Relatório do Projeto: Sistema de Gestão Inteligente de Estoque para PHX Consulting

1. Introdução

Este documento detalha o desenvolvimento do Sistema de Gestão Inteligente de Estoque, uma solução criada para a PHX Consulting. O objetivo principal do projeto é automatizar e otimizar o processo de compra de inventário de uma empresa de e-commerce, substituindo o uso de planilhas manuais por um sistema robusto que utiliza Inteligência Artificial para gerar recomendações precisas. A solução visa mitigar problemas comuns como ruptura de estoque e excesso de inventário (overstock).

Desenvolvido por **Christian Kringel**.

2. Arquitetura da Solução

O sistema é composto por uma arquitetura moderna e desacoplada, utilizando tecnologias eficientes para cada parte da aplicação:

- **Front-end:** Uma interface de usuário rica e interativa construída com **React** e **TypeScript**, apresentando os dados e insights de forma clara.
- **Back-end:** Orquestrado pela plataforma de automação de workflows **N8N**, que serve como a camada de API, conectando-se ao banco de dados e a serviços de IA.
- **Banco de Dados:** Um banco de dados relacional **PostgreSQL**, provisionado via Docker, para armazenar todos os dados da aplicação.
- **Gerador de Dados:** Um script em **Python** para popular o banco de dados com dados fictícios, permitindo testes realistas e demonstrações.
- **Inteligência Artificial:** O modelo **Gemini 2.5 Flash** é utilizado para tarefas de previsão de demanda e geração de justificativas para as recomendações.

3. Front-end

A interface do usuário foi projetada para ser intuitiva, permitindo que a equipe de compras visualize rapidamente os principais indicadores e tome decisões informadas.

Tecnologias Utilizadas

- **React 18** com **TypeScript**: Para uma base de código robusta, escalável e com tipagem estática.
- **Vite**: Como ferramenta de build e servidor de desenvolvimento, garantindo alta performance.

- **TailwindCSS**: Framework CSS utilitário para a construção de um design moderno e responsivo.
- **ShadCN/UI**: Biblioteca de componentes React para agilizar o desenvolvimento de UI.

Estrutura do Dashboard

O Dashboard principal é o coração do front-end, centralizando todas as informações críticas:

- **KPICard**: Exibe os principais indicadores de desempenho (KPIs), como valor total do estoque, produtos zerados, etc.
- **PurchaseRecommendationsTable**: Apresenta a tabela de recomendações de compra geradas pela IA.
- **OverstockAlerts**: Mostra alertas para produtos com excesso de estoque.
- **ABCAnalysis**: Exibe a análise de classificação ABC dos produtos.
- **SKUSalesTable**: Detalha as vendas por cada SKU.
- **StockCoverageTable**: Informa a cobertura de estoque em dias para os produtos.

Scripts de Execução

- npm run dev: Inicia o servidor de desenvolvimento.
- npm run build: Gera a build de produção.
- npm run lint: Executa a verificação de qualidade do código.
- npm run preview: Visualiza a build de produção localmente.

4. Back-end com N8N

O back-end foi implementado utilizando N8N, uma plataforma de automação que permitiu a criação de workflows poderosos para servir como endpoints da API.

Workflows Principais

- **dashboard-kpis**: Retorna as métricas chave para o dashboard principal (produtos ativos, valor do estoque, vendas recentes).
- **purchase-recommendations**: O workflow mais complexo.
  1. Realiza uma consulta no PostgreSQL para obter os dados históricos dos produtos.
  2. Envia esses dados para um **AI Agent** configurado com o modelo **Gemini 2.5 Flash**.
  3. A IA prevê a demanda diária, considerando fatores como sazonalidade e tendências de vendas.
  4. Retorna a quantidade a ser comprada e uma justificativa detalhada para a recomendação.
- **overstock-alerts**: Identifica e retorna produtos com excesso de estoque.
- **sku-sales-analysis**: Agrupa e retorna métricas mensais de vendas por produto.
- **stock-coverage**: Calcula e retorna a cobertura de estoque, risco de ruptura e outras métricas.
- **abc-classification**: Realiza a classificação ABC dos produtos com base no faturamento.

Modelo de IA Utilizado

- **Gemini 2.5 Flash**: Escolhido por seu excelente equilíbrio entre baixo custo e alta performance, sendo ideal para as tarefas de análise e previsão de demanda exigidas pelo projeto.

5. Ambiente de Desenvolvimento e Dados

Para garantir um ambiente de desenvolvimento isolado e replicável, o Docker foi utilizado.

Docker Compose

A infraestrutura do back-end é gerenciada pelo Docker Compose, que provisiona os seguintes serviços:

- **PostgreSQL 13**: Banco de dados principal, acessível na porta 5433.
- **N8N**: Plataforma de automação, acessível na porta 5679.

Para iniciar todo o ambiente, execute o comando:

```
cd Docker
docker-compose up -d
```

Gerador de Dados Fictícios

O script fake_data.py gera um volume de dados realista para testes, incluindo:

- 250 produtos em 8 categorias.
- 12.000 registros de vendas.
- 3.000 registros de compras.

O script gera um arquivo inserts_dados_ficticios.txt com os comandos SQL para popular o banco.

6. Dificuldades Encontradas no Desenvolvimento

O processo de desenvolvimento apresentou alguns desafios que foram superados e serviram como aprendizado.

1. **Configuração do Docker e Gestão de Volumes**: A configuração inicial do Docker apresentou obstáculos significativos. Problemas com permissões de usuário no sistema operacional levaram à perda do projeto em uma ocasião. Além disso, a gestão de volumes persistentes foi um desafio; uma configuração incorreta causou a perda dos dados do banco de dados ao parar os containers.
2. **Definição do Contrato de Dados (Front-end, Back-end e IA)**: Houve um período de iteração para definir exatamente quais dados o front-end precisava consumir. Isso impactou diretamente a estrutura das queries no N8N e, mais criticamente, o que seria

enviado para a IA. Foi necessário refinar os prompts e a seleção de dados para garantir que o modelo Gemini recebesse o contexto necessário para gerar previsões úteis e que seu retorno (JSON) fosse facilmente consumível pelo front-end.

3. **Integração com o Front-end**: O desenvolvimento da interface foi acelerado com o auxílio da ferramenta **Lovable**. Meu foco principal foi na criação e no teste dos webhooks no N8N e na subsequente vinculação desses endpoints com os componentes React, garantindo que o fluxo de dados fosse coeso e eficiente.

7. Implementações Futuras

Para evoluir o sistema, as seguintes melhorias são sugeridas:

- **Uso de IA para Enriquecimento de Dados**: O workflow stock-coverage atualmente retorna métricas calculadas. Uma melhoria futura seria aplicar um agente de IA sobre esses dados para gerar insights mais profundos, como identificar padrões de risco não óbvios ou sugerir ajustes dinâmicos nos parâmetros de cobertura de estoque.
- **Otimização do Uso da API de IA**: A implementação acima não foi realizada na versão inicial para respeitar os limites do tier gratuito do Gemini. Com o grande volume de dados de teste (mais de 400 produtos com dados históricos), as chamadas para a IA poderiam facilmente exceder a cota diária. No futuro, podem ser implementadas estratégias de cache ou processamento em lote para viabilizar essa funcionalidade de forma econômica.