WebII

Utilizando modelos e Angular

Utilizando modelos

Aprendemos a criar um Model a partir de um Schema. Disponbilizamos o Model para nosso controller através do express-load utilizando a convenção de acesso app.models.contato. Agora, precisamos reescrever o código de nosso controller.

Vamos editar app/controllers/contato.js removendo a implementação de todas as funções do controller e apagando a lista fxa de contatos e a variável de autoincremento que criamos. No fnal, nosso controller deve fcar assim:

```
// app/controllers/contato.js
module.exports = function (app) {
  //referencia da nossa classe model
  var Contato = app.models.contato;
  var controller = {};
  controller.listaContatos = function (req, res) {};
  controller.obtemContato = function (req, res) {};
  controller.removeContato = function (req, res) {};
  controller.salvaContato = function (req, res) {};
  return controller;
};
```

Guardamos uma referência para o Model na variável Contato iniciando com letra maiúscula, uma convenção bastante utilizada para funções construtoras. Essas funções permitem utilizar o operador new para criarmos novas instâncias, objetos que representam nossos documentos.

Buscando documentos

Vamos iniciar pela função controller.listaContatos. Não temos interesse em instanciar um contato, apenas listar todos os cadastrados em nosso banco. Para isso, a própria função construtora Contato possui a função find. Como queremos todos os contatos, ela não receberá um critério como parâmetro.

Existem diferentes maneiras de realizarmos esta consulta, porém a função find pode retornar uma promise através da chamada encadeada à função exec:

```
controller.listaContatos = function(req, res) {

        var promise = Contato.find().exec();

}
```

```
controller.listaContatos = function (req, res) {
    Contato.find().exec()
      .then(
        function (contatos) {
          res.json(contatos);
        },
        function (erro) {
          console.error(erro);
          res.status(500).json(erro);
        }
    );
  };
```

Recebemos no callback de sucesso a lista de contatos retornada e logo em seguida a enviamos através da função res.json. Já no callback de erro, logamos a informação e, antes de enviá-la para o cliente, modifcamos o status da resposta para 500 (internal server error).

Buscando pelo ID

Nosso próximo passo será implementar a busca de um contato, só que dessa vez utilizaremos a função Contato.findById:

```
controller.listaContatoId = function (req, res) {
    var _id = req.params.id;
    Contato.findById(_id).exec()
      .then(function (contato) {
        if (!contato) throw new Error('Contato não encontrado');
        res.json(contato);
      },
        function (erro) {
          console.log(erro);
          res.status(404).json(erro);
        }
    );
  };
```

Removendo documentos

Agora atacaremos a remoção de contatos. Utilizaremos a função Contato.remove que recebe como critério o ObjectId procurado. Se a operação for realizada com sucesso, enviaremos o status padrão 200 como resposta.

```
controller.deleteContato = function (req, res) {
    var _id = req.params.id;
    Contato.remove({'_id': _id}).exec()
      .then(
        function () {
          res.status(204).end();
        },
        function (erro) {
          return console.error(erro);
        }
    );
  };
```

Atualizando Contatos

utilizaremos a função Contato.findByIdAndUpdate para atualizá-lo. Seu primeiro parâmetro é o ID do contato procurado; o segundo, os dados do contato:

```
controller.updateContato = function (req, res) {
   var _id = req.params.id;
   Contato.findByIdAndUpdate(_id, req.body).exec()
     .then(
     function (contato) {
       res.json(contato);
     },
     function (erro) {
       console.error(erro);
       res.status(500).json(erro);
     });
 };
```

Bower: gerenciador de dependências front-end

Criado pela equipe do Twitter é um gerenciador de pacotes voltado para front-end.

O Bower (http://bower.io) é um gerenciador de pacotes para web voltado para front-end que realiza grande parte das tarefas que faríamos manualmente, inclusive gerencia as dependências de sua aplicação. Outro ponto interessante é que ele não gerencia pacotes exclusivamente JavaScript, mas também CSS e HTML.

Bower vs. npm

Podemos comparar o Bower com npm , com a diferença de que o primeiro é voltado para pacotes front-end e o segundo para pacotes backend, apesar de alguns desenvolvedores tentarem utilizar o npm para as duas finalidades.

Instalação do Bower

npm install bower -g

bower.json e as dependências

No Bower, o arquivo que lista nossas dependências é chamado bower.json. Dentro da pasta raiz do projeto, ele pode ser criado pelo comando bower init

Baixando dependências front-end

emos o Bower instalado e o arquivo bower.json dentro da pasta contatooh. Nosso projeto utilizará duas bibliotecas: AngularJS e Bootstrap. A primeira é uma biblioteca JavaScript; a segunda, uma biblioteca CSS. Vamos instalar o AngularJS primeiro e mais tarde o Bootstrap

Utilizamos o comando bower install passando como parâmetro o nome da biblioteca seguido opcionalmente de # e o número da versão. Utilizaremos o parâmetro --save que adicionará a biblioteca como dependênciaem bower.json. Por exemplo se quisermos a versão 1.3 do AngularJS:

bower install angular#1.3 --save

Alterando a pasta destino com .bowerrc

Podemos alterar o caminho e o nome da pasta criada pelo Bower para armazenar nossas dependências através do arquivo oculto .bowerrc. Vamos criar o arquivo dentro da raiz do projeto com a seguinte estrutura:

```
// /.bowerrc
{
     "directory": "public/vendor"
}
```

O .bowerrc também é um JSON assim como bower.json com primeiro possui a propriedade directory. É nesta propriedade que defnimos o local onde serão baixados os pacotes do projeto. Em nosso caso, além de alterarmos o caminho da pasta, também mudamos o nome do diretório para public/vendor

Outros comandos

search

o comando search que lista todas as bibliotecas registradas que possuem o texto procurado, por exemplo:

bower search angular

Serão listados no terminal bibliotecas que contenham como parte de seu nome a palavra angular.

Info

Quais versões de uma determinada biblioteca estão disponíveis pelo Bower? Podemos ter essa informação facilmente pelo comando:

bower info angular

Esse comando listará no terminal todas as versões disponíveis e que podem ser baixadas através do Bower:

uninstall

O que fazer quando uma biblioteca deixar de ser uma dependência do nosso projeto? Precisamos apagar sua pasta com todos os seus arquivos, inclusive alterar nosso arquivo bower.json removendo a biblioteca.

Podemos fazer tudo em um unico comando:

bower uninstall angular --save

--ofine

Todo pacote baixado pelo Bower fca armazenado em um cache que pode ser utilizado ofine, muito bem-vindo quando estamos sem acesso à internet e não queremos perder tempo. Para isso, basta adicionar o parâmetro --offline em conjunto com o comando bower install:

bower install angular#1.3 --save --offline

AngularJS: o framework MVC da Google

O código que escrevemos no lado do cliente manipula algo que não existe no lado do servidor: o Document Object Model (DOM).

O DOM é uma árvore de elementos, isto é, um espelho em memória de nossa página criado automaticamente pelo navegador quando ela é carregada. Inclusive, ele possui funções e propriedades que, ao serem modifcadas, disparam alterações instantâneas no que é exibido para o usuário.

Vejamos um exemplo de manipulação do DOM que incrementa o total de contatos cadastrados para cada clique no botão.

```
<!-- página html -->
<button class="botao-grava">Novo</button>
<p class="contatos">Contatos cadastrados: 0 </p>
```

Agora o JavaScript:

```
var contatos = document.querySelector('.contatos');
var total = 0;
var botao = document.querySelector('.botao-grava');
botao.addEventListener('click', function(event) {
total++;
contatos.textContent = 'Contatos cadastrados: ' + total;
});
```

Dificuldades que surgem da manipulação do DOM

Apesar de ser uma tarefa rotineira para programadores front-end, a manipulação do DOM traz algumas difculdades.

Presos à estrutura

Nos exemplos anteriores, por mais que tenhamos utilizado classes nos seletores para não fcarmos amarrados à estrutura do documento, ainda somos obrigados a conhecê-lo para saber que o elemento X possui a classe tal e, portanto, pode ser selecionado:

Testabilidade

Outro ponto que salta à vista é que a não separação entre a lógica e os elementos do DOM torna o teste do código uma tarefa não trivial:

Consistência entre model e view

Também somos os responsáveis em garantir a consistência entre o dado e sua apresentação. Em outras palavras, toda vez que o model for atualizado, precisamos atualizar sua apresentação na view.

Outra opção: frameworks MVC client-side

Com base nos problemas listados, foram criados frameworks MVCclient side como BackBone (http://backbonejs.org) , Ember (http://emberjs.com) React (https://facebook.github.io/react/) Vuejs (https://vuejs.org/) entre outros.

Características do AngularJS

AngularJS (http://angularjs.org) é um framework MVC client-side que trabalha com tecnologias já estabelecidas: HTML, CSS e JavaScript. Criado na Google e liberado como projeto open-source para o público em 2009, seu foco reside na criação de Single Page Applications (SPA)

Miško Hevery, pai do framework, procurou trazer os mesmos benefícios do modelo MVC aplicados no lado do servidor para o navegador como a facilidade de manutenção, reusabilidade através de componentes e testabilidade, uma vez que a lógica é desacoplada de elementos do DOM. Ainda há recursos como injeção de dependências!

Preparando o ambiente

Para que possamos entender com clareza como o AngularJS funciona, vamos criar uma página simples que envolve um botão que ao ser clicado incrementa um contador que simula o total de contatos cadastrados.

Habilitando o AngularJS em nossa página

Para habilitarmos o AngularJS em nossa página public/index.html, precisamos primeiro importar seu script antes da tag de fechamento </body>:

```
<script src="vendor/angular/angular.js"></script>
```

Importar a biblioteca não é sufciente. Precisamos adicionar o atributo ng-app na área que será controlada pelo AngularJS:

Isso é necessário, porque o AngularJS gerencia apenas o bloco com o atributo ng-app. Em nosso exemplo, estamos gerenciando o HTML inteiro, mas poderíamos gerenciar uma área menor, deixando o restante para outros frameworks que também gerenciam o HTML, como Emberjs (http://emberjs.com/) , evitando conflito.

Nosso primeiro módulo com AngularJS

Um módulo em AngularJS nada mais é do que um código JavaScript de clarado em seu arquivo. Vamos criar o arquivo public/js/main.js importando-o logo em seguida em public/index.html:

AngularJS disponibiliza o objeto angular globalmente. É através dele que acessamos vários recursos do framework, inclusive o de criação de módulos, justamente aquele de que precisamos.

Criamos um módulo através da função angular.module. Ela recebe dois parâmetros: o primeiro é o nome do módulo; o segundo, um array com todas as suas dependências. Não temos nenhuma dependência por enquanto, ainda assim, precisamos passar o array vazio como parâmetro.

Angular expression (AE)

Em public/index.html, nosso parágrafo exibe o total de contatos cadastrados, mas o valor está fixo. É necessário que essa informação varie toda vez que o usuário clicar no botão “Novo”. Para isso, usaremos uma Angular Expression (AE):

```
<p>Contatos cadastrados: {{total}}</p>
```

Uma AE é representada por {{ }}. Tudo o que estiver entre as chaves duplas será buscado de algum lugar e mesclado com nossa view. Nesse ponto,podemos dizer que nossa view é uma espécie de template, já que possui uma lacuna que precisa ser preenchida. Em nosso caso. usamos a AE {{total}}.

Se o AngularJS é um framework MVC, quem é o responsável em disponibilizar os dados para a view? O controller

Nosso primeiro controller

A diretiva ng-app apenas indica que a tag html é gerenciada pelo AngularJS, inclusive todos seus elementos flhos, mas não indica quem fornecerá os dados ou quem responderá às ações levadas com esses elementos. Em AngularJS, dizemos que estas são responsabilidades de um controller.

A diretiva ng-controller

Associamos um controller a um elemento de nossa página através da diretiva ng-controller. Em nosso caso, ela será associada à tag <body>:

```
<body ng-controller="ContatosController">
```

O controller ContatosController será o responsável em fornecer o dado {{total}} para view

É uma boa prática do AngularJS criar cada controller em um arquivo em separado, inclusive podemos agrupá-los dentro de uma pasta chamada controllers

```
// public/js/controllers/ContatosController.js
angular.module('contatooh').controller('ContatosController', function ($scope) {
  $scope.total = 0;
});
```

Executando ações no controller através de diretiva

Como incrementar o total exibido quando o botão for clicado? É através da diretiva ng-click que conseguiremos este resultado:

```
<button ng-click="incrementa()">Novo</button>
```

```
// public/js/controllers/ContatosController.js
$scope.incrementa = function() {
$scope.total++;
};
```

Data binding

O mecanismo que faz a associação entre view e model é chamada data binding. Por enquanto vimos o tipo one-way data binding, no qual a view só pode ler o model, mas não gravá-lo.

A diretiva ng-repeat

Exercicíos

Criar as rotas do app de series usando mongoose

E criar uma calculadora utilizando angular