# Desafio Técnico Takeat: Full Stack Developer (Pleno)

## Bem-vindo à Takeat!

Somos um ecossistema completo para gestão de restaurantes. Nosso propósito é substituir múltiplos sistemas desconexos por uma única tecnologia fluida. Nosso princípio é claro: **Experiência do usuário (menos cliques, mais comida e rapidez)**.

## O Contexto do Desafio: "Gerenciamento de Pedidos com Estoque Atômico"

No mundo real dos restaurantes, o pior pesadelo não é um bug visual, mas vender um prato cujo ingrediente acabou. Isso gera frustração no cliente final e caos na cozinha. Sua missão é criar uma mini-versão do nosso sistema de pedidos que garanta a integridade do estoque baseado na **Ficha Técnica** dos produtos.

### Stack Obrigatória

- **Backend:** Node.js (Express), Sequelize ORM, Banco Relacional (Postgres).
- **Frontend:** React.

---

## Parte 1: Backend (A Lógica de Negócio)

Você deve construir uma API RESTful que gerencie pedidos e estoques complexos. Não queremos apenas baixar a quantidade de um "Hambúrguer", queremos baixar os ingredientes que compõem aquele hambúrguer.

### Requisitos de Modelagem (Sugestão):

1. **Products:** O item vendável (ex: "X-Burger").
2. **Inputs:** O insumo físico (ex: "Pão", "Carne", "Queijo").
3. **ProductInputs:** A tabela pivô que define a ficha técnica (ex: 1 X-Burger gasta 1 Pão e 1 Carne).
4. **Orders:** O registro da venda.

### Requisitos Funcionais:

- **Seed:** Crie um script para popular o banco com alguns produtos e ingredientes iniciais.

- **Endpoint de Pedido (POST /orders):** Deve receber uma lista de produtos e quantidade.
- **Validação de Estoque:** O sistema deve verificar se há ingredientes suficientes para **todos** os itens do pedido.
- **Atomicidade (Crucial):**
    - A operação deve ser **transacional**.
    - Se o pedido contém 3 itens e o terceiro não tem estoque de ingredientes suficiente, **nenhum** deve ser salvo e nenhum estoque deve ser decrementado. (Rollback total).

---

# Parte 2: Frontend (A Experiência do Usuário)

Crie uma interface em **React** focada em agilidade para o garçom.

### Requisitos Funcionais:

1. **Cardápio:** Listagem simples dos produtos.
2. **Carrinho:** Resumo do que será pedido.
3. **Feedback de Erro:** Se o backend retornar erro de estoque (ex: "Acabou a Carne"), a interface deve informar isso claramente ao usuário, permitindo que ele remova o item problemático e tente pedir o restante.

---

# Parte 3: O Diferencial (Desafio Extra)

**Cenário:** O restaurante tem uma conexão instável. O garçom lança o pedido, mas a internet cai exatamente no momento do clique.

**O Desafio:** Implemente uma estratégia de resiliência (Offline-first/Queue).

1. Se a requisição falhar por erro de rede, o app não pode travar ou perder os dados.
2. O pedido deve ser salvo em uma **fila local** (Client-side).
3. Quando a conexão voltar, o sistema deve tentar processar a fila automaticamente.
4. **Tratamento de Conflito Tardio:** Se, ao sincronizar, o estoque tiver acabado, como você avisa o garçom que aquele pedido antigo falhou?

# O que vamos avaliar?

Não estamos olhando apenas se "funciona", mas **como** foi construído.

1. **Arquitetura e Organização:** Separação de responsabilidades (Controllers, Services, Models).
2. **Uso do Sequelize:** Implementação correta de Relacionamentos (N:N) e, principalmente, o uso correto de Transactions.
3. **Código Limpo:** Nomenclatura, componentização no React e estrutura do CSS.
4. **UX/UI:** O sistema é intuitivo? O feedback visual é claro?
5. **Resiliência:** Como sua aplicação lida com falhas (banco fora do ar, internet caindo, estoque negativo).

# Entrega

1. Disponibilize o código em um repositório público (GitHub/GitLab).
2. Inclua um README.md com instruções claras de como rodar o projeto (Front e Back).
3. (Opcional) Se tiver implementado o Desafio Extra, explique brevemente sua estratégia no README.

**Boa sorte!**