AngularJs

A diretiva ng-repeat

Você já deve ter percebido que a lista de contatos está fixa em nossa parcial contatos.html . Em uma aplicação real, a lista seria construída a partir de dados armazenados em banco de dados, tudo processado no lado do servidor.O AngularJS pede ao servidor os dados de que precisa e se encarrega de atualizar a view com esses dados no lado do cliente, o famoso data binding . Qualquer dado no objeto $scope é visível pela view, sendo assim, vamos criar uma lista de contatos no escopo de ContatosController :

Essa diretiva permite que o AngularJS repita a criação de um elemento de nossa página a partir dos dados de um array. Em nosso caso, vamos adicionála na <tr> que exibe os dados do contato:

```
<tr ng-repeat="contato in contatos">
    <td>
        <a>{{contato.nome}}</a>
    </td>
    <td>{{contato.email}}</td>
    <td>
        <button>Remover</button>
    </td>
</tr>
```

A diretiva ng-model e filtragem da lista

Podemos filtrar nossa lista de contatos, mas, primeiro, precisamos adicionar um input em nossa view

```
<input type="search" placeholder="parte do nome">
```

Não podemos simplesmente utilizar uma AE {{}} em nosso input,porque toda AE é somente leitura. Em nosso caso, queremos ler e gravar em uma propriedade no escopo do controller, isto é, queremos fazer two-way data binding. Para isso, usamos a diretiva ng-model:

```
<input ng-model="filtro" type="search" placeholder="parte do nome"
>
```

Vamos indicar que a diretiva ng-repeat possui um filtro com | filter:

```
<tr ng-repeat="contato in contatos | filter: filtro">
```

O serviço $http

Nosso front-end atualmente trabalha com dados estáticos definidos em nossos controllers. A ideia agora é buscar esses dados diretamente de nosso servidor.

O AngularJS possui o serviço $http responsável por requisições Ajax. Injetamos este serviço em nossos controllers como qualquer outro artefato do framework:

```
function($scope, $http) {
```

Para obtermos a lista de contatos de nosso servidor através de uma requisição do tipo GET, configuramos o serviço da seguinte maneira:

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

Promises: combatendo o callback HELL

O $http, no lugar de receber um callback diretamente em sua chamada, trabalha um pouco diferente: ela devolve um valor.

O padrão Promise

O primeiro passo para entender este mistério é saber que $http não retorna a lista de contatos, mas uma promise (promessa) de que ele tentará buscar esses dados:

```
var promise = $http({method: 'GET', url: '/contatos'});
```

Uma promise é um objeto que fornecerá o resultado futuro de uma ação. No exemplo anterior, como estamos executando uma requisição assíncrona, não sabemos quando ela nos devolverá seu resultado, sendo assim, ficamos com a promessa de sua devolução, sua promise

Fazendo uma analogia: quando alguém nos promete algo, temos apenas sua promessa, pois não sabemos quando ela será cumprida. Independente de ela ser cumprida ou não, tocamos nossa vida. Porém, sabemos o que pode acontecer se a promessa for cumprida ou não. É por isso que uma promise possui estados.

Estados de uma promise

Uma promise possui três estados e dependendo desses estados, ações são executadas:

- fulfilled: quando a promise é bem-sucedida;
- rejected: quando a promise é rejeitada;
- failed: quando não é nem bem-sucedida nem rejeitada.

As funções then e catch

Uma promise possui o método then, que recebe como parâmetros callbacks. O primeiro é executado quando o status da promise for fulfilled;

o segundo, para os estados rejected e failed.

```
var promise = $http.get('/contatos');
promise
.then(obterDados, function(erro) {
console.log(erro.status)
console.log(erro.statusText)
});
);
```

A função catch

Porém, o AngularJS introduz a função catch, que permite isolar o callback dos estados rejected e failed:

```
var promise = $http.get('/contatos');
promise
.then(obterDados)
.catch(erro) {
console.log(erro.status)
console.log(erro.statusText)
}
```

Nesse exemplo, através do parâmetro retorno, temos acesso a propriedades especiais que nos permitem acessar os dados retornados, inclusive obter mensagens de erro enviadas pelo servidor:

- data: o body da resposta transformado e pronto para usar;
- status: número que indica o status HTTP da resposta;
- statusText: texto HTTP da resposta.

Ainda é possível ter acesso ao objeto header e config, este último com as configurações utilizadas na requisição.

```
function exibeContatos(contatos) {
//Exibe os contatos na tela
return contatos;
}
function modificaContatos(contatos) {
//Modifica os contatos seguindo algum critério
return contatos;
}
function atualizaContatos(contatos) {
//Recebe os contatos modificados e
envia novamente para o servidor para que sejam gravados.
return contatos;
}
var promise = $http.get('/contatos');
promise
.then(exibeContatos)
.then(modificaContatos)
.then(AtualizaContatos)
.then(function(contatos) {
$scope.mensagem =
{texto: 'Contatos atualizados com sucesso'};
})
.catch(erro) { /* se algo der errado, trata */
console.log(erro.status)
console.log(erro.statusText)
});
```

As funções success e error

O objeto $http adiciona convenientemente em suas promises as funções success e error. A primeira recebe o callback que será executado quando a promise for fulfilled. Já a segunda, o callback que será executado quando a promise for rejected ou failed.

```
$http.get('/contatos')
.success(function(data) {
$scope.contatos = data;
})
.error(function(statusText) {
console.log(statusText);
});
```

Obtendo contatos com $http

O $http retorna uma promise e como executar um código quando ela for bemsucedida, já podemos refatorar nosso nontatosController para deixar de trabalhar com dados estáticos e obtêlos do nosso back-end feito com Express:

```
$http.get('/contatos')
.success(function(data) {
$scope.contatos = data;
})
.error(function(statusText) {
console.log("Não foi possível obter a lista de
contatos");
console.log(statusText);
});
```

Single page application (SPA)

Single Page Application (SPA) é uma aplicação entregue para o navegador que não recarrega a página durante seu uso. Aplicações deste tipo tendem a dar uma experiência mais fluida para os usuários, ao mesmo tempo em que favorece o servidor, enviando uma quantidade de dados menor para ser processada.

Em SPA, a página principal, por exemplo, index.html, é carregada apenas uma vez e possui uma grande lacuna preenchida com o conteúdo de outras páginas através das URLs acessadas. Essas páginas são chamadas de parciais

Usando este tipo de aplicação não podemos simplesmente digitar a URL da parcial que desejamos carregar. Se fizermos isso, a página inteira será recarregada. Uma estratégia é realizar requisições Ajax, assíncronas por natureza e independentes da requisição da página principal. No final, o desenvolvedor precisa atualizar a página já carregada com o novo conteúdo da parcial acessada manipulando DOM diretamente.

SPA utilizam URLs especiais que apontam sempre para a mesma página adicionando alguma informação sobre a parcial a ser carregada. É comum que essa informação seja adicionada imediatamente após um #, uma espécie de marcador que permite extrair qual parcial deve ser processada.

Tudo é feito através de um sistema de rotas que evita a submissão da URL ao mesmo tempo em que remove a responsabilidade do programador de atualizar a página principal com o conteúdo das parciais:

O módulo ngRoute

AngularJS possui um sistema de rotas que visa blindar o desenvolvedor da complexidade pela atualização de áreas da página utilizando Ajax, mais uma vez, evitando que ele manipule o DOM diretamente.

O AngularJS contem um módulo em separado chamado ngRoute. Rode o comando:

bower install angular-route --save

o módulo ngRoute como sua dependência. Isso é importante, caso contrário, não teremos acesso à artefatos injetáveis deste módulo. Vamos fazer isso adicionando-o no array passando como segundo parâmetro da função angular.module:

```
angular.module('contatooh', ['ngRoute']);
```

Sistema de rotas ativado, devemos configurá-lo,vamos alterar a página public/index.html substituindo o conteúdo da tag <body> por uma com a diretiva ng-view, mas mantendo a importação dos scripts.

Não podemos esquecer de remover a diretiva ng-controller de <body>, o que é necessário porque a associação do controller será feita através da configuração de nossas rotas. Isso permite que parciais possam utilizar controllers diferentes de acordo com quem as usa

A diretiva ng-view sinaliza para o sistema de rotas a área da página que receberá views parciais. A diferença de uma view parcial para uma view como a index.html é que a primeira não possui as tags <html>, <head> e <body>, logo, para serem exibidas, precisam ser incluídas dinamicamente dentro de uma página com a diretiva ng-view, em nosso caso, a página index.html.

Criando views parciais

Temos nossa view principal index.html pronta, agora precisamos criar nossas primeiras parciais que popularão a diretiva ng-view. Vamos criar a pasta app/public/partials e dentro dela guardar todas as nossas views parcials

Primeiro criaremos a view parcial que será responsável pela listagem dos contatos. Repare que movemos para ela nosso botão e o parágrafo com a Angular Expression:

Se abrirmos a página no navegador apenas veremos uma página em branco. Precisamos configurar as rotas da aplicação e definir que a URL http://localhost:3000/index.html#/contatos deve carregar a view parcial public/partials/contatos.html

Configurando rotas com $routeProvider

A configuração das rotas da aplicação costuma ser feita no módulo principal da aplicação, ou seja, no arquivo main.js. Utilizamos a função config, que recebe uma função que tem como parâmetro um artefato injetado pelo AngularJS responsável pela criação de rotas, o objeto $routeProvider. Caso não tivéssemos importado o módulo ngRoute, ele não estaria disponível para injeção:

```
.config(function($routeProvider) {
$routeProvider.when('/contatos', {
templateUrl: 'partials/contatos.html',
controller: 'ContatosController'
});
```

O objeto $routeProvider possui a função when. Nela informamos a rota (sem o #) e no segundo parâmetro um objeto que define qual template (parcial) será carregado para a rota e qual será seu controller através das propriedades templateURL e controller, respectivamente.

Precisamos criar a rota responsável pela exibição da parcial partials/contato.html. Esta página exibirá o contato selecionado da lista, logo, precisa receber como parâmetro o id do contato selecionado:

```
$routeProvider.when('/contato/:contatoId', {
templateUrl: 'partials/contato.html',
controller: 'ContatoController'
});
```

O objeto $routeParams

Vimos a necessidade de obtermos o ID do contato a partir de uma rota do AngularJS. Para isso, criaremos o controller public/js/controllers/ContatoController.js:

```
angular.module('contatooh').controller('ContatoController',
function($scope, $routeParams) {
console.log($routeParams.contatoId);
});
```

Adicionando rota padrão

Por fim, podemos adicionar uma rota padrão caso o endereço da rota não exista. Fazemos isso através da função $routeProvider.otherwise.

Nela, passamos um objeto com a propriedade redirectTo, que aponta para um rota alternativa, em nosso caso, aquela que lista os contatos:

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

Exercicios

Criar 2 Partias na aplicação de series uma para listar todos as series cadastradas

E outra para apena uma e cadastrar

Criar uma tabela com todos os elementos vindos do servidor rest