02

Ah se eu pudesse ordenar minha tabela clicando na coluna...

Melhorando a experiência do usuário

Nossa aplicação é capaz de exibir uma lista de negociações em uma tabela, que pode ser alimentada pelo usuário ou importada de serviços na web. Para deixarmos a aplicação ainda melhor, que tal permitir que o usuário ordene a tabela, clicando em cada coluna?

Por exemplo, se o usuário clicar na coluna "**QUANTIDADE**", ordenaremos pela `quantidade`, se ele clicar na coluna "**DATA**", ordenaremos pela `data`. Além disso, se ele clicar mais de uma vez na mesma coluna, ele ordenará a tabela ascendentemente ou descendentemente. Nesse exercício, mostrarei uma "receita" para resolver este problema.

A primeira coisa que faremos é criar um método em nosso `NegociacaoController`, que será o responsável em ordenar a lista de negociações de `ListaNegociacoes`. A ordenação da lista envolve sua alteração, logo, a view `NegociacoesView` precisará ser atualizada. Ainda bem que implementamos um mecanismo caseiro de *data binding* (associação de dados) entre o model e view, no qual a alteração no modelo automaticamente renderiza a view ao qual foi associado.

Altere `aluraframe/client/js/app/controllers/NegociacaoController.js` e adicione o método `ordena`:

```
// aluraframe/client/js/app/controllers/NegociacaoController.js
// código anterior omitido

ordena(coluna) {
    // ainda vamos implementar o método!
}
```

Veja que o método `ordena` recebe como parâmetro a `coluna` que queremos ordenar. Lembre-se que ordenaremos nosso modelo pela coluna que o usuário clicar, sendo assim, nada mais justo do que associar o método `ordena` ao evento `click` do cabeçalho de cada coluna em `NegociacoesView`:

Alterando `aluraframe/client/js/app/views/NegociacoesView.js`:

```
<!-- aluraframe/client/js/app/views/NegociacoesView.js -->
<!-- código anterior omitido -->
<thead>
    <tr>
        <th onclick="negociacaoController.ordena('data')">DATA</th>
        <th onclick="negociacaoController.ordena('quantidade')">QUANTIDADE</th>
        <th onclick="negociacaoController.ordena('valor')">VALOR</th>
        <th onclick="negociacaoController.ordena('volume')">VOLUME</th>
    </tr>
</thead>
<!-- código posterior omitido -->
```

Quando associamos a chamada do método ao evento, na chamada do método passamos como parâmetro a coluna que desejamos ordenar. É importante que cada parâmetro passado exista como uma propriedade em nosso modelo `Negociacao`.

Você deve estar pensando "Mas Flávio, queremos é ordenar a lista de negociações que `ListaNegociacoes` guarda". Sim, mas cada negociação da lista é uma instância da classe `Negociacao`.

Primeiramente, vamos implementar a solução de ordenação sem nos preocupar em alternar ascendentemente ou descendentemente, resolveremos isso depois. Além disso, antes de partir para a solução, que tal entender como é feito o processo de ordenação de uma lista a partir de algum critério da lista.

Entendendo primeiro: Array.sort

Um Array em Javascript possui o método `sort`. Este método recebe uma estratégia de ordenação, ou seja, essa estratégia deve ser passada pelo desenvolvedor, mas deve seguir algumas regras. Vejamos um exemplo com escopo menor:

```
let lista = [10,1, 5, 9, 8, 12, 15];
```

Queremos ordenar essa lista em ordem crescente:

```
let lista = [10,1, 5, 9, 8, 12, 15];
lista.sort();
console.log(lista); // exibe a lista na ordem crescente
```

E se quisermos em ordem decrescente? Ordenamos primeiro de maneira ascendente e depois invertemos a ordem do array com `reverse`:

```
let lista = [10,1, 5, 9, 8, 12, 15];
lista.sort();
lista.reverse();
console.log(lista); // exibe a lista ordenada em ordem decrescente
```

Na verdade, mesmo a ordenação numérica tem problemas no JavaScript. Faça o teste, o resultado é um pouco inesperado. Contudo temos uma [explicação detalhada sobre o ordenamento numérico do JavaScript em nosso blog (http://blog.alura.com.br/ordenacao-de-numeros-no-javascript-nao-funciona/)](http://blog.alura.com.br/ordenacao-de-numeros-no-javascript-nao-funciona/).

Podemos até mesmo ordenar uma lista de strings, que o procedimento é o mesmo. A ordenação funcionou porque o padrão do `sort` é classificar os elementos em ordem crescente na ordem da tabela ASCII. Vamos para um exemplo mais complexo?

Agora temos uma lista de negociações:

```
let negociacoes = [
    new Negociacao(new Date(), 7, 200),
    new Negociacao(new Date(), 1, 300),
    new Negociacao(new Date(), 8, 100)
]
```

Queremos que a lista seja ordenada pela propriedade `quantidade`. O que será que vai acontecer se chamarmos `lista.sort`?

```
let negociacoes = [
    new Negociacao(new Date(), 7, 200),
    new Negociacao(new Date(), 1, 300),
    new Negociacao(new Date(), 8, 100)
]
negociacoes.sort();
negociacoes.forEach(negociacao => console.log(negociacao));
```

Pois é, o método `sort` não fez curso de "Mãe Diná" para saber qual critério deve usar para ordenar nossa lista. Além disso, a lista continua do jeito que está. O método `sort` não consegue aplicar a estratégia de ordenar de maneira crescente porque um objeto da classe `Negociacao` não tem representação na tabela ASCII. E agora?

Quando temos uma lista de objetos que não sejam strings, números ou `boolean` (com este tipo, `false` vem primeiro e depois `true`), precisamos passar o critério de ordenação para o método `sort`:

```
let negociacoes = [
    new Negociacao(new Date(), 7, 200),
    new Negociacao(new Date(), 1, 300),
    new Negociacao(new Date(), 8, 100)
]
negociacoes.sort((a, b) => a.quantidade -  b.quantidade);
negociacoes.forEach(negociacao => console.log(negociacao));
```

A função passada para `sort` recebe dois parâmetros que representam pares de elementos, isso porque toda comparação envolve um par de elementos. A regra é a seguinte: com o critério selecionado, se o valor retornado for `0` não há alteração a ser feita, se o valor retornado for positivo, `b` deve vir antes de `a`, se o valor for negativo, `a` deve vir antes de `b`.

Que tal ordenar pela data?

```
let negociacoes = [
    new Negociacao(new Date(), 7, 200),
    new Negociacao(new Date(), 1, 300),
    new Negociacao(new Date(), 8, 100)
]
negociacoes.sort((a, b) => b.data - a.data); // agora é b menos a!
negociacoes.forEach(negociacao => console.log(negociacao));
```

Não fique chocado, quando subtraímos uma data pela outra é retornado um número que pode ser zero, positivo ou negativo, atendendo a regra do `sort`. Faça um teste no console do Chrome e veja você mesmo:

```
new Date(2016,4,12) - new Date(2016,5,1) // negativo
new Date(2016,5,1) - new Date(2016,4,12) // positivo
new Date(2016,5,1) -  new Date(2016,5,1) // 0
```

E se quisermos uma ordem decrescente? Só inverter a subtração:

```
let negociacoes = [
    new Negociacao(new Date(), 7, 200),
    new Negociacao(new Date(), 1, 300),
    new Negociacao(new Date(), 8, 100)
]
negociacoes.sort((a, b) => b.quantidade -  a.quantidade); // agora é b menos a!
negociacoes.forEach(negociacao => console.log(negociacao));
```

Agora que você já sabe definir um critério de ordenação para `Array.sort`, vamos voltar para o método `ordena` de `NegociacaoController`.

Implementando nossa solução

Já sabemos como ordenar um Array segundo um critério, mas o problema é que ao acessarmos `this._listaNegociacoes.negociacoes` nós recebemos uma cópia da lista original e qualquer alteração na lista não afeta a instância de `ListaNegociacoes` (ainda lembra da programação defensiva?). Para resolvermos isso, vamos criar o método `ordena` em `ListaNegociacoes`. Este método receberá o critério de ordenação, que será passado para a lista de negociações encapsulada pela classe:

```
// aluraframe/client/js/app/models/ListaNegociacoes.js

class ListaNegociacoes {

    // código anterior omitido

    // novo método!
    ordena(criterio) {
        this._negociacoes.sort(criterio);
    }
}
```

Agora, vamos voltar para `NegociacaoController` e alterar seu método `ordena` e implementá-lo:

```
class NegociacaoController {

    // código anterior omitido

    ordena(coluna) {
        this._listaNegociacoes.ordena((a, b) => a[coluna] - b[coluna]);
    }
}
```

Veja que interessante. Não podemos fazer `a.quantidade` ou `a.data`, porque a propriedade usada no critério de ordenação é escolhida pelo usuário. Sendo assim, usamos a sintaxe `objeto[nomePropriedade]` para acessar a propriedade do objeto. Essa

forma mais verbosa é interessantíssima quando queremos acessar as propriedades de um objeto dinamicamente .

Apesar de termos feitos essas mudanças, nada acontecerá. Precisamos atualizar a view quando o método `ordena` do nosso modelo for chamado, para isso, precisamos adicioná-lo na lista de métodos ou propriedades que desejamos monitorar do nosso modelo. Alterando `NegociacaoController` :

```
class NegociacaoController {

    constructor() {
        // propriedades omitidas

        this._listaNegociacoes = new Bind(
            new ListaNegociacoes(),
            new NegociacoesView($('#    negociacoesView')),
            'adiciona', 'esvazia', 'ordena');

        // outras propriedades omitidas
    }
}
```

Perfeito, faça um teste agora. Alterne cliques em algumas colunas e veja o resultado. Gostou? Contudo, nossa solução está incompleta. Precisamos efetuar uma ordenação ascendente ou descendente quando o usuário clicar na mesma coluna. Como implementar isso?

Há sempre uma solução

A lógica é seguinte. Se a ordenação atual é X e ele clicou em outra coluna, trocando a ordenação para Y, não fazemos nada e deixamos a lista ser ordenada por Y. No entanto, se a ordenação atual é X e ele clica na coluna que solicita novamente uma ordenação por X, invertemos a ordem atual.

Vamos criar como propriedade de `NegociacaoController` a propriedade `this._ordemAtual` .

```
class NegociacaoController {

    constructor() {
        this._ordemAtual = ''; // quando a página for carregada, não tem critério. Só passa a ter quand
    }
    // código posterior omitido
}
```

```
class NegociacaoController {

    // código anterior omitido

    ordena(coluna) {
        if(this._ordemAtual == coluna) {
            // inverte a ordem da lista!
        } else {
```

```
            this._listaNegociacoes.ordena((a, b) => a[coluna] - b[coluna]);
        }
        this._ordemAtual = coluna;
    }
}
```

E para invertermos a lista? Precisamos criar em nosso modelo `ListaNegociacoes` o método `inverteOrdem`, que chama `this._negociacoes.reverse()` para nós:

```
// aluraframe/client/js/app/models/ListaNegociacoes.js

class ListaNegociacoes {

    // código anterior omitido

    inverteOrdem() {
        this._negociacoes.reverse();
    }
}
```

Agora, podemos terminar nossa controller com esta última alteração:

```
class NegociacaoController {

    // código anterior omitido

    ordena(coluna) {
        if(this._ordemAtual == coluna) {
            this._listaNegociacoes.inverteOrdem();
        } else {
            this._listaNegociacoes.ordena((a, b) => a[coluna] - b[coluna]);

        }
        this._ordemAtual = coluna;
    }
}
```

E claro, não podemos nos esquecer de adicionar o método `inverteOrdem` como um dos métodos que estamos monitorando para atualizar automaticamente a `View`:

```
class NegociacaoController {

    constructor() {
        // propriedades omitidas

        this._listaNegociacoes = new Bind(
            new ListaNegociacoes(),
            new NegociacoesView($('#negociacoesView'),
            'adiciona', 'esvazia', 'ordena', 'inverteOrdem');
```

```
            // outras propriedades omitidas
        }
    }
```

Perfeito! Experimente agora brincar com os critérios de ordenação clicando "igual a um louco" nos cabeçalhos das colunas da nossa tabela.