🚀 Guia de Desenvolvimento: E-commerce Fullstack

Este documento é o seu manual para modificar e expandir o projeto.

A Regra de Ouro: O Fluxo de Trabalho

Quase toda alteração que você fizer seguirá um fluxo de 6 passos, sempre começando pelo "fundo" (Banco de Dados) e subindo para o "topo" (Interface do Usuário).

Fluxo de Modificação (Backend para Frontend):

1. **Banco (Modelo):** Alterar o arquivo `.cs` do Modelo.
2. **Banco (Migração):** Rodar os comandos `dotnet ef migrations` para atualizar o banco.
3. **Backend (API):** Atualizar o `Program.cs` se a lógica de salvar ou buscar mudou.
4. **Frontend (Modelo):** Alterar o arquivo `.ts` de Interface.
5. **Frontend (Formulário):** Adicionar o campo no `CadastrarProduto.tsx`.
6. **Frontend (Visualização):** Exibir o campo no `ListarProdutos.tsx`.

🔧 Tutorial 1: Adicionar um Novo Campo (Ex: "SKU")

Vamos adicionar um campo `Sku` (Código de Unidade) do tipo `string` ao `Produto`.

Passo 1: Backend (Modelo)

Edite o arquivo `API/Models/Produto.cs` e adicione a nova propriedade dentro da classe `Produto`. Adicione esta linha: `public string Sku { get; set; } = string.Empty;`

Passo 2: Backend (Migração)

No terminal, dentro da pasta `API`, crie e aplique a migração.

1. Crie a migração com o comando: `dotnet ef migrations add AddSkuProduto`
2. Aplique no banco de dados com: `dotnet ef database update` O seu banco de dados (`Ecommerce.db`) agora possui uma coluna "Sku" na tabela "Produtos".

Passo 3: Backend (API)

Edite o arquivo `API/Program.cs`.

- Para `MapPost` (Cadastrar), nenhuma mudança é necessária.

- Para `MapPatch` (Alterar), precisamos adicionar a lógica de atualização. Na rota `app.MapPatch("/api/produto/alterar/{id}", ...)` adicione esta linha antes do `Update`: `resultado.Sku = produtoAlterado.Sku;`

Passo 4: Frontend (Modelo)

Edite o arquivo `Frontend/src/Models/Produto.ts` para que o React "saiba" da existência do SKU. Adicione esta linha na interface `Produto`: `sku: string;`

Passo 5: Frontend (Formulário)

Edite o arquivo `Frontend/src/Components/Pages/Produtos/CadastrarProduto.tsx`.

1. Adicione um novo `useState` para o campo: `const [sku, setSku] = useState("");`
2. Adicione o campo no objeto enviado para a API, dentro do `const produto: Produto = { ... }`: `sku,`
3. Adicione o `input` no formulário (JSX). Crie um novo `<div>` contendo uma `label` ("SKU:") e um `input` com os atributos `type="text"` e `onChange={(e: any) => setSku(e.target.value)}`.

Passo 6: Frontend (Visualização)

Edite o arquivo `Frontend/src/Components/Pages/Produtos/ListarProdutos.tsx`.

1. Adicione a coluna no cabeçalho da tabela `<thead>`: `<th>SKU</th>`
2. Adicione a célula `<td>` no corpo da tabela `<tbody>`, dentro do `.map()`: `<td>{produto.sku}</td>`

Pronto! Você modificou o sistema de ponta a ponta.

📦 Tutorial 2: Adicionar uma Nova Função (Vincular "Categorias")

Seu código já possui um `Models/Categoria.cs`, mas ele não está conectado a `Produto`. Vamos conectá-los (Relacionamento 1-N).

Passo 1: Backend (Modelos)

Precisamos criar a "chave estrangeira".

1. Em `API/Models/Categoria.cs`, adicione a lista de produtos (propriedade de navegação): `public List<Produto>? Produtos { get; set; }`

2. Em `API/Models/Produto.cs`, adicione a referência à Categoria. Adicione estas duas linhas: `public int CategoriaId { get; set; }` (a chave estrangeira) e `public Categoria? Categoria { get; set; }` (a propriedade de navegação).

Passo 2: Backend (Migração)

Rode os comandos na pasta `API`: `dotnet ef migrations add AddRelacionamentoProdutoCategoria` `dotnet ef database update`

Passo 3: Backend (API)

Precisamos de novas rotas para Categorias e atualizar as rotas de Produto. Edite o `API/Program.cs`.

1. **Novas Rotas (Categorias):** Adicione estas rotas:
    - Uma rota `app.MapGet("/api/categoria/listar", ...)` que retorna `Results.Ok(ctx.Categorias.ToList())` se houver categorias.
    - Uma rota `app.MapPost("/api/categoria/cadastrar", ...)` que faz `ctx.Categorias.Add(categoria)` e `ctx.SaveChanges()`.
2. **Atualizar Rotas (Produto):** Precisamos usar `.Include()` para trazer os dados da Categoria junto com o Produto (*Eager Loading*).
    - Em `app.MapGet("/api/produto/listar", ...)`: substitua `ctx.Produtos.ToList()` por `ctx.Produtos.Include(p => p.Categoria).ToList()`.
    - Em `app.MapGet("/api/produto/buscar/{id}", ...)`: substitua `Produto? resultado = ctx.Produtos.Find(id);` por `Produto? resultado = ctx.Produtos.Include(p => p.Categoria).FirstOrDefault(p => p.Id == id);`

Passo 4: Frontend (Modelos)

Crie os modelos TypeScript para refletir o C#.

1. Crie o arquivo `Frontend/src/Models/Categoria.ts` e defina uma interface `Categoria` com os campos `categoriaId: number;`, `nome: string;` e `criadoEm: string;`.
2. Atualize o `Frontend/src/Models/Produto.ts`. Importe a `Categoria` e adicione estas linhas na interface `Produto`: `categoriaId: number;` e `categoria?: Categoria;` (o `?` indica que é opcional).

Passo 5: Frontend (Formulário)

Em `CadastrarProduto.tsx`, vamos usar um `<select>` (dropdown) que busca as categorias da API.

1. Importe `useEffect` (do React) e `Categoria` (do seu modelo).
2. Adicione novos `useState`: `const [categoriaId, setCategoriaId] = useState(0);` e `const [categorias, setCategorias] = useState<Categoria[]>([]);`
3. Adicione um `useEffect` para carregar as categorias quando a página abrir. O código deve ser: `useEffect(() => { carregarCategorias(); }, []);`
4. Crie a função `async function carregarCategorias()` que usa `axios.get("http://localhost:5011/api/categoria/listar")` e então chama `setCategorias(resposta.data)`.
5. Atualize o `submeterProdutoAPI` para enviar o `categoriaId`. Dentro do objeto `produto`, adicione: `categoriaId: Number(categoriaId),`
6. Atualize o JSX para mostrar um `<select>`. Adicione um `<div>` com uma `label` ("Categoria:") e um `<select onChange={(e: any) => setCategoriaId(e.target.value)}>`.
    - Dentro do select, adicione a opção padrão: `<option value={0}>Selecione uma categoria</option>`.
    - Abaixo, mapeie o array `categorias` para criar as opções: `{categorias.map((categoria) => (<option key={categoria.categoriaId} value={categoria.categoriaId}>{categoria.nome}</option>))}`.

Passo 6: Frontend (Visualização)

Em `ListarProdutos.tsx`, mostre o *nome* da categoria.

1. Adicione a coluna em `<thead>`: `<th>Categoria</th>`
2. Adicione a célula `<td>` em `<tbody>`: `<td>{produto.categoria?.nome}</td>` Usamos `produto.categoria?.nome` (com `?`) por segurança. Se um produto for cadastrado sem categoria, isso evita que o React quebre.