Desafio de Desenvolvimento - Sistema de Controle de Pagamentos

Contexto

Uma empresa deseja um sistema para gerenciar pagamentos de seus clientes. Cada pagamento está associado a um cliente e deve ser controlado com base em seu status (Pendente, Pago, Cancelado).

Objetivo

Criar uma aplicação simples que permita cadastrar clientes, registrar pagamentos, listar os pagamentos de um cliente e atualizar o status de um pagamento.

Requisitos

Back-End (ASP.NET Core Web API com DDD)

1. **Modelagem do Domínio:**
    a. **Entidades principais:**
        i. **Cliente:** `Id`, `Nome`, `Email`.
        ii. **Pagamento:** `Id`, `ClienteId`, `Valor`, `Data`, `Status` (Pendente, Pago, Cancelado).
    b. **Agregado:**
        i. Pagamento é o agregado raiz, garantindo a consistência do status.
2. **Regras de Negócio:**
    a. Um pagamento pode ser criado apenas se o cliente existir.
    b. O status de um pagamento só pode ser alterado nas seguintes condições:
        i. De "Pendente" para "Pago".
        ii. De "Pendente" para "Cancelado".
    c. Não é possível alterar um pagamento que já está "Pago" ou "Cancelado".
3. **Endpoints necessários:**
    a. **Clientes:**
        i. **GET /api/clientes:** Listar todos os clientes.
        ii. **POST /api/clientes:** Criar um cliente.

b. **Pagamentos:**
    i. **GET /api/pagamentos?clienteId={clienteId}:** Listar pagamentos de um cliente.
    ii. **POST /api/pagamentos:** Criar um novo pagamento.
    iii. **PUT /api/pagamentos/{id}/status:** Atualizar o status de um pagamento.

4. **Persistência:**
    a. Usar um banco relacional.

5. **Estrutura baseada em DDD:**
    a. Camadas principais:
        i. **Domain:** Contém entidades e regras de negócio.
        ii. **Application:** Implementa casos de uso, como "CriarPagamento" e "AtualizarStatusPagamento".
        iii. **Infrastructure:** Para persistência (usando EF Core ou outra abordagem).

Front-End (React)

1. Criar uma interface para:
    a. Listar clientes e cadastrar novos.
    b. Listar pagamentos de um cliente.
    c. Criar um novo pagamento para um cliente.
    d. Atualizar o status de um pagamento (botões para marcar como "Pago" ou "Cancelar").

2. Usar Axios ou Fetch para consumir a API.

3. Implementar validações no front-end:
    a. Não permitir criar pagamentos sem cliente ou valor.
    b. Bloquear alterações de status para pagamentos já "Pagos" ou "Cancelados".

4. Estilos básicos com CSS.

Extras (Opcional, não obrigatório)

- Adicionar paginação ou filtros na listagem de pagamentos.
- Exibir um resumo no front-end (ex.: total de pagamentos pendentes e pagos).
- Testes unitários básicos no back-end para as regras de negócio (ex.: mudança de status).

Critérios de Avaliação

1. **Modelagem do Domínio:** Estrutura das entidades e agregados.
2. **Organização do Código:** Separação de responsabilidades e uso de camadas.
3. **Funcionalidade:** Implementação correta dos fluxos e regras.

Entrega do Projeto no Git

1. **Criar um repositório público ou privado (compartilhado):**
    a. O candidato deve criar um repositório no **GitHub**
    b. O repositório deve conter as pastas e arquivos do projeto, organizados de forma clara.