Registration App

Uma breve explicação e exemplo sobre o micro-serviço

**Visão Geral:**

O projeto "Registration App" é uma aplicação desenvolvida em C# utilizando uma arquitetura de microservices. A finalidade principal é gerenciar informações relacionadas ao cadastro de dados pessoais e integração com serviços externos.

Tecnologia Sugerida: **C#**

Prazo de Entrega: 26/01/2023 às 16:00

**Critérios de Avaliação:**

**Cenário 1: Receber Dados Cadastrais e Salvar Internamente**

Dado que um usuário fornece seus dados cadastrais através do endpoint de cadastro

Quando o sistema recebe esses dados

Então os dados devem ser salvos internamente para referências futuras

**Cenário 2: Integração com Serviço Externo para Cadastro no Sistema Parceiro**

Dado que os dados cadastrais de um usuário foram salvos internamente,

Quando o sistema se integra com o serviço externo para o cadastro no sistema parceiro,

Então os dados salvos internamente devem ser enviados para o sistema parceiro,

E o sistema deve receber uma confirmação de que o cadastro foi realizado com sucesso.

**Cenário 3: Integração com Outro Serviço Externo para Cadastro no Sistema Parceiro**

Dado que os dados cadastrais foram salvos internamente após a integração com o primeiro serviço externo,

Quando o sistema se integra com outro serviço externo para o mesmo objetivo,

Então os dados previamente salvos devem ser enviados para o segundo sistema parceiro,

E o sistema deve receber uma confirmação de que o cadastro foi realizado com sucesso no segundo sistema parceiro.

**Cenário 4: Tratamento de Falhas na Integração**

Dado que o sistema está integrado com os serviços externos,

Quando ocorrer uma falha durante a integração com qualquer um dos sistemas parceiros,

Então o sistema deve registrar a falha para fins de monitoramento e análise,

E deve notificar os administradores sobre a falha para tomada de ação corretiva.

**Cenário 5: Verificação de Dados no Sistema Parceiro**

Dado que o cadastro foi realizado com sucesso nos sistemas parceiros,

Quando necessário

Então o sistema deve permitir a verificação dos dados no sistema parceiro para garantir consistência entre os sistemas.

* **Microservices:**

O sistema é composto por dois microservices principais:

1. **Microsserviço de Clientes:**
   * Responsável pelo gerenciamento de informações relacionadas aos clientes, incluindo cadastro, atualização e remoção.
   * Fornece serviços HTTP através de controllers.
2. **Microsserviço Sistema Parceiro:**
   * Integra os cadastros de clientes
   * Fornece serviços HTTP através de controllers.

**Design Patterns Utilizados:** Repository Pattern com Unit of Work.

**Linguagem:** c# (.Net 6)

* **Domain-Driven Design (DDD):**
  + Para estruturar o código ao redor do domínio do problema, promovendo um entendimento claro dos conceitos de negócios.
  + Entidades, agregados, serviços de domínio e objetos de valor são identificados e modelados de acordo com o contexto específico de cada microsserviço.
* **Clean Architecture:**
  + Para organizar o código de forma que as camadas de dependência diminuam à medida que nos aproximamos do núcleo da aplicação.
  + As camadas externas (como a interface do usuário e os detalhes de infraestrutura) dependem apenas das camadas internas, mantendo a flexibilidade e facilitando a manutenção.
* **Unit of Work:**
  + Padrão Unit of Work para gerenciar transações e operações de banco de dados de forma coesa em cada microsserviço.
  + Isso permite que as operações de banco de dados sejam agrupadas em uma única transação, garantindo consistência nos dados.
* **Dependency Injection:**
  + Para facilitar a manutenção e teste de cada componente.
  + Os serviços e dependências são injetados por meio de um container de DI, promovendo a inversão de controle e a desacoplação de componentes.