salvar</button>
        <br/>
        <span ng-show="meuFormulario.$invalid">
            Formulário com erros!
        </span>
        <br/>
    </form>
</body>
</html>
```

Neste formulário, usamos mais algumas propriedades, como por exemplo ***ng-show*** que irá exibir ou não a tag contendo a mensagem de erro do formulário, e ***ng-disabled*** que desativa o botão de submissão do formulário. O uso do ***meuFormulario.$invalid*** é um recurso do AngularJS que define se um formulário está inválido ou não. Como usamos uma caixa de texto com a propriedade ***required***, se o campo não estiver preenchido, o formulário ficará inválido.

exemplo.controller.js

```
angular
    .module('app', [])
    .controller('ExemploController', exemploController);

function exemploController($scope, $window) {

    $scope.nomeFormulario = '';

    /**
     * @description Simula salvamento dos dados do formulário
     */
    $scope.salvar = function() {

        $scope.nomeFormulario = '';
        $window.alert('Operação realizada com sucesso!')
    };
}
```

No controlador injetamos além do **$scope**, o serviço do angular **$window** para ter acesso ao objeto window do browser.

1.3.5. Filtros

Os filtros são usados para formatar os dados exibidos para o usuário.

Eles podem ser usados para visualizar modelos, controladores ou serviços. O AngularJS vem com uma coleção de filtros embutidos, mas também é fácil definir os seus próprios.

A sintaxe geral nos modelos é a seguinte:

```
{{ expessao | nome_filtro : parametros }}
```

filter	Seleciona um subconjunto de itens e o retorna como uma nova formatação.
currency	Formata um número como uma moeda (ou seja, R$ 1.234,56). Quando nenhum símbolo de moeda é fornecido, o símbolo padrão para a localidade atual é usado.
number	Formata um número como texto
date	Formata data para uma string baseada no formato informado.
json	Permite converter um objeto JavaScript em string JSON.
lowercase	Converte uma string para que todas as letras fiquem minúsculas.
uppercase	Converte uma string para que todas as letras fiquem maiúsculas.

limitTo	Cria uma nova matriz ou sequência contendo apenas um número especificado de elementos. Os elementos são obtidos do início ou do fim da matriz de origem, sequência ou número, conforme especificado pelo valor e sinal (positivo ou negativo) do limite. Outros objetos do tipo array também são suportados (por exemplo, subclasses de array, NodeLists, coleções jqLite / jQuery etc.). Se um número for usado como entrada, ele será convertido em uma sequência.
orderBy	Returns an array containing the items from the specified `collection`, ordered by a `comparator` function based on the values computed using the `expression` predicate.

1.3.5.1. Criando seu próprio filtro

Criar nosso próprio filtro é muito fácil, só precisamos registrá-lo utilizando a *function* **filter** no módulo do AngularJS. A syntax pode ser vista abaixo. O primeiro argumento é o nome do filtro e o segundo é a *factory function* para o filtro.

filtros.app.js

```
angular.module('filtrosApp', [])
    .filter('cpf', formatarCPF);

function formatarCPF() {
    return function(input) {
        var str = input + '';
        if (str.length <= 11) {
            str = str.replace(/\D/g, '');
            str = str.replace(/(\d{3})(\d)/, "$1.$2");
            str = str.replace(/(\d{3})(\d)/, "$1.$2");
            str = str.replace(/(\d{3})(\d{1,2})$/, "$1-$2");
        }
        return str;
    };
}
```

filtro-controller.js

```
angular
    .module('app', ['filtrosApp'])
    .controller('FiltroController', filtroController);

function filtroController($scope) {

    $scope.cpfDigitado = '02470868513';
}
```

filtro.htlml

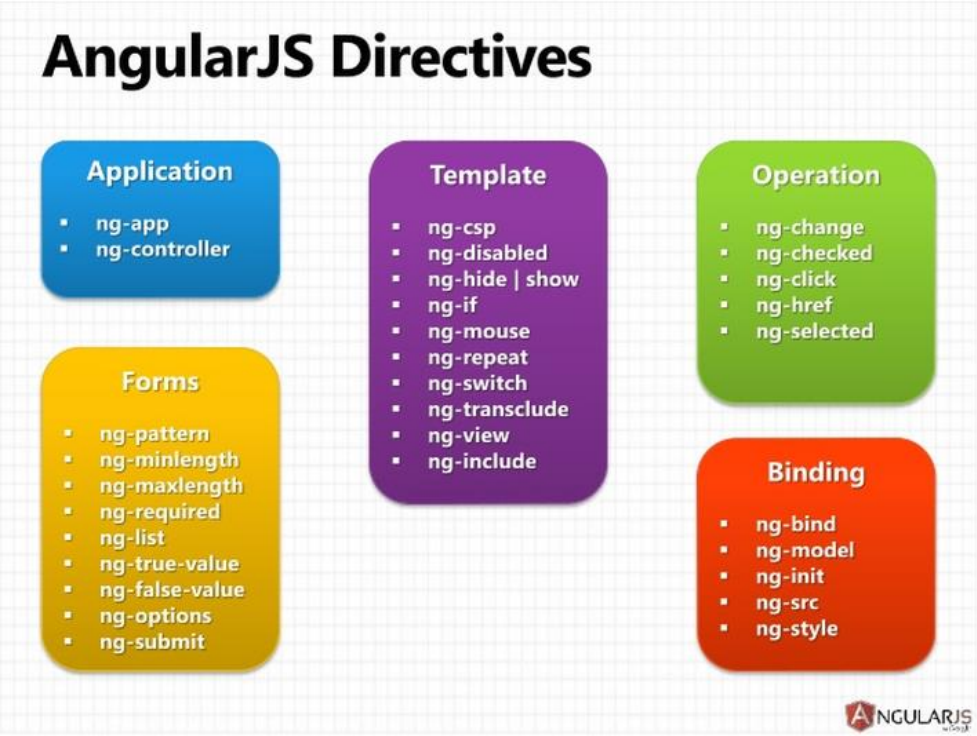
```
<html ng-app="app">
<meta charset="utf-8" />

<head>
    <title>DataBind - Controller</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script
>
    <script src="filtros.js"></script>
    <script src="filtro.controller.js"></script>
</head>
<body ng-controller="FiltroController">
    <hr/> CPF: <input type="text" ng-model="cpfDigitado" />
    <h1>CPF digitado: {{cpfDigitado | cpf}}</h1>
    <hr/>
</body>

</html>
```

1.3.6. Diretivas

Diretivas são extensões da linguagem HTML, que fornecem a possibilidade de estender/ampliar o comportamento de elementos HTML. Este recurso permite a implementação de novos comportamentos de forma declarativa.

Ao selecionar um elemento HTML, a diretiva pode ampliar seu comportamento de diversas formas:

- para adicionar um novo HTML,
- associar eventos à funções Javascript,
- manipular o DOM.



https://docs.angularjs.org/guide/directive

1.3.6.1. Criando sua própria diretiva

diretivas.js

```
angular.module('diretivaApp', [])
    .directive('primeiraDiretiva', primeriaDiretiva);

function primeriaDiretiva() {
    return {
        restrict: 'AECM',
        template: '<p>Fiz minha primeira diretiva!!</p>',
        replace: true // restrict M só funciona com o replace true
    };
}
```

app.js

```
angular
    .module('app', ['diretivaApp']);
```

diretiva.html

```
<html ng-app="app">
<meta charset="utf-8" />

<head>
    <title>Diretivas</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></s
cript>
    <script src="diretivas.app.js"></script>
    <script src="app.js"></script>
</head>

<body>
    <h1>Diretivas</h1>

    <h2> restrict A</h2>
    <span primeira-diretiva></span>

    <h2> restrict E</h2>
    <primeira-diretiva></primeira-diretiva>

    <h2> restrict C</h2>
    <span class="primeira-diretiva"></span>

    <h2> restrict M</h2>
    <!-- directive: primeira-diretiva -->

</body>

</html>
```

1.3.7. Rotas

O AngularJS possui um recurso chamado Deep Linking, que consiste em criar rotas na URI do documento HTML para manipular partes do código HTML de forma independente, podendo assim separar ainda mais as camadas da sua aplicação. No caso mais simples, suponha que exista uma lista de dados que são exibidos em uma tabela, e que ao clicar em um item desta lista, deseja-se exibir um formulário com os dados daquela linha.

O uso de DeepLinking usa Ajax para carregar templates de forma dinâmica, então é necessário que todo o exemplo seja testado em um servidor web.

Nós utilizaremos o http-server do node, para isso devemos instalar de forma global

```
npm install http-server -g
```

Se organizarmos esta pequena aplicação em arquivos, teremos:

app.html

O arquivo principal da aplicação, que contém o código html, o ng-app, a inclusão do AngularJS, entre outras propriedades.

app.js

Contém o código javascript que define o módulo e o comportamento das rotas.

lista.controller.js

Contém o código javascript para a injeção das variáveis declaradas no scope.

list.tpl.html

Contém a tabela que lista os dados.

formulario-edicao.controller.js

Contém o código javascript que define a regra negocial da edição.

formulario-novo.controller.js

Contém o código javascript que define a regra negocial para se cadastrar um novo registro.

formulario.tpl.html

Contém o formulário de edição e criação de um novo registro.

app.html

```
<html ng-app="app">
<meta charset="utf-8" />

<head>
   <title>Rotas</title>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular.min.js"></script>
   <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.7.8/angular-
route.min.js"></script>
    <script src="app.js"></script>
    <script src="formulario-edicao.controller.js"></script>
    <script src="formulario-novo.controller.js"></script>
    <script src="lista.controller.js"></script>
</head>

<body>
    <h1>Exemplo Rotas</h1>
    <div ng-view></div>
</body>

</html>
```

Inicialmente criamos o arquivo app.html, que contém a chamada aos arquivos javascript da aplicação. Além dos arquivos javascript, também usamos a propriedade **ng-app**, que já aprendemos a usar em qualquer aplicação que use o framework.

Também adicionamos um segundo arquivo javascript, que é responsável pelo gerenciamento da rota, chamado de *´angular-route.min.js´.*

Este módulo é criado através da definição de um nome para o ***ng-app***, ficando desta forma: ng-app="app". Assim, estamos criando um módulo chamado *app* que deve estar definido pela aplicação. Como podemos ver, o arquivo app.html não tem nenhum conteúdo, apenas o cabeçalho e uma *div* que possui a propriedade ***ng-view***. Esta propriedade configura o AngularJS para que toda a geração de código seja renderizada dentro desta tag. Esta definição é realizada no arquivo rotas.app.js, cuja parte inicial está descrita a seguir.

app.js

```
angular
    .module('app', ['ngRoute'])
    .config(['$routeProvider', '$locationProvider', definirRotas])
    .run(['$rootScope', inicializarVariaveisGlobais]);

function definirRotas($routeProvider, $locationProvider) {
```

```
    $locationProvider.hashPrefix('');
    $routeProvider.
    when('/', {
        controller: 'ListaController',
        templateUrl: 'lista.tpl.html'
    }).

    when('/edicao/:fruta', {
        controller: 'EdicaoController',
        templateUrl: 'formulario.tpl.html'
    }).
    when('/novo', {
        controller: 'NovoController',
        templateUrl: 'formulario.tpl.html'
    }).
    otherwise({ redirectTo: '/' });
}

/**
 * @description Inicializa variáveis globais no rootScope
 * @param {*} $rootScope Injeção angular
 */
function inicializarVariaveisGlobais($rootScope) {

    $rootScope.frutas = ['banana', 'maçã', 'laranja'];
    console.log(app.run');
}
```

Nesta primeira parte, usamos o método ***angular.module*** para criar um módulo, cujo o nome é *app*. O segundo parâmetro é a referência ao módulo ***ngRoute***, que é usado para criar as rotas (lembre-se que ele deve ser incuído, conforme visto na linha 7 do arquivo app.html).

Após criar o módulo, usamos o método config para configurar o módulo, neste caso estamos configurando uma funcionalidade chamada ***Router(***que possui a função de carregar templates e controllers de acordo com uma URI, ou seja, um endereço repassado pelo navegador) e ***Location*** (define o prefixo padrão das rotas). Na linha Na linha 12 temos a primeira configuração através do método when, que informa ao Router que, ao acessar a raiz do endereço web que o arquivo incio.html está, deve ser carregado o controller *ListaController* e o template lista.tpl.html.

Tanto o template quanto o controller serão carregados no elemento html que contém a propriedade ng-view do app.html.

Na linha 17 adicionamos mais uma rota, e agora configuramos que quando a URI for */edicao/:fruta*, o controller *EdicaoControlle*r e o template *formulário.tpl.html* serão carregados. O atributo *:fruta* será uma variável que poderá ser obtida no controller.

Tanto na linha 17 quanto na linha 22 usamos o mesmo template formulário.tpl.html que contém um formulário para edição ou inserção de um registro.

Na linha 27, configuramos a rota padrão da URI, que é ativada quando nenhuma rota configurada é encontrada.

Voltando para a linha 4 usamos o método ***run*** para configurar a variável ***$scope*** da aplicação, em um contexto global ao módulo. Neste método criamos a variável frutas que possui um contexto global à aplicação.

Criamos três controllers para a aplicação, ListaController, EdicaoController e NovoController.

O primeiro, ListaController, usado apenas para a injeção do $scope.

lista.controller.js

```
angular
    .module('app')
    .controller('ListaController', listaController);

function listaController($scope) {

    console.log('listaController');
}
```

Já o controller EdicaoController possui três parâmetros:

• **scope**: É o escopo da aplicação que pode ser utilizada no template do controller criado.
• **location:** Usada para realizar redirecionamentos entre as rotas
• **routeParams:** São os parâmetros repassados pela URI

formulario.edicao.controller.js

```
angular
    .module('app')
    .controller('EdicaoController', edicaoController);

function edicaoController($scope, $location, $routeParams) {

    // Variáveis Públicas
    $scope.titulo = 'Editar Fruta';
    $scope.fruta = $routeParams.fruta;

    // variáveis privadas
    var _indiceFruta = $scope.frutas.indexOf($scope.fruta);

    /**
     * @description Salva os dados editados da fruta
     */
    $scope.salvar = function() {

        // Atualiza os dados da fruta editada
        $scope.frutas[_indiceFruta] = $scope.fruta;

        // Redireciona para a página principal
        $location.path('/');
    };
}
```

Na linha 8 preenchemos a variável *$scope.titulo*, para que o título do formulário mude, lembrando que o formulário é usado tanto para criar um novo registro quando editá-lo. Na linha 9 pegamos como parâmetro a fruta que foi repassada pela URI. Este valor é pego de acordo com o parâmetro *:fruta* criado pela rota(linha 17 do arquivo rotas.app.js). Na linha 12 obtemos o índice do item que está para ser editado. Usamos isso para poder editar o item no método salvar criado logo a seguir. Na linha 17 temos o método salvar que é usado para atualizar o registro no array global. Em uma aplicação real estaríamos utilizando ajax para que o servidor persistisse o dado. Na linha 25 redirecionamos a aplicação e com isso outro template será carregado.

Criamos também o controller NovoController, que é semelhante ao EdicaoController e possui o método salvar onde um novo registro é inserido no array frutas.

```
angular
    .module('app')
```

```
    .controller('NovoController', novoController);

function novoController($scope, $location, $routeParams) {

    $scope.titulo = 'Nova Fruta';
    $scope.fruta = '';


    /**
     * @description Inclui uma nova fruta no array de frutas
     */
    $scope.salvar = function() {

        // Adiciona um novo item(fruta) no array(frutas)
        $scope.frutas.push($scope.fruta);

        // Redireciona para a página principal
        $location.path('/');
    };
}
```

Vamos agora analisar o arquivo lista.tpl.html que é um template e carregado diretamente pelo roteamento do módulo (app.js, linha 14).

lista.tpl.html

```
<h2>Frutas ({{frutas.length}})</h2>
<ul>
    <li ng-repeat="fruta in frutas">
        <a href="#/edicao/{{fruta}}">{{fruta}}</a>
    </li>
</ul>
<a href="#/novo">Novo</a>
```

O template não necessita informar o seu controller, pois isso já foi feito pelo módulo do AngularJS (rotas.app.js, linha 13). Como a variável frutas possui um escopo global, ela pode ser usada pelo template e na linha 1, exibindo quantos itens existem no array. Na linha 3 iniciamos a repetição dos elementos que pertencem ao array frutas e incluímos na repetição um link para #/edicao/. Esta é a forma com que o roteamento do AngularJS funciona, iniciando com # e repassando a URI logo a seguir. Na linha 7, criamos outro link, para incluir um novo registro. Novamente usamos a URI que será utilizada pelo roteamento do AngularJS.

O último arquivo deste pequeno exemplo é o formulário que irá editar ou inserir um novo registro.

formulário.tpl.html

```
<h2> {{titulo}} </h2>
<form name="meuFormulario">
    <input type="text" ng-model="fruta" name="fruta" required>
    <button ng-click="salvar()" ng-disabled="meuFormulario.$invalid">Salvar</button>
</form>
<a href="#/">Cancelar</a>
```

Na linha 1 usamos o {{titulo}} para inserir um título que é criado pelo controller. O formulário possui apenas um campo cujo ***ng-model*** é fruta que será utilizado pelo controllers de edição e novo. Neste formulário também utilizamos ***ng-disabled*** para que o botão seja ativado somente se houver algum texto digitado na caixa de texto. O botão salvar possui a propriedade ***ng-click***, que irá chamar o método salvar() do controller.

1.4. Promessas

Uma promessa é um método de resolver um valor (ou não) de maneira assíncrona. Promessas são objetos que representa o valor de retorno ou exceção lançada que uma função pode eventualmente fornecer. São incrivelmente úteis para lidar com objetos remotos.

Para criar uma promessa no Angular, podemos usar o serviço $q interno. O serviço $ q fornece alguns métodos. Primeiro, precisamos injetar o serviço $q no objeto em que queremos usá-lo, conforme linha 5 no caso abaixo.

formulario-novo.controller.js

```
angular
    .module('app')
    .controller('NovoController', novoController);

function novoController($scope, $location, $routeParams, $q) {

    $scope.titulo = 'Nova Fruta';
    $scope.fruta = '';

    /**
     * @description Inclui uma nova fruta no array de frutas
     */
    $scope.salvar = function() {

        $scope.mensagem = 'carregando...';
        adicionarFrutaNaLista()
            .then(function() {

                $scope.mensagem = '';
                $location.path('/');
            });

    };

    function adicionarFrutaNaLista() {
        var deferred = $q.defer();

        setTimeout(() => {

            $scope.frutas.push($scope.fruta);
            deferred.resolve();
        }, 1000);

        return deferred.promise;
    }
}
```

Perceba que ao criar a função ***adicionarFrutaNaLista()*** *na liha 25,* ela me devolve uma promessa e não um valor, dessa forma precisamos “esperar” o tratamento da promessa com o comando ***then***(linha 17) para assim continuar com o processamento desejado.

1.5. Conectando AngularJS ao o servidor

Agora que conhecemos um pouco sobre o AngularJS, podemos entender como funciona a sua comunicação com o servidor. Assim como é feito com jQuery e até com javascript puro, a melhor forma de obter e enviar dados para o servidor é através de Ajax e o formato de dados para se usar nesta comunicação é JSON.

Existem diversas formas de conexão entre cliente (neste caso, AngularJS) e servidor, e neste curso estaremos utilizando um conceito chamado RESTful, que é uma comunicação HTTP que segue um padrão bastante simples, utilizando cabeçalhos HTTP como POST, GET, PUT, DELETE.
Na forma mais simples de comunicação de dados, onde temos um objeto e as ações de criar, editar, listar e deletar objetos, resumimos o padrão RESTful às seguintes ações:

Método http://site.com/produtos

- GET Listar todos os produtos
- POST Editar uma lista de produtos
- PUT Criar um novo produto na lista de produtos
- DELETE Excluir uma lista de produtos

No AngularJS a forma mais simples de trabalhar com estas conexões é através do serviço ***$http***, que pode ser injetado em um controller.

1.5.1. Uso do $http

$http é uma implementação ajax através do XMLHttpRequest utilizando JSONP. Iremos sempre usar JSON para troca de dados entre cliente e servidor.

A forma mais simples de uso do $http está descrito no exemplo a seguir:

```
$http({method: 'GET', url: '/algumUrl'});
```

O método get pode ser generalizado para:

```
$http.get('/algumUrl');
```

Assim como existe o get, existem os outros métodos também, conforme a lista a seguir:

- $http.get
- $http.head
- $http.post
- $http.put
- $http.delete
- $http.jsonp

1.6. Services

Até agora, nos preocupamos apenas com a forma como a visualização está vinculada ao $scope e como o controlador gerencia os dados. Para fins de memória e desempenho, os controladores são instanciado somente quando eles são necessários e descartados quando não são. Isso significa que toda vez que trocamos
uma rota ou recarregar uma view, o controlador atual é limpo pelo Angular.
Os serviços fornecem um método para mantermos os dados por toda a vida útil do aplicativo e nos comunicarmos controladores de maneira consistente.

Os serviços são objetos singleton que são instanciados apenas uma vez por aplicativo (pelo injetor $) e carregados com preguiça (criados apenas quando necessário). Eles fornecem uma interface para manter juntos esses métodos relacionados a uma função específica.

$http, por exemplo, é um exemplo de serviço AngularJS. Ele fornece acesso de baixo nível ao objeto XMLHttpRequest do navegador. Em vez de precisar sujar o aplicativo com chamadas de baixo nível para o objeto XMLHttpRequest, podemos simplesmente interagir com a API $http.

Para exemplificar o uso de services criamos o arquivo *fruta.service.js* para acessar a API Rest com todas as funcionalidades de manipulação da base de dados.

Por boas práticas de programação todo serviço é um módulo para possibilitar a chamada de qualquer parte da aplicação com a sua injeção.

fruta.service.js

```
angular.module('frutaServiceApp', [])
    .service('FrutaService', frutaService);

function frutaService($http) {
    var ENDERECO_BACKEND = 'https://ng-curso-api.herokuapp.com/frutas';
    return {
        listar: function() {
            return $http.get(ENDERECO_BACKEND)
        },
        incluir: function(nome) {
            return $http.post(ENDERECO_BACKEND, nome);
        },
        atualizar: function(fruta) {
            return $http.put(ENDERECO_BACKEND, fruta);
        },
        deletar: function(id) {
            var parametros = {
                data: id,
                headers: {
                    'Content-type': 'application/json;charset=utf-8'
                }
            }
            return $http.delete(ENDERECO_BACKEND, parametros);
        }
    }
}
```

No serviço acima - FrutaService - usamos os métodos ***get*** para listar, ***post*** para incluir, ***put*** para atualizar e ***delete*** para deletar.

Para usar o serviço em um controller, deve-se injetar o módulo dele na declaração do app e depois injetá-lo na declaração do controller conforme podemos ver na linha 4 do arquivo *rotas.app.js* e na linha 3 do arquivo *formulario-novo.controller.js.*

app.js

```
angular
    .module('app', [
        'ngRoute',
        'frutaServiceApp'
    ])
    .config(['$routeProvider', '$locationProvider', definirRotas]);

function definirRotas($routeProvider, $locationProvider) {

    $locationProvider.hashPrefix('');
    $routeProvider.
    when('/', {
        controller: 'ListaController',
        templateUrl: 'lista.tpl.html'
    }).
    when('/edicao/:id', {
        controller: 'EdicaoController',
        templateUrl: 'formulario.tpl.html'
    }).

    when('/novo', {
        controller: 'NovoController',
```

```
        templateUrl: 'formulario.tpl.html'
    }).
    otherwise({ redirectTo: '/' });
}
```

formulario-novo.controller.js

```
angular
    .module('app')
    .controller('NovoController', novoController);

novoController.$inject = ['$scope', '$location', 'FrutaService'];

function novoController($scope, $location, frutaService) {

    $scope.titulo = 'Nova Fruta';
    $scope.fruta = {
        id: undefined,
        nome: undefined
    };

    /**
     * @description Inclui uma nova fruta na base de dados
     */
    $scope.salvar = function() {

        frutaService.incluir($scope.fruta.nome)
            .then(function() {

                // Redireciona para a página principal
                $location.path('/');
            });
    };
}
```

1.7. Gerindo dependências com NPM

O **npm** é o Gerenciador de Pacotes do Node (Node Package Manager) que vem junto com ele e que é muito útil no desenvolvimento node. Por anos, o Node tem sido amplamente usado por desenvolvedores JavaScript para compartilhar ferramentas, instalar vários módulos e gerenciar suas dependências. Sabendo disso, é realmente importante para pessoas que trabalham com Node.js entendam o que é npm.

1.7.1. Como o npm Funciona?

O npm funciona baseado nesses dois ofícios:

- Ele é um repositório amplamente usado para a publicação de projetos Node.js de código aberto (open-source). Isso significa que ele é uma plataforma online onde qualquer pessoa pode publicar e compartilhar ferramentas escritas em JavaScript.
- O npm é uma ferramenta de linha de comando que ajuda a interagir com plataformas online, como navegadores e servidores. Essa utilidade auxilia na instalação e desinstalação de pacotes, gerenciamento das versões e gerenciamento de dependências necessárias para executar um projeto.

Para usá-lo, você precisa instalar o **node.js** – visto que, eles são empacotados juntos.

1.7.2. Começar um projeto com npm

Se você já tiver o **Node** e o **npm** e deseja começar a construir seu projeto, execute o comando **npm init**. Com isso, você dará procedimento à inicialização do seu projeto.

Esse comando funciona como uma ferramenta para criar o arquivo package.json de um projeto. Depois de executar as etapas do npm init, um arquivo package.json será gerado e colocado no diretório atual.

1.8. Automatizando com o Grunt

Em diversas ocasiões, nós desenvolvedores, enfrentamos tarefas repetitivas que podem ser facilmente automatizadas. É aí que entra o Grunt. Ele é usado para: minificação, geração de builds, execução de testes, entre outros.

Instalação: ***npm install -g grunt-cli***

Em nosso projeto , após a criação do package.json, iremos adicionar as dependências abaixo nele para podermos usar o grunt como ferramenta de automação de tarefas repetitivas:

```
"devDependencies": {
        "grunt": "^1.0.1",
        "grunt-concurrent": "^2.3.1",
        "grunt-contrib-clean": "^1.0.0",
        "grunt-contrib-concat": "^1.0.1",
        "grunt-contrib-connect": "^1.0.2",
        "grunt-contrib-copy": "^1.0.0",
        "grunt-contrib-cssmin": "^1.0.2",
        "grunt-contrib-jshint": "^1.1.0",
        "grunt-contrib-uglify": "^2.0.0",
        "grunt-contrib-watch": "^1.0.0",
        "grunt-htmllint": "^0.2.7",
        "grunt-karma": "^2.0.0",
        "jit-grunt": "^0.10.0"
    }
```

Executar o comando **npm install** para instalar as dependências no seu projeto.

Devemos também criar o arquivo Gruntfile.js:

```
module.exports = function(grunt) {

    // Pasta onde se encontra os códigos do seu projeto
    var srcDir = 'src';

    // Pasta para onde deseja cobiar os códigos do build para execução no browser
    var buildDir = 'www';

    grunt.initConfig({

        // https://github.com/gruntjs/grunt-contrib-clean
        clean: {
            all: {
                src: [buildDir]
            }
        },
```

```
        // https://github.com/gruntjs/grunt-contrib-connect
        connect: {
            server: {
                options: {
                    port: 8300,
                    base: buildDir,
                    hostname: '*',
                    middleware: (connect, options, middlewares) => {
                        middlewares.unshift((req, res, next) => {
                            res.setHeader('Access-Control-Allow-Origin', '*');
                            res.setHeader('Access-Control-Allow-Methods', '*');
                            return next();
                        });
                        return middlewares;
                    }
                }
            }
        },

        // https://github.com/gruntjs/grunt-contrib-copy
        copy: {
            all: {
                cwd: srcDir,
                src: ['**', '!**/*-spec.js'],
                dest: buildDir,
                expand: true
            }
        },

        // https://github.com/gruntjs/grunt-contrib-cssmin
        cssmin: {
            all: {
                files: [{
                    expand: true,
                    cwd: srcDir,
                    src: '**/*.css',
                    dest: buildDir,
                    ext: '.min.css'
                }]
            }
        },

        // https://github.com/htmllint/grunt-htmllint
        htmllint: {
            all: {
                options: {
                    force: true,
                    'id-class-style': false,
                    'attr-name-style': false,
                    'attr-req-value': false,
                    'tag-bans': []
                },
                src: [srcDir + '/**/*.html']
            }
        },

        // https://github.com/gruntjs/grunt-contrib-jshint
        // Mais detalhes sobre a configuração do arquivo .jshintrc em http://jshint.com/docs/
        jshint: {
            options: {
                jshintrc: true
            },
            all: [srcDir + '/**/*.js',
                '!' + srcDir + '/**/*.min.js',
                '!' + srcDir + '/**/*-min.js',
```

```
                '!' + srcDir + '/**/*-spec.js'
            ]
        },

        // https://github.com/gruntjs/grunt-contrib-uglify
        uglify: {
            all: {
                files: [{
                    expand: true,
                    cwd: buildDir,
                    src: ['**/*.js', '!**/*-spec.js'],
                    dest: buildDir,
                    ext: '.min.js'
                }]
            }
        },

        // https://github.com/gruntjs/grunt-contrib-watch
        watch: {
            build: {
                files: [srcDir + '/**'],
                tasks: ['fastbuild']
            }
        },

        // https://github.com/karma-runner/grunt-karma
        karma: {
            unit: {
                logLevel: 'ERROR',
                configFile: 'karma.conf.js'
            }
        }
    });

    // Carregamento das tarefas já definidas pelo grunt
    grunt.loadNpmTasks('grunt-contrib-clean');
    grunt.loadNpmTasks('grunt-contrib-connect');
    grunt.loadNpmTasks('grunt-contrib-copy');
    grunt.loadNpmTasks('grunt-contrib-cssmin');
    grunt.loadNpmTasks('grunt-contrib-jshint');
    grunt.loadNpmTasks('grunt-contrib-uglify');
    grunt.loadNpmTasks('grunt-contrib-watch');
    grunt.loadNpmTasks('grunt-htmllint');
    grunt.loadNpmTasks('grunt-karma');

    // definição de tarefas
    grunt.registerTask(
        'build',
        'Compila todos os assets e copia os arquivos para o diretório de build.', ['clean', 'js
hint', 'karma', 'copy', 'cssmin', 'uglify']
    );

    grunt.registerTask(
        'fastbuild',
        'Compila todos os assets e copia os arquivos para o diretório de build.', ['clean', 'js
hint', 'karma', 'copy']
    );

    grunt.registerTask(
        'default',
        'Observa o projeto por mudanças, automaticamente contrói e executa o servidor.', ['fast
build', 'connect', 'watch']
    );
};
```

Após a instalação e configuração basta executar o comando **grunt** para levantar o servidor.

1.9. Testes automatizados com Jasmine/Karma

1.9.1. Jasmine

O Jasmine é um framework para escrever testes para código JavaScript. Ele é independente de navegador web e não precisa de outras bibliotecas para funcionar. Com o Jasmine, vamos escrever nossos testes unitários.

Documentação: https://jasmine.github.io/tutorials/your_first_suite

1.9.2. Karma

Karma é um test runner feito para o AngularJs. O principal objetivo do Karma é automatizar os testes em diversos navegadores web com um único comando. Mesmo ele tendo sido criado para o AngularJs, atualmente ele é usado em outros frameworks JavaScript.

O Karma suporta diversos tipos de testes, mas o foco do nosso curso serão os testes unitários

Instalação: ***npm install -g karma-cli***

1.9.3. Configuração

Precisamos adicionar as dependências do Jasmine e do Karma no package.json (devDependencies) do nosso projeto para podermos usá-los, executar o **npm install** para instalá-los e definir o script para execução automatizada no Gruntfile.js.

package.json

```
        "jasmine-core": "^2.5.2",
        "karma": "^1.7.1",
        "karma-coverage": "^1.1.1",
        "karma-jasmine": "^1.0.2",
        "karma-phantomjs-launcher": "^1.0.2",
        "phantomjs-prebuilt": "^2.1.16"
```

Lembre-se que toda vez que alterar as dependências no package.json, deve-se executar o comando **npm install.**

Gruntfile.js

```
        karma: {
            unit: {
                logLevel: 'ERROR',
                configFile: 'karma.conf.js'
            }
        },
```

```
    grunt.loadNpmTasks('grunt-karma');
```

Ainda no Gruntfile.js precisamos adicionar a chamada 'karma' no build e no fastbuild antes do script 'copy'.

Por fim precisamos configurar a execução dos nossos testes através da criação do karma.conf:

- Na pasta base do projeto devemos executar o comando ***karma init***

Após esses procedimentos vamos criar nosso primeiro teste:

teste.spec.js

```
describe('Primeiros Testes', function() {

    it('verificarAposentadoria', function() {

        expect(verificarAposentadoria(65, 20)).toEqual('QUALIFICADO PARA APOSENTARIA');
        expect(verificarAposentadoria(70, 20)).toEqual('QUALIFICADO PARA APOSENTARIA');
        expect(verificarAposentadoria(59, 20)).toEqual('NÃO QUALIFICADO PARA APOSENTARIA')
;

        expect(verificarAposentadoria(59, 30)).toEqual('QUALIFICADO PARA APOSENTARIA');
        expect(verificarAposentadoria(59, 35)).toEqual('QUALIFICADO PARA APOSENTARIA');

        expect(verificarAposentadoria(60, 25)).toEqual('QUALIFICADO PARA APOSENTARIA');
        expect(verificarAposentadoria(61, 26)).toEqual('QUALIFICADO PARA APOSENTARIA');
        expect(verificarAposentadoria(60, 24)).toEqual('NÃO QUALIFICADO PARA APOSENTARIA')
;
        expect(verificarAposentadoria(59, 25)).toEqual('NÃO QUALIFICADO PARA APOSENTARIA')
;

        expect(verificarAposentadoria(32, 14)).toEqual('NÃO QUALIFICADO PARA APOSENTARIA')
;
    });
    it('verificarTipoTriangulo', function() {

        expect(verificarTipoTriangulo(1, 1, 1)).toEqual('EQUILÁTERO');
        expect(verificarTipoTriangulo(2, 1, 1)).toEqual('ISÓSCELES');
        expect(verificarTipoTriangulo(1, 2, 1)).toEqual('ISÓSCELES');
        expect(verificarTipoTriangulo(1, 1, 2)).toEqual('ISÓSCELES');
        expect(verificarTipoTriangulo(1, 2, 3)).toEqual('ESCALENO');
    });
});
/**
Uma empresa quer verificar se um empregado está qualificado para a aposentadoria ou não.
Para estar em condições, um dos seguintes requisitos deve ser satisfeito:
- Ter no mínimo 65 anos de idade.
- Ter trabalhado no mínimo 30 anos.
- Ter no mínimo 60 anos e ter trabalhado no mínimo 25 anos.
 */
function verificarAposentadoria(idade, tempoTrabalho) {

}
/**
Crie um programa para verificar se um triangulo é Equilátero, Isósceles ou Escaleno
· um triângulo isósceles é um triângulo que possui dois lados de mesma medida.
· um triângulo equilátero é todo triângulo em que os três lados são iguais.
· um triângulo escaleno nenhum dos lados são iguais.
 */
function verificarTipoTriangulo(lado1, lado2, lado3) {

}
```

Executar o teste com o comando **grunt karma**.

1.9.4. Testanto o service

```
describe('FrutaService', function() {

    var serviceTest;
    var $httpBackend;

    beforeEach(function() {

        module('frutaServiceApp');
        inject(function(_FrutaService_, _$httpBackend_) {

            serviceTest = _FrutaService_;
            $httpBackend = _$httpBackend_;
        });
    });

    it('listar', function() {

        $httpBackend.expectGET('https://ng-curso-
api.herokuapp.com/frutas').respond(200, [{ id: 1, nome: 'banana' }]);
        serviceTest.listar()
            .then(function(response) {
                var listaFrutas = response.data;
                expect(listaFrutas.length).toEqual(1);
                expect(listaFrutas[0].id).toEqual(1);
                expect(listaFrutas[0].nome).toEqual('banana');
            });
        $httpBackend.flush();
    });
});
```

1.9.5. Testando o controlador

```
describe('NovoController', function() {

    var controllerTest;
    var $scope;
    var $location;
    var frutaService;

    beforeEach(function() {

        module('app', 'frutaServiceApp', 'ngRoute');
        inject(function(_$controller_, _$rootScope_, _$location_, _FrutaService_) {

            $scope = _$rootScope_.$new();
            $location = _$location_;
            frutaService = _FrutaService_;
            controllerTest = _$controller_('NovoController', { $scope: $scope, $location: $loca
tion, frutaService: frutaService });

        });
    });

    it('NovoController', function() {

        expect($scope.titulo).toBe('Nova Fruta');
        expect($scope.fruta).toBeDefined();
    });
```

```
    it('salvar', function() {

        $scope.fruta = { id: 1, descricao: 'teste' };
        $scope.salvar();
    });
});
```

1.10. Git - Controle de versão

Git é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de software, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo.

Comandos básicos git: https://rogerdudler.github.io/git-guide/index.pt_BR.html