01

Evoluindo a Garconapp

Transcrição

Seguindo a implementação da Garçonapp, o próximo passo será criarmos a marcação do pedido efetivamente. Nós temos uma lista de produtos e quando a garçonete clicar em um produto, nós queremos que apareça no aplicativo qual item foi marcado e quantas vezes ele foi pedido, com um número visível ao lado. Para fazer isto, vamos trabalhar com um componente chamado `badge`, que vem com o *Materialize*.

Vamos ver como ficará no código. Usaremos o exemplo com o elemento "Bolo com Nutella".

```
<a class="collection-item waves-effect black-text">
    Com Nutella
</a>
```

Iremos adicionar um novo elemento nesta linha, a *tag* `span`, onde estará a classe `badge`. Podemos configurar a cor e usar um `brown-text`.

```
<a class="collection-item waves-effect black-text">
    Com Nutella
    <span class="badge brown-text">1</span>
</a>
```

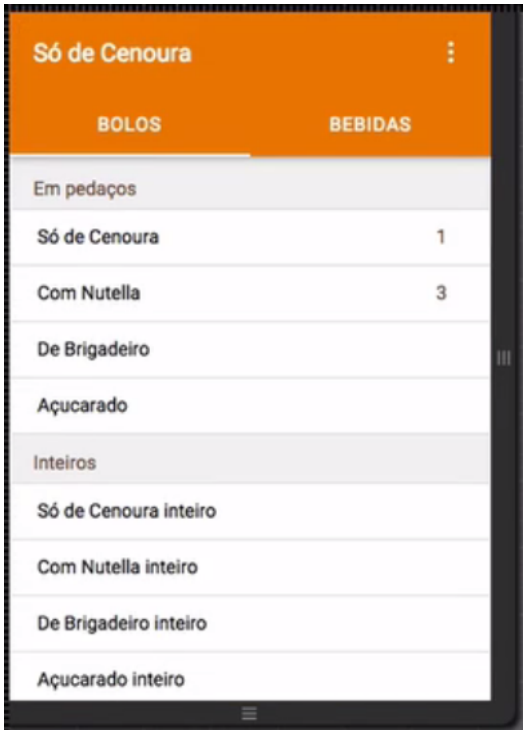
Podemos incluir quantas instâncias. Se clicarmos uma vez sobre o bolo, ao lado do item, irá aparecer um número 1 no app. Se quisermos pedir três, teremos o número 3 ao lado do item.

```
<a class="collection-item waves-effect black-text">
    Com Nutella
    <span class="badge brown-text">3</span>
</a>
```

Vamos acrescentar um "Bolo só de cenoura". Vamos adicionar o número 1 ao lado do item.

```
<a class="collection-item waves-effect black-text">
    Só de Cenoura
    <span class="badge brown-text">1</span>
</a>
```

Se testarmos o app no navegador, veremos o número 1 ao lado do item "Só de Cenoura" e o número 3 ao lado do item "Com Nutella".



Mas nós iremos escrever HTML para fazer o pedido? Não. Já entendemos como funciona o componente, mas nosso objetivo é que ele coloque o `badge`, quando clicarmos sobre os itens. Inclusive, queremos que ele incremente, se clicarmos novamente no mesmo produto. Vamos transformar o `badge` em algo mais dinâmico, por isso, trabalharemos com o Javascript.

Primeiramente, iremos criar o arquivo novo, o `app.js`, e vou importá-lo para o arquivo HTML:

```
<script src="js/jquery.min.js"></script>
<script src="js/materialize.min.js"></script>
<script src="js/app.js"></script>
```

Este arquivo irá ajudar na implementação. Faremos isto com o jQuery. Na aba JS, usaremos todos os `collection-item`. Especificaremos que ele o `badge` seja usado quando um item for clicado.

```
$('.collection-item').on('click', function(){
    var $badge;
    $badge = $('<span class="badge brown-text">1</span>');
});
```

Este é o HTML que criamos anteriormente. Mas iremos começar com o 0.

```
$('.collection-item').on('click', function(){
    var $badge;
    $badge = $('<span class="badge brown-text">0</span>');
});
```

Queremos colocá-lo agora no elemento `collection-item`. Para isto, usaremos o `appendTo(this)` - referente ao elemento clicado atualmente.

```
$('.collection-item').on('click', function(){
    var $badge;
    $badge = $('<span class="badge brown-text">0</span>').appendTo(this);
});
```

A implementação ainda não está completa, mas se testarmos o app no navegador, veremos que ele adiciona o número 0 ao lado item, quando clicado. Mas se clicarmos novamente, ele não irá incrementar e permanecerá marcado com 0. O pior é que ele criará vários `badge` s no código. Teremos que continuar trabalhando nisto.

Começaremos por descobrir se existe algum item com a classe `badge` dentro do elemento atualmente clicado. Se (`if`) não existir, isto significa que a propriedade `length` virá com 0. Nós iremos criar o `badge` novo apenas se não existir um antigo.

```
$('.collection-item').on('click', function(){
    var $badge = $('.badge', this);
    if ($badge.length == 0) {
        $badge = $('<span class="badge brown-text">0</span>').appendTo(this);
    }
});
```

Em ambas as situações, queremos incrementar o valor do `badge` , então iremos pegar o valor dele com o jQuery. Vamos transformá-lo em um número com `parseInt` e somaremos com mais um. Em seguida, colocamos de volta no `badge.text` .

```
$badge.text(parseInt($badge.text()) + 1);
```

Colocamos dentro do `badge` o texto que significa o texto anterior somado com 1.

Se testarmos o app no navegador, veremos que quando clicamos mais de uma vez no mesmo item, ele irá indicar a quantidade de cliques dados no produto. Criamos uma maneira fácil de anotar os pedidos. Em seguida, precisamos encontrar uma forma de efetivar o pedido. Para que isto aconteça, iremos adicionar um botão chamado `floating action button` , que ficará no canto direito abaixo, aonde você finalizará a ação.

Vamos criar o botão. No arquivo `index.html` , abaixo da lista com os `collection-item` , iremos adicionar um `div` que irá receber a classe vinda do *Materialize*, `fixed-action-btn` .

```
<div class="fixed-action-btn"></div>
```

Vamos criar um novo link com `href` que terá algumas classes especiais, `btn-floating` . Podemos configurar o tamanho com `btn-large` e adicionar o efeito `waves-effect waves-light` . Para adicionar a cor marrom, usaremos a classe `brown` . Inicialmente, vamos usar um botão simples, em que visualizaremos `xx` .

```
<div class="fixed-action-btn">
    <a href=""
        class="btn-floating btn-large waves-effect
        waves-light brown">xx</a>
</div>
```

Agora ele criou um botão marrom redondo no aplicativo, que quando clicamos tem um efeito de onda.

Para adicionar um ícone no botão, precisamos adicionar uma *tag* `i` com a classe `material-icons`. Vamos utilizar o ícone `done` que tem o símbolo de `V`.

```
<div class="fixed-action-btn">
    <a href=""
        class="btn-floating btn-large waves-effect
            waves-light brown">
        <i class="material-icons"></i>
    </a>
</div>
```

Após marcados quais itens foram pedido, iremos clicar no botão marrom. Mas antes, queremos que o app forneça um resumo do pedido que foi feito e que solicite a informação sobre o número da mesa. Por isso, vamos implementar um `modal` que fará estas ações.

No HTML, criaremos mais um `div` que terá a classe `modal`. Nós utilizaremos o do tipo `modal-fixed-footer`, em que o rodapé será fixo. Vamos adicionar também outro `div` s. um com `modal-content` (com o conteúdo com do modal) e outro com `modal-footer` (com o rodapé).

```
<div class="modal modal-fixed-footer">
    <div class="modal-content">
        conteudo
    </div>
    <div class="modal-footer">
        rodape
    </div>
</div>
```

Como ele saberá que se clicarmos no botão de cima, ele terá que abrir no que está posicionado abaixo? Ele ainda não sabe. Então, teremos que criar o `id="confirmacao"`. Usaremos o mesmo `id` no link do botão que quero que abra o modal.

```
<div class="fixed-action-btn">
    <a href="#confirmacao"
        class="btn-floating btn-large waves-effect
            waves-light brown">
        <i class="material-icons"></i>
    </a>
</div>

<div class="modal modal-fixed-footer"
    id="confirmacao">
    <div class="modal-content">
        conteudo
    </div>
    <div class="modal-footer">
        rodape
    </div>
</div>
```

Precisaremos voltar no arquivo JS e inicializar o `modal`, com um `modal-trigger`.

```
$('.modal-trigger').leanModal();
```

Ele irá inicializar o plugin de modal do Materialize.

> Observação: se você baixar as versões mais recentes do Materialize direto no site deles ao invés de seguir com os arquivos do curso, é importante observar que essa função foi renomeada para apenas `.modal()`

Vamos fazer um novo teste no navegador. Porém, ao clicarmos no botão de finalização do pedido, não teremos o resultado esperado. Isto, porque faltou um detalhe: no HTML, precisamos indicar na linha do botão que estamos usando um `modal-trigger`. Ele será o responsável por abrir o modal.

```
<div class="fixed-action-btn">
    <a href="#confirmacao"
        class="btn-floating btn-large waves-effect waves-light brown modal-trigger">
        <i class="material-icons">done</i>
    </a>
</div>
```

Agora, irá abrir um pequena tela com o conteúdo e um rodapé simples. Iremos aproveitá-lo com o conteúdo do pedido, mais adiante.

Onde estava escrito `conteudo` no nosso código, vamos substituir por `Resumo do Pedido` e iremos acrescentar quais itens foram pedidos pelo cliente. Vamos adicionar em seguida, um elemento que já vem pronto com o Materialize: o `blockquote`. Depois, daremos um `id` para referenciá-lo, junto com `resumo`. Para incluirmos o resumo do pedido, precisaremos do JS.

```
<div class="modal modal-fixed-footer"
    id="confirmacao">
    <div class="modal-content">
        <h5>Resumo do pedido</h5>

        <blockquote id="resumo"></blockquote>
    </div>
    <div class="modal-footer">
        rodape
    </div>
</div>
```

Iremos adicionar também um `input` e o funcionário poderá incluir o número da mesa. Ao adicionar o `id`, poderemos recuperá-lo no Java Script

```
<input type="number" class="validate" placeholder="Número da mesa" id="numero-mesa">
```

O nosso código ficará assim:

```
<div class="modal modal-fixed-footer"
    id="confirmacao">
    <div class="modal-content">
        <h5>Resumo do pedido</h5>
        <input type="number" class="validate" placeholder="Número da mesa" id="numero-mesa">
        <blockquote id="resumo"></blockquote>
    </div>
    <div class="modal-footer">
        rodape
    </div>
</div>
```

No navegador, veremos que o app já irá apresentar o "Resumo do pedido", além do campo "Número da mesa". A vantagem de utilizarmos o `type=number` é que ele irá apresentar o teclado numérico no celular - no nosso caso, representa uma grande facilidade.

Agora queremos que os itens selecionado apareçam no corpo do "Resumo". Faremos isto com jQuery.

Quando alguém clicar no link do `floating-action-button` - o botão `done` - iremos criar a tela do resumo. Para isto, iremos incluir o `id=confirmar` e assim, poderemos usá-lo no arquivo JS.

```
<div class="fixed-action-btn">
    <a href="#confirmacao" id="confirmar"
        class="btn-floating btn-large waves-effect waves-light brown modal-trigger">
        <i class="material-icons">done</i>
    </a>
</div>
```

Vamos incluir `#confirmar` no código, e iremos configurar o resumo. Nosso objetivo, é mudar o texto dele para `os produtos selecionados` . Então, quando clicarmos no botão de confirmação do pedido, iremos receber a mensagem: "os produtos selecionados são...". Ela ira se referir aos produtos que efetivamente foram selecionados .

```
$('#confirmar').on('click', function() {
    $('#resumo').text('os produtos selecionados');
});
```

Como iremos identificar quais itens foram pedidos? Eles serão os que possuem um elemento `badge` , logo, basta selecionarmos todas as vezes que este aparecer no projeto. O `badge` irá se referir à quantidade, mas como identificaremos o item. O `badge` estará dentro da *tag* `a` . Então, usaremos o `parent` no nosso código e depois iremos percorrer a lista.

```
$('#confirmar').on('click', function() {
    $('.badge').parent().each(function() {
    });
    $('#resumo').text('os produtos selecionados');
});
```

Na prática, ao fazermos um teste no *Console* (do Inspect), quando selecionarmos $('.badge') e não tivermos nenhum produto selecionado no app, não acontecerá nada. Mas se marcarmos por exemplo três produtos, aparecerão três

elementos no *Console*.

```
$('.badge')
[<span class="badge brown-text">1</span>;
<span class="badge brown-text">1</span>;
<span class="badge brown-text">1</span>;]
```

Se selecionarmos `$('.badge').parent()`, irão aparecer os itens que estão acima, como o produto "Só de Cenoura".

```
$('.badge').parent()
[<a class="collection-item waves-effect black-text">
    "Só de Cenoura"
    <span class="badge brown-text">1</span>
</a>]
```

Queremos agora que identificar em cada um destes links, qual é o produto e qual é a quantidade. Para isto criaremos uma variável `produto`, na qual o elemento atual (`this`) será o `firstChild.textContent`. Criaremos também a variavel `quantidade` que será o `lastChild.textContent`. Podemos incluir um `alert` para observarmos o processo sendo realizado.

```
$('.badge').parent().each(function() {
        var produto = this.firstChild.textContent;
        var quantidade = this.lastChild.textContent;

        alert(produto + quantidade)
    });
```

Testaremos nosso app no navegador. Ao marcamos duas vezes o produto "Só de Cenoura" duas vezes, veremos surgir a mensagem do `alert`:

Só de Cenoura 2

Nosso objetivo é que esta mensagem apareça dentro da tela do "Resumo de pedido". Para isto, criaremos a variável `texto`, que começará vazia, e vamos acrescentar o `produto` e a `quantidade` na nova `var`. Iremos configurar também a mensagem de texto.

```
var texto = ' ';

$('.badge').parent().each(function() {
    var produto = this.firstChild.textContent;
    var quantidade = this.lastChild.textContent;

    texto += produto + ': ' + quantidade + ', ';
});
```

Ao clicarmos o botão de confirmação do pedido, ja veremos a tela de Resumo com a mensagem: "Só de Cenoura: 1. Com Nutella: 1, De Brigadeiro: 1...". Agora o app apresentar um resumo correto.

Por último, queremos incluir dois botões no rodapé que deixarão o app mais bonito, como os de "Cancelar" e "Efetuar pedido", por exemplo. Voltaremos para o HTML, no trecho referente ao `modal` do rodapé e criaremos novos botões: "Pedir" e "Cancelar". No código, iremos incluir a classe `modal-close` , assim, quando alguém clicar nos botões, o resumo será fechado automaticamente.

```
<div class="modal-footer">
    <button class="btn deep-orange waves-effect waves-light modal-close">
        Pedir
    </button>

    <button class="btn-flat waves-effect waves-red modal-close">
        Cancelar
    </button>
</div>
```

Os dois terão pequenas diferenças no design. O botão de "Cancelar" irá receber ondas vermelhas ao ser clicado, recurso muito usado em elementos com a mesma função. A tela com o resumo fechará após clicarmos em algum dos botões, mas ainda precisamos implantar suasfuncionalidades.

Um detalhe que faltou no app: a opção de fazermos alterações no pedido. Então, queremos encontrar uma forma de remover o `badge` que inserido, quando clicamos em algum produto.

Faremos esta implementação no arquivo JS. Abaixo do `modal-trigger` , iremos adiconar o `collection` . Ao clicarmos sobre o `badge` dentro dela - no número que indica a quantidade no app - depois de ter sido selecionado, o produto será desmarcado.

```
$('.collection').on('click', '.badge', function() {
    $(this).remove();
    return false;
});
```

Por exemplo, se fizemos o pedidos de um produto, ao clicarmos sobre o número que indica a quantidade, ele irá desfazer o pedido.

Em seguida, iremos incluir a opção "Cancelar" no Menu do app - posicionado no canto superior direito. Para isto, iremos criar outro JS que chamaremos de `acao-limpar` . a classe irá limpar o número da mesa e removerá todos os elementos `badges` que existem no sistema.

```
$('.acao-limpar').on('click', function() {
    $('#numero-mesa').val( '' );
    $('.badge').remove();
});
```

Para testarmos isto, adicionaremos a classe `acao-limpar` no botão de "Cancelar" no "Resumo do pedido", que criamos no arquivo HTML.

```
<button class="acao-limpar btn-flat waves-effect waves-red modal-close">
    Cancelar
</button>
```

Assim, o botão "Cancelar", além de fechar modal, ele irá limpar as anotações do app.

Se fizermos um teste, após marcamos alguns itens e confimarmos o pedido, ele irá para o resumo. Quando clicarmos no botão de "Cancelar", ele voltará para a lista de produtos, mas sem nenhuma marcação.

Agora, incluiremos a mesma ação em uma opção do Menu. O trecho do código referente ao Menu, está no topo do HTML. Iremos criar um `ul`, com a classe `dropdown-content`. Trata-se de uma lista de opções que podemos adicionar no Menu da lateral. Por enquanto, iremos incluir apenas a opção "Limpar", que terá a classe `acao-limpar`. Quando alguém acionar o botão, ele limpará as marcações do app.

```
<ul class="dropdown-content">
    <li><a class="black-text acao-limpar">Limpar</a></li>
</ul>
```

Criaremos com isto, um menu que será aberto ao clicarmos no ícone das três bolinhas verticais. Vamos incluir no código um `id= submenu` - da mesma forma como fizemos no modal.

```
<ul class="dropdown-content" id="submenu">
```

Na linha com a *tag* `i`, referente ao ícone, adicionaremos `data-activates="submenu"` e configurações, como o `data-gutter` e `data-contrainwidth`.

```
<div>
    <i data-activates="submenu" data-gutter="5"
        data-contrainwidth="false"
        class="material-icons waves-effect waves-light waves-circle">more_vert</i>
</div>
```

Nas classes do ícone, iremos adicionar também a classe nova `dropdown-button`.

```
<i data-activates="submenu" data-gutter="5"
    data-contrainwidth="false"
    class="material-icons waves-effect waves-light waves-circle dropdown-button">more_vert</i>
```

Com isto, estaremos indicando que ao clicarmos no ícone, ele acionará um *dropdown*, que o `id` é o `submenu`. Ele terá apenas um link: o botão "Limpar".

Ao clicarmos no ícone da lateral superior do app, um Menu será aberto com a opção "Limpar". Quando clicamos na opção, ele limpará todas as anotações nos produtos anteriormente selecionados.

Temos um menu com um item único, porém, se temos a possibilidade de incluir novas opções. É o que faremos mais adiante.

Observe que já implementamos a funcionalidade do pedido, da listagem dos produtos selecionados, só nos falta efetivar o pedido. Para isto, teremos que acionar um ***Ajax***. Por enquanto, iremos focar no design e componentes do app. A parte do *Ajax*, veremos mais adiante.