Desafio 1:

User Story: Pré-cadastro de clientes

Como área de Comercialização da Cielo, desejo manter um pré-cadastro de clientes (prospect) para possibilitar uma futura oferta de produtos e serviços a esses clientes.

Regras:

1) Informações do cadastro:

a) Se Pessoa Jurídica:

- CNPJ
  - número de 14 dígitos formatado com zeros à esquerda
- Razão Social
  - máximo de 50 caracteres
- MCC - "*Merchant Category Code*"
  - número com no máximo 4 caracteres
- CPF do contato do estabelecimento
  - número de 11 dígitos formatado com zeros à esquerda
- Nome do contato do estabelecimento
  - máximo de 50 caracteres
- Email do contato do estabelecimento
  - expressão regular para validação:
    "^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\.]+)\\.([a-zA-Z]{2,5})$")

b) Se Pessoa Física:

- CPF da pessoa
  - número de 11 dígitos formatado com zeros à esquerda
- MCC – *"Merchant Category Code"*
  - número com no máximo 4 caracteres
- Nome da pessoa
  - máximo de 50 caracteres
- Email da pessoa
  - expressão regular para validação:
    "^([a-zA-Z0-9_\\-\\.]+)@([a-zA-Z0-9_\\-\.]+)\\.([a-zA-Z]{2,5})$"

* Todas as informações são obrigatórias

2) Consistências:

a) A operação de **cadastrar** cliente deverá validar se o cadastro não existe.

Se o cadastro **já** existir, o sistema deverá retornar um status coerente informando que o cliente já está cadastrado e não realizar qualquer alteração nos dados existentes.

b) A operação de **alterar** cliente deverá validar se o cadastro já existe.

Se o cadastro **não** existir, o sistema deverá retornar um status coerente informando que o cliente ainda não está cadastrado e não deverá realizar a inclusão de um novo registro.

c) Ambas as operações de **cadastrar** ou **alterar** cliente deverão validar se todos os dados foram informados, se estão consistentes conforme tamanhos, tipos de dados e formatações disponibilizadas na regra “1”.

Em caso de qualquer inconsistência, o sistema deverá retornar um status coerente informando os detalhes do erro.

d) A operação de **consultar** um cliente deverá validar se o cadastro já existe.

Se o cadastro **não** existir, o sistema deverá retornar um status coerente informando que o cliente ainda não está cadastrado.

Desafio:

a) modelar uma API REST com operações que possibilitem a **criação**, **alteração**, **exclusão** e **consulta** de pré-cadastros de clientes. O entregável deverá ser um documento ***swagger***.

b) implementar na linguagem java utilizando o framework spring boot as APIs modeladas no item 1. Os dados podem ser armazenados em memória.

c) Implementar cobertura de 70% de testes unitários

Desafio 2:

User Story: Fila de atendimento

Como área de Comercialização da Cielo, desejo ter uma fila de atendimento aos prospect, para que cada cliente possa ser analisado de forma sequencial pelos gestores comerciais.

Regras:

1) Toda vez que um **novo cadastro** ou uma **alteração de cadastro** for realizada no sistema, o cliente deverá entrar na **última** posição da fila de atendimento.

2) Possibilitar a **retirada** do cliente na **primeira** posição da fila de atendimento, apresentando seus dados para o tratamento.

3) Caso o gestor comercial solicite um *prospect* da fila para atendimento e não houver nenhum cliente na fila, deverá retornar um status coerente informando que a fila de atendimento está vazia.

Desafio:

a) incluir na API criada no desafio "1" uma nova operação que possibilite a **retirada do próximo cliente** da fila de atendimento e retorne os dados disponíveis

b) implementar na linguagem java uma estrutura de dados para uma **fila** utilizando apenas tipos de **dados primitivos** (sem utilizar classes java.util.*), onde seja possível acrescentar e retirar clientes na fila no modelo **FIFO** (*First In, First Out*).

c) contemplar as **regras da história de usuário** através da implementação da operação modelada no item "a", utilizando a estrutura de fila criada no item "b"

d) Implementar cobertura de 70% de testes unitários

Desafio 3:

Technical Debt: Escalabilidade da Fila de atendimento

Ao realizar um teste de carga na aplicação, o time de performance da Cielo identificou um problema de escalabilidade na fila de atendimento criada no desafio "2":

Durante o teste de carga foi possível identificar que a fila em memória suportava um limite insuficiente para os requisitos de negócio. Ao atingir o limite de memória, a aplicação abortou.

Na tentativa de fazer com que a aplicação suportasse um volume maior de requisições, o time de performance solicitou ao time de *devops* a configuração de provisionamento automático de novas instâncias da aplicação conforme atingisse um certo consumo de memória. A configuração acabou por afetar os requisitos de negócio, porque como a fila foi implementada internamente, novas instâncias resultaram em filas apartadas, afetando os requisitos de negócio.

Além disso, a área de negócio notou que ao reiniciar a aplicação, todos os clientes da fila foram perdidos. Isso ocorria porque a aplicação armazenava os clientes em memória, e os prospects da fila eram perdidos quando a aplicação era finalizada.

Para resolver este débito técnico o time de arquitetura orientou a utilização de uma solução de fila mais robusta e escalável, recomendando também a utilização da solução de mensageria SQS da AWS.

Desafio:

Desenhe e implemente uma nova solução para a fila de atendimento, utilizando a solução de mensageria SQS da AWS.

Desafio 4:

Technical Debt: Segurança da Informação

Desafio:

a) identifique um débito técnico de Segurança da Informação na aplicação

b) detalhe o débito técnico identificado, informando a criticidade e possíveis consequências

c) planeje as atividades técnicas para o desenvolvimento da solução

d) implemente a solução