1. EPIC (Grande Iniciativa Estratégica)

Título do Epic: Modernização e Escalabilidade da Plataforma de Inventário Comercial

- **Descrição do Valor de Negócio:** Atualmente, a operação depende de processos manuais e planilhas descentralizadas para gerenciar o catálogo de mercadorias. A centralização e automação da gestão de produtos e categorias em uma plataforma web segura e reativa visa mitigar erros de estoque, reduzir o tempo de cadastro de novos itens em 40% e fornecer uma base de dados íntegra e relacional para futuras integrações com módulos de vendas e faturamento.
- **Métricas de Sucesso:**
    1. Disponibilidade de uma API RESTful capaz de processar operações CRUD completas com tempo de resposta inferior a 200ms por requisição.
    2. Redução a zero de inconsistências de órfãos no banco de dados através da aplicação de integridade referencial rígida.
    3. Interface do usuário reativa que dispense o recarregamento total da página (`window.location.reload()`) durante a manutenção de dados.

2. FEATURE (Escopo Funcional e Critérios de Alto Nível)

Título da Feature: Módulo de Gestão de Catálogo (Produtos e Categorias)

- **Feature Mãe:** Epic #10024 - Modernização e Escalabilidade da Plataforma de Inventário Comercial
- **Descrição Funcional:** Desenvolvimento de ponta a ponta (Full Stack) de um ecossistema funcional de cadastro, listagem, edição e deleção de Produtos e Categorias. O candidato deve estruturar uma API em .NET acoplada a um banco de dados real e uma interface em Nuxt 3 onde o grid de listagem funciona como a central de ações (inline ou modal) para manipulação dos dados.
- **Critérios de Aceite de Alto Nível:**
    - O sistema deve persistir informações em tabelas relacionais de verdade através do EF Core. Uso de simuladores de banco de dados em memória desclassificará o candidato por não simular o cenário de concorrência real de produção.
    - O grid do front-end deve sincronizar seu estado imediatamente após o sucesso de mutações (PUT e DELETE) no servidor.

3. USER STORIES (Histórias de Usuário)

User Story 1 (Back-end): API RESTful de Persistência e Ciclo de Vida do Catálogo (.NET)

- **Template:**
  - **Como** Desenvolvedor Backend da aplicação,
  - **Quero** expor endpoints estruturados para criação, leitura, atualização e exclusão (CRUD) de produtos e categorias,
  - **Para** que a camada cliente possa manipular e consumir dados íntegros, validados e de forma performática.
- **Acceptance Criteria (Objetivos e Mensuráveis):**
  - **AC 01 - Arquitetura e Persistência:** Configurar o **Entity Framework Core** utilizando exclusivamente um banco relacional permanente (SQL Server, PostgreSQL ou MySQL). Variáveis estáticas no código ou o provedor *InMemory* do EF Core **não** são aceitos.
  - **AC 02 - Modelagem Relacional:** Criar as tabelas de `Categoria` (`Id`, `Nome`, `Descrição`) e `Produto` (`Id`, `Nome`, `Descrição`, `Preço`, `CategoriaId`). O mapeamento via Fluent API ou Data Annotations deve explicitar o relacionamento **1 para Muitos (1:N)** com Chaves Estrangeiras (`FK`).
  - **AC 03 - Contrato de Endpoints:** Expor estritamente as rotas:
    - `GET /api/categorias` e `POST /api/categorias`
    - `PUT /api/categorias/{id}` e `DELETE /api/categorias/{id}`
    - `GET /api/produtos` (deve incluir o objeto aninhado da categoria vinculada utilizando `.Include()`)
    - `POST /api/produtos`, `PUT /api/produtos/{id}` e `DELETE /api/produtos/{id}`
  - **AC 04 - Regra de Integridade Referencial:** Ao receber um `DELETE /api/categorias/{id}`, o back-end deve verificar se existem produtos associados a ela. Caso existam, a exclusão deve ser impedida, retornando HTTP Status `409 Conflict` ou `400 Bad Request` com a mensagem textual: *"Não é possível excluir uma categoria que possua produtos vinculados."*
  - **AC 05 - Validação de Payload:** Os endpoints de criação (`POST`) e edição (`PUT`) devem rejeitar requisições em que o campo `Nome` seja nulo ou possua menos de 5 caracteres, respondendo com HTTP Status `400 Bad Request` e um sumário dos erros.
  - **AC 06 - Segurança de Comunicação (CORS):** Implementar uma política explícita de CORS no arquivo `Program.cs`, liberando requisições unicamente da URL/porta oficial do projeto frontend Nuxt 3. O uso de `.AllowAnyOrigin()` é proibido.

User Story 2 (Front-end): Grid Reativo de Gerenciamento e Manutenção de Dados (Vue 3 / Nuxt 3)

- **Template:**
  - **Como** Operador do sistema de estoque,
  - **Quero** uma interface fluida que liste os dados e me permita criar, editar ou excluir produtos e categorias diretamente pelo Grid,
  - **Para** manter o catálogo comercial atualizado com agilidade e sem transições desnecessárias de página.
- **Acceptance Criteria (Objetivos e Mensuráveis):**
  - **AC 01 - Estado e Comunicação Assíncrona:** Gerenciar o formulário e os dados do Grid usando a Composition API (`ref`, `reactive`). A comunicação com a API .NET deve utilizar **Axios** ou o cliente nativo do Nuxt (`$fetch`/`useFetch`).
  - **AC 02 - Coluna de Ações no Grid:** A tabela/grid de listagem de categorias e de produtos deve possuir obrigatoriamente uma coluna final chamada "Ações", contendo dois botões distintos por linha: **Editar** e **Excluir**.
  - **AC 03 - Fluxo de Edição Dinâmica:** Ao clicar em "Editar", o candidato deve abrir um componente de diálogo/modal preenchido com as informações atuais daquela linha, ou transformar a linha do grid em campos editáveis (*inline*). O salvamento deve disparar um `PUT` para a API.
  - **AC 04 - Validação em Tela:** O botão "Salvar" (do cadastro e da edição) deve permanecer desativado (`disabled`) enquanto o campo `Nome` não atender à regra de negócio (mínimo de 5 caracteres preenchidos).
  - **AC 05 - Confirmação de Deleção:** Ao acionar o botão "Excluir" em uma linha, a interface deve obrigatoriamente interceptar o clique e exibir um alerta de confirmação (*Dialog de confirmação* ou *Popconfirm* do navegador). A requisição HTTP `DELETE` só pode ser disparada se o usuário confirmar a intenção.
  - **AC 06 - Sincronização Sem Refresh:** Após o retorno de sucesso das operações de `PUT` ou `DELETE` vindas da API, a interface deve atualizar a lista local em memória de forma reativa instantaneamente. É proibido recarregar a aba ou a página forçadamente.
  - **AC 07 - Tratamento Visual de Exceções:** Caso a API rejeite uma exclusão devido à regra de integridade referencial, o front-end deve capturar o erro HTTP e exibir uma mensagem amigável na tela (ex: Toast notification ou banner de erro) contendo o texto explicativo enviado pelo servidor.
  - **AC 08 - Consumo de Componente Vinculado:** No formulário/modal de manipulação de *Produtos*, a escolha da categoria deve ser feita através de um componente do tipo *Select* alimentado assincronamente a partir dos dados do endpoint `GET /api/categorias`.

4. TASKS TÉCNICAS (Quebra de Atividades de Execução)

Para concluir o escopo das User Stories, o candidato precisará realizar as seguintes tarefas:

- [ ] **Task 01 (Back):** Inicializar o projeto Web API .NET Core, configurar as classes de modelo (`Produto`, `Categoria`) e o arquivo `appsettings.json` com a string de conexão local.
- [ ] **Task 02 (Back):** Criar a classe de contexto do banco de dados (`DbContext`), configurar o relacionamento `1:N` via Fluent API e executar o comando do EF Core CLI para gerar e aplicar as `Migrations` no banco de dados.
- [ ] **Task 03 (Back):** Desenvolver os Controllers e implementar os métodos CRUD injetando o contexto do banco, adicionando as regras de validação de caracteres do nome e o tratamento de erro na deleção de categorias associadas.
- [ ] **Task 04 (Front):** Inicializar a aplicação Nuxt 3, estruturar as rotas de páginas separadas para `/categorias` e `/produtos`, e configurar o layout base de navegação.
- [ ] **Task 05 (Front):** Desenvolver o formulário de cadastro e o componente de Grid/Tabela de listagem consumindo os dados da API de forma assíncrona.
- [ ] **Task 06 (Front):** Implementar a lógica de captura de cliques e passagem de parâmetros (`ID`) no Grid para gerenciar a abertura do modal de edição (`PUT`) e a janela de confirmação de exclusão (`DELETE`).
- [ ] **Task 07 (Doc):** Escrever o arquivo `README.md` detalhando os pré-requisitos, instruções de como restaurar os pacotes Nuxt/Nuget, como rodar as migrations e os payloads esperados para teste.