05

Criando nosso ListModel

Transcrição

A negociação está pronta! Agora temos que adicionar as demais negociações cadastradas na lista - lembrando que uma vez adicionadas, estas não serão mais removidas e tampouco poderemos alterá-las.

Se optarmos por trabalhar com *array* de `negociacoes`, não temos esse tipo de controle e podemos fazer várias operações sobre o *array*. A solução é criar um modelo (*model*) que vai encapsular a regra de uma lista de negociações. Dentro da pasta `models`, criaremos o arquivo `ListaNegociacoes.js`. A nova classe criada terá como atributo uma lista de negociações que começará com `0`:

```
class ListaNegociacoes {

    constructor() {
        this._negociacoes = []
    }

    adiciona(negociacao) {
        this._negociacoes.push(negociacao);
    }
}
```

Observe que usamos o prefixo `_` para indicar que a lista não deve ser alterada. Se ninguém pode acessar as propriedades de negociação dentro da lista, adicionamos o método `adiciona` que receberá uma `negociacao`. Precisamos de um método que nos permita ler a lista de negociações para então, podermos exibi-la.

Em seguida, criaremos o getter de `negociacoes`. Ao instanciarmos uma lista de negociações, ela estará vazia. Por meio do método `adiciona()`, podemos adicionar negociações e com o método `get` poderemos listá-las.

```
adiciona(negociacao) {
    this._negociacoes.push(negociacao);
}

get negociacoes() {

    return this._negociacoes;
}
```

Depois, no `index.html`, temos que importar a nossa classe.

```
<script src="js/app/models/Negociacao.js"></script>
<script src="js/app/controllers/NegociacaoController.js"></script>
<script src="js/app/helpers/DateHelper.js"></script>
<script src="js/app/models/ListaNegociacoes.js"></script>
```

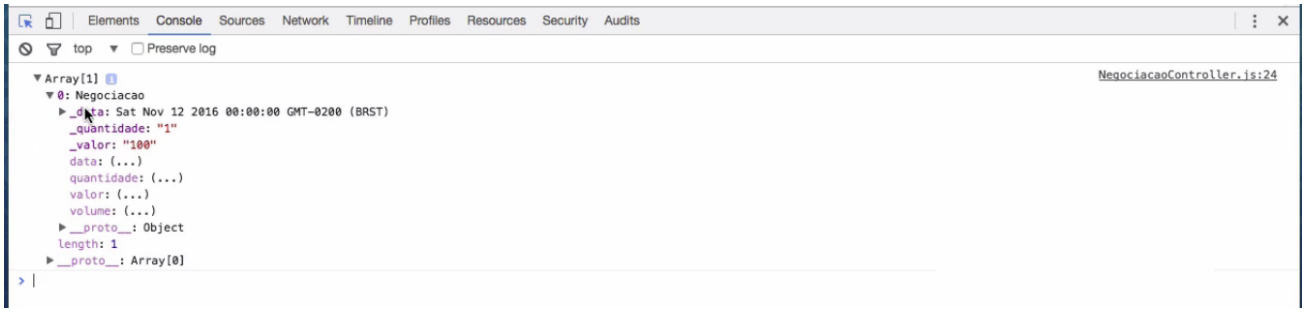
De volta ao `NegociacaoController`, adicionaremos um atributo que chamarei `_listaNegociacoes`.

```
class NegociacaoController {

    constructor() {

        let $ = document.querySelector.bind(document);
        this._inputData = $('#data');
        this._inputQuantidade = $('#quantidade');
        this._inputValor = $('#valor');
        this._listaNegociacoes = new ListaNegociacoes();

    }
    //...
```

Temos um atributo da `Controller`. Assim que cadastramos uma nova negociação, é preciso também construir uma nova `Negociacao` com os dados do formulário. Para isto, adicionaremos um novo `this` ao método `adiciona()`.

```
adiciona(event) {

        event.preventDefault();

        let negociacao =
            new Negociacao(
                DateHelper.textoParaData(this._inputData.value),
                this._inputQuantidade.value,
                this._inputValor.value
            );

        this._listaNegociacoes.adiciona(negociacao);
        console.log(this._listaNegociacoes.negociacoes);
```

Para testar se funciona, usamos o `console.log()` para exibir a lista de negociações. No navegador vamos preencher o formulário adicionando `12/11/2016` no campo `Data`, `1` em `Quantidade` e `100` no `Valor`. Veremos que os dados serão exibidos corretamente.



Queremos adicionar os dados de uma nova negociação, mas o formulário ainda tem nos campos as informações preenchidas anteriormente. Vamos encontrar uma forma de limpar o formulário para que a ação não seja realizada pelos usuários. Podemos adicionar também o foco no campo data. Faremos tudo isso, adicionando abaixo do método `adiciona()`, o `_limpaFormulario`, que só poderá ser chamado pela própria classe `NegociacaoController`.

```
_limpaFormulario() {
        this._inputData.value = '';
        this._inputQuantidade.value = 1;
        this._inputValor.value = 0.0
```

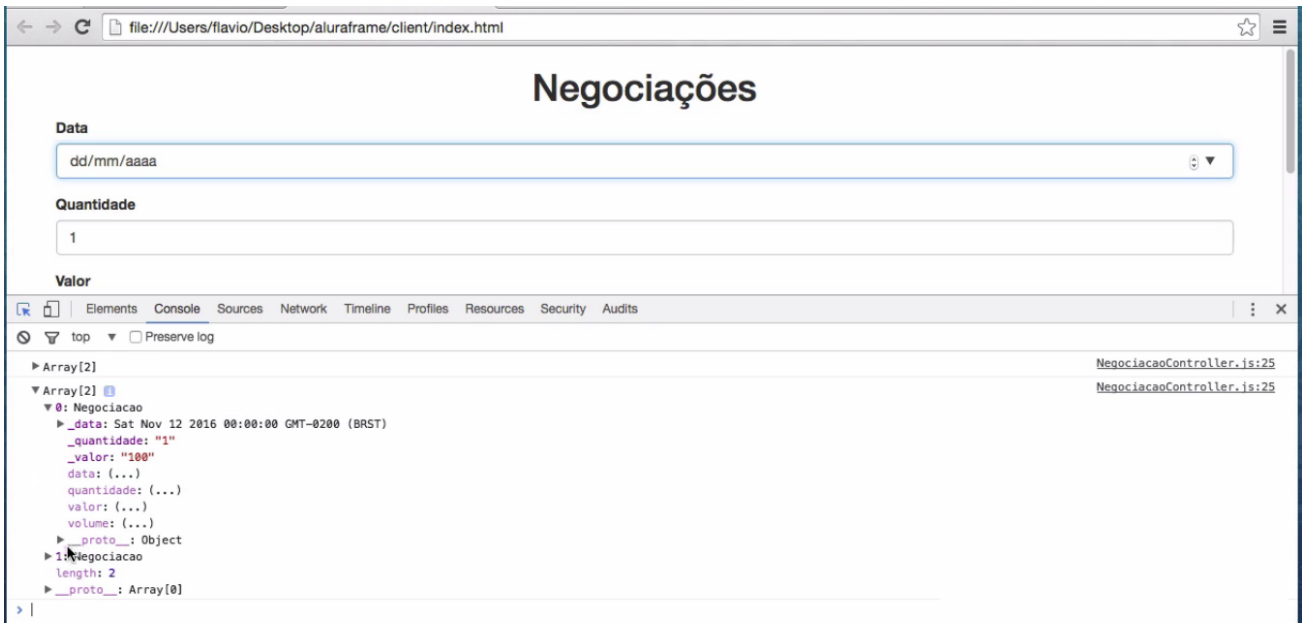
```
        this._inputData.focus();

    }
```

Assim que acabarmos de fazer a negociação, chamaremos o `this._limpaFormulario()` :

```
adiciona(event) {

    event.preventDefault();

    let negociacao = new Negociacao{
        DateHelper.textoParaData(this._inputData.value),
        this._inputQuantidade.value,
        this._inputValor.value
    };

    this._listaNegociacoes.adiciona(negociacao);
    this._limpaFormulario();

    console.log(this._listaNegociacoes.negociacoes);
}
```

Não podemos nos esquecer de importar o script em `index.html` . No lugar de importá-lo como último script, vamos agrupá-lo com a importação de `Negociacao.js` ; sendo assim, tudo o que for model será importado um após o outro. Só estamos imprimindo, por enquanto, porque ainda não estamos exibindo a negociação na tela. Vamos testar novamente e preencher os dados do formulário.

Após preenchermos uma primeira vez todos os campos e enviarmos os dados, o formulário ficará vazio. Mas o dados da negociação foram salvos. O mesmo ocorrerá quando preenchermos pela segunda vez os dados do formulário.



Temos duas negociações dentro do *array*, cada uma com a sua devida configuração. Já estamos conseguindo adicionar elementos à lista. Mas podemos melhorar ainda o código. Nós criamos a negociações dentro do método `adiciona()` . Logo abaixo, criaremos outro método auxiliar que se chama `_criaNegociacao` . Neste, aproveitaremos o `return` do `adiciona()` . Agora o novo método será responsável por criar `Negociacao` :

```
_criaNegociacao() {

  return new Negociacao(
        DateHelper.textoParaData(this._inputData.value),
        this._inputQuantidade.value,
        this._inputValor.value
    );
//...
```

Removeremos a variável `negociacao` do `adiciona()` e em vez de passarmos `negociacao` no `this.`, passaremos o `this._criaNegociacao`:

```
adiciona(event) {

    event.preventDefault();

    this._listaNegociacoes.adiciona(this._criaNegociacao());
    this._limpaFormulario();

    console.log(this._listaNegociacoes.negociacoes);
}
```

Até aqui, os três métodos estão assim:

```
adiciona(event) {

    event.preventDefault();

    this._listaNegociacoes.adiciona(this._criaNegociacao());
    this._limpaFormulario();

    console.log(this._listaNegociacoes.negociacoes);
}

_criaNegociacao() {
    return new Negociacao(
        DateHelper.textoParaData(this._inputData.value),
        this._inputQuantidade.value,
        this._inputValor.value);
  }

_limpaFormulario() {
    this._inputData.value = '';
    this._inputQuantidade.value = 1;
    this._inputValor.value = 0.0

    this._inputData.focus();

            }
```

Testaremos o formulário para conferir se não quebramos nada.

Tudo está funcionando corretamente. Então, criamos dois métodos auxiliares da `controller` que não devem ser chamados por fora da classe. A convenção é que apenas a classe pode chamar propriedades e métodos sinalizados com `_` .

Mas será que a classe, o modelo de negociações, é imutável? Será que podemos interagir com a lista de negociação? Será que podemos alterar sem passar pelo `adiciona()` ? Vamos fazer um teste para descobrir.