**DESAFIO TÉCNICO – DESENVOLVIMENTO DE API DE CONSULTA DE CRÉDITOS**

**OBJETIVO**

O candidato deverá desenvolver uma API RESTful utilizando Spring Boot para a consulta de créditos constituídos. A API fornecerá informações essenciais como número do crédito constituído, número da NFS-e, data da constituição do crédito, valor do ISSQN, tipo do crédito e outros atributos. Além disso, será necessário desenvolver um front-end em Angular para consumir essa API e exibir os dados ao usuário.

**REQUISITOS TÉCNICOS**

·   Back-end: Java 8+, Spring Boot, Spring Data JPA, Hibernate

·   Banco de Dados: PostgreSQL ou MariaDB

·   Front-end: Angular 2+

·   Containerização: Docker

·   Mensageria: Kafka, Azure Service Bus

·   Testes Automatizados: JUnit, Mockito

·   Padrões de Projeto: Uso adequado de padrões como MVC, Repository, Factory, Singleton e outros conforme aplicável

**DESAFIO A SER DESENVOLVIDO**

ESTRUTURA DA API

GET /api/creditos/{numeroNfse}

Descrição: Retorna uma lista de créditos constituídos com base no número da NFS-e.

Parâmetro:

·   numeroNfse (String) - Número identificador da NFS-e

Resposta esperada:

[

  {

"numeroCredito": "123456",

"numeroNfse": "7891011",

"dataConstituicao": "2024-02-25",

"valorIssqn": 1500.75,

"tipoCredito": "ISSQN",

"simplesNacional": "Sim",

"aliquota": 5.0,

"valorFaturado": 30000.00,

"valorDeducao": 5000.00,

"baseCalculo": 25000.00

  }

]

GET /api/creditos/credito/{numeroCredito}

Descrição: Retorna os detalhes de um crédito constituído específico com base no número do crédito constituído.

Parâmetro:

* numeroCredito (String) - Número identificador do crédito constituído

Resposta esperada:

{

  "numeroCredito": "123456",

  "numeroNfse": "7891011",

  "dataConstituicao": "2024-02-25",

  "valorIssqn": 1500.75,

  "tipoCredito": "ISSQN",

  "simplesNacional": "Sim",

  "aliquota": 5.0,

  "valorFaturado": 30000.00,

  "valorDeducao": 5000.00,

  "baseCalculo": 25000.00

}

**MODELAGEM DE DADOS**

Entidade Credito

@Entity

public class Credito {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String numeroCredito;

private String numeroNfse;

private LocalDate dataConstituicao;

private BigDecimal valorIssqn;

private String tipoCredito;

private boolean simplesNacional;

private BigDecimal aliquota;

private BigDecimal valorFaturado;

private BigDecimal valorDeducao;

private BigDecimal baseCalculo;

}

Script de Criação da Tabela

CREATE TABLE credito

(

id            BIGINT GENERATED BY DEFAULT AS IDENTITY,

numero\_credito VARCHAR(50) NOT NULL,

numero\_nfse   VARCHAR(50) NOT NULL,

data\_constituicao DATE       NOT NULL,

valor\_issqn   DECIMAL(15, 2) NOT NULL,

tipo\_credito  VARCHAR(50) NOT NULL,

simples\_nacional  BOOLEAN    NOT NULL,

aliquota      DECIMAL(5, 2)  NOT NULL,

valor\_faturado DECIMAL(15, 2) NOT NULL,

valor\_deducao DECIMAL(15, 2) NOT NULL,

base\_calculo  DECIMAL(15, 2) NOT NULL

);

Script de População da Tabela

INSERT INTO credito (numero\_credito, numero\_nfse, data\_constituicao, valor\_issqn, tipo\_credito, simples\_nacional, aliquota, valor\_faturado, valor\_deducao, base\_calculo)

VALUES

('123456', '7891011', '2024-02-25', 1500.75, 'ISSQN', true, 5.0, 30000.00, 5000.00, 25000.00),

('789012', '7891011', '2024-02-26', 1200.50, 'ISSQN', false, 4.5, 25000.00, 4000.00, 21000.00),

('654321', '1122334', '2024-01-15', 800.50, 'Outros', true, 3.5, 20000.00, 3000.00, 17000.00);

**DESENVOLVIMENTO DO FRONT-END**

O candidato deverá desenvolver um front-end em Angular para consumir a API e apresentar os dados ao usuário.

Funcionalidades Esperadas

·   Tela de consulta permitindo apenas a busca por número da NFS-e ou número do crédito.

·   Tabela exibindo os resultados da consulta.

·   Responsividade para dispositivos móveis.

**IMPLANTAÇAO E INFRAESTRUTURA**

·   A API e o front-end devem ser containerizada utilizando Docker

·   O banco de dados PostgreSQL ou MariaDB deve ser provisionado localmente.

**CRITÉRIOS DE AVALIAÇÃO**

Os seguintes critérios serão considerados na avaliação do desafio:

·   Código Limpo: Boas práticas de desenvolvimento e organização do código.

·   Qualidade do Código: Uso de padrões como SOLID, DRY e KISS.

·   Funcionamento da API: Implementação correta dos endpoints e retorno adequado dos dados.

·   Testes Automatizados: Cobertura de testes unitários e de integração.

·   Uso de Git: Histórico de commits organizados.

·   Documentação: README explicativo com instruções para rodar o projeto.

**ENTREGA DO DESAFIO**

O candidato deverá entregar o código em um repositório público do GitHub, contendo as instruções de instalação e execução no arquivo README.md.

Desafios adicionais:

Mensageria

·   Como um desafio extra, adicionar um publisher Kafka ou Azure Service Bus para notificar um tópico/fila sempre que uma consulta for realizada. Isso simula um caso de uso real onde logs ou eventos podem ser armazenados para auditoria.

Testes Automatizados

·   Cobrir a API com testes unitários usando JUnit e Mockito.