Desafio/Banco Digital Simplificado

Desafio: Serviço de Backend para um Banco Digital Simplificado

Você foi contratado para criar um serviço de backend que gerencie contas e transações de um banco digital simplificado. A aplicação deve ser capaz de armazenar e consultar informações sobre contas e transações financeiras, além de processar operações básicas. O objetivo é avaliar sua capacidade de implementar uma solução funcional, bem estruturada e escalável.

Requisitos:

Funcionalidades:

1. Gerenciamento de Contas:

- Criar uma conta bancária com os seguintes campos:
  - Nome do cliente.
  - CPF do cliente (validar formato e unicidade).
  - Saldo inicial (opcional, mas deve ser maior ou igual a 0).

2. Movimentações Financeiras:

- **Depósito**:
  - Endpoint para processar depósitos em uma conta existente.
  - Validar se a conta existe antes de realizar o depósito.
  - O valor do depósito deve ser maior que 0.
- **Saque**:
  - Endpoint para realizar saques de uma conta existente.
  - Validar se a conta existe antes de realizar o saque.
  - Validar se há saldo suficiente (saldo nunca pode ser negativo).
  - O valor do saque deve ser maior que 0.

3. Histórico de Transações:

- Para cada depósito ou saque, registrar no banco de dados:
  - Tipo da transação: **depósito** ou **saque**.
  - Data/hora da transação.
  - Valor da transação.

4. Consultas:

- **Consultar saldo**:
  - Endpoint que retorne o saldo atual de uma conta.
- **Listar transações por conta**:
  - Endpoint que liste o histórico de transações de uma conta específica com paginação.
  - A resposta deve incluir data, tipo e valor de cada transação.
- **Relatório simples**:
  - Endpoint que retorne o total de depósitos e saques realizados em um período (ex.: último dia).

Requisitos Técnicos:

1. **Tecnologias:**
   - Use C# com ASP.NET (Core ou Framework) para o backend.
   - Banco de dados relacional (SQLServer, SQLite ou PostgreSQL) ou InMemory.
   - Code-first (opcional)
   - ORM (opcional)
2. **Boas Práticas de Desenvolvimento:**
   - Utilize princípios de SOLID e Clean Code.
   - Separe as responsabilidades em camadas.
   - Aplique validações e tratamento de erros adequados.
3. **Testes:**

- Implemente testes unitários para a lógica de negócios.
- Se possível, adicione testes de integração para os endpoints e health checks dos serviços.

4. **Documentação:**
   - Use Swagger/OpenAPI para documentar os endpoints.
5. **Conteinerização:**
   - Forneça um `Dockerfile` para empacotar a aplicação.
   - Inclua um arquivo `docker-compose.yml` para subir o banco de dados, serviços e aplicação juntos.
6. **Instruções de Execução:**
   - Forneça um README no repositório contendo:
     - Passos para executar o projeto localmente e via Docker.
     - Exemplos de requisições para cada endpoint.

Critérios de Avaliação:

1. **Estrutura e organização do código**: O código deve ser modular e de fácil compreensão.
2. **Aderência às boas práticas de programação**: Uso correto de padrões arquiteturais e princípios de design.
3. **Funcionalidade**: A solução deve cumprir todos os requisitos funcionais.
4. **Validações e tratamento de erros**: Verificar se há validações apropriadas e respostas HTTP significativas (400, 404, etc.).
5. **Testabilidade**: Presença e qualidade dos testes.
6. **Conteinerização**: Capacidade de rodar o projeto em um ambiente Docker.
7. **Documentação**: Clareza e organização da documentação no README e Swagger.

Entrega:

1. Envio de email para vagas-core@pinpag.com.br com o assunto “Entrega desafio - [Nome]” com o link do repositório do GitHub (ou similar).