Solução de Pagamentos para Checkout

Você foi contratado para desenvolver a **API Pay**, uma solução de pagamentos para o checkout de um e-commerce. O objetivo principal é que a API Pay sirva como um gateway intermediário, integrando-se a provedores de pagamento externos (como Stripe ou Braintree).

O prazo esperado para a conclusão do teste é de 7 dias, qualquer dúvida sobre o desafio ou problemas no meio do processo, fique à vontade para entrar em contato conosco por esse mesmo email.

A implementação deve atender os seguintes requisitos:

Requisitos Funcionais

1. **Processamento de Pagamentos:**
    - A API deve receber as informações do pedido e processar o pagamento com um dos provedores externos.
    - Caso um dos provedores externos esteja indisponível, a API deve utilizar um provedor de backup de forma automática e transparente.
2. **Redundância de Provedores:**
    - A API deve ser resiliente e capaz de alternar entre provedores automaticamente em caso de falha ou indisponibilidade.
3. **Métodos Opcionais (Bônus):**
    - **Estorno de Pagamentos:** Implementar um endpoint para realizar estornos utilizando os provedores externos.
    - **Consulta de Transações:** Implementar um endpoint para leitura de dados de transações processadas, retornando informações detalhadas como status, valor e data.

Requisitos Técnicos

1. **Endpoints Mínimos:**
    - **POST /payments**: Processar um pagamento.
    - **POST /refunds** *(bônus)*: Realizar estorno de um pagamento previamente processado.
    - **GET /payments/{id}** *(bônus)*: Obter informações de uma transação.
    - Você está livre para pensar nos contratos de entrada e saída dos endpoints acima
2. **Provedores de Pagamento Simulados:**
    - Simular dois provedores externos (e.g., Stripe e Braintree) utilizando APIs mock (você pode criar mocks locais ou usar ferramentas externas como Mocky, MirageJS, JSONPlaceholder, etc). No final deste email estarão os contratos para os endpoints mockados.

3. **Mensagens de Erro e Logs:**
    - A API deve retornar mensagens claras para falhas e erros.
    - Logs detalhados devem ser registrados para depuração de problemas.
4. **Resiliência e Tolerância a Falhas:**
    - Implementar um mecanismo para detectar falhas em um provedor e alternar para o próximo.

Critérios de Avaliação

1. **Qualidade do Código:**
    - Organização, legibilidade e boas práticas de desenvolvimento.
2. **Arquitetura:**
    - Estrutura do projeto e escolhas arquiteturais.
3. **Resiliência:**
    - Capacidade de alternar provedores de forma transparente.
4. **Endpoints Opcionais:**
    - Implementação de métodos de estorno e consulta (bônus).
5. **Testes Automatizados:**
    - Cobertura de testes unitários e/ou de integração.

Instruções

- Utilize qualquer linguagem de programação e framework de sua preferência.
- Crie uma documentação breve explicando como rodar a aplicação, configurar provedores e testar os endpoints, pode ser no Readme.
- Submeta o projeto em um repositório público no GitHub e envie o link do repositório por email.

Mocks dos provedores externos

Utilize desses mocks para que você possa integrar com os dois provedores diferentes, esse são os contratos que você deve criar nos seus mocks:

Provedor 1

Criação de pagamento:
Endpoint:
POST /charges

Request:
{
  “amount”: number,

```
  “currency”: string formato ISO 4217,
  “description”: string,
  “paymentMethod”: {
     “type”: “card”,
     “card”: {
       “number”: string,
       “holderName” string,
       “cvv”: string,
       “expirationDate”: “DD/YYYY”,
       “installments”: number
     }
}
```

Response:

```
{
  “id”: uuid,
  “createdAt”: “yyyy-mm-dd”
  “status”: “authorized” | “failed” | “refunded”,
  “originalAmount”: number,
  “currentAmount”: number,
  “currency”:  string formato ISO 4217,
  “description”: string,
  “paymentMethod”: “card”,
  “cardId”: uuid
}
```

Criação de estorno:
Endpoint:
POST /refund/{id}

Request:

```
{
  “amount”: number,
}
```

Response:

```
{
  “id”: uuid,
  “createdAt”: “yyyy-mm-dd”
  “status”: “authorized” | “failed” | “refunded”,
  “originalAmount”: number,
  “currentAmount”: number,
  “currency”:  string formato ISO 4217,
  “description”: string,
  “paymentMethod”: “card”,
  “cardId”: uuid
```

}

Buscar pagamento:
Endpoint:
GET /charges/{id}

Response:
{
“id”: uuid,
“createdAt”: “yyyy-mm-dd”
“status”: “authorized” | “failed” | “refunded”,
“originalAmount”: number,
“currentAmount”: number,
“currency”: string formato ISO 4217,
“description”: string,
“paymentMethod”: “card”,
“cardId”: uuid
}

—

Provedor 2

Criação de pagamento:
Endpoint:
POST /transactions

Request:
{
“amount”: number,
“currency”: string formato ISO 4217,
“statementDescriptor”: string,
“paymentType”: “card”,
“card”: {
“number”: string,
“holder” string,
“cvv”: string,
“expiration”: “DD/YY”,
“installmentNumber”: number
}
}

Response:
{
“id”: uuid,
“date”: “yyyy-mm-dd”
“status”: “paid” | “failed” | “voided”,
“amount”: number,

“originalAmount”: number,
“currency”: string formato ISO 4217,
“statementDescriptor”: string,
“paymentType”: “card”,
“cardId”: uuid
}

Criação de estorno:
Endpoint:
POST /void/{id}

Request:
{
“amount”: number,
}

Response:
{
“id”: uuid,
“date”: “yyyy-mm-dd”
“status”: “paid” | “failed” | “voided”,
“amount”: number,
“originalAmount”: number,
“currency”: string formato ISO 4217,
“statementDescriptor”: string,
“paymentType”: “card”,
“cardId”: uuid
}

Buscar pagamento:
Endpoint:
GET /transactions/{id}

Response:
{
“id”: uuid,
“date”: “yyyy-mm-dd”,
“status”: “paid” | “failed” | “voided”,
“amount”: number,
“originalAmount”: number,
“currency”: string formato ISO 4217,
“statementDescriptor”: string,
“paymentType”: “card”,
“cardId”: uuid
}