02

Associando métodos do controller às ações do usuário

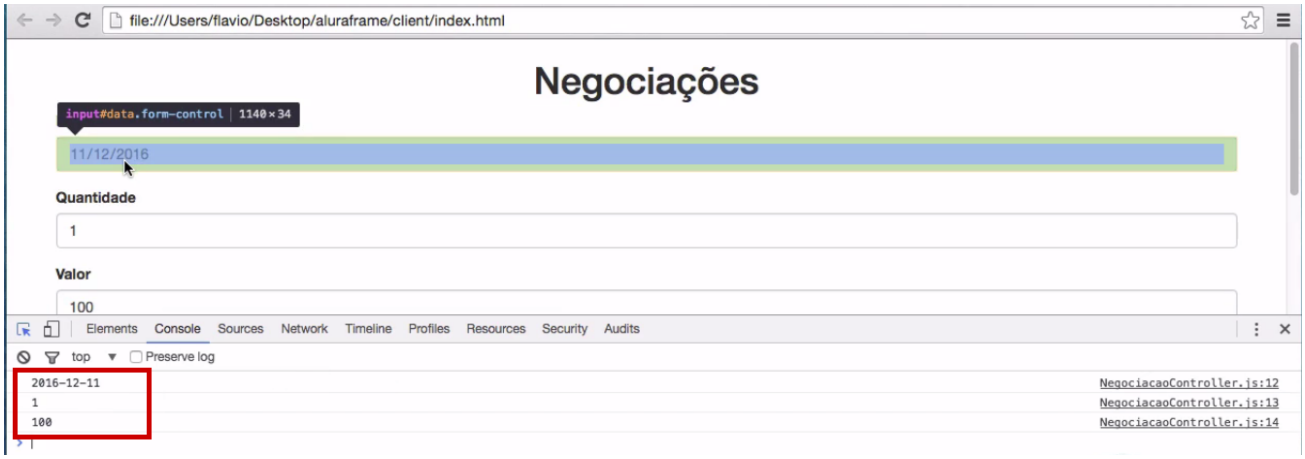
Transcrição

Ao clicarmos no botão "Incluir" do formulário, já conseguimos executar as ações da `controller`. Agora, precisamos capturar os dados preenchidos. Primeiramente, iremos exibi-los no Console. Para isto, criaremos a variável `inputQuantidade`.

```
class NegociacaoController {

  adiciona(event) {
        event.preventDefault();

        let inputData = document.querySelector('#data');
        let inputQuantidade =  document.querySelector('#quantidade');
        let inputValor = document.querySelector('#valor');

        console.log(inputData.value);
        console.log(inputQuantidade.value);
        console.log(inputValor.value);

  }
}
```

O `document.querySelector()` será o responsável por buscar no DOM os elementos com id `#data`, `#quantidade` e `#valor` - observe que conseguimos utilizar os seletores CSS. Os itens foram dispostos na ordem do formulário. Lembrando que cada `input` tem `id` no arquivo `index.html`.

Vamos executar o código para ver se todos os dados aparecem no Console do navegador. Preencheremos o formulário com a data `11/12/2016`, a quantidade `1` e o valor `100`.



Os dados apareceram corretamente no Console, nós já conseguimos capturá-los. Mas veja que repetimos bastante o código, ficou uma sintaxe trabalhosa de ser digitada. Como em JavaScript temos as *First Class Functions*, podemos declarar a variável `$` - como usado no jQuery - e dentro, jogaremos o `document.querySelector`. Poderemos fazer as seguintes substituições no código:

```
class NegociacaoController {

  adiciona(event) {
        event.preventDefault();

        let $ = document.querySelector;
        let inputData = $('#data');
        let inputQuantidade =  $('#quantidade');
        let inputValor = $('#valor');

        console.log(inputData.value);
        console.log(inputQuantidade.value);
        console.log(inputValor.value);

  }
}
```

Desta forma, ficou menos trabalhoso escrever o código. Mas se o executarmos como está, veremos uma mensagem de erro no navegador.

Não funcionou colocarmos o `querySelector` na variável `$` para criarmos um *alias*. Por que não funcionou? O `querySelector` é uma função que pertence ao objeto `document` - chamaremos tal função de método. Internamente, o `querySelector` tem uma chamada para o `this`, que é o contexto pelo qual o método é chamado. Logo, o `this` é o `document`. No entanto, quando colocamos o `querySelector` dentro do `$`, ele passa a ser executado fora do contexto de `document` e isto não funciona. O que devemos fazer, então? Queremos tratar o `querySelector` como uma função separada. Nós queremos que ao colocarmos o `querySelector` para o `$`, ele mantenha a associação com o `document`. Para isto, usaremos o `bind()`:

```
class NegociacaoController {

  adiciona(event) {
        event.preventDefault();

        let $ = document.querySelector.bind(document);
        let inputData = $('#data');
        let inputQuantidade =  $('#quantidade');
        let inputValor = $('#valor');

        console.log(inputData.value);
        console.log(inputQuantidade.value);
        console.log(inputValor.value);

  }
}
```

Agora, estamos informando que o `querySelector` irá para a variável `$`, mas ainda manterá uma associação com `document`. Desta vez, o código funcionará corretamente no navegador. Vimos um truque que copia um pouco o que o jQuery faz de melhor e cria um "mini-framework", ao associarmos a variável `$` com o `querySelector` - um seletor CSS - e mantendo a ligação com o `document`.

Até aqui, vimos como manipular os dados do DOM, além do truque citado para otimizar nosso código. Precisaremos ainda construir uma negociação com base nestes dados. Mas será que esta forma de organizar os dados é performática?

Veremos isto mais adiante.