alura

Removendo itens e navegação em elementos

Agora que já temos o valor total e a quantidade dos itens no nosso carrinho, vamos criar um link para remover um produto. Na nossa página, criamos mais um `<td>` para as ações e dentro dessa tag vamos criar um link Remover.

```
<td><a href="">(Remover)</td>
```

Copiamos e colamos o código em todas as linhas de nosso carrinho.

Mas quando alguém clicar nesse link, queremos fazer alguma coisa. Para conseguir fazer isso precisamos identificar esse elemento, como são vários, vamos utilizar **class**:

```
<td><a class="remove-item" href="">(Remover)</td>
```

Ficando então com o código da última linha, por exemplo:

```
<tr>
    <td></td>
    <td>Nome</td>
    <td>Quantidade</td>
    <td>Valor</td>
    <td>Total</td>
    <td><a class="remove-item" href="">(Remover)</td>
</tr>
```

Implementamos a ação no jQuery, selecionando todos o elementos com a classe **remove-item** e adicionamos um evento de **click** que ira chamar uma função. Lembrando que devemos fazer isso dentro do método **aposInicializado**.

```
$(".remove-item").click(removeItem);
```

Vamos criar essa função que se chamará **removeItem**.

```
var removeItem = function() {

};
```

Como a função é remover um item do carinho, teremos que subtrair 1 no valor do nosso total de produtos. Primeiro lemos a quantidade total:

```
var quantidadeComoString = $("#quantidade-de-itens").text();
```

Lembrando que devemos converter para inteiro

```
var quantidadeComoString = $("#quantidade-de-itens").text();
var atual = parseInt(quantidadeComoString);
```

O novo total de produtos é o nosso **atual** menos um:

```
var novaQuantidade = atual - 1;
```

Escrevemos de volta para o usuário o valor atualizado:

```
$("#quantidade-de-itens").text(novaQuantidade);
```

Ficando com o código final:

```
var removeItem = function() {
    var quantidadeComoString = $("#quantidade-de-itens").text();
    var atual = parseInt(quantidadeComoString);
    var novaQuantidade = atual - 1;
    $("#quantidade-de-itens").text(novaQuantidade);
};
```

Quando testamos esse código, percebemos que o valor não é alterado e que a página é atualizada. Mas por qual motivo isso acontece? Toda vez que clicamos no link ele nos redireciona para uma página, como não passamos nenhuma ele entende que é para mesma. É preciso cancelar esse redirecionamento do link, não queremos atualizar a página! Queremos **evitar** o comportamento **padrão**. Fazemos isso na nossa função que **previne** o **padrão**, o **preventDefault** que cancela o comportamento nativo do elemento.

Recebemos como argumento na função esse evento e através dele conseguimos usar o preventDefault.

```
var removeItem = function(event) {
    event.preventDefault();
    // resto do código
};
```

Testamos o código e vemos que o valor total de produtos esta diminuindo, mas o produto, o botão, ainda esta lá.

Então voltamos para nosso código e implementamos essa remoção. No JavaScript existe o **this** que representa o elemento onde o evento está ocorrendo. Como colocamos o evento no botão, o **this** é o botão. Utilizamos o método **remove** e ele retira o elemento da página:

```
var removeItem = function(event) {
    event.preventDefault();
    var self = $(this);
    self.remove();
    // resto do código
}
```

Atualizamos a página e removemos um item. Ele diminuiu o total de produtos, removeu o botão, mas não removeu a linha (o nosso produto). Como assim? Claro, pedimos para remover o elemento onde ocorreu o evento - o nosso botão.

Desejamos remover a tag **tr** inteira! Isto é, desejamos remover o nosso avô. O avô do **this**. Mas não existe um método chamado **granfather** ou similar. Como acessar a tag avô da tag atual? Para isso temos o método **parent**, que acessa a tag pai. Para acessar o avô, fazemos **self.parent().parent()**.

```
var removeItem = function(event) {
    event.preventDefault();
    var self = $(this);
    self.parent().parent().remove();
    // resto do código
}
```

Atualizamos a página e tudo está funcionando como esperávamos. Mas espera um instante, **parent.parent** parece ser algo muito frágil. Já vimos anteriormente que nos atrelar a estrutura das nossas tags faz com que qualquer mudança leve de design quebre todo nosso javascript. Não queremos isso. Desejamos criar um código javascript que aceite algumas mudanças de design. Como fazer então para procurar, ao invés de meu avô, a primeira tag **tr** que é meu pai, ou meu avô, ou bisavô etc? Para procurar seu ancestral mais próximo com determinadas características usamos o método **closest**:

```
var tr = self.closest("tr");
```

Navegando na árvore de hierarquia e encontrando o elemento mais próximo em relação ao **this** resolvemos nosso problema de uma maneira mais maleável. Vamos usá-lo passando o **tr** como mencionado e em seguida colocamos **remove** para remover o elemento.

```
var removeItem = function(event) {
    event.preventDefault();

    var self = $(this);
    self.closest("tr").remove();
```

```
    // resto do código
}
```

Testamos e agora o botão remover além de atualizar a quantidade de produto também remove o produto do carrinho. Agora só falta atualizar o valor total a ser pago que veremos na nossa próxima aula.

Ver video | Começar exercícios