Desafio Técnico - Desenvolvedor Python Júnior

Contexto: Você foi contratado para modernizar o sistema de atendimento de uma pizzaria delivery. O objetivo é criar um assistente virtual inteligente que não apenas converse, mas tenha acesso a dados reais do negócio (cardápio e preços) e siga um fluxo de atendimento estruturado, garantindo que o pedido seja montado corretamente.

Objetivo: Desenvolver uma aplicação containerizada em **um único repositório (Monorepo)**, composta por:

1. **Banco de Dados:** Para armazenar o cardápio.
2. **Backend (Python/FastAPI):** Para orquestrar o agente de IA e expor a API.
3. **Frontend (React):** Para interação via chat.

Estrutura de Pastas Esperada O projeto deve seguir uma organização onde o `docker-compose.yml` fique na raiz e os serviços em pastas separadas. Exemplo:

```
/my-challenge-repo
├── docker-compose.yml
├── README.md
├── /backend      # Código do FastAPI + LangGraph
├── /frontend     # Código do React
└── .env          # Variáveis de ambiente
```

Requisitos

1. Infraestrutura e Containerização (Docker)

- **Monorepo:** Todo o código deve estar em um único repositório git.
- **Execução:** A solução completa deve subir com um único comando na raiz do projeto: `docker-compose up`.
- **Orquestração:** O arquivo `docker-compose.yml` deve gerenciar três serviços simultâneos:
  - `db`: Banco de dados relacional.
  - `backend`: API e Agente.
  - `frontend`: Interface Web.
- **Configuração:** As variáveis de ambiente (DB credentials, API Keys) devem ser injetadas via arquivo `.env`.

2. Banco de Dados (Persistência)

- Utilizar um banco relacional (PostgreSQL ou MySQL).
- **Modelagem:** Criar uma tabela para armazenar o cardápio (ex: `pizzas`), contendo pelo menos: `nome`, `ingredientes` e `preço`.
- **Seeding:** O banco deve ser populado automaticamente com dados fictícios na inicialização (ex: Pizza de Calabresa, R$ 40,00), para que o agente tenha o que consultar imediatamente.
- **ORM:** A interação do Python com o banco deve ser feita via ORM (**SQLModel** ou **SQLAlchemy**).

3. Backend (FastAPI + LangGraph)

- **API:** Desenvolver endpoints assíncronos (`async/await`) usando FastAPI.
- **Orquestração de IA:** Utilizar **LangGraph** para definir o fluxo de conversa.
    - *Nota:* Não utilize apenas cadeias simples (*Chains*); é obrigatório o uso de um `StateGraph`.
- **Gestão de Estado:** O estado da conversa (mensagens, itens do pedido, total) deve ser gerenciado pelo grafo.
- **Ferramentas (Tools):** Criar pelo menos uma ferramenta (função decorada com `@tool`) que o LLM possa chamar.
    - *Requisito:* A ferramenta `get_pizza_price(pizza_name: str)` deve consultar o preço **no banco de dados** via ORM.
- **LLM:** Utilizar um modelo gratuito (Groq API (Recomendado), OpenAI, etc).

4. Frontend (React)

- Interface limpa de chat (janela de mensagens + campo de input).
- O histórico da conversa deve ser mantido visualmente durante o uso.
- Conexão com o backend via HTTP (REST).

Exemplo de Fluxo de Conversa (Casos de Uso)

O candidato deve se basear neste diálogo para desenhar seus *Nodes* e *Edges* no LangGraph.

> **Usuário:** "Oi, boa noite."
>
> **Assistente:** "Olá! Bem-vindo à Pizza Bot. Gostaria de ver o cardápio ou fazer um pedido?" *(Dica: Node inicial de saudação)*
>
> **Usuário:** "Quanto custa a pizza de Calabresa?"
>
> **Assistente:** (Processando...) *Chamada interna à tool `get_pizza_price` no banco de dados.*
>
> **Assistente:** "A pizza de Calabresa custa R$ 40,00 e leva molho de tomate, queijo e calabresa." *(Dica: Decisão do LLM de usar Tool -> Node de Tool -> Retorno ao LLM)*
>
> **Usuário:** "Vou querer uma."
>
> **Assistente:** "Perfeito. Adicionei 1x Calabresa ao seu carrinho. O total é R$ 40,00. Algo mais?" *(Dica: Atualização do State do grafo com o item do pedido)*
>
> **Usuário:** "Não, pode fechar."
>
> **Assistente:** "Pedido confirmado! 1x Calabresa. Total: R$ 40,00. Obrigado!" *(Dica: Node final ou reset de estado)*

Critérios de Avaliação

1. **Arquitetura:** Organização do código e separação de responsabilidades (Backend vs. Frontend vs. Database).
2. **Docker:** O `docker-compose` funciona de primeira? Os serviços se comunicam corretamente na rede interna do Docker?
3. **Domínio do LangGraph:** Uso correto de *Nodes*, *Edges* e *Conditional Edges*.
4. **Qualidade do Código:** Tipagem (*Type Hints*), tratamento de erros e documentação (README.md com instruções de como rodar).
5. **Funcionalidade:** O bot consegue consultar o banco e informar o preço correto?

Dicas Úteis

- Lista de provedores de LLM para facilitar integração com Langchain: LangChain Providers
- Certifique-se de que o backend aguarde o banco de dados estar pronto antes de tentar conectar (Healthchecks no Docker são bem-vindos).