

UC: Projetos - FrontEnd

DIAS: 17 e 18/10/2024 – Parte 2

Assunto:

Projeto 24 – Lista de Produtos com Imagens

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

<> index.html > ...

```
1  <!DOCTYPE html>
2  <!-- A declaração DOCTYPE é uma instrução para o navegador
3  | sobre qual versão do HTML a página está escrita. Neste caso,
4  | indica HTML5, o padrão atual da web, assegurando que os navegadores
5  | renderizem a página usando as regras de sintaxe do HTML5. -->
6
7  <html lang="pt-br">
8  <!-- A tag <html> marca o início do documento HTML. O atributo 'lang'
9   | define o idioma principal do documento como português do
10  | Brasil (pt-br), o que ajuda os motores de busca e as tecnologias
11  | assistivas a entenderem o conteúdo da página. -->
12
13 <head>
14 <!-- A tag <head> contém metadados, links para folhas de estilo, scripts e
15  | outras informações que não são diretamente visíveis na área
16  | de conteúdo do navegador. -->
17
18 <meta charset="UTF-8">
19 <!-- A tag <meta> com o atributo 'charset' definido como "UTF-8"
20  | especifica a codificação de caracteres para o documento.
21  | UTF-8 é uma codificação de caracteres universal que inclui
22  | praticamente todos os caracteres e símbolos do mundo, assegurando
23  | que qualquer texto seja exibido corretamente. -->
24
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

<> index.html > html > head > meta

```
25   <meta name="viewport" content="width=device-width, initial-scale=1.0">
26   <!-- Outra tag &lt;meta&gt; que configura a viewport para controlar como a
27   página é dimensionada e visualizada em diferentes dispositivos.
28   O atributo 'content' com 'width=device-width' instrui o navegador a
29   definir a largura da viewport de acordo com a largura do dispositivo, o
30   que é crucial para design responsivo. 'initial-scale=1.0'
31   estabelece o nível de zoom inicial quando a página é carregada,
32   promovendo uma melhor acessibilidade e visualização inicial. --&gt;
33
34   &lt;title&gt;Lista de Produtos&lt;/title&gt;
35   <!-- A tag &lt;title&gt; define o título da página, que é exibido na aba do
36   navegador. Este título é também utilizado pelos motores de
37   busca e pode influenciar o SEO (Search Engine Optimization) da
38   página. "Lista de Produtos" é um título que dá uma ideia clara
39   do conteúdo da página. --&gt;
40
41   &lt;link rel="stylesheet" href="styles.css"&gt;
42   <!-- A tag &lt;link&gt; é usada aqui para vincular uma folha de estilo externa,
43   localizada no arquivo 'styles.css'. Esse arquivo contém todas as
44   regras de estilo CSS que definem a aparência da página. Isso ajuda a
45   separar o conteúdo (HTML) da apresentação (CSS), facilitando a
46   manutenção e atualizações do design. --&gt;
47
48   &lt;script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.18.5/xlsx.full.min.js"&gt;&lt;/script&gt;
49   <!-- A tag &lt;script&gt; com um atributo 'src' que aponta para um URL
50   CDN (Content Delivery Network) que hospeda a biblioteca 'xlsx.full.min.js'.
51   Esta biblioteca JavaScript é utilizada para ler e manipular arquivos
52   do Excel diretamente no navegador, permitindo que a página processe e
53   exiba dados de arquivos Excel sem a necessidade de um backend para
54   parseamento dos dados. --&gt;</pre>
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

<> index.html > □ html > □ head

```
55
56    </head>
57
58    <body>
59        <!-- A tag <body> contém todo o conteúdo visível da página web. Aqui são
60            definidos todos os elementos que os usuários podem interagir e
61            visualizar no navegador. -->
62
63        <h1>Lista de Produtos</h1>
64            <!-- A tag <h1> define um cabeçalho de nível um, que é o mais importante e
65                mais influente em termos de SEO dentro de uma página. "Lista de Produtos"
66                serve como o título principal da página, indicando claramente o
67                conteúdo principal para os usuários. -->
68
69        <div class="total-vendas" id="total-vendas">Total de Vendas: R$ 0,00</div>
70            <!-- Um <div> que exibe o total de vendas inicializado com R$ 0,00.
71                Este elemento será atualizado dinamicamente via JavaScript
72                conforme os dados forem processados. A classe e o id fornecem
73                ganchos para estilização e manipulação via script. -->
74
75        <div class="conteudo">
76            <!-- Uma <div> que agrupa os elementos principais da interface, facilitando a
77                estilização e organização do layout. -->
78
79            <div class="lista-produtos">
80                <!-- Uma <div> interna que contém a interface para a filtragem e
81                    listagem dos produtos. -->
82
83                <select id="filtro-vendedor">
84
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

<> index.html > ⚡ html > ⚡ body > ⚡ div.conteudo > ⚡ div.lista-produtos > ⚡ select#filtro-vendedor

```
85     <option value="">Todos os Vendedores</option>
86     <!-- Um campo de seleção que permite ao usuário filtrar produtos
87         por vendedor. Inicialmente, oferece a opção de ver
88         todos os vendedores. Outras opções serão adicionadas
89         dinamicamente com base nos dados. -->
90
91     </select>
92
93     <input type="text" id="filtro-produto" placeholder="Filtrar por Produto">
94     <!-- Um campo de entrada de texto que permite ao usuário filtrar a
95         lista de produtos por nome do produto. -->
96
97     <ul id="lista-produtos">
98
99         <!-- Um elemento <ul> onde a lista de produtos será dinamicamente
100            inserida via JavaScript. -->
101
102    </ul>
103
104    <button id="mostrar-todos">Mostrar Todos os Produtos</button>
105    <!-- Um botão que quando clicado, redefinirá quaisquer filtros
106        aplicados e mostrará todos os produtos. -->
107
108    </div>
109
110    <div class="detalhes-produto" id="detalhes-produto">
111        <!-- Uma <div> para exibir detalhes sobre um produto selecionado.
112            Os detalhes serão adicionados dinamicamente quando um
113            produto for clicado na lista. -->
114
```

EXPLORER ... <> index.html X JS script.js # styles.css ⌂ ⋮

✓ P24 - LISTA DE PRODUTOS COM IMAGENS

- > imagemProduto
- <> index.html
- 🖼 p24_screenshot.jpg
- JS script.js
- # styles.css
- ⓧ Vendedor.xlsx

<> index.html > 📂 html > 📂 body > 📂 div.conteudo > 📂 div#detalhes-produto.detalhes-produto

115	</div>
116	</div>
117	
118	<script src="script.js"></script>
119	<!-- A tag <script> com o atributo 'src' que aponta para o
120	arquivo 'script.js'. Este script contém a lógica para
121	manipular os dados do Excel, aplicar filtros e atualizar o
122	DOM com informações sobre produtos e vendas. -->
123	
124	</body>
125	</html>

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > body

```
1  body {  
2  
3      font-family: Arial, sans-serif;  
4      /* Define a família de fontes do corpo do documento como 'Arial',  
5         com 'sans-serif' como fallback. Arial é uma fonte limpa e  
6         profissional que melhora a legibilidade do texto. */  
7  
8      margin: 0;  
9      /* Remove as margens padrão do elemento <body>, permitindo que o  
10         conteúdo ocupe toda a largura e altura da janela do navegador  
11         sem espaços adicionais nas bordas. */  
12  
13     padding: 0;  
14     /* Remove o preenchimento interno padrão do elemento <body>, garantindo  
15         que não haja espaço extra dentro do limite do corpo que  
16         afete o layout interno. */  
17  
18     background-color: #f5f5f5;  
19     /* Define a cor de fundo para um cinza muito claro (#f5f5f5),  
20         proporcionando um contraste suave que é fácil para os olhos e  
21         melhora a estética geral da página. */  
22  
23 }  
24  
25 h1 {  
26  
27     text-align: center;  
28     /* Centraliza o texto dentro do elemento <h1>, garantindo que o título  
29         principal esteja alinhado ao meio horizontalmente, o que é  
30         ideal para cabeçalhos. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > h1

```
31
32     background-color: #4CAF50;
33     /* Aplica uma cor de fundo verde (#4CAF50) ao título, o que destaca
34         visualmente o cabeçalho e ajuda a definir a identidade
35             visual da página. */
36
37     color: white;
38     /* Define a cor do texto dentro do <h1> para branco, criando um alto
39         contraste com o fundo verde, o que facilita a leitura. */
40
41     padding: 20px;
42     /* Adiciona um preenchimento de 20px em todas as direções dentro do
43         elemento <h1>, aumentando o espaço em torno do texto, o que
44             melhora a legibilidade e a presença visual do título. */
45
46     margin: 0;
47     /* Define a margem do <h1> para 0, removendo qualquer espaço extra
48         ao redor do cabeçalho que poderia afetar o layout vertical. */
49
50 }
51
52 .total-vendas {
53
54     text-align: center;
55     /* Centraliza o texto da classe '.total-vendas', que é útil para
56         garantir que o total de vendas exibido seja facilmente visível e
57             esteticamente alinhado ao centro da área de exibição. */
58
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > .total-vendas

```
59     margin: 10px 0;  
60     /* Define a margem superior e inferior para 10px e as margens laterais  
61      para 0, criando um espaçamento vertical adequado em torno do  
62      elemento sem afetar o alinhamento horizontal. */  
63  
64     font-size: 18px;  
65     /* Ajusta o tamanho da fonte para 18px, garantindo que o texto seja  
66      grande o suficiente para ser lido confortavelmente sem  
67      dominar o layout. */  
68  
69     color: #333;  
70     /* Define a cor do texto para um cinza escuro (#333), que é suave  
71      para os olhos enquanto mantém uma legibilidade excelente  
72      contra o fundo claro. */  
73  
74 }  
75  
76 .conteudo {  
77  
78     display: flex;  
79     /* Utiliza o modelo de layout flexível (Flexbox) para o  
80      container '.conteudo', permitindo um layout responsivo e  
81      adaptável dos elementos filhos. Flexbox facilita o alinhamento e  
82      a distribuição do espaço entre itens em um contêiner, mesmo quando o  
83      tamanho dos itens é desconhecido ou dinâmico. */  
84
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

< index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > .conteudo

```
85     padding: 20px;
86      /* Aplica um preenchimento de 20px em todas as direções dentro do
87       elemento '.conteudo'. Isso cria um espaço entre os limites do
88       contêiner e seus filhos, melhorando a estética geral e
89       evitando que o conteúdo toque diretamente as bordas. */
90
91 }
92
93 .lista-produtos {
94
95   width: 30%;
96   /* Define a largura da '.lista-produtos' para 30% da largura de
97    seu contêiner pai. Isso estabelece um layout proporcional
98    dentro do flex container, permitindo que o conteúdo restante
99    use o espaço disponível restante. */
100
101  padding: 10px;
102  /* Adiciona um preenchimento de 10px em todas as direções dentro
103   de '.lista-produtos', o que melhora a separação visual entre o
104   conteúdo da lista e as bordas do contêiner. */
105
106  background-color: white;
107  /* Define a cor de fundo de '.lista-produtos' para branco,
108   proporcionando um contraste limpo com a maioria dos esquemas
109   de cores e destacando visualmente o contêiner na interface. */
110
111  border-right: 1px solid #ddd;
112  /* Adiciona uma borda direita sólida de 1px com uma cor cinza claro (#ddd).
113   Isso serve como uma divisão visual entre a '.lista-produtos' e
114   outros conteúdos ao lado dela no layout flexível. */
```

styles.css > .lista-produtos

```
115
116     box-shadow: 2px 0 5px □rgba(0, 0, 0, 0.1);
117     /* Aplica uma sombra à direita do contêiner '.lista-produtos'.
118      Esta sombra adiciona profundidade visual, melhorando a distinção
119      entre este contêiner e outros elementos adjacentes na página. */
120
121 }
122
123 .lista-produtos select,
124 .lista-produtos input {
125
126     width: 100%;
127     /* Define a largura dos elementos 'select' e 'input' dentro de
128      '.lista-produtos' para ocupar 100% do espaço disponível de seu
129      contêiner pai, garantindo uma apresentação consistente e alinhada
130      dos controles de entrada. */
131
132     padding: 10px;
133     /* Adiciona um preenchimento de 10px, aumentando a área de interação e
134      melhorando a estética das caixas de seleção e campos de entrada. */
135
136     margin-bottom: 10px;
137     /* Aplica uma margem inferior de 10px, criando um espaço entre
138      este elemento e qualquer outro elemento abaixo dele no
139      layout, ajudando a separar visualmente os diferentes
140      controles de entrada. */
141
142     border: 1px solid ■#ddd;
143     /* Adiciona uma borda sólida de 1px com cor cinza clara para
144      definir claramente os limites dos campos de entrada. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

< index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > .lista-produtos select

```
145
146     border-radius: 5px;
147     /* Arredonda os cantos da borda com um raio de 5px, suavizando a
148      |   aparência dos elementos de entrada e proporcionando um
149      |   toque de design moderno e amigável. */
150
151 }
152
153 .lista-produtos ul {
154
155     list-style-type: none;
156     /* Remove qualquer estilo de lista padrão (como marcadores ou
157      |   números) do elemento 'ul', limpa a apresentação visual para
158      |   que apenas o conteúdo da lista seja exibido sem
159      |   distrações adicionais. */
160
161     padding: 0;
162     /* Remove o preenchimento interno do elemento 'ul', garantindo
163      |   que os itens da lista alinhem exatamente com as bordas
164      |   do container '.lista-produtos'. */
165
166 }
167
168 .lista-produtos li {
169
170     padding: 10px;
171     /* Adiciona um preenchimento de 10px em todas as direções dentro
172      |   de cada item de lista, aumentando a área de interação e
173      |   melhorando o conforto visual ao separar o texto das bordas. */
174
```

P24 - LISTA DE PRODUTOS COM IMAGENS

- > imagemProduto
- ▷ index.html
- 🖼️ p24_screenshot.jpg
- JS script.js
- # styles.css
- /vnd Vendedor.xlsx

```
# styles.css > .lista-produtos li
175    border: 1px solid #ddd;
176    /* Aplica uma borda sólida de 1px com uma cor cinza claro (#ddd),
177       delimitando claramente cada item de lista, o que ajuda na
178       organização visual e na distinção entre itens adjacentes. */
179
180
181    margin-bottom: 5px;
182    /* Define uma margem inferior de 5px para cada item de lista,
183       criando um espaço vertical entre eles e evitando uma
184       aparência aglomerada. */
185
186    cursor: pointer;
187    /* Muda o cursor para um ponteiro quando o usuário passa o mouse
188       sobre um item de lista, indicando que o item é clicável. */
189
190    border-radius: 5px;
191    /* Arredonda as bordas dos itens de lista com um raio de 5px,
192       suavizando a aparência geral e tornando o design mais
193       moderno e amigável. */
194
195    transition: background-color 0.3s;
196    /* Aplica uma transição suave para mudanças de cor de fundo que
197       ocorrem, por exemplo, ao passar o mouse. A duração da
198       transição é de 0.3 segundos. */
199
200 }
201
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

< index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > lista-produtos li:hover

```
202 .lista-produtos li:hover {  
203  
204     background-color: #e0e0e0;  
205     /* Muda a cor de fundo de um item de lista para um cinza  
206      | claro (#e0e0e0) quando o mouse está sobre ele,  
207      | fornecendo um feedback visual imediato de que o  
208      | item é interativo. */  
209 }  
210  
211 .lista-produtos button {  
212  
213     width: 100%;  
214     /* Estende a largura dos botões para ocupar 100% da largura  
215      | disponível dentro de seu contêiner, garantindo um  
216      | alinhamento e uma apresentação consistentes. */  
217  
218     padding: 10px;  
219     /* Adiciona um preenchimento de 10px, aumentando a área de  
220      | clique e melhorando a usabilidade, especialmente em  
221      | dispositivos com tela sensível ao toque. */  
222  
223     margin-top: 10px;  
224     /* Define uma margem superior de 10px, separando visualmente os  
225      | botões de outros elementos acima deles. */  
226  
227  
228     background-color: #4CAF50;  
229     /* Define a cor de fundo dos botões para um verde vibrante (#4CAF50),  
230      | tornando-os visualmente distintos e fáceis de identificar  
231      | como elementos clicáveis. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

```
# styles.css > .lista-produtos button
232
233     color: white;
234     /* Define a cor do texto dentro dos botões para branco, criando um
235      | alto contraste com o fundo verde, o que facilita a leitura. */
236
237     border: none;
238     /* Remove a borda padrão dos botões para uma aparência mais
239      | limpa e moderna. */
240
241     border-radius: 5px;
242     /* Arredonda as bordas dos botões para harmonizar com outros
243      | elementos arredondados na interface. */
244
245     cursor: pointer;
246     /* Muda o cursor para um ponteiro quando sobre os botões,
247      | indicando que são interativos. */
248
249     transition: background-color 0.3s;
250     /* Permite uma transição suave de cor de fundo quando o estado
251      | do botão muda, por exemplo, ao passar o mouse, melhorando a
252      | resposta visual. */
253
254 }
255
256 .lista-produtos button:hover {
257
258     background-color: #45a049;
259     /* Escurece a cor de fundo dos botões para um verde mais
260      | profundo (#45a049) quando o mouse está sobre eles,
261      | proporcionando um feedback visual de interatividade. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

< index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > .lista-produtos button:hover

```
262
263 }
264
265 .detalhes-produto {
266
267     width: 70%;
268     /* Define a largura do contêiner `detalhes-produto` para 70% do
269      | seu contêiner pai, permitindo espaço amplo para exibir
270      | informações detalhadas sobre os produtos selecionados. */
271
272     padding: 20px;
273     /* Adiciona um preenchimento interno ao `detalhes-produto`, criando
274      | espaço ao redor do conteúdo, o que melhora a legibilidade
275      | e o apelo visual. */
276
277     background-color: white;
278     /* Estabelece a cor de fundo da seção `detalhes-produto` como branca,
279      | proporcionando um fundo limpo e neutro para os detalhes do produto,
280      | fazendo com que textos ou imagens se destaquem. */
281
282     box-shadow: 2px 0 5px rgba(0, 0, 0, 0.1);
283     /* Aplica uma sombra sutil ao redor do contêiner, ajudando a destacá-lo
284      | visualmente do restante do layout, aumentando a percepção de
285      | profundidade e foco no conteúdo dos detalhes do produto. */
286
287 }
288
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

styles.css > .detalhes-produto img

```
289 .detalhes-produto img {  
290  
291     max-width: 400px;  
292     /* Define a largura máxima das imagens dentro de `detalhes-produto` para  
293      | 400px, garantindo que as imagens não se tornem excessivamente  
294      | grandes e dominem a interface. */  
295  
296     display: block;  
297     /* Configura as imagens para serem exibidas como blocos, o que  
298      | significa que elas ocuparão a linha inteira por si só,  
299      | facilitando a centralização. */  
300  
301     margin: 0 auto 20px;  
302     /* Centraliza a imagem horizontalmente dentro do contêiner e adiciona  
303      | uma margem inferior de 20px, separando visualmente a imagem  
304      | dos outros conteúdos abaixo dela. */  
305  
306     border-radius: 10px;  
307     /* Arredonda os cantos das imagens com um raio de 10px, suavizando a  
308      | aparência geral e tornando o design mais moderno e amigável. */  
309  
310 }  
311  
312 .miniaturas {  
313  
314     display: flex;  
315     /* Utiliza o modelo de caixa flexível para organizar as miniaturas  
316      | dentro do contêiner, permitindo um layout responsivo e adaptável. */  
317
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

```
# styles.css > .detalhes-produto img
318     flex-wrap: wrap;
319     /* Permite que os itens da flexbox sejam envolvidos em várias linhas,
320      |   ajustando-se ao espaço disponível sem esticar os itens. */
321
322     gap: 10px;
323     /* Define um espaço de 10px entre cada miniatura, garantindo que haja
324      |   espaço suficiente para evitar uma aparência aglomerada. */
325
326     justify-content: center;
327     /* Centraliza as miniaturas horizontalmente dentro do container, criando
328      |   um alinhamento visual agradável. */
329
330 }
331
332 .miniaturas img {
333
334     width: 100px;
335     height: 100px;
336     /* Define a largura e altura das miniaturas para 100px, garantindo
337      |   uniformidade no tamanho das imagens, o que ajuda a manter
338      |   um layout consistente. */
339
340     object-fit: cover;
341     /* Garante que as imagens se ajustem às dimensões sem distorcer,
342      |   cobrindo completamente o espaço designado sem perder a proporção. */
343
344     cursor: pointer;
345     /* Muda o cursor para um ponteiro quando o usuário passa o mouse
346      |   sobre as miniaturas, indicando que elas são interativas. */
347
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

```
# styles.css > .miniaturas img
348     border-radius: 10px;
349     /* Arredonda os cantos das miniaturas, alinhando-se ao estilo das
350      |   imagens maiores e mantendo a coerência visual. */
351
352     transition: transform 0.3s, box-shadow 0.3s;
353     /* Adiciona uma transição suave para transformações e sombras,
354      |   melhorando a resposta visual quando as miniaturas são interagidas. */
355
356 }
357
358 .miniaturas img:hover {
359
360     transform: scale(1.1);
361     /* Aumenta a escala da miniatura em 10% quando o mouse passa sobre ela,
362      |   destacando a imagem selecionada e adicionando um efeito visual atraente. */
363
364     box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
365     /* Aplica uma sombra mais pronunciada durante o estado de hover,
366      |   aumentando a percepção de profundidade e destacando ainda
367      |   mais a miniatura sobre a qual o usuário está interagindo. */
368
369 }
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > ...

```
1  document.addEventListener('DOMContentLoaded', function () {  
2      /* Adiciona um ouvinte de evento ao documento que executa  
3          uma função quando todo o conteúdo HTML foi completamente  
4          carregado. Isso garante que o JavaScript não tentará  
5          manipular elementos que ainda não foram renderizados na página. */  
6  
7      const listaProdutos = document.getElementById('lista-produtos');  
8      /* Declara e atribui à variável 'listaProdutos' o elemento HTML  
9          que tem o ID 'lista-produtos'. Este elemento será usado para  
10         listar os produtos disponíveis na página, onde as informações  
11         dos produtos serão inseridas dinamicamente. */  
12  
13      const detalhesProduto = document.getElementById('detalhes-produto');  
14      /* Declara e atribui à variável 'detalhesProduto' o elemento HTML  
15          que tem o ID 'detalhes-produto'. Este espaço é destinado a  
16          mostrar detalhes mais específicos de um produto quando o  
17          usuário clica em algum item da lista. */  
18  
19      const filtroProduto = document.getElementById('filtro-produto');  
20      /* Obtém e armazena o elemento de entrada para filtrar produtos  
21          por nome, que permitirá aos usuários refinar a lista de  
22          produtos exibida baseada em seu input. */  
23  
24      const filtroVendedor = document.getElementById('filtro-vendedor');  
25      /* Acessa e armazena o elemento de seleção que permite ao usuário  
26          filtrar a lista de produtos por vendedor, melhorando a experiência  
27          do usuário ao permitir a visualização segmentada de produtos. */  
28
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > ⚡ document.addEventListener('DOMContentLoaded') callback

```
29 const mostrarTodosBtn = document.getElementById('mostrar-todos');
30 /* Acessa e armazena o botão que, quando clicado, mostrará todos os
31 |   produtos disponíveis, ignorando quaisquer filtros aplicados anteriormente. */
32
33 const totalVendas = document.getElementById('total-vendas');
34 /* Acessa e armazena o elemento que exibe o total de vendas. Esse
35 |   valor será atualizado dinamicamente com base nos produtos
36 |   mostrados na lista ou com base nos filtros aplicados. */
37
38 let dadosTabela = [];
39 /* Declara uma variável 'dadosTabela' como um array vazio, que
40 |   será preenchido com os dados carregados de um arquivo Excel.
41 |   Este array conterá os dados brutos que serão processados e
42 |   exibidos na página. */
43
44 let produtosUnicos = [];
45 /* Declara uma variável 'produtosUnicos' como um array vazio,
46 |   que será usado para armazenar informações de produtos de
47 |   uma maneira que evite repetições, permitindo uma representação
48 |   mais clara e eficiente dos produtos. */
49
50 let vendedoresUnicos = [];
51 /* Declara uma variável 'vendedoresUnicos' como um array vazio,
52 |   que armazenará os nomes dos vendedores sem duplicatas, usado
53 |   para preencher opções de filtro e permitir análises baseadas
54 |   em vendedores específicos. */
55
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback

```
56 // Função para carregar o arquivo Excel
57 function carregarExcel() {
58     /* Define a função 'carregarExcel', responsável por carregar os
59      dados de um arquivo Excel. Esta função não recebe argumentos e
60      encapsula todo o processo de acesso ao arquivo, leitura e
61      extração de dados. */
62
63     fetch('Vendedor.xlsx')
64     /* Utiliza a função global 'fetch' para fazer uma requisição HTTP GET
65      ao arquivo 'Vendedor.xlsx'. 'fetch' retorna uma promessa que,
66      quando resolvida, fornece um objeto de resposta a partir do
67      qual podemos extrair o conteúdo do arquivo. */
68
69     .then(response => response.arrayBuffer())
70     /* O primeiro '.then' lida com a resposta da 'fetch', convertendo o
71      corpo da resposta em um ArrayBuffer. Esta conversão é
72      necessária porque o XLSX.js, a biblioteca usada para ler o
73      arquivo Excel, pode operar com dados binários diretamente
74      para parsear o arquivo. */
75
76     .then(data => {
77         /* O segundo '.then' processa o ArrayBuffer contendo os dados do Excel. */
78
79         const workbook = XLSX.read(data, { type: 'array' });
80         /* Utiliza a biblioteca XLSX.js para converter o ArrayBuffer em
81          um objeto 'workbook', que representa o arquivo Excel carregado. */
82
83         const sheetName = workbook.SheetNames[0];
84         /* Acessa o nome da primeira planilha dentro do arquivo Excel, assumindo
85          que os dados necessários estão na primeira planilha. */
```

JS script.js > ⚡ document.addEventListener('DOMContentLoaded') callback > ⚡ carregarExcel > ⚡ then() callback

```
86
87         const sheet = workbook.Sheets[sheetName];
88         /* Acessa a planilha especificamente pelo nome obtido na etapa anterior. */
89
90         const json = XLSX.utils.sheet_to_json(sheet);
91         /* Converte os dados da planilha especificada em um array de objetos
92             JSON, onde cada objeto representa uma linha da planilha,
93             facilitando a manipulação e o acesso aos dados. */
94
95         dadosTabela = json;
96         /* Armazena os dados convertidos em JSON na variável 'dadosTabela',
97             preparando-os para serem usados nas funções subsequentes de
98             manipulação e display. */
99
100        criarListaProdutos();
101        /* Chama a função 'criarListaProdutos', que processa 'dadosTabela'
102            para extrair e listar produtos únicos. */
103
104        criarListaVendedores();
105        /* Chama a função 'criarListaVendedores', que processa 'dadosTabela'
106            para extrair e listar vendedores únicos. */
107
108        calcularTotalVendas(dadosTabela);
109        /* Chama a função 'calcularTotalVendas', que soma todas as
110            vendas dos produtos e atualiza a visualização do total de vendas. */
111
112    })
113
```

✓ P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > carregarExcel

```
114     .catch(error => console.error('Erro ao carregar o arquivo Excel:', error));
115     /* Captura e lida com qualquer erro que ocorra durante o processo
116      de carregamento ou processamento do arquivo Excel, exibindo
117      uma mensagem de erro no console para diagnóstico. */
118
119 }
120
121 // Função para criar a lista de produtos únicos
122 function criarListaProdutos() {
123     /* Define a função 'criarListaProdutos' que não recebe argumentos
124       externos. Esta função é responsável por processar os dados brutos
125       de 'dadosTabela' para identificar e consolidar produtos únicos,
126       calculando o total de vendas para cada um. */
127
128     const produtosMap = new Map();
129     /* Cria um objeto Map para armazenar os produtos de forma única. O Map
130       permite associar chaves únicas a valores específicos, neste
131       caso, os nomes dos produtos a seus dados correspondentes,
132       facilitando a agregação e atualização de informações sem duplicatas. */
133
134     dadosTabela.forEach(linha => {
135         /* Itera sobre cada 'linha' no array 'dadosTabela', que contém os
136           dados de cada produto listado no arquivo Excel. */
137
138         if (!produtosMap.has(linha.Produto)) {
139             /* Verifica se o 'produtosMap' já possui uma entrada para o
140               produto desta linha. Se não tiver, uma nova entrada é criada. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener("DOMContentLoaded") callback > criarListaProdutos > dadosTabela.forEach() callback

```
142     produtosMap.set(linha.Produto, {  
143         ...linha,  
144         Total: 0  
145         /* Adiciona o produto ao Map com uma cópia dos dados da  
146         linha e inicializa o total de vendas como 0. Isso prepara a  
147         estrutura para acumular vendas sem alterar os outros  
148         atributos do produto. */  
149     });  
150 }  
151  
152     produtosMap.get(linha.Produto).Total += linha.Total;  
153     /* Acessa o produto correspondente no Map e soma o valor 'Total'  
154     da linha atual ao total acumulado do produto, agregando  
155     as vendas de múltiplas entradas que podem existir  
156     para o mesmo produto. */  
157  
158 );  
159  
160  
161     produtosUnicos = Array.from(produtosMap.values());  
162     /* Converte os valores do Map para um array, que agora contém  
163     apenas representações únicas de cada produto com seus  
164     totais de vendas agregados. Este array 'produtosUnicos'  
165     será utilizado para preencher a interface do usuário com  
166     dados limpos e consolidados. */  
167  
168     preencherListaProdutos(produtosUnicos);  
169     /* Chama a função 'preencherListaProdutos', passando o array  
170     'produtosUnicos'. Esta função atualiza a lista visível de  
171     produtos na página com os dados processados. */
```

JS script.js > ⚡ document.addEventListener('DOMContentLoaded') callback > ⚡ criarListaProdutos

```
172
173     exibirMiniaturasProdutos(produtosUnicos);
174     /* Chama a função 'exibirMiniaturasProdutos', também com 'produtosUnicos',
175      para mostrar visualizações em miniatura dos produtos, permitindo
176      uma navegação visual pelos itens disponíveis. */
177
178 }
179
180 // Função para preencher a lista de produtos no HTML
181 function preencherListaProdutos(produtos) {
182
183     // Limpa o conteúdo atual da lista de produtos no HTML
184     listaProdutos.innerHTML = '';
185
186     // Itera sobre cada produto no array 'produtos'
187     produtos.forEach(produto => {
188
189         // Cria um novo elemento 'li' (item de lista) para cada produto
190         const li = document.createElement('li');
191
192         // Define o texto do item de lista como o nome do produto
193         li.textContent = produto.Produto;
194
195         // Adiciona um evento de clique ao item de lista para exibir
196             // os detalhes do produto quando ele for clicado
197         li.addEventListener('click', () => exibirDetalhesProduto(produto));
198
199         // Adiciona o item de lista à lista de produtos no HTML
200         listaProdutos.appendChild(li);
201 }
```

✓ P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

🖼 p24_screenshot.jpg

JS script.js

styles.css

/vndedor.xlsx

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > preencherListaProdutos > produtos.forEach() callback

```
202      });
203  }
204
205  // Função para exibir os detalhes do produto ao clicar
206  function exibirDetalhesProduto(produto) {
207
208      // Define o conteúdo HTML do elemento 'detalhesProduto' com
209      // os detalhes do produto clicado.
210  detalhesProduto.innerHTML =
211      <!-- Insere uma imagem do produto. O caminho da imagem é gerado
212      dinamicamente com base no nome do produto.
213      Atributo 'alt' é usado para fornecer um texto alternativo
214      descritivo para a imagem. --&gt;
215      &lt;img src="imagemProduto/${produto.Produto}.jpg" alt="${produto.Produto}"&gt;
216
217      <!-- Exibe o nome do produto em um cabeçalho de nível 2 (h2) --&gt;
218      &lt;h2&gt;${produto.Produto}&lt;/h2&gt;
219
220      <!-- Exibe o total de vendas do produto em um parágrafo (p),
221      formatado como moeda brasileira (BRL).
222      'Intl.NumberFormat' é uma API que permite formatar números de
223      acordo com a localidade especificada.
224      No caso, a localidade é 'pt-BR' para português do Brasil e o
225      estilo é 'currency' para formatar como moeda. --&gt;
226      &lt;p&gt;Total de Vendas: ${new Intl.NumberFormat('pt-BR', { style: 'currency', currency: 'BRL' })
227      .format(produto.Total)}&lt;/p&gt;
228
229  }</pre>
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback

```
231 // Função para exibir miniaturas de todos os produtos
232 function exibirMiniaturasProdutos(produtos) {
233
234     // Define o conteúdo HTML do elemento 'detalhesProduto' como
235     // um div vazio com a classe 'miniaturas'
236     detalhesProduto.innerHTML = '<div class="miniaturas"></div>';
237
238     // Seleciona o elemento 'div' com a classe 'miniaturas' que
239     // foi inserido dentro de 'detalhesProduto'
240     const miniaturasDiv = detalhesProduto.querySelector('.miniaturas');
241
242     // Itera sobre cada produto no array 'produtos'
243     produtos.forEach(produto => {
244
245         // Cria um novo elemento 'img' para representar a miniatura do produto
246         const img = document.createElement('img');
247
248         // Define o atributo 'src' da imagem para o caminho da imagem
249         // do produto, baseado no nome do produto
250         img.src = `imagemProduto/${produto.Produto}.jpg`;
251
252         // Define o atributo 'alt' da imagem com o nome do produto
253         // para acessibilidade e em caso de falha no carregamento da imagem
254         img.alt = produto.Produto;
255
256         // Adiciona um evento de clique à miniatura para que, ao clicar,
257         // os detalhes completos do produto sejam exibidos
258         img.addEventListener('click', () => exibirDetalhesProduto(produto));
259
260
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > exibirMiniaturasProdutos > produtos.forEach() callback

```
261         // Adiciona a imagem da miniatura ao 'div' com a classe 'miniaturas'
262         miniaturasDiv.appendChild(img);
263     });
264 }
265
266
267 // Função para criar a lista de vendedores únicos
268 function criarListaVendedores() {
269     /* Define a função 'criarListaVendedores', que é responsável por
270      | processar os dados de 'dadosTabela' para extrair uma
271      | lista de vendedores sem duplicidades. */
272
273     const vendedoresSet = new Set(dadosTabela.map(linha => linha.Vendedor));
274     /* Utiliza a classe 'Set', que armazena elementos únicos, para criar
275      | uma coleção de vendedores sem repetições. O método 'map' é
276      | usado para extrair o nome do vendedor de cada linha em
277      | 'dadosTabela', e cada nome é então passado para o 'Set',
278      | garantindo que apenas nomes únicos sejam mantidos. */
279
280     vendedoresUnicos = Array.from(vendedoresSet);
281     /* Converte o 'Set' de vendedores em um array 'vendedoresUnicos',
282      | facilitando iterações futuras e manipulações. Esta lista
283      | será usada para popular o menu dropdown de filtro de
284      | vendedores na interface do usuário. */
285
286     vendedoresUnicos.forEach(vendedor => {
287         /* Itera sobre cada vendedor único no array 'vendedoresUnicos'. */
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > criarListaVendedores > vendedoresUnicos.forEach() callback

```
289     const option = document.createElement('option');
290     /* Cria um novo elemento 'option' para o elemento 'select'
291      do HTML. Este 'option' representa uma opção de filtro
292      para o usuário selecionar. */
293
294     option.value = vendedor;
295     /* Define o atributo 'value' do 'option', que é o valor que será
296      enviado quando o formulário for submetido. Neste caso, é o
297      nome do vendedor. */
298
299     option.textContent = vendedor;
300     /* Define o texto visível dentro do 'option', que também é o nome do
301      vendedor. Este é o texto que o usuário verá no menu dropdown. */
302
303     filtroVendedor.appendChild(option);
304     /* Adiciona o elemento 'option' ao elemento 'select' do
305      filtro de vendedores na página, efetivamente atualizando a
306      interface do usuário com a nova opção de filtro. */
307
308   });
309 }
310
311 // Função para calcular o total de vendas de uma lista de produtos
312 function calcularTotalVendas(produtos) {
313
314   // Usa o método reduce para somar o total de vendas de todos os produtos.
315   // O reduce percorre cada produto no array 'produtos' e
316   // acumula o valor total de vendas
317   const total = produtos.reduce((sum, produto) =>
318
```

✓ P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

🖼 p24_screenshot.jpg

JS script.js

styles.css

☒ Vendedor.xlsx

```
JS script.js > ⚡ document.addEventListener('DOMContentLoaded') callback > ⚡ calcularTotalVendas > [⌚] total > ⚡ produtos.reduce() callback

319         // Para cada produto, adiciona o valor da propriedade
320         |           // 'Total' ao acumulador 'sum'
321         |           // Se 'Total' for undefined ou null, considera 0 para
322         |           // evitar erros na soma
323         |           sum + (produto.Total || 0), // Inicia a soma em 0
324
325     );
326
327     // Formata o total acumulado como moeda em Real (BRL)
328     |           // usando a API Intl.NumberFormat
329     // E exibe esse valor formatado dentro do elemento 'totalVendas' no HTML
330     totalVendas.textContent = `Total de Vendas: ${new Intl.NumberFormat('pt-BR', { style:
331         'currency', currency: 'BRL' }).format(total)}`;
332
333
334     // Função para filtrar a lista de produtos com base nos
335     |           // filtros de produto e vendedor
336     function filtrarListaProdutos() {
337
338         // Obtém o valor digitado no campo de filtro de produto e
339         |           // converte para minúsculas para garantir que a comparação
340         |           // seja case-insensitive (ignora maiúsculas e minúsculas)
341         const filtro = filtroProduto.value.toLowerCase();
342
343         // Obtém o valor selecionado no campo de filtro de vendedor
344         const filtroVend = filtroVendedor.value;
345
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > filtrarListaProdutos

```
346     // Filtra os dados da tabela, retornando apenas as linhas que
347     |       // atendem aos critérios dos filtros
348     const produtosFiltrados = dadosTabela.filter(linha => {
349
350         // Verifica se o nome do produto na linha inclui o texto do filtro
351         |       // de produto (ignorando maiúsculas e minúsculas)
352         const produtoMatch = linha.Produto.toLowerCase().includes(filtro);
353
354         // Verifica se o vendedor na linha corresponde ao filtro de vendedor
355         // Se o filtro de vendedor estiver vazio, considera que a linha
356         |       // corresponde a todos os vendedores
357         const vendedorMatch = filtroVend === "" || linha.Vendedor === filtroVend;
358
359         // Retorna true apenas se ambos os filtros (produto e vendedor)
360         |       // forem atendidos
361         return produtoMatch && vendedorMatch;
362
363     });
364
365
366     // Cria um mapa para armazenar produtos únicos, onde a
367     |       // chave é o nome do produto
368     // e o valor é um objeto contendo os detalhes do produto e
369     |       // o total acumulado de vendas
370     const produtosMap = new Map();
371
372     // Itera sobre os produtos filtrados para agrupar e acumular o
373     |       // total de vendas por produto
374     produtosFiltrados.forEach(linha => {
375
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > filtrarListaProdutos > produtosFiltrados.forEach() callback

```
376     // Verifica se o mapa já contém uma entrada para o produto
377         // atual (usando o nome do produto como chave)
378         if (!produtosMap.has(linha.Produto)) {
379
380             // Se o produto ainda não estiver no mapa, cria uma
381             // nova entrada com o produto
382             // A chave é o nome do produto, e o valor é um objeto que
383             // contém todas as propriedades da linha
384             // Inicializa o total de vendas do produto como 0,
385             // pronto para ser acumulado
386             produtosMap.set(linha.Produto, {
387                 ...linha, // Copia todas as propriedades da linha atual para o novo objeto
388                 Total: 0 // Inicializa o total de vendas com 0
389             });
390         }
391
392         // Incrementa o total de vendas do produto no mapa,
393         // somando o valor da venda atual
394         // Isso permite que, se houver várias ocorrências do mesmo
395         // produto, seus totais de vendas sejam somados
396         produtosMap.get(linha.Produto).Total += linha.Total;
397
398     });
399
400
401     // Converte os valores do mapa (produtos únicos com totais acumulados) em um array
402     // Isso facilita a manipulação dos dados e permite passar o array para outras funções
403     const produtosUnicosFiltrados = Array.from(produtosMap.values());
404
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > document.addEventListener('DOMContentLoaded') callback > filtrarListaProdutos

```
405     // Preenche a lista de produtos na interface com os produtos únicos filtrados
406     preencherListaProdutos(produtosUnicosFiltrados);
407
408     // Exibe as miniaturas dos produtos filtrados na interface
409     exibirMiniaturasProdutos(produtosUnicosFiltrados);
410
411     // Calcula e exibe o total de vendas dos produtos que
412     // passaram pelo filtro
413     calcularTotalVendas(produtosFiltrados);
414
415 }
416
417
418     // Adiciona um evento ao campo de filtro de produto que dispara a
419     // função de filtrar a lista de produtos
420     // toda vez que o usuário digitar algo no campo ('input' é
421     // acionado em cada entrada do usuário)
422     filtroProduto.addEventListener('input', filtrarListaProdutos);
423
424     // Adiciona um evento ao campo de filtro de vendedor que dispara a
425     // função de filtrar a lista de produtos
426     // toda vez que o usuário mudar a seleção do vendedor ('change' é
427     // acionado ao mudar a opção selecionada)
428     filtroVendedor.addEventListener('change', filtrarListaProdutos);
429
430     // Adiciona um evento ao botão "Mostrar Todos" que, ao ser
431     // clicado, executa as seguintes ações:
432     mostrarTodosBtn.addEventListener('click', () => {
433 }
```

P24 - LISTA DE PRODUTOS COM IMAGENS

> imagemProduto

<> index.html

p24_screenshot.jpg

JS script.js

styles.css

Vendedor.xlsx

JS script.js > ⚡ document.addEventListener('DOMContentLoaded') callback > ⚡ filtrarListaProdutos

```
434     // Preenche a lista de produtos na interface com todos
435     // os produtos únicos (sem filtro)
436     preencherListaProdutos(produtosUnicos);
437
438     // Exibe as miniaturas de todos os produtos únicos na interface
439     exibirMiniaturasProdutos(produtosUnicos);
440
441     // Calcula e exibe o total de vendas considerando todos
442     // os dados da tabela (sem filtro)
443     calcularTotalVendas(dadosTabela);
444
445 );
446
447     // Chama a função carregarExcel para iniciar o processo de
448     // carregamento dos dados a partir de um arquivo Excel
449     carregarExcel();
450
451 );
```