Criando uma aplicação simples com AngularJS

por Davi Ferreira

AngularJS é o mais novo lançamento do time de desenvolvedores do Google. Diferentemente de outros frameworks JavaScript, ele adota uma abordagem mais ligada à sintaxe HTML, funcionando como uma espécie de extensão da linguagem.

Neste artigo, vamos criar uma aplicação simples de lista de compras e aprender os conceitos básicos do framework no que diz respeito à associação, manipulação e exibição de dados.

Estrutura inicial

Assim como qualquer aplicação web, nosso ponta-pé inicial acontece com a criação de um página básica. A diferença aqui é que vamos informar um nova propriedade na tag do nosso arquivo: **ng-app**.

```
<html ng-app>
  <head>
    <title>Lista de compras</title>
    <script src="http://code.angularjs.org/1.0.1/angular-1.0.1.min.js"></script>
  </head>
  <body>
  </body>
</html>
```

Essa é a grande sacada do AngularJS. Ao declarar a propriedade ng-app, estamos inicializando a nossa aplicação. É a primeira de algumas novas propriedades que iremos utilizar. Todo o funcionamento do framework gira em torno dessas novas declarações.

O atributo ng-app na tag informa que o nosso DOM, além de HTML, é também um documento AngularJS. Esta propriedade pode ser utilizada em qualquer elemento do DOM — em alguns casos, apenas uma parte do seu HTML será uma aplicação Angular. Por baixo dos panos, o framework define o elemento com o atributo ng-app como a raiz da aplicação.

Olá, Tableless!

Para provar que o foco do Angular está no HTML e não no JavaScript, vamos implementar um exemplo simples em nossa estrutura:

```
<html ng-app>
  <head>
    <title>AngularJS - Tableless</title>
    <script src="http://code.angularjs.org/1.0.1/angular-1.0.1.min.js"></script>
  </head>
  <body>
    <input type="text" ng-model="nome">
    <p>Olá, Tableless! Meu nome é: {{ nome }}</p>
  </body>
</html>
```

Ao carregarmos esse HTML no navegador e digitarmos qualquer coisa no input, o parágrafo é atualizado automagicamente. Perceberam que até agora não escrevemos nenhum código JavaScript?

A propriedade *ng-model* funciona como um canal entre a nossa view e o form. Ela pode ser utilizada em inputs do tipo texto, selects, textareas, checkboxes e radio buttons.

O model, seus dados e suas validações ficam automaticamente disponíveis no escopo da nossa aplicação, como veremos a seguir.

A associação de dados é feita através do famoso “bigode-bigode” ({{ }}), passando nomes presentes no escopo (no exemplo acima, o model **nome**).

Enfim, JavaScript!

Chegou a hora de escrevermos nosso primeiro trecho de código JavaScript. Vamos criar um controller para nossa aplicação que carrega uma lista inicial de ítens. Os ítens são armazenados no escopo da aplicação ($scope).

```
function ListaComprasController($scope) {
   $scope.itens = [
      {produto: 'Leite', quantidade: 2, comprado: false},
      {produto: 'Cerveja', quantidade: 12, comprado: false}
   ];
}
```

E é só isso por enquanto!

Exibindo nossos ítens

Vamos agora adicionar um novo bloco HTML com a tabela de listagem dos produtos:

```
<div ng-controller="ListaComprasController">
   <table>
    <thead>
     <tr>
      <th>Produto</th>
      <th>Quantidade</th>
     </tr>
    </thead>
    <tbody>
     <tr ng-repeat="item in itens">
      <td><strong>{{ item.produto }}</strong></td>
      <td>{{ item.quantidade }}</td>
     </tr>
    </tbody>
   </table>
</div>
```

Duas novidades foram apresentadas no HTML acima:

- o atributo **ng-controller** informa o nome do controller JavaScript responsável pelo bloco contido no elemento, no nosso caso o controller *ListaComprasController* definido anteriormente.
- o atributo **ng-repeat** executa um *loop* na variável **itens** declarada no escopo da aplicação, retornando cada

ítem e imprimindo o produto e a quantidade em uma linha da nossa tabela. O formato é: <retorno> in <coleção>.

Adicionando produtos

Para não ficarmos apenas com 4 linhas de JavaScript, vamos adicionar uma funcionalidade que inclui ítens em nossa lista de compras.

O primeiro passo é criar um formulário que será responsável pela ação:

```
<form class="form-inline" name="formItem">
 <input type="text" ng-model="item.produto">
 <input type="number" ng-model="item.quantidade">
 <button ng-click="adicionaItem()">adicionar ítem</button>
</form>
```

Estamos utilizando de novo o atributo **ng-model** para definir um model para os nossos inputs. O controller passa a receber diretamente informações sobre esses campos.

A novidade dessa vez fica por conta do atributo **ng-click**, que recebe uma função que precisamos declarar no controller:

```
function ListaComprasController($scope) {
   $scope.itens = [
      {produto: 'Leite', quantidade: 2, comprado: false},
      {produto: 'Cerveja', quantidade: 12, comprado: false}
   ];
$scope.adicionaItem = function () {
      $scope.itens.push({produto: $scope.item.produto,
                  quantidade: $scope.item.quantidade,
                  comprado: false});
      $scope.item.produto = $scope.item.quantidade = '';
   };
}
```

Ao clicarmos no botão, o produto é adicionado à tabela. Aqui o model poderia estar realizando diversas validações disponíveis na API do framework entre outras coisas. Porém, no nosso exemplo, apenas adicionamos um novo ítem à lista de produtos e em seguida limpamos os models (os campos do formulário).

Testes

Por ser um framework que demanda um código JavaScript mais estruturado, fica bem simples testar sua aplicação. Utilizando Jasmine, por exemplo, poderíamos facilmente testar o controller dessa forma:

```
describe('Lista Compras Unitário', function () {
   describe('ListaComprasController', function () {
      beforeEach(function () {
         this.$scope = {};
         this.controller = new ListaComprasController(this.$scope);
      });
it('deve criar "itens" com 2 ítens', function () {
         expect(this.$scope.itens.length).toBe(2);
```

```
    });
describe('adicionaItem', function () {
        it('deve adicionar um novo ítem à lista com dados do escopo', function () {
            this.$scope.item = {};
            this.$scope.item.produto = 'Carne';
            this.$scope.item.quantidade = 5;
            this.$scope.adicionaItem();
            expect(this.$scope.itens.length).toBe(3);
            expect(this.$scope.itens[2].produto).toBe('Carne');
            expect(this.$scope.itens[2].quantidade).toBe(5);
            expect(this.$scope.itens[2].comprado).toBeFalse;
        });
    });
  });
});
```

AngularJS é muito mais do que isso!

Deixei muitos tópicos de fora por enquanto. O objetivo aqui era mostrar o potencial do framework AngularJS. Seus recursos ainda incluem rotas, múltiplas views, AJAX e serviços REST e a criação de componentes personalizados. O que vocês viram foi o básico do básico, uma introdução.

No site do framework há uma documentação bem completa, com diversos tutoriais.

O código fonte do nosso exemplo vocês encontram no Github do Tableless. E nesse link vocês conseguem visualizar nossa lista de compras em ação.

Finalizando, nosso camarada Vedovelli gravou um screencast bem completo sobre o AngularJS, recomendo!