Criando um Frontend com HTML, CSS e JavaScript para Gerenciar Produtos 🛍️

Objetivos:

- Estruturar uma página web semanticamente com HTML.
- Estilizar elementos da página usando CSS.
- Adicionar interatividade e manipular o conteúdo da página dinamicamente com JavaScript.
- Consumir dados de uma API backend para buscar e enviar informações de produtos.
- Construir uma interface funcional para listar e cadastrar produtos.

Pré-requisitos conceituais:

- Lógica de programação básica.
- Entendimento do que é uma API RESTful (GET, POST) e o formato JSON.
- Consciência da existência de um backend para produtos (id, nome, preco).

1. O Que é Desenvolvimento Frontend (Sem Frameworks)?

O **Desenvolvimento Frontend** é a arte de construir a interface com a qual o usuário interage diretamente em um navegador web. Quando não utilizamos frameworks ou bibliotecas como React, Angular ou Vue.js, estamos trabalhando com as tecnologias fundamentais da web:

- **HTML (HyperText Markup Language):** Define a **estrutura** e o conteúdo da página (títulos, parágrafos, imagens, formulários, etc.). É o esqueleto da sua aplicação.
- **CSS (Cascading Style Sheets):** Descreve a **apresentação** e o estilo visual da página HTML (cores, fontes, layout, espaçamento, animações básicas). É a "roupa" e a "maquiagem" da sua aplicação.
- **JavaScript (JS):** Adiciona **interatividade** e comportamento dinâmico à página (responder a cliques de botão, validar formulários, buscar dados de um servidor, atualizar o conteúdo da página sem recarregá-la). É o "cérebro" e os "músculos" da sua aplicação no lado do cliente.

Trabalhar diretamente com essas três tecnologias fornece uma compreensão profunda de como a web funciona.

2. Ferramentas Essenciais

Para começar, você precisará de:

1. **Um Editor de Código:**

- Visual Studio Code (VS Code) - Altamente recomendado, gratuito e com muitas extensões úteis.
- Sublime Text
- Atom
- Notepad++ (Windows) ou TextEdit (Mac) - Para algo bem básico.

2. **Um Navegador Web Moderno:**
   - Google Chrome (excelente para ferramentas de desenvolvedor)
   - Mozilla Firefox (também com ótimas ferramentas de desenvolvedor)
   - Microsoft Edge (baseado no Chromium, similar ao Chrome)
3. **Acesso às Ferramentas de Desenvolvedor do Navegador:**
   - Geralmente acessadas clicando com o botão direito na página e selecionando "Inspecionar" ou "Inspecionar Elemento", ou pressionando F12.
   - Essas ferramentas são cruciais para depurar HTML, CSS e JavaScript, inspecionar requisições de rede e muito mais.

3. HTML: A Estrutura da Página de Produtos

Vamos criar um arquivo chamado index.html. Este arquivo conterá a estrutura básica da nossa aplicação de gerenciamento de produtos.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Catálogo de Produtos</title>
  <link rel="stylesheet" href="estilos.css"> </head>
<body>
  <header>
    <h1>Meu Catálogo de Produtos</h1>
  </header>

  <main>
    <section id="formulario-secao">
      <h2>Adicionar Novo Produto</h2>
      <form id="form-produto">
        <div>
          <label for="nome-produto">Nome:</label>
          <input type="text" id="nome-produto" name="nome-produto" required>
```

```
            </div>
            <div>
                <label for="preco-produto">Preço (R$):</label>
                <input type="number" id="preco-produto" name="preco-produto"
step="0.01" min="0" required>
            </div>
            <button type="submit">Adicionar Produto</button>
            <p id="mensagem-formulario" class="mensagem"></p>
        </form>
    </section>

    <hr>

    <section id="lista-produtos-secao">
        <h2>Lista de Produtos</h2>
        <div id="lista-produtos-container">
            <p id="mensagem-lista" class="mensagem">Carregando produtos...</p>
        </div>
    </section>
</main>

<footer>
    <p>© 2025 Minha Loja de Produtos</p>
</footer>

<script src="app.js"></script> </body>
</html>
```

Principais Elementos HTML Utilizados:

- <!DOCTYPE html>: Define o tipo de documento.
- <html lang="pt-BR">: Elemento raiz, com o idioma definido para português do Brasil.
- <head>: Contém meta-informações sobre o HTML (não visíveis diretamente na página).
    - <meta charset="UTF-8">: Define o conjunto de caracteres.
    - <meta name="viewport" ...>: Configura a página para ser responsiva em diferentes tamanhos de tela.
    - <title>: O título que aparece na aba do navegador.

    - <link rel="stylesheet" href="estilos.css">: Vincula nosso arquivo CSS externo.
- <body>: Contém o conteúdo visível da página.
- <header>, <main>, <footer>: Tags semânticas que estruturam as principais seções da página.
- <h1>, <h2>: Títulos de diferentes níveis.
- <section>: Agrupa conteúdo tematicamente relacionado.
- <form id="form-produto">: Define um formulário para entrada de dados.
    - <label for="...">: Rótulo para um campo de formulário. O atributo for deve corresponder ao id do input.
    - <input type="..." id="..." name="..." required>: Campos de entrada.
        - type="text": Para texto.
        - type="number": Para números, com step="0.01" para decimais e min="0" para evitar preços negativos.
        - required: Torna o campo obrigatório.
    - <button type="submit">: Botão que envia o formulário.
- : Um container genérico, usado aqui para agrupar a lista de produtos.
- <p>: Parágrafo.
- <hr>: Linha horizontal para separação visual.
- <script src="app.js"></script>: Vincula nosso arquivo JavaScript externo. É comum colocar scripts no final do <body> para que o HTML seja carregado e analisado antes que o JavaScript tente manipulá-lo.

4. CSS: Estilizando a Página (estilos.css)

Vamos criar um arquivo chamado estilos.css na mesma pasta do index.html. Este arquivo adicionará um pouco de estilo visual.

```
/* estilos.css */
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
    color: #333;
}

header {
    background: #333;
```

```
    color: #fff;
    padding: 1rem 0;
    text-align: center;
}

header h1 {
    margin: 0;
}

main {
    padding: 20px;
    max-width: 900px;
    margin: 20px auto;
    background: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
}

section {
    margin-bottom: 20px;
    padding: 15px;
    border-bottom: 1px solid #eee;
}

section:last-child {
    border-bottom: none;
}

h2 {
    color: #333;
    margin-top: 0;
}

form div {
    margin-bottom: 15px;
}

label {
    display: block;
```

```
    margin-bottom: 5px;
    font-weight: bold;
}

input[type="text"],
input[type="number"] {
    width: calc(100% - 22px); /* Considera padding e borda */
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
    box-sizing: border-box; /* Garante que padding e borda não aumentem a largura
total */
}

button[type="submit"] {
    display: inline-block;
    background: #5cb85c;
    color: #fff;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    font-size: 16px;
}

button[type="submit"]:hover {
    background: #4cae4c;
}

.produto-item {
    background: #f9f9f9;
    border: 1px solid #eee;
    padding: 15px;
    margin-bottom: 10px;
    border-radius: 4px;
}

.produto-item h3 {
    margin-top: 0;
```

```
    color: #555;
}

.produto-item p {
    margin-bottom: 5px;
}

.mensagem {
    margin-top: 10px;
    padding: 10px;
    border-radius: 4px;
}

.mensagem.sucesso {
    background-color: #dff0d8;
    color: #3c763d;
    border: 1px solid #d6e9c6;
}

.mensagem.erro {
    background-color: #f2dede;
    color: #a94442;
    border: 1px solid #ebccd1;
}

footer {
    text-align: center;
    padding: 20px;
    background: #333;
    color: #fff;
    margin-top: 30px;
}
```

Conceitos de CSS Aplicados:

- **Seletores:** body, header, input[type="text"], .produto-item, #form-produto.
- **Propriedades:** font-family, background-color, padding, margin, border-radius, display, width.
- **Box Model:** padding, border, margin, e como box-sizing: border-box; afeta o

cálculo da largura/altura.
- **Classes (.):** Para aplicar estilos a múltiplos elementos (ex: .produto-item, .mensagem).
- **IDs (#):** Para aplicar estilos a um elemento único (geralmente usado com moderação em CSS, mais comum para JavaScript).

5. JavaScript: Interatividade e Comunicação com o Backend (app.js)

Crie um arquivo app.js na mesma pasta.

```
// app.js
document.addEventListener('DOMContentLoaded', () => {
    // URLs da API (ajuste a porta se o seu backend rodar em outra)
    const API_URL = 'http://localhost:PORTA_DO_BACKEND/api/produtos'; // <<--
AJUSTE A PORTA AQUI

    // Elementos do DOM
    const formProduto = document.getElementById('form-produto');
    const nomeProdutoInput = document.getElementById('nome-produto');
    const precoProdutoInput = document.getElementById('preco-produto');
    const listaProdutosContainer =
document.getElementById('lista-produtos-container');
    const mensagemFormulario = document.getElementById('mensagem-formulario');
    const mensagemLista = document.getElementById('mensagem-lista');

    // Função para exibir mensagens
    function exibirMensagem(elemento, texto, tipo = 'info') {
        elemento.textContent = texto;
        elemento.className = 'mensagem'; // Reseta classes
        if (tipo === 'sucesso') {
            elemento.classList.add('sucesso');
        } else if (tipo === 'erro') {
            elemento.classList.add('erro');
        }
        // Remove a mensagem após alguns segundos
        setTimeout(() => {
            elemento.textContent = '';
            elemento.className = 'mensagem';
        }, 5000);
```

```
    }

    // Função para renderizar um único produto na lista
    function renderizarProduto(produto) {
        const produtoDiv = document.createElement('div');
        produtoDiv.classList.add('produto-item');
        produtoDiv.setAttribute('data-id', produto.id); // Útil para futuras manipulações
(editar/deletar)

        const nomeEl = document.createElement('h3');
        nomeEl.textContent = produto.nome;

        const precoEl = document.createElement('p');
        precoEl.textContent = `Preço: R$ ${parseFloat(produto.preco).toFixed(2)}`;

        const idEl = document.createElement('p');
        idEl.textContent = `ID: ${produto.id}`;

        produtoDiv.appendChild(nomeEl);
        produtoDiv.appendChild(precoEl);
        produtoDiv.appendChild(idEl);

        // Adicionar botões de ação (exemplo para o futuro)
        // const btnEditar = document.createElement('button');
        // btnEditar.textContent = 'Editar';
        // btnEditar.onclick = () => console.log('Editar produto ID:', produto.id);
        // produtoDiv.appendChild(btnEditar);

        // const btnDeletar = document.createElement('button');
        // btnDeletar.textContent = 'Deletar';
        // btnDeletar.onclick = () => console.log('Deletar produto ID:', produto.id);
        // produtoDiv.appendChild(btnDeletar);

        return produtoDiv;
    }

    // Função para buscar e exibir todos os produtos
    async function carregarProdutos() {
        mensagemLista.textContent = 'Carregando produtos...';
```

```
        mensagemLista.className = 'mensagem'; // Reset
        listaProdutosContainer.innerHTML = ''; // Limpa a lista antiga antes de carregar
        listaProdutosContainer.appendChild(mensagemLista);


        try {
            const response = await fetch(API_URL);
            if (!response.ok) {
                throw new Error(`Erro na requisição: ${response.status}
${response.statusText}`);
            }
            const produtos = await response.json();

            mensagemLista.style.display = 'none'; // Esconde "Carregando..."

            if (produtos.length === 0) {
                exibirMensagem(mensagemLista, 'Nenhum produto cadastrado.', 'info');
                mensagemLista.style.display = 'block';
            } else {
                produtos.forEach(produto => {
                    const produtoEl = renderizarProduto(produto);
                    listaProdutosContainer.appendChild(produtoEl);
                });
            }
        } catch (error) {
            console.error('Erro ao carregar produtos:', error);
            exibirMensagem(mensagemLista, `Erro ao carregar produtos:
${error.message}`, 'erro');
            mensagemLista.style.display = 'block';
        }
    }

    // Função para lidar com o envio do formulário de novo produto
    async function handleAdicionarProduto(event) {
        event.preventDefault(); // Previne o recarregamento da página

        const nome = nomeProdutoInput.value.trim();
        const preco = parseFloat(precoProdutoInput.value);
```

```
    if (!nome || isNaN(preco) || preco <= 0) {
        exibirMensagem(mensagemFormulario, 'Por favor, preencha o nome e um
preço válido.', 'erro');
        return;
    }

    const novoProduto = { nome, preco };

    try {
        const response = await fetch(API_URL, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify(novoProduto),
        });

        if (!response.ok) {
            // Tenta ler a mensagem de erro do backend, se houver
            let errorMsg = `Erro ao cadastrar produto: ${response.status}
${response.statusText}`;
            try {
                const errorData = await response.json();
                if (errorData && errorData.message) {
                    errorMsg = errorData.message;
                }
            } catch (e) { /* Ignora se não conseguir parsear o JSON do erro */ }
            throw new Error(errorMsg);
        }

        // const produtoAdicionado = await response.json(); // O backend retorna o
produto criado
        exibirMensagem(mensagemFormulario, 'Produto adicionado com sucesso!',
'sucesso');

        formProduto.reset(); // Limpa o formulário
        await carregarProdutos(); // Recarrega a lista de produtos

    } catch (error) {
```

```
            console.error('Erro ao adicionar produto:', error);
            exibirMensagem(mensagemFormulario, error.message, 'erro');
        }
    }

    // Adicionar Event Listeners
    if (formProduto) {
        formProduto.addEventListener('submit', handleAdicionarProduto);
    }

    // Carregar produtos inicialmente
    carregarProdutos();
});
```

Principais Conceitos de JavaScript Aplicados:

- **document.addEventListener('DOMContentLoaded', () => { ... });**: Garante que o script só rode depois que todo o HTML da página foi completamente carregado e analisado pelo navegador. Isso evita erros ao tentar acessar elementos que ainda não existem.
- **Seleção de Elementos do DOM:**
  - document.getElementById('id-do-elemento'): Seleciona um elemento pelo seu id.
  - (Outros seletores: document.querySelector(), document.querySelectorAll(), document.getElementsByClassName(), etc.)
- **Manipulação do DOM:**
  - elemento.textContent = 'novo texto': Altera o texto de um elemento.
  - elemento.innerHTML = 'html aqui': Altera o HTML interno de um elemento (use com cuidado devido a riscos de segurança XSS se o HTML vier de fontes não confiáveis).
  - document.createElement('div'): Cria um novo elemento HTML.
  - elemento.classList.add('nova-classe'), elemento.classList.remove('classe'): Adiciona ou remove classes CSS.
  - elemento.appendChild(filho): Adiciona um elemento filho a outro.
  - elemento.setAttribute('atributo', 'valor'): Define um atributo.
- **Event Listeners:**
  - formProduto.addEventListener('submit', handleAdicionarProduto);: Executa a função handleAdicionarProduto quando o evento submit (envio) do formulário ocorrer.

    - event.preventDefault(): Impede o comportamento padrão de um evento (no caso do submit de um formulário, impede que a página seja recarregada).
- **Funções Assíncronas (async/await):**
    - Usadas para lidar com operações que levam tempo, como chamadas de API, sem bloquear a thread principal do navegador (o que congelaria a interface).
    - async function minhaFuncao() { ... }: Declara uma função assíncrona.
    - await expressao: Pausa a execução da função async até que a Promise retornada pela expressao seja resolvida ou rejeitada.
- **fetch API:**
    - Usada para fazer requisições HTTP para o backend.
    - fetch(API_URL): Faz uma requisição GET por padrão.
    - fetch(API_URL, { method: 'POST', headers: { ... }, body: JSON.stringify(dados) }): Faz uma requisição POST, enviando dados no corpo.
        - 'Content-Type': 'application/json': Informa ao servidor que estamos enviando dados em formato JSON.
        - JSON.stringify(novoProduto): Converte um objeto JavaScript em uma string JSON.
    - response.ok: Verifica se a resposta HTTP foi bem-sucedida (status entre 200-299).
    - response.json(): Converte o corpo da resposta (que se espera ser JSON) em um objeto JavaScript.
- **Tratamento de Erros (try...catch):**
    - Usado com async/await para capturar erros que podem ocorrer durante as chamadas de API ou outras operações.
- **Validação de Formulário Simples:** Verifica se os campos não estão vazios e se o preço é um número válido.

Importante:

- **AJUSTE A PORTA:** No app.js, substitua 'http://localhost:PORTA_DO_BACKEND/api/produtos' pela URL correta e porta onde seu backend está rodando (ex: http://localhost:3000/api/produtos ou http://localhost:5000/api/produtos).
- **CORS (Cross-Origin Resource Sharing):** Se o seu frontend (servido diretamente do sistema de arquivos ou por um servidor local simples como o Live Server do VS Code) estiver em uma "origem" diferente do seu backend (ex: localhost:8080 vs localhost:3000), o backend precisará ser configurado para permitir requisições do frontend. Isso geralmente envolve adicionar headers como Access-Control-Allow-Origin: * (para desenvolvimento) ou Access-Control-Allow-Origin: http://sua-origem-frontend no servidor backend.

6. Executando a Aplicação

1. Salve os três arquivos (index.html, estilos.css, app.js) na mesma pasta.
2. Abra o arquivo index.html diretamente no seu navegador.
3. Certifique-se de que seu servidor backend (que você criou anteriormente ou que foi fornecido) esteja rodando e acessível na URL configurada no app.js.

Você deverá ver o formulário para adicionar produtos e a lista de produtos (que será preenchida após a chamada à API).