Escopo do Projeto

Objetivo:

Desenvolver um sistema eficiente para o processamento de pagamentos de operadoras de cartão de crédito. Nosso objetivo principal é receber os dados das transações com cartão de crédito e validar se o cartão do cliente possui limite disponível para a realização da compra. Este sistema garantirá a verificação precisa e em tempo real do limite de crédito dos clientes, proporcionando uma experiência de pagamento segura e confiável.

Funcionalidades Principais

1. Registro do Cliente

Vamos criar uma API backend para fazer o registro dos cliente com os dados básico.

- Rota: /api/cliente
- Método: POST
- Contrato:

```
"{
    "cpf":"1111111111",
    "nome":"João da Silva",
    "email":"joao@example.com",
    "telefone":"+55 11 91234-5678",
     "rua":"Rua A",
     "cidade":"Cidade",
      "estado":"Estado",
     "cep":"12345-678",
     "pais":"Brasil"

  }
```

- Retorno:
    - 200 para sucesso {"id_cliente":"XXXXXX"}
    - 401 para erro de autorização
    - 500 para um erro de negócio
- Autenticação: Requer autenticação JWT.

2. Gerar do Cartão

Vamos criar uma API backend para gerar o cartão para um cliente. Cada cliente não pode ter mais que 2 cartões.
Ponto 1: Ficar atento ao número do cartão de crédito permitido por cliente.

- Rota: /api/cartao

- Método: POST
- Contrato:

```
"{
    "cpf":"1111111111",
    "limite":1000,
     "numero":"**** **** **** 1234",
      "data_validade":"12/24",
      "cvv":"123”
}
```

- Retorno:
    - 200 para sucesso
    - 401 para erro de autorização
    - 403 para erro número máximo de cartões atingido
    - 500 para um erro de negócio
- Autenticação: Requer autenticação JWT.

3. Registro do Pagamento

Vamos criar uma API backend para receber as solicitações de autorização de pagamento de cartão de crédito.
Ponto 1: Ficar atento ao limite do cartão de crédito.
Ponto 2: Ficar atento ao validar se o cartão existe para aquele cliente.
Ponto 3: validade do cartão
Ponto 4 validar o código de verificação do cartão

- Rota: /api/pagamentos
- Método: POST
- Contrato:

```
{
        “cpf”:”1111111111”,
       "numero":"**** **** **** 1234",
       "data_validade":"12/24",
       "cvv":"123",
      "valor":100.00
}
```

- Retorno:
    - 200 para sucesso {"chave_pagamento":"XXXXXX"}
    - 401 para erro de autorização
    - 402 para caso o cartão do cliente estourou
    - 500 para um erro de negócio
- Autenticação: Requer autenticação JWT.

4. Consulta de Pagamentos por Cliente

O objetivo e listar os pagamentos efetuados por um único cliente e mostrar a situação desse pagamento.

- Rota: /api/pagamentos/cliente/{Chave}
- Método: GET
- Retorno:
    - 200 para sucesso

```
[{
    "valor":100.00,
    "descricao":"Compra de produto X",
    "metodo_pagamento":"cartao_credito",
    "status":"aprovado"
}
]
```

    - 401 para erro de autorização
    - 500 para um erro de negócio
- Autenticação: Requer autenticação JWT.

5. Autenticação

O objetivo autenticar o acesso de uma api passando o usuário e a senha e a api vai retornar um token de acesso.

- Rota: /api/autenticacao
- Método: POST
- Contrato:

```
{
  "usuario":xxxxx,
  "senha":"XXXX"
}
```

-
- Retorno:
    - 200 para sucesso

```
{
    "token":"XXXXXXXXX"
}
```

    - 401 para erro de autorização
    - 500 para um erro de negócio

Fluxo de Trabalho

1. A api deve solicitar um token de acesso usando o usuário e senha.

2. Devemos fazer o cadastro de cliente.

3. Gerar o cartão de crédito para o cliente informando os dados e o limite para o cartão de crédito.

4. Solicitar a autorização de um pagamento de um cartão.

Segurança:

Implementar autenticação JWT para todos os endpoints.
Criar um único endpoint para criação do bearer token.
Validar as permissões do usuário para acessar os endpoints.
Criar um endpoint que tenha uma expiração de 2 min e os demais podem ser uma expiração maior.
O usuário e senha que deve ser usado é:
usuário: adj2
senha: adj@1234

Armazenamento de Dados:

Utilizar um banco de dados SQL ou NoSQL para armazenar os dados de pagamento.
Estruturar uma tabela (ou coleção, se NoSQL) para armazenar os detalhes dos pagamentos.

Documentação:

Documentar os endpoints da API usando o Swagger ou alguma outra ferramenta de documentação.

Tecnologias Requeridas:

- **Docker:** A aplicação deve subir totalmente via Docker sem a necessidade de intervenção dos professores.
- **API: D**evem seguir o padrão fornecido para que seja possível executar os testes de jmeter.
- **Banco de Dados:** SQL (por exemplo, MySQL, PostgreSQL) ou NoSQL (por exemplo, MongoDB)
- **Autenticação:** JWT (JSON Web Token)
- **Documentação da API:** Swagger/OpenAPI
- **Documentação**: Desenho de arquitetura
- **Cobertura Testes:** Garantir que a aplicação está coberta com 80% de cobertura de testes.
- **Arquitetura Limpa:** Garantir que a aplicação está usando os padrões de arquitetura limpa.

Pontos de atenção

Os endpoints e os contratos devem ser seguidos de acordo com a documentação pois a validação do mesmo será feita através de jmeter.

As aplicações devem ter no máximo 512 de memória para termos um teste de performance com resultados iguais para todos os grupos.

Critério de Avaliação.

1. **Docker (Sua aplicação deve subir totalmente com um docker-composer incluindo banco de dados, micro serviço, eureka, gateway. Qualquer coisa que for necessário para o seu ecossistema)**
2. **Cobertura Testes: 80%**
3. **Api Funcionais**
4. **Arquitetura Limpa**
5. **Documentação (Swagger e Diagramas da Arquitetura)**
6. **Autenticação**

Critério de Desempate.

1. **Maior Cobertura de Testes.**
2. **Maior quantidade de casos de testes**