

INTENSIVÃO DE JAVASCRIPT

Apostila Completa Aula 4

Guia passo a passo para construir seu próprio site
de E-Commerce a partir do zero!



Parte 1

Revisando Primeiros Passos do Projeto



Revisando Primeiros Passos do Projeto

Para começar, vamos relembrar os passos iniciais do nosso projeto. Abra a pasta do projeto no seu ambiente de desenvolvimento. Agora, clique com o botão direito do mouse e selecione a opção "Abrir Terminal" (no caso do Linux/Mac) ou "Abrir PowerShell" (no caso do Windows).

Dentro do terminal ou PowerShell, vamos executar um comando importante: **npm run dev**. Esse comando irá iniciar o Vite, que é a ferramenta que estamos utilizando para desenvolver o nosso site. O Vite irá preparar tudo e colocar o nosso site no ar.

Agora, com o site em execução, abra o seu navegador e digite o endereço correspondente à página inicial do site. Com isso, você poderá ver o resultado atual do seu projeto.

A grande vantagem é que qualquer mudança que fizermos no projeto a partir de agora será imediatamente refletida no navegador. Isso significa que podemos experimentar e fazer alterações no código, e ver como essas mudanças afetam o site em tempo real.

```
PS C:\Users\OneDrive\Documentos\Projetos\Magazine Hashtag> npm run dev

> magazine-hashtag@0.0.0 dev
> vite

VITE v4.4.9 ready in 588 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

Parte 2

Aprimorando a Funcionalidade do Carrinho de Compras

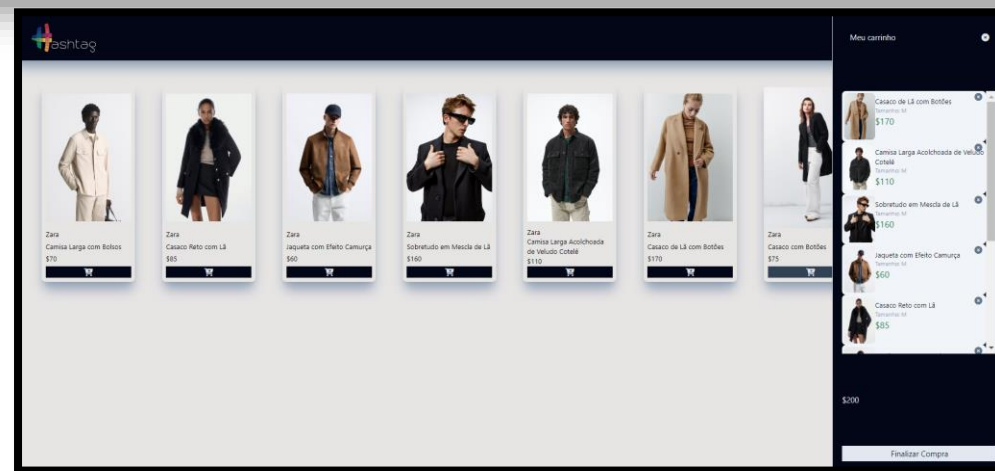


Aprimorando a Funcionalidade do Carrinho de Compras

Vamos fazer algumas melhorias no nosso carrinho de compras. Primeiro, iremos estabelecer um limite para o espaço onde os cartões dos produtos são exibidos. Isso ajudará a manter tudo organizado. Além disso, vamos garantir que exista um espaço adequado entre cada cartão, para que não fiquem muito próximos uns dos outros. Por fim, vamos aplicar alguns estilos especiais do Tailwind para dar um visual mais atraente e profissional ao carrinho de compras.

Dentro do arquivo index.html, na **tag <section>** de id **"produtos-carrinho"** vamos adicionar a **classe h-3/5**, que define a altura de um elemento como 60% da altura do seu contêiner pai, e utilizaremos a **classe overflow-y-auto** permite a rolagem vertical dentro de um elemento quando o conteúdo excede o espaço disponível.

```
27 <section id="produtos-carrinho" class="h-3/5 overflow-y-auto"></section>
28 <p id="preco-total">$200</p>
29 <button class="bg-slate-200 text-slate-900 p-1">Finalizar Compra</button>
30 </section>
```

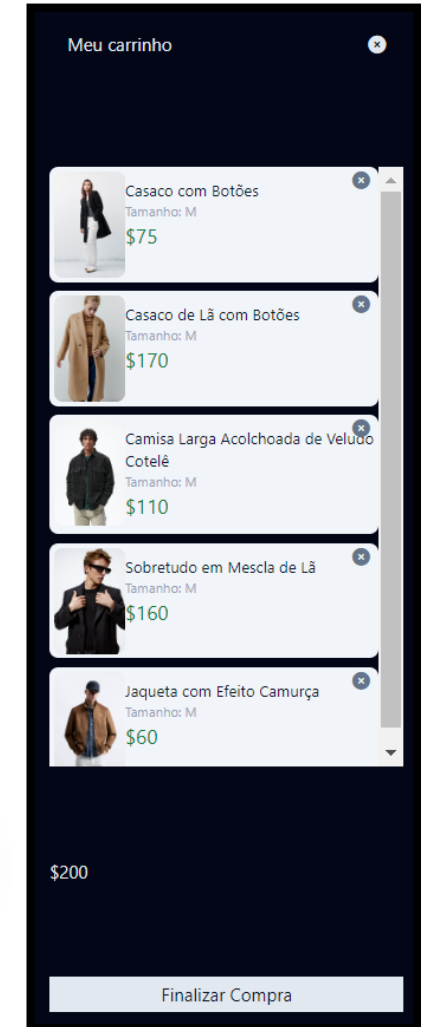


Aprimorando a Funcionalidade do Carrinho de Compras

Agora vamos aplicar algumas classes, ainda dentro da nossa **<section>**, que já utilizamos em alguns elementos do nosso projeto, vamos relembrar os seus conceitos:

- **flex:** Cria um contêiner de elementos flexíveis, permitindo o alinhamento e dimensionamento flexíveis.
- **flex-col:** Define a direção dos elementos filhos como coluna, organizando-os verticalmente.
- **gap-2:** Adiciona um espaço de 0.5rem (8px) entre os elementos filhos.

```
27 <section id="produtos-carrinho" class="h-3/5 overflow-y-auto flex flex-col gap-2"></section>
```



Aprimorando a Funcionalidade do Carrinho de Compras

Agora vamos fazer alterar a estrutura do nosso cartão de produtos, dentro do **arquivo menuCarrinho.js**, vamos adicionar ao elemento **<div>**, que encapsula as informações de nome, preço e tamanho do nosso produto, as seguintes classes do tailwind:

```
33 <div class="p-2 flex flex-col justify-between">
```

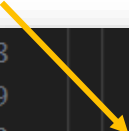
Para aprimorar como o carrinho de compras funciona, vamos fazer com que um produto seja adicionado somente uma vez. Se o usuário optar por adicionar o mesmo produto várias vezes, nós vamos controlar a quantidade. Para isso, vamos incorporar botões que permitam aumentar e diminuir a quantidade, exibindo o número total de vezes que o produto foi adicionado.

```
38 <div class="flex text-slate-950 items-end absolute bottom-0 right-2">
39   <button>-</button>
40   <p>2</p>
41   <button>+</button>
42 </div>
```

Aprimorando a Funcionalidade do Carrinho de Compras

Ao escrevermos HTML dentro de um texto Javascript, precisamos construí-lo com boas práticas e uma delas é o uso de **identação**.

A **identação** em HTML se refere à organização hierárquica do código, onde elementos filhos são recuados com espaços ou **tabs** para melhorar a legibilidade.



```
38 <div class="flex text-slate-950 items-end absolute bottom-0 right-2">
39   <button>-</button>
40   <p>2</p>
41   <button>+</button>
42 </div>
```

Vamos ver os conceitos das classes que vamos utilizar para estilizar container que engloba nossos botões:

- **text-slate-950**: Define a cor do texto como uma tonalidade específica de cinza.
- **items-end**: Alinha os elementos filhos no extremo inferior do contêiner.
- **absolute**: Remove o elemento do fluxo normal e o posiciona de forma absoluta em relação ao contêiner pai.
- **bottom-0**: Alinha o elemento no canto inferior do contêiner pai.
- **right-2**: Posiciona o elemento a uma distância de 0.5rem (8px) a partir da borda direita do contêiner pai.

Aprimorando a Funcionalidade do Carrinho de Compras

```
38 <div class="flex text-slate-950 items-end absolute bottom-0 right-2 text-lg">
39   <button>-</button>
40   <p class="ml-2">2</p>
41   <button class="ml-2">+</button>
42 </div>
```

Continuando a estilização, utilizamos as classes `ml-2` `text-lg`:

- **ml-2**: Adiciona uma margem esquerda de 0.5rem (8px) a um elemento.
- **text-lg**: Define o tamanho do texto como grande, proporcionando um estilo tipográfico aumentado.

Nesse momento precisamos entender o conceito de **Objeto** em JavaScript. Um objeto é uma estrutura de dados que armazena informações em **pares de chave e valor**.

A chave é uma string que atua como um identificador único para acessar o valor associado a ela. O valor pode ser qualquer tipo de dado, como números, strings, arrays, funções ou até mesmo outros objetos.

Essa combinação de chave e valor permite organizar e acessar os dados de forma eficiente, tornando os objetos fundamentais para representar estruturas complexas em programação.

Aprimorando a Funcionalidade do Carrinho de Compras

Dentro do arquivo menuCarrinho.js, vamos criar um objeto vazio, que armazenará dinamicamente a quantidade de cada produto que o usuário adicionar ao carrinho de compras.

```
3    const idsProdutoCarrinhoComQuantidade = {};
```

Para obter os valores do nosso objeto, utilizamos a notação de ponto (**objeto.chave**). No caso do objeto estar vazio, a notação de colchetes (**objeto[]**) oferece maior flexibilidade ao acessar os valores associados.

Então dentro da **função adicionarAoCarrinho()**, vamos adicionar a linha de código abaixo, que atribui o valor 1 à chave produto.id dentro do objeto idsProdutoCarrinhoComQuantidade, essencialmente começando o controle de quantidade desse produto no carrinho de compras.


```
37    idsProdutoCarrinhoComQuantidade[idProduto] = 1;
```

Aprimorando a Funcionalidade do Carrinho de Compras

Para melhorarmos a capacidade de aumentar a quantidade de um produto e evitarmos adicionar o mesmo cartão de produto várias vezes, vamos explorar um conceito crucial do JavaScript: as estruturas condicionais.

As estruturas condicionais são como "decisões" no código. Elas permitem que o programa escolha entre diferentes caminhos com base em uma condição. O **"if"** é usado para verificar se algo é verdadeiro. Se for verdadeiro, o programa executa uma ação específica. Se não for verdadeiro, o **"else"** entra em ação e o programa faz outra ação alternativa. É como decidir se você leva um guarda-chuva se estiver chovendo (if) ou usa óculos de sol se estiver ensolarado (else). Isso ajuda o código a se adaptar e tomar ações diferentes com base no que está acontecendo.

Então vamos criar um código que checa se o **"idProduto"** já existe na lista de produtos no carrinho. Se estiver lá, isso indica que o produto já foi adicionado antes. Nesse caso, a função **"incrementarQuantidadeProduto(idProduto)"** é chamada para aumentar a quantidade desse produto no carrinho (função que vamos criar).



```
23 export function adicionarAoCarrinho(idProduto) {  
24   if(idProduto in idsProdutoCarrinhoComQuantidade) {  
25     incrementarQuantidadeProduto(idProduto);  
26   }
```

Aprimorando a Funcionalidade do Carrinho de Compras

Para não ocorrer nenhum erro enquanto criamos nossas condições e obtermos os valores corretos dos ids do nosso objeto, no arquivo utilidades.js, vamos alterar os ids que estão em formato de números em textos, basta colocar os números entre aspas ("1", "2", "3"...).

```
src > JS utilidades.js > [⌘] catalogo
1   export const catalogo = [
2     {
3       id: "1",
4       marca: 'Zara',
5       nome: 'Camisa Larga com Bolsos',
6       preco: 70,
7       imagem: 'product-1.jpg',
8       feminino: false,
9     },
10    {
11      id: "2",
12      marca: 'Zara',
13      nome: 'Casaco Reto com Lã',
14      preco: 85,
15      imagem: 'product-2.jpg',
16      feminino: true,
17    },
18  ],
```

Agora vamos criar a nossa função:

- **incrementarQuantidadeProduto()**

```
23  function incrementarQuantidadeProduto(idProduto) {
24    idsProdutoCarrinhoComQuantidade[idProduto]++;
25  }
```

Essa função aumenta a quantidade de um produto específico no carrinho, **incrementando o valor associado ao seu "idProduto"** na lista **"idsProdutoCarrinhoComQuantidade"**. Em seguida, a função encerra sua execução com "return".

- o **return** está sendo utilizado sem especificar um valor. Nesse caso, o **return** sem um valor simplesmente finaliza a execução da função.
- O **operador "++"** é usado para incrementar o valor de uma variável em 1.

Aprimorando a Funcionalidade do Carrinho de Compras

E já pensando na funcionalidade de diminuir a quantidade do mesmo produto no carrinho de compras, vamos criar a nossa função **decrementarQuantidadeProduto()**, ela vai utilizar o **operador "--"** para decrementar o valor da variável em 1.

```
27 function decrementarQuantidadeProduto(idProduto) {
28     idsProdutoCarrinhoComQuantidade[idProduto]--;
29 }
```

E para garantir que o usuário não possa adicionar mais de um cartão do mesmo produto, vamos adicionar **"return"** após a execução da nossa estrutura de repetição **"if"**.

Nesse caso o **return** é usado para interromper a execução da função imediatamente após chamar a função **incrementarQuantidadeProduto(idProduto)**.

```
32 if(idProduto in idsProdutoCarrinhoComQuantidade) {
33     incrementarQuantidadeProduto(idProduto);
34     return;
35 }
```

No elemento **<p>** que armazena a quantidade de produtos adicionados ao carrinho de compras, precisamos deixar ele dinâmico, e para isso vamos alterar o código:

```
56 <p id="quantidade-{{produto.id}}"
57   class="ml-2">{{idsProdutoCarrinhoComQuantidade[produto.id]}}
58 </p>
```

Os trechos do código acima, **{{produto.id}}** e **{{idsProdutoCarrinhoComQuantidade[produto.id]}}** está inserindo dinamicamente os valores do **produto.id** e da **idsProdutoCarrinhoComQuantidade[produto.id]**, respectivamente, no conteúdo do parágrafo.

Aprimorando a Funcionalidade do Carrinho de Compras

Agora vamos criar uma função que será responsável por alterar dinamicamente a quantidade do produto adicionado ao carrinho de compras.

Ela recebe um parâmetro **idProduto**, que é o identificador único do produto cuja quantidade precisa ser atualizada no carrinho de compras.

Ela usa a função **document.getElementById** para encontrar um elemento HTML no documento com um ID específico. O ID é construído usando a template string **quantidade-\${idProduto}**, onde **idProduto** é o valor do parâmetro que recebemos. Esse ID provavelmente corresponde a um elemento que exibe a quantidade desse produto no carrinho.

Depois de selecionar o elemento HTML, a função atribui o valor da quantidade do produto ao conteúdo desse elemento usando a propriedade **innerText**. Essa quantidade é obtida a partir de um objeto chamado **idsProdutoCarrinhoComQuantidade**, onde as chaves são os IDs dos produtos e os valores são as quantidades correspondentes no carrinho. A quantidade correta é obtida usando **idsProdutoCarrinhoComQuantidade[idProduto]**.

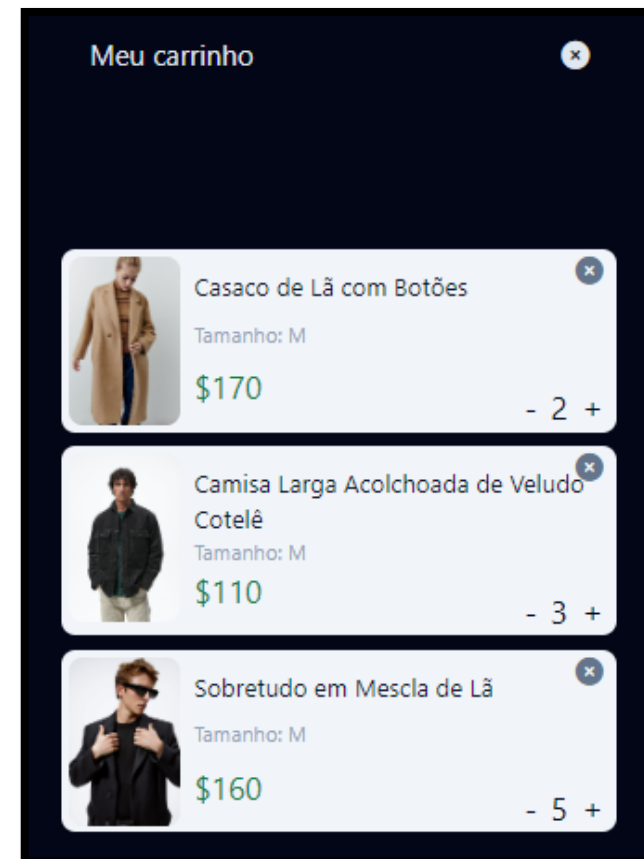
```
31 function atualizarInformacaoQuantidade(idProduto) {  
32     document.getElementById(`quantidade-${idProduto}`).innerText = idsProdutoCarrinhoComQuantidade[idProduto];  
33 }
```

Aprimorando a Funcionalidade do Carrinho de Compras

A função **atualizarInformacaoQuantidade(idProduto)** será adicionada nas funções **incrementarQuantidadeProduto(idProduto)** e **decrementarQuantidadeProduto(idProduto)**, para que toda vez que aumente ou diminua a quantidade de produtos seja atualizado no nosso site.

```
23 function incrementarQuantidadeProduto(idProduto) {  
24     idsProdutoCarrinhoComQuantidade[idProduto]++;  
25     atualizarInformacaoQuantidade(idProduto)  
26 }  
27  
28 function decrementarQuantidadeProduto(idProduto) {  
29     idsProdutoCarrinhoComQuantidade[idProduto]--;  
30     atualizarInformacaoQuantidade(idProduto)  
31 }
```

Agora, o JavaScript obtém a informação da quantidade atual desse produto, do objeto `idsProdutosCarrinhoComQuantidade`.



Aprimorando a Funcionalidade do Carrinho de Compras

Agora aplicaremos a funcionalidade nos botões + e -, para isso vamos criar um id em cada botão.

Além disso, vamos selecionar o elemento **<button>** - usando o ID de cada um. Adicionaremos um ouvinte de evento para quando esse elemento for clicado. Quando o elemento for clicado, chama a função **decrementarQuantidadeProduto()** e passa o ID do produto como argumento. Isso resultará na redução da quantidade desse produto no carrinho, graças à lógica implementada na função **decrementarQuantidadeProduto()**.

E vamos aplicar a mesma lógica para o **<button>** + e utilizaremos a função **incrementarQuantidadeProduto()**.

```
67 document.getElementById(`decrementar-produto-${produto.id}`).addEventListener('click', () => decrementarQuantidadeProduto(produto.id));
68 document.getElementById(`incrementar-produto-${produto.id}`).addEventListener('click', () => incrementarQuantidadeProduto(produto.id));
```


Aprimorando a Funcionalidade do Carrinho de Compras

Mas repare que os nossos botões apenas funcionam no último cartão de produtos adicionado, isso está ocorrendo porque o que temos ao renderizar eles através do Javascript é um texto e não elementos Html:

- o código HTML está armazenado como uma string na variável **cartaoProdutoCarrinho**. Quando o JavaScript analisa essa template string, ele trata o conteúdo como uma sequência de caracteres, ou seja, O JavaScript trata o conteúdo da template string como texto. Ele não interpreta ou processa o código HTML dentro dela. No contexto do JavaScript, isso é apenas uma sequência de caracteres e não é interpretado como estrutura de página HTML.

Para corrigirmos a funcionalidade dos botões, precisamos que o Javascript crie o nosso `<article>` como elemento Html e não como texto.

Primeiro vamos retirar o container principal do nosso cartão de produto do nosso arquivo `menuCarrinho.js`, ou seja, vamos excluir a linha que contém **`<article class="flex bg-slate-100 rounded-lg p-1 relative">`** e vamos criar ele com Javascript.

Aprimorando a Funcionalidade do Carrinho de Compras

Escrevendo um código em JavaScript que cria um elemento HTML **article** e adiciona algumas classes a ele.

Primeiro, usamos a função **document.createElement("article")** para criar uma nova tag HTML **<article></article>**. Isso seria como moldar um bloco de conteúdo dentro do qual queremos inserir informações.

Em seguida, temos um array chamado **articleClasses**, que contém várias strings representando classes CSS, como "flex", "bg-slate-100", "rounded-lg", "p-1" e "relative". As classes CSS são usadas para aplicar estilos a elementos HTML e definir como eles devem ser exibidos visualmente.

```
const elementoArticle = document.createElement("article"); //cria a tag <article></article>
const articleClasses = ["flex", "bg-slate-100", "rounded-lg", "p-1", "relative"];
```

Depois disso, entramos em um loop chamado **for...of**. Isso é como uma lista de tarefas, onde para cada classe na lista de **articleClasses**, fazemos algo.

Aprimorando a Funcionalidade do Carrinho de Compras

Dentro desse loop, usamos a função **elementoArticle.classList.add(articleClass)**. Aqui, estamos pegando cada classe do array **articleClasses** e a adicionando ao elemento **elementoArticle** que criamos anteriormente. Basicamente, estamos dizendo "adicione esta classe ao elemento que estamos construindo".

Assim, à medida que percorremos o loop, cada classe é adicionada ao nosso elemento **article**, que agora possui várias classes CSS. Essas classes vão definir como o elemento **article** será estilizado e posicionado na página.

```
for (const articleClass of articleClasses) {  
  elementoArticle.classList.add(articleClass);  
} // adiciona as classes <article class="flex bg-slate-100 rounded-lg p-1 relative"></article>
```

O resultado final desse trecho de código é que ele cria um elemento **article** e aplica a ele um conjunto específico de classes Tailwind CSS.

Aprimorando a Funcionalidade do Carrinho de Compras

Para finalizar a implementação desse novo formato de criar elementos html com Javascript, precisamos ao final o código, que renderizará o nosso cartão de forma correta:

```
elementoArticle.innerHTML = cartaoProdutoCarrinho;  
containerProdutosCarrinho.appendChild(elementoArticle);
```

Já temos um elemento **article** criado e estilizado previamente. Agora, queremos preencher esse elemento com algum conteúdo e, em seguida, adicioná-lo a um contêiner maior na página.

Primeiro, temos a linha **elementoArticle.innerHTML = cartaoProdutoCarrinho;**. Aqui, **cartaoProdutoCarrinho** é uma string que contém algum código HTML. Estamos pegando essa string e atribuindo-a ao **innerHTML** do elemento **article** que criamos anteriormente. Isso é como dizer "coloque o conteúdo HTML que está na string dentro deste elemento **article** que criamos".

Depois disso, temos a linha **containerProdutosCarrinho.appendChild(elementoArticle);**. **containerProdutosCarrinho** é provavelmente o contêiner maior no qual queremos adicionar nosso elemento **article**.

Aprimorando a Funcionalidade do Carrinho de Compras

Usamos o método **appendChild(elementoArticle)** para adicionar o nosso elemento **article** como um "filho" do **containerProdutosCarrinho**. Visualmente, é como colocar o nosso elemento **article** dentro do contêiner maior.

Assim, ao executar essas duas linhas de código, conseguimos preencher nosso elemento **article** com o conteúdo da string **cartaoProdutoCarrinho** e então adicionamos esse elemento a um contêiner maior. Isso é fundamental para exibir dinamicamente o "cartão" de um produto no carrinho em uma página da web. É como montar o conteúdo visual dentro da estrutura do nosso layout.

```
46 const elementoArticle = document.createElement("article"); //cria a tag <article></article>
47 const articleClasses = ["flex", "bg-slate-100", "rounded-lg", "p-1", "relative"];
48 for (const articleClass of articleClasses) {
49   elementoArticle.classList.add(articleClass);
50 } // adiciona as classes <article class="flex bg-slate-100 rounded-lg p-1 relative"></article>
51
52 const cartaoProdutoCarrinho = `<button id="fechar-carrinho" class="absolute top-0 right-2">
53   <i class="fa-solid fa-circle-xmark text-slate-500 hover:text-slate-800"></i>
54 </button>
55 
60 <div class="p-2 flex flex-col justify-between">
61   <p class="text-slate-900 text-sm">${produto.nome}</p>
62   <p class="text-slate-400 text-xs">Tamanho: M</p>
63   <p class="text-green-700 text-lg">${produto.preco}</p>
64 </div>
65 <div class="flex text-slate-950 items-end absolute bottom-0 right-2 text-lg">
66   <button id="decrementar-produto-${produto.id}"></button>
67   <p id="quantidade-${produto.id}" class="ml-2">${idsProdutoCarrinhoComQuantidade[produto.id]}</p>
68   <button id="incrementar-produto-${produto.id}" class="ml-2"></button>
69 </div>`;
70
71 elementoArticle.innerHTML = cartaoProdutoCarrinho;
72 containerProdutosCarrinho.appendChild(elementoArticle);
73
74 document.getElementById(`decrementar-produto-${produto.id}`).addEventListener('click', () => decrementarQuantidadeProduto(produto.id));
75 document.getElementById(`incrementar-produto-${produto.id}`).addEventListener('click', () => incrementarQuantidadeProduto(produto.id));
76 }
```

Aprimorando a Funcionalidade do Carrinho de Compras

Vamos fazer algumas alterações para garantir que os produtos sejam removidos do carrinho quando sua quantidade for menor do que 1. Isso envolve atualizar tanto o nosso dicionário/objeto quanto o elemento HTML correspondente. Para tornar o código mais organizado, vamos criar uma nova função chamada

desenharProdutoNoCarrinho(). A responsabilidade dessa função será armazenar todo o processo de criação dos elementos HTML.

Dessa forma, o conteúdo relacionado à criação dos elementos será removido da função **adicionarAoCarrinho()**, mantendo-a mais focada em outras tarefas. A nova função **desenharProdutoNoCarrinho()** será chamada para desenhar ou atualizar o visual do produto no carrinho. Se a quantidade do produto ficar menor que 1, a função também removerá o produto do carrinho, tanto do dicionário quanto do elemento HTML.

```
function desenharProdutoNoCarrinho(idProduto) {
  const produto = catalogo.find((p) => p.id === idProduto);
  const containerProdutosCarrinho = document.getElementById("produtos-carrinho");

  const elementoArticle = document.createElement("article"); //cria a tag <article></article>
  const articleClasses = ["flex", "bg-slate-100", "rounded-lg", "p-1", "relative"];
  for (const articleClass of articleClasses) {
    elementoArticle.classList.add(articleClass);
  } // adiciona as classes <article class="flex bg-slate-100 rounded-lg p-1 relative"></article>

  const cartaoProdutoCarrinho = `<button id="fechar-carrinho" class="absolute top-0 right-2">
    <i class="fa-solid fa-circle-xmark text-slate-500 hover:text-slate-800"></i>
  </button>
  
  <div class="p-2 flex flex-col justify-between">
    <p class="text-slate-900 text-sm">${produto.nome}</p>
    <p class="text-slate-400 text-xs">Tamanho: M</p>
    <p class="text-green-700 text-lg">${produto.preco}</p>
  </div>
  <div class="flex text-slate-950 items-end absolute bottom-0 right-2 text-lg">
    <button id="decrementar-produto-${produto.id}">-</button>
    <p id="quantidade-${produto.id}" class="ml-2">${idsProdutoCarrinhoComQuantidade[produto.id]}</p>
    <button id="incrementar-produto-${produto.id}" class="ml-2">+</button>
  </div>`;

  elementoArticle.innerHTML = cartaoProdutoCarrinho;
  containerProdutosCarrinho.appendChild(elementoArticle);

  document.getElementById(`decrementar-produto-${produto.id}`).addEventListener('click', () => {
    function() { void } decrementarQuantidadeProduto(produto.id);
  });
  document.getElementById(`incrementar-produto-${produto.id}`).addEventListener('click', () => {
    function() { void } incrementarQuantidadeProduto(produto.id);
  });
}
```

Aprimorando a Funcionalidade do Carrinho de Compras

Então a função **adicionarAoCarrinho()** terá a seguinte estrutura nesse momento do nosso projeto:

```
export function adicionarAoCarrinho(idProduto) {  
  if (idProduto in idsProdutoCarrinhoComQuantidade) {  
    incrementarQuantidadeProduto(idProduto);  
    return;  
  }  
  idsProdutoCarrinhoComQuantidade[idProduto] = 1;  
  desenharProdutoNoCarrinho(idProduto);  
}
```

Estamos no caminho certo para o nosso cartão de produtos no carrinho de compras, mas ainda precisamos criar mais algumas funções, para que ele funcione da maneira como queremos.

Vamos criar a função **renderizarProdutosCarrinho()**, é como falamos para o nosso programa desenhar os produtos.

Ela terá a seguinte estrutura:

```
function renderizarProdutosCarrinho() { // renderizar = desenhar  
  const containerProdutosCarrinho = document.getElementById("produtos-carrinho");  
  containerProdutosCarrinho.innerHTML = "";  
  for (const idProduto in idsProdutoCarrinhoComQuantidade) {  
    desenharProdutoNoCarrinho(idProduto);  
  }  
}
```

- **Obtenção do Elemento de Contêiner:** A linha `const containerProdutosCarrinho = document.getElementById("produtos-carrinho");` obtém o elemento HTML que possui o id "produtos-carrinho", ou seja, o local onde os produtos do carrinho serão exibidos na página.

Aprimorando a Funcionalidade do Carrinho de Compras

- **Limpeza do Conteúdo do Contêiner:** A linha **containerProdutosCarrinho.innerHTML = ""**; remove todo o conteúdo dentro do elemento do contêiner, preparando-o para exibir os produtos atualizados do carrinho.
- **Laço de Iteração:** A linha **for (const idProduto in idsProdutoCarrinhoComQuantidade)** inicia um loop que passa por cada **idProduto** presente na estrutura **idsProdutoCarrinhoComQuantidade**, que é o objeto mapeia IDs de produtos para quantidades no carrinho.
- **Chamada de Função para Desenhar Produtos:** Dentro do loop, a função **desenharProdutoNoCarrinho(idProduto)**; é chamada para cada **idProduto**. Isso significa que a função **desenharProdutoNoCarrinho()** será responsável por adicionar as informações do produto correspondente ao carrinho na página.

Em resumo, a função **renderizarProdutosCarrinho()** é responsável por atualizar a exibição dos produtos no carrinho na página. Ela primeiro obtém o elemento de contêiner onde os produtos do carrinho serão exibidos, limpa o conteúdo existente desse contêiner e, em seguida, itera sobre os IDs dos produtos presentes no carrinho. Para cada ID de produto, chama outra função (**desenharProdutoNoCarrinho()**) para desenhar as informações do produto na página.

Aprimorando a Funcionalidade do Carrinho de Compras

A próxima função que vamos criar é a **removerDoCarrinho()**, que como o próprio nome diz a sua responsabilidade será remover o cartão de produtos do carrinho de compras. E para isso primeiro, precisamos deixar o id do botão que irá ter essa funcionalidade dinâmico.

```
const cartaoProdutoCarrinho = `<button id="remove-item-${produto.id}" class="absolute top-0 right-2">  
  <i class="fa-solid fa-circle-xmark text-slate-500 hover:text-slate-800"></i>  
</button>
```

Agora a lógica da nossa função é que a função **removerDoCarrinho(idProduto)** é usada para remover um produto específico do carrinho. Primeiro, o produto é removido do objeto que armazena os IDs dos produtos no carrinho com suas quantidades correspondentes. Em seguida, a função **renderizarProdutosCarrinho()** é chamada para atualizar a exibição dos produtos restantes no carrinho na página. Isso garante que a página reflita corretamente a remoção do produto do carrinho.

```
function removerDoCarrinho(idProduto) {  
  delete idsProdutoCarrinhoComQuantidade[idProduto];  
  renderizarProdutosCarrinho();  
}
```

Aprimorando a Funcionalidade do Carrinho de Compras

- **Exclusão do Produto do Carrinho:** A linha **delete idsProdutoCarrinhoComQuantidade[idProduto];** remove o produto do objeto ou mapa **idsProdutoCarrinhoComQuantidade** com base no ID do produto fornecido como argumento para a função. Isso efetivamente remove o produto do carrinho, excluindo a entrada correspondente no objeto.
- **Chamada para Atualizar a Renderização:** Após remover o produto do carrinho, a linha **renderizarProdutosCarrinho();** é chamada. Essa chamada de função é usada para atualizar a renderização da lista de produtos no carrinho, refletindo a remoção do produto que acabou de ser excluído.

Vamos adicionar a lógica do remover a todos os botões de todos os cartões de produtos que estiverem renderizados no nosso carrinho de compra:

```
document.getElementById(`decrementar-produto-${produto.id}`).addEventListener('click', () => decrementarQuantidadeProduto(produto.id));  
document.getElementById(`incrementar-produto-${produto.id}`).addEventListener('click', () => incrementarQuantidadeProduto(produto.id));  
document.getElementById(`remover-item-${produto.id}`).addEventListener('click', () => removerDoCarrinho(produto.id));
```

Aprimorando a Funcionalidade do Carrinho de Compras

Para finalizar a configuração do painel de produtos no nosso carrinho de compras, abordaremos o problema atual em que a redução da quantidade de produtos resulta em um erro, persistindo na diminuição mesmo quando não há mais produtos no carrinho. Nossa intenção é garantir que, ao reduzir a quantidade de 1 para 0, o cartão do produto seja automaticamente removido do carrinho de compras.

Então dentro da função **decrementarQuantidadeProduto()**, a estrutura condicional **if** verifica se a quantidade de um determinado produto no carrinho é igual a 1. Se essa condição for atendida, significa que a quantidade do produto foi reduzida para 1 e você deseja remover automaticamente esse produto do carrinho.

```
function decrementarQuantidadeProduto(idProduto) {  
  if (idsProdutoCarrinhoComQuantidade[idProduto] === 1) {  
    removerDoCarrinho(idProduto);  
    return;  
  }  
  idsProdutoCarrinhoComQuantidade[idProduto]--;  
  atualizarInformacaoQuantidade(idProduto)  
}
```

Aprimorando a Funcionalidade do Carrinho de Compras

Nesse momento nosso cartão de produtos está funcional e com a interface que queríamos, abaixo segue como o código do arquivo menuCarrinho.js está até esse momento do projeto.

JS menuCarrinho.js X

src > JS menuCarrinho.js > ...

```
1 import { catalogo } from "../utilidades";
2
3 const idsProdutoCarrinhoComQuantidade = {};
4
5 function abrirCarrinho() {
6   document.getElementById("carrinho").classList.add('right-[0px]');
7   document.getElementById("carrinho").classList.remove('right-[-360px]');
8 }
9
10 function fecharCarrinho() {
11   document.getElementById("carrinho").classList.remove('right-[0px]');
12   document.getElementById("carrinho").classList.add('right-[-360px]');
13 }
14
15 export function inicializarCarrinho() {
16   const botaoFecharCarrinho = document.getElementById("fechar-carrinho");
17   const botaoAbrirCarrinho = document.getElementById("abrir-carrinho");
18
19   botaoFecharCarrinho.addEventListener("click", fecharCarrinho);
20   botaoAbrirCarrinho.addEventListener("click", abrirCarrinho);
21 }
```

1

```
23 function removerDoCarrinho(idProduto) {
24   delete idsProdutoCarrinhoComQuantidade[idProduto];
25   renderizarProdutosCarrinho();
26 }
27
28 function incrementarQuantidadeProduto(idProduto) {
29   idsProdutoCarrinhoComQuantidade[idProduto]++;
30   atualizarInformacaoQuantidade(idProduto)
31 }
32
33 function decrementarQuantidadeProduto(idProduto) {
34   if (idsProdutoCarrinhoComQuantidade[idProduto] === 1) {
35     removerDoCarrinho(idProduto);
36     return;
37   }
38   idsProdutoCarrinhoComQuantidade[idProduto]--;
39   atualizarInformacaoQuantidade(idProduto)
40 }
41
42 function atualizarInformacaoQuantidade(idProduto) {
43   document.getElementById(`quantidade-${idProduto}`).innerText = idsProdutoCarrinhoComQuantidade[idProduto];
44 }
```

2

Aprimorando a Funcionalidade do Carrinho de Compras

```

46 function desenharProdutoNoCarrinho(idProduto) {
47   const produto = catalogo.find((p) => p.id === idProduto);
48   const containerProdutosCarrinho = document.getElementById("produtos-carrinho");
49
50   const elementoArticle = document.createElement("article"); //cria a tag <article></article>
51   const articleClasses = ["flex", "bg-slate-100", "rounded-lg", "p-1", "relative"];
52   for (const articleClass of articleClasses) {
53     elementoArticle.classList.add(articleClass);
54   } // adiciona as classes <article class="flex bg-slate-100 rounded-lg p-1 relative"></article>
55
56   const cartaoProdutoCarrinho = `<button id="remover-item-${produto.id}" class="absolute top-0 right-2">
57     <i class="fa-solid fa-circle-xmark text-slate-500 hover:text-slate-800"></i>
58   </button>
59   
64   <div class="p-2 flex flex-col justify-between">
65     <p class="text-slate-900 text-sm">${produto.nome}</p>
66     <p class="text-slate-400 text-xs">Tamanho: M</p>
67     <p class="text-green-700 text-lg">${produto.preco}</p>
68   </div>
69   <div class="flex text-slate-950 items-end absolute bottom-0 right-2 text-lg">
70     <button id="decrementar-produto-${produto.id}"></button>
71     <p id="quantidade-${produto.id}" class="ml-2">${idsProdutoCarrinhoComQuantidade[produto.id]}</p>
72     <button id="incrementar-produto-${produto.id}" class="ml-2"></button>
73   </div>;
74
75   elementoArticle.innerHTML = cartaoProdutoCarrinho;
76   containerProdutosCarrinho.appendChild(elementoArticle);
77
78   document.getElementById(`decrementar-produto-${produto.id}`).addEventListener('click', () => decrementarQuantidadeProduto(produto.id));
79   document.getElementById(`incrementar-produto-${produto.id}`).addEventListener('click', () => incrementarQuantidadeProduto(produto.id));
80   document.getElementById(`remover-item-${produto.id}`).addEventListener('click', () => removerDoCarrinho(produto.id));
81 }

```

3

```

83 function renderizarProdutosCarrinho() { // renderizar = desenhar
84   const containerProdutosCarrinho = document.getElementById("produtos-carrinho");
85   containerProdutosCarrinho.innerHTML = "";
86   for (const idProduto in idsProdutoCarrinhoComQuantidade) {
87     desenharProdutoNoCarrinho(idProduto);
88   }
89 }
90
91 export function adicionarAoCarrinho(idProduto) {
92   if (idProduto in idsProdutoCarrinhoComQuantidade) {
93     incrementarQuantidadeProduto(idProduto);
94     return;
95   }
96   idsProdutoCarrinhoComQuantidade[idProduto] = 1;
97   desenharProdutoNoCarrinho(idProduto);
98 }

```

4

Na próxima aula finalizaremos o nosso site E-Commerce!

Ainda não segue a gente no Instagram e nem é inscrito no nosso canal do Youtube? Então corre lá!



@hashtagprogramacao



youtube.com/hashtag-programacao

