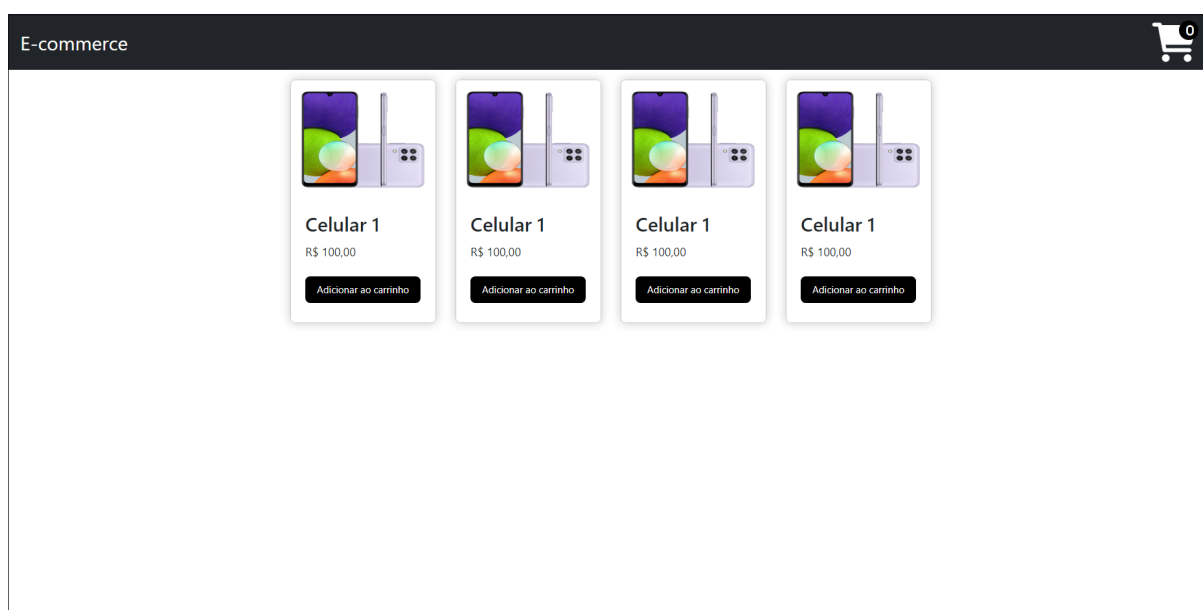


Carrinho

Exemplo de carrinho utilizando HTML, CSS e JS, armazenando os produtos no localStorage do navegador.

link para repositório do Github: <https://github.com/netsoncavina/carrinho-exemplo>

Dentro do repositório há 2 pastas: inicial e finalizado. Dentro da pasta inicial, há os arquivos HTML e CSS configurados, faltando somente o JS para adicionar as funcionalidades.



Bibliotecas utilizadas no projeto:

Bootstrap

Bootstrap

Read installation docs Install Bootstrap's source Sass and JavaScript files via npm, RubyGems, Composer, or Meteor. Package managed installs don't include documentation or our

 <https://getbootstrap.com/>



FontAwesome

Font Awesome

The world's most popular and easiest to use icon set just got an upgrade. More icons. More styles. More Options.

 <https://fontawesome.com/>



Sweet Alert

SweetAlert2

Show success message \$ npm install/sweetalert2 Or grab from jsdelivr CDN // ES6 Modules or TypeScript import Swal from 'sweetalert2' // CommonJS const Swal = require('sweetalert2')

 <https://sweetalert2.github.io/>



O que é Local Storage?

A API de Web Storage fornece duas formas de armazenar dados no navegador:

- **Session Storage** → Armazena informações enquanto o navegador estiver aberto (mesmo que atualize a página), mas, caso a janela seja fechada, as informações são excluídas.
- **Local Storage** → Funciona de forma semelhante a Session Storage, porém os dados continuam armazenados após o navegador ser fechado.

Como utilizar o Local Storage?


A API de Web Storage fornece algumas funções para acesso ao localStorage:


- `Storage.length` → Retorna a quantidade de itens armazenados no localStorage.
- `Storage.key(posição)` → Retorna o nome da chave em determinada posição.
- `Storage.getItem(nomeChave)` → Retorna o valor de determinada chave.
- `Storage.setItem(nomeChave, valorChave)` → Salva e/ou atualiza o valor de determinada chave.
- `Storage.removeItem(chave)` → Exclui o valor de uma chave.
- `Storage.clear()` → Exclui todas as chaves e valores armazenados.

Para mais informações sobre a API:

Storage - APIs da Web | MDN

Aqui acessamos um objeto Storage chamando localStorage. Primeiro testamos se o armazenamento local contém itens de dados usando! localStorage.getItem('bgcolor'). Se isso

 <https://developer.mozilla.org/pt-BR/docs/Web/API/Storage>

 mdn web docs

Utilizando o Local Storage

Primeiro criamos uma função para armazenar os dados no localStorage. Neste projeto iremos armazenar o nome do produto, seu preço e sua imagem, então a função fica assim no javascript:

```
function adicionaProduto(nome, imagem, preco) {  
  localStorage.setItem(nome, [nome, preco, imagem]);  
}
```

A função armazena, no localStorage, um vetor contendo nome, preço e imagem do produto, e a chave é o nome do produto.

Para utilizar a função, precisamos adicionar uma função onclick ao botão do produto, que fica assim:

```
<div class="card-produto">  
    
  <div class="descricao-produto">  
    <h5>Celular 1</h5>  
    <p>R$ 100,00</p>  
    <button  
      class="btn botao-comprar"  
      onclick="adicionaProduto('Celular 1', 'imagem1.jpg', '100')"  
    >  
      Adicionar ao carrinho  
    </button>  
  </div>  
</div>
```

Ao clicar no botão de adicionar ao carrinho, podemos ver no localStorage do navegador que o produto foi salvo.

Key	Value
Celular 1	Celular 1,100,imagem1.jpg

Atualizar numero no carrinho

Atualmente o carrinho possui espaço para o número de produtos nele, porém sempre exibe 0, vamos implementar a função que atualiza a quantidade.

Primeiro temos que dar um id ao numero no carrinho, irei dar o id “numero_carrinho”:

```
<i class="fa-solid fa-cart-shopping carrinho">
  <span class="badge" id="numero_carrinho">0</span>
</i>
```

ATENÇÃO: O id deve ser adicionado ao numero, dentro da tag span, não ao icone do carrinho.

Feito isso, no javascript devemos criar a função que atualiza o valor. Essa função irá atualizar o numero de acordo com a quantidade de items do localStorage, utilizando a função localStorage.length e atualizando o innerHTML daquele id.

A função fica assim:

```
function atualizaCarrinho() {
  count = document.getElementById("numero_carrinho");
  count.innerHTML = localStorage.length;
}
```

Para ter sempre o valor atualizado, devemos chamar a função na inicialização do site e toda vez que a função “adicionaProduto” for chamada. Então nosso arquivo javascript irá ficar assim:

```
function adicionaProduto(nome, imagem, preco) {
  localStorage.setItem(nome, [nome, preco, imagem]);
  atualizaCarrinho();
}
function atualizaCarrinho() {
  count = document.getElementById("numero_carrinho");
  count.innerHTML = localStorage.length;
}

atualizaCarrinho();
```

Isso é tudo para a pagina inicial.

Pagina do carrinho

Na pagina do carrinho, nós precisamos:

- Acessar os produtos armazenados no localStorage
- Exibir os produtos
- Ter a opção de remover o produto do carrinho
- Exibir os valores dos produtos
- Calcular o total (soma dos valores)
- Finalizar o pedido

Acessar os produtos armazenados

Para acessar os produtos, iremos utilizar a função “getItem” da API de Web Storage.

```
function getProdutos() {  
  var valores = [],  
      chaves = Object.keys(localStorage),  
      i = chaves.length;  
  
  while (i--) {  
    valores.push(localStorage.getItem(chaves[i]));  
  }  
  
  return valores;  
}
```

Nesta função nós criamos um vetor chamado valores, onde nós iremos armazenar os valores de cada chave do localStorage (nome, preço e imagem). Utilizando a função “Object.keys()” do javascript, nós obtemos as chaves do localStorage.

Dentro do laço while, nós adicionamos os valores das chaves para o vetor valores utilizando a função getItem.

Depois, retornamos o vetor valores.

Exibir os produtos

Para exibir os produtos, será necessário escrever HTML dentro do Javascript, para isso utilizamos os id e o metodo innerHTML.

```
function exibeProdutos() {
  let produtos = document.getElementById("produtos");
  let items = getProdutos();
  if (items.length == 0) {
    produtos.innerHTML = "Nenhum item no carrinho!";
  }
  for (let i = 0; i < items.length; i++) {
    let produto = document.createElement("div");
    produto.innerHTML = `
    <div class="card-produto-carrinho">
    

    <div class="col">
      <h2>${items[i].split(",")[0]}</h2>
      <h4>R$ ${items[i].split(",")[1]}</h4>
    </div>
    <button
      class="btn botao-remover"
      onclick="removeProduto('${items[i].split(",")[0]}')
    >
      Remover do carrinho
    </button>
  </div>

  `;
    produtos.appendChild(produto);
  }
}
```

Primeiro, acessamos o elemento com id “produtos” e o armazenamos na variavel produtos.

A variavel items é utilizada para armazenar o vetor que a função “getProdutos()” retorna.

Dentro do laço for, percorremos o vetor items para que, com os valores de cada indice, nós criemos uma div que irá se tornar o “card” de cada produto, com imagem, nome, preço e um botão para excluir o produto. No final do laço, essa div é adicionada a div com id=”produtos”.

Para que os produtos sejam exibidos na pagina do carrinho, é necessário realizar a chamada da função.

Remover produtos

Para remover os produtos do carrinho, será utilizada função “removeItem()” da API de Web Storage

```
function removeProduto(nome) {
  localStorage.removeItem(nome);
  atualizaCarrinho();
  location.reload();
}
```

Nesta função é necessário passar o nome da chave como parâmetro.

Após remover o item, nós chamamos a função `atualizaCarrinho()` para atualizar o número de produtos no carrinho e `location.reload()` para atualizar a página.

Exibir os valores dos produtos

Esta função será bem parecida com a função de exibir os produtos, tendo como únicas diferenças o local onde os dados serão exibidos (outra div, com outro id) e que não será necessário exibir as imagens.

```
function exibeValores() {
  let items = getProdutos();
  let valores = document.getElementById("valores");
  for (let i = 0; i < items.length; i++) {
    let valor = document.createElement("div");
    valor.innerHTML = `
    <div class="row">
      <h5 class="pedido_item text-left">${items[i].split(",")[0]}</h5>
      <div class="col">
        <h5 class="pedido_preco text-right">R$ ${items[i].split(",")[1]}</h5>
      </div>
    </div>`;
    valores.appendChild(valor);
  }
}
```

Calculo do valor total do pedido

Nesta função iremos acessar os valores dos produtos, da mesma forma que fizemos anteriormente, transformar os valores de string para float, soma-los e exibi-los no local adequado.

```
function calculaTotal() {
  let items = getProdutos();
  let total = document.getElementById("total");
  let valor = 0;
  for (let i = 0; i < items.length; i++) {
    valor += parseFloat(items[i].split(",")[1]);
  }
}
```

```
}  
total.innerHTML = "R$ " + valor;  
}
```

Obs: Os valores armazenados no localStorage são sempre strings, então sempre que for realizar operações matemáticas, é necessário convertê-los para float ou int.

Finalizar pedido

Nesta função iremos utilizar a função “clear()” da API de Web Storage.

Esta função irá servir somente para limpar o carrinho e exibir um alert de pedido concluído.

```
function finalizaPedido() {  
    alert("Pedido finalizado com sucesso!");  
    localStorage.clear();  
    location.reload();  
}
```

Funções que devem ser chamadas ao fim do arquivo javascript

```
atualizaCarrinho();  
exibeProdutos();  
exibeValores();  
calculaTotal();
```