Processo Seletivo

Desenvolvedor C# Pleno

Desafio

Tema: Implementar um sistema de processamento de transações financeiras

Versão: 1.0

Stack: C# (.NET 9)

Candidato: Antonio Eduardo Silveira Demarchi

Recrutadora: Renata Mota

Rio de Janeiro
11/2025

Tutorial para Execução do Projeto .NET 9:

Link:https://github.com/DemarchiWorking/sistema-processamento-transa-es-pagueveloz

- Instalar o Docker, iniciar o ambiente virtualizado
- Baixar os projetos com todos arquivos
- Entrar na pasta
- Rodar o comando:
- docker compose up [ou docker compose up -d --build]
- docker compose ps
- docker ps -a

  Pegue os ID's do meta-contas_api, meta-corefinanceiro_api, meta-transferencias_api e execute o comando com ids respectivos (ou de o start via docker desktop). Exemplo:

- docker start f6b74e6cf2f5 c6732820e09b d65e28bca404
- Vai executar um console mostrando a porta que a aplicação está rodando.

Como Executar Operações	

http://localhost:5000/api/Clientes
{
"clienteId": "1",
"nome": "Antonio Demarchi"
}
 	
1°
	http://localhost:5000/api/Contas
	{
	"clienteId": "1",
	"initialBalance": 0,
	"creditLimit": 1000
	}
	2°

http://localhost:5001/api/Operacao
{
"operation": "credit",
"account_id": "ACC-1",
"amount": 100,
"currency": "BRL",
"reference_id": "TXN-1",
"metadata": {
"description": "credito inicial"
}
}
 	
3°
	http://localhost:5001/api/Transacao
	{
	"operation": "transfer",
	"origin_account_id": "ACC-1",
	"destination_account_id": "ACC-2",
	"amount": 50,
	"currency": "BRL",
	"reference_id": "TXN-2",
	"metadata": {
	"description": "transferencia inicial"
	}
	}
	4°

Documentação

Camada	Tecnologia	Justificativa
Framework	.NET 9.0 (C#)	Performance
Arquitetura	Clean Architecture + CQRS (MediatR)	Separação e Testabilidade
Banco de Dados	PostgreSQL + EF Core	Transações distribuídas
Mensageria	MassTransit (RabbitMQ) + Outbox	Entrega
Logging	Serilog	Registro de ocorrências log
Container	Docker Compose	Fácil execução

Projetos Principais:	
PagueVeloz.Contas	PagueVeloz.CoreFinanceiro

A solução que usei baseia-se em 2 microsserviços construídos sobre o ecossistema .NET 9.0, utilizando Clean Architecture para organização interna com 4 projetos (Api, Aplicacao,Dominio e Infra) e CQRS para segregação de responsabilidades. O sistema prioriza a consistência eventual e a garantia de entrega de mensagens através do Outbox.

O PostgreSQL é um banco relacional confiável e o EF Core atua como ORM para agilizar o desenvolvimento, permitindo o mapeamento objeto-relacional.

Docker Compose para que seja possível subir toda a topologia (APIs, Banco de Dados, RabbitMQ) com um comando, fazendo com que o ambiente de desenvolvimento local seja igual ao de produção.

PagueVeloz.Contas

1. Introdução

1.1 Propósito

Este documento descreve a arquitetura do microsserviço PagueVeloz.Contas, responsável pelo gerenciamento de Clientes e Contas no sistema de processamento de transações financeiras da PagueVeloz.
Requisitos funcionais:

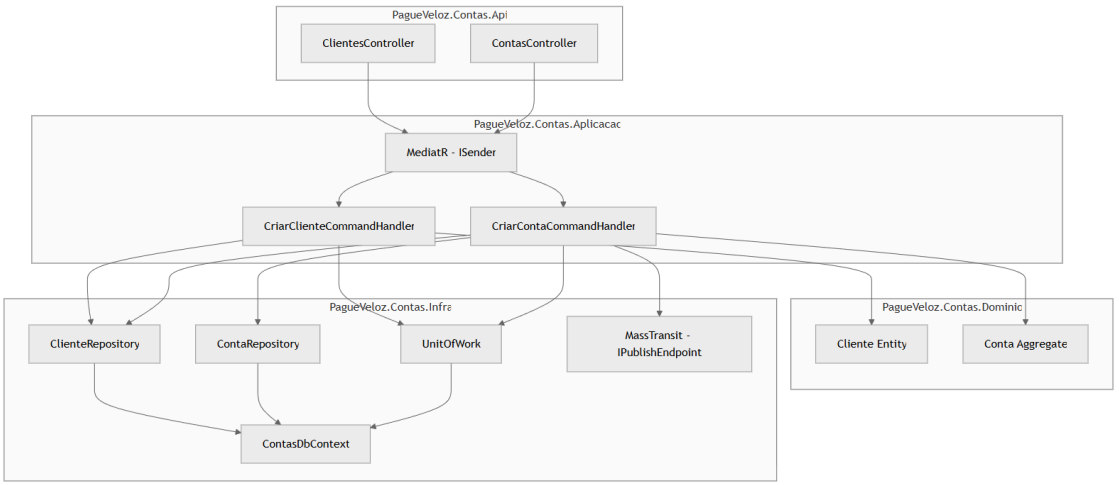
RF01: Criar conta para um cliente [client_id], com saldo inicial, limite de crédito e status inicial active.

RF02: Suportar múltiplas contas por cliente.

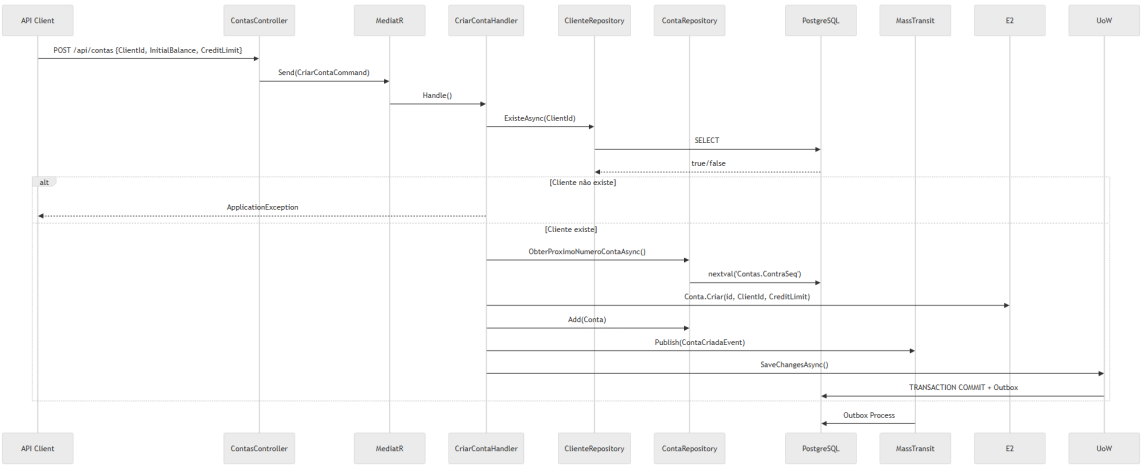
RF03: Validações, existência de cliente, unicidade, valores não negativos.

Gerenciamento de saldos e transações transferido ao CoreFinanceiro.
Saldo inicial é informado no evento ContaCriadaEvent.

2.1 Diagrama de Camadas



2.2 Fluxo de Criação de Conta



3. Modelo de Domínio

3.1 Entidades

Entidade	Propriedades
Cliente	Id, Nome
Conta	Id (ACC-{seq}), ClienteId, LimiteDeCredito, Status

Observação: Sem campos de saldo (gerenciados externamente). LockVersion para concorrência otimista.

3.2 Evento

Evento	Payload
ContaCriadaEvent	AccountId, ClientId, CreditLimit, Status, CreatedAt, InitialBalance

4. Camada de Aplicação (Comandos)

Comando	Handler	Dependências
CriarClienteCommand	CriarClienteCommandHandler	IClienteRepository, IUnitOfWork
CriarContaCommand	CriarContaCommandHandler	IClienteRepository, IContaRepository, IUnitOfWork, IPublishEndpoint

5. Camada de Infraestrutura

5.1 Repositórios

Repositório	Métodos Principais
ClienteRepository	Adicionar(), ExisteAsync()
ContaRepository	GetByIdAsync(), Add(), Update(), ObterProximoNumeroContaAsync() (Sequence)

5.2 Persistência

Banco: PostgreSQL

Tabela	Colunas Chave	Constraints
Clientes	Id (PK)	
Contas	Id (PK, manual), ClienteId (FK), LimiteDeCredito, Status, lock_version (Concurrency)	Sequence ContaSeq para numeração.

5.3 UnitOfWork

Wrapper simples para SaveChangesAsync() com suporte a transações.

6. API REST

Endpoint (POST)	Request DTO	Response DTO (201)
/api/clientes	CriarClienteRequest (ClienteId, Nome)	CriarClienteResponse (ClienteId, Nome, CreatedAt)
/api/contas	CriarContaRequest (ClienteId, InitialBalance, CreditLimit)	CriarContaResponse (AccountId, ClienteId, CreditLimit, Status, CreatedAt)

Erros: Cliente já existe, Cliente não encontrado...

Criar Cliente
curl -X POST http://localhost:5000/api/clientes -H "Content-Type: application/json" -d '{"clienteId": "CLI-001", "nome": "João Silva"}'
Criar Conta
curl -X POST http://localhost:5000/api/contas -H "Content-Type: application/json" -d '{"clienteId": "CLI-001", "initialBalance": 0, "creditLimit": 100000}'

PagueVeloz.CoreFinanceiro

1. Introdução

1.1 Propósito

Responsável pelo processamento de operações financeiras (credit, debit, reserve, capture, reversal) e transferências entre contas no sistema da PagueVeloz.
Processamento de Operações, Validações, Eventos e Resiliência.

1.2 Escopo

Operações Básicas: Processar credit, debit, reserve, capture, reversal com validações de saldo, limite e idempotência.Transfere valores entre contas com atomicidade e retry em concorrência.

Funcionalidades:

RF05: Processar credit: Adicionar valor ao saldo disponível.

RF06: Processar debit: Remover valor do saldo disponível, considerando limite de crédito.

RF07: Processar reserve: Mover valor do saldo disponível para reservado.

RF08: Processar capture: Remover valor do saldo reservado.

RF09: Processar reversal: Reverter operação anterior.

RF10: Processar transfer: Mover valor entre contas (debit na origem, credit no destino, atômico).

RF11: Garantir idempotência via reference_id (evitar duplicatas).

RF12: Impedir operações que deixem saldo disponível negativo.

RF13: Respeitar limite de crédito em débitos.

RF14: Reservas apenas com saldo disponível suficiente.

RF15: Capturas apenas com saldo reservado suficiente.

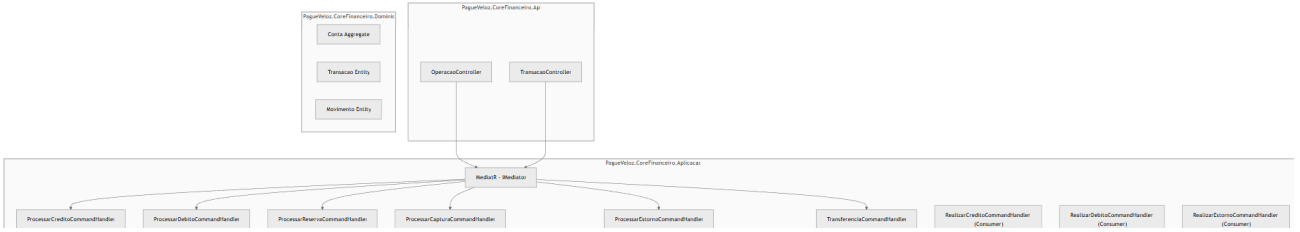
RF16: Validar campos obrigatórios.

RF17: Suportar metadata opcional.

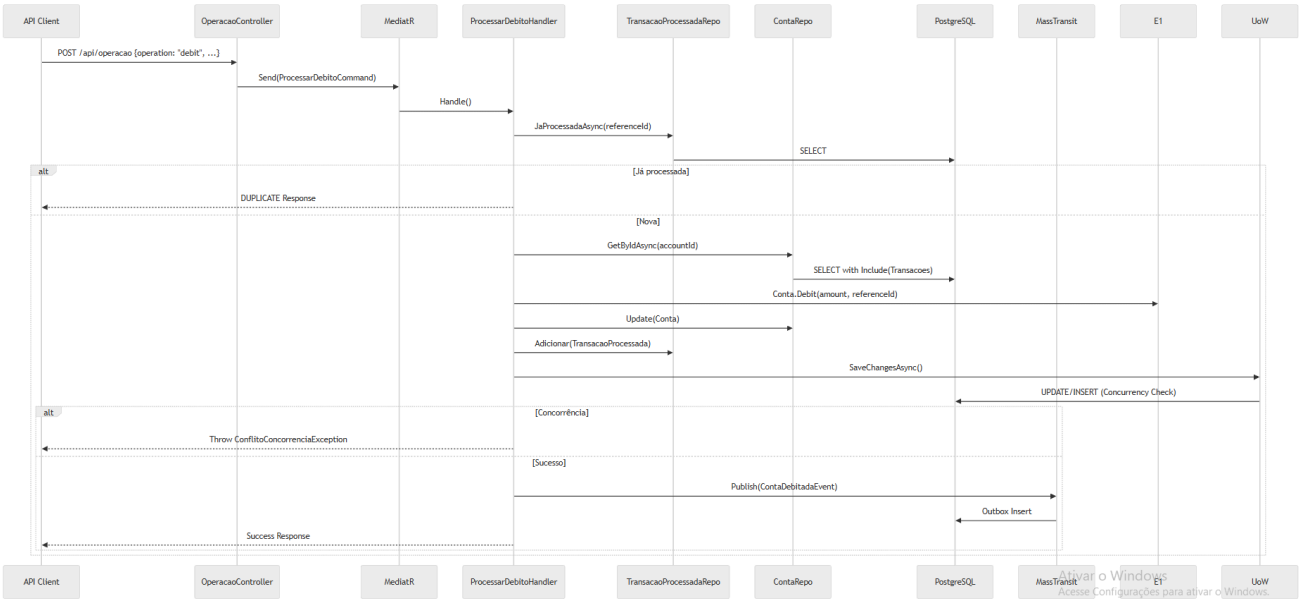
RF18: Gerar eventos assíncronos.

2. Visão Geral da Arquitetura

2.1 Diagrama de Camadas



2.2 Fluxo de Processamento de Débito



3. Modelo de Domínio

3.1 Entidades e Aggregates

Entidade/Aggregate	Propriedades Principais	Comportamentos
Conta	AccountId, Currency, Status, SaldoDisponivelEmCentavos, SaldoReservadoEmCentavos, LimiteDeCreditoEmCentavos, LockVersion, Transacoes	Factory CriarNova(); Métodos: Debit(), Credit(), Reserve(), Capture(), Estornar(), Block(); Valida status, valores positivos, saldos suficientes; Lança Domain Events.
Transacao	Id, Tipo (Enum:	Registrada em Conta;

	Credit, Debit, etc.), Valor, ReferenceId, Currency, Status, FinalBalance …	Suporta reversão.
Movimento	Id, AccountId, Tipo, Value, Coin, ReferenceId, TransactionId, Timestamp, MetadadosJson	Registro de movimentos financeiros.

4. Camada de Aplicação (Comandos)

Comando	**Handler**	**Validações Principais**
ProcessarCreditoCommand	ProcessarCreditoCommandHandler	Idempotência, Conta existe, Moeda, Saldo.
ProcessarDebitoCommand	ProcessarDebitoCommandHandler	Limite de crédito.
ProcessarReservaCommand	ProcessarReservaCommandHandler	Saldo disponível suficiente.

ProcessarCapturaCommand	ProcessarCapturaCommandHandler	Saldo reservado suficiente.
ProcessarEstornoCommand	ProcessarEstornoCommandHandler	originalReferenceId em metadata, Transação original existe.
TransferenciaCommand	TransferenciaCommandHandler	Contas existem, Ordenação IDs para locks.

Transações atômicas via UnitOfWork (Begin/Commit/Rollback).

5. Camada de Infraestrutura

5.1 Repositórios

Repositório	**Métodos Principais**
ContaRepository	GetByIdAsync(), Add(), Update(), GetByAccountNumber().
IdempotenciaRepository	GetByReferenceIdAsync(), Add().
TransacaoProcessadaRepository	

5.2 Persistência

Banco: PostgreSQL ("Financeiro").

Tabelas: Contas, TransacoesProcessadas, Movimentos

5.3 UnitOfWork

Gerencia transações, commit, rollback em falhas.

6. API REST

Endpoint (POST)	**Request DTO**	**Response DTO (201)**
/api/operacao	TransacaoRequest	TransacaoResponse
/api/transacao	TransacaoFinanceiraRequest	TransacaoResponse

 # Débito
curl -X POST http://localhost:5001/api/operacao -H "Content-Type: application/json" -d '{"operation": "debit", "accountId": "ACC-001", "amount": 20000, "currency": "BRL", "referenceId": "TXN-002"}'
 # Transferência
curl -X POST http://localhost:5001/api/transacao -H "Content-Type: application/json" -d '{"operation": "transfer", "originAccountId": "ACC-001", "destinationAccountId": "ACC-002", "amount": 50000, "currency": "BRL", "referenceId": "TXN-004"}'

Testes

1. Introdução

Foram feitos alguns testes unitários para a entidade Conta do PagueVeloz.CoreFinanceiro.
Os testes validam o comportamento do domínio conforme regras de negócio definidas.

Framework de Testes: xUnit.net com FluentAssertions para asserções fluentes e legíveis.

Cobertura:
- Criação de contas (RF01, RF04).
- Operações de crédito/débito.
- Operações de reserva/captura.

Princípio AAA Pattern
[Arrange (preparação), Act (execução), Assert (verificação) nos testes]

2. Visão Geral dos Testes

Os testes estão organizados em classes temáticas para facilitar manutenção:

- ContaTests: Foco em criação e validações iniciais.
- ContaTests_Operacao: Operações de crédito/débito.
- ContaTests_ReservaCaptura: Operações de reserva/captura.

3. Detalhes dos Testes por Classe

==========

Foco: Validação do CriarNova para garantir integridade inicial da conta.

==========

CriarNova_DeveCriarContaValida_ComParametrosCorretos

Verifica criação de conta válida com ID, moeda e limite, garantindo status active e saldos zerados.

Arrange: Parâmetros válidos.
Act: Chama CriarNova.
Assert: Objeto não nulo, status, saldos e poder de compra.

==========

CriarNova_DeveLancarExcecao_QuandoParametrosInvalidos

Testar cenários inválidos (ID/moeda vazios/nulos) para validar exceções de argumento.

Arrange: Dados inline inválidos.
Act: Chama CriarNova.
Assert: Lança ArgumentNullException.
==========

CriarNova_DeveLancarExcecao_QuandoLimiteNegativo

Garante que limite negativo lance exceção, prevenindo estados inválidos

Arrange: Limite -100.
Act: Chama CriarNova.
Assert: Lança ArgumentException com mensagem específica.
==========

Foco: Operações de crédito/débito, incluindo uso de limite e prevenção de duplicatas/saldos negativos.

==========

Credit_DeveAumentarSaldo_QuandoOperacaoValida

Validar adição ao saldo disponível e registro de transação.

Arrange: Conta nova.
Act: Credit(10000, "TX-01").

Assert: Saldo atualizado, transação existe.

==========

Credit_DeveLancarExcecao_SeTransacaoDuplicada

Testa idempotência: Duplicata lança exceção para evitar processamentos repetidos.

Arrange: Crédito inicial com ref duplicada.
Act: Crédito novamente.
Assert: Lança TransacaoJaProcessadaException.

==========

Debit_DeveDiminuirSaldo_QuandoHaSaldoSuficiente

Verifica débito com saldo positivo.

Arrange: Crédito inicial 20000.
Act: Debit(5000, "TX-DEBIT-01").
Assert: Saldo 15000.
==========

Debit_DeveUsarLimite_QuandoSaldoZero

Confirma uso de limite de crédito quando saldo é zero.

Arrange: Conta com limite 50000.
Act: Debit(10000, "TX-LIMIT-01").
Assert: Saldo -10000, poder de compra 40000.

==========

Debit_NaoDeveProcessar_QuandoExcedePoderDeCompra

Garante que débito excedendo poder de compra não altere saldos nem registre transação.

Arrange: Limite 1000.
Act: Debit(2000, "TX-FAIL").
Assert: Saldo inalterado, sem transação.

==========

Foco: Operações de reserva e captura, para cenários de pagamento.

==========

Reserve_DeveMoverSaldoDeDisponivelParaReservado

Valida movimentação de saldo disponível para reservado sem alterar total

Arrange: Crédito 1000.
Act: Reserve(300, "TX-RES-01").
Assert: Disponível 700, Reservado 300, Total 1000.

==========

Capture_DeveConsumirSaldoReservado

Confirma remoção de valor reservado, reduzindo total.

Arrange: Crédito 1000 + Reserva 300.
Act: Capture(300, "TX-CAP-01").
Assert: Reservado 0, Disponível 700, Total 700.

==========

Capture_DeveFalhar_SeNaoHouverReservaSuficiente

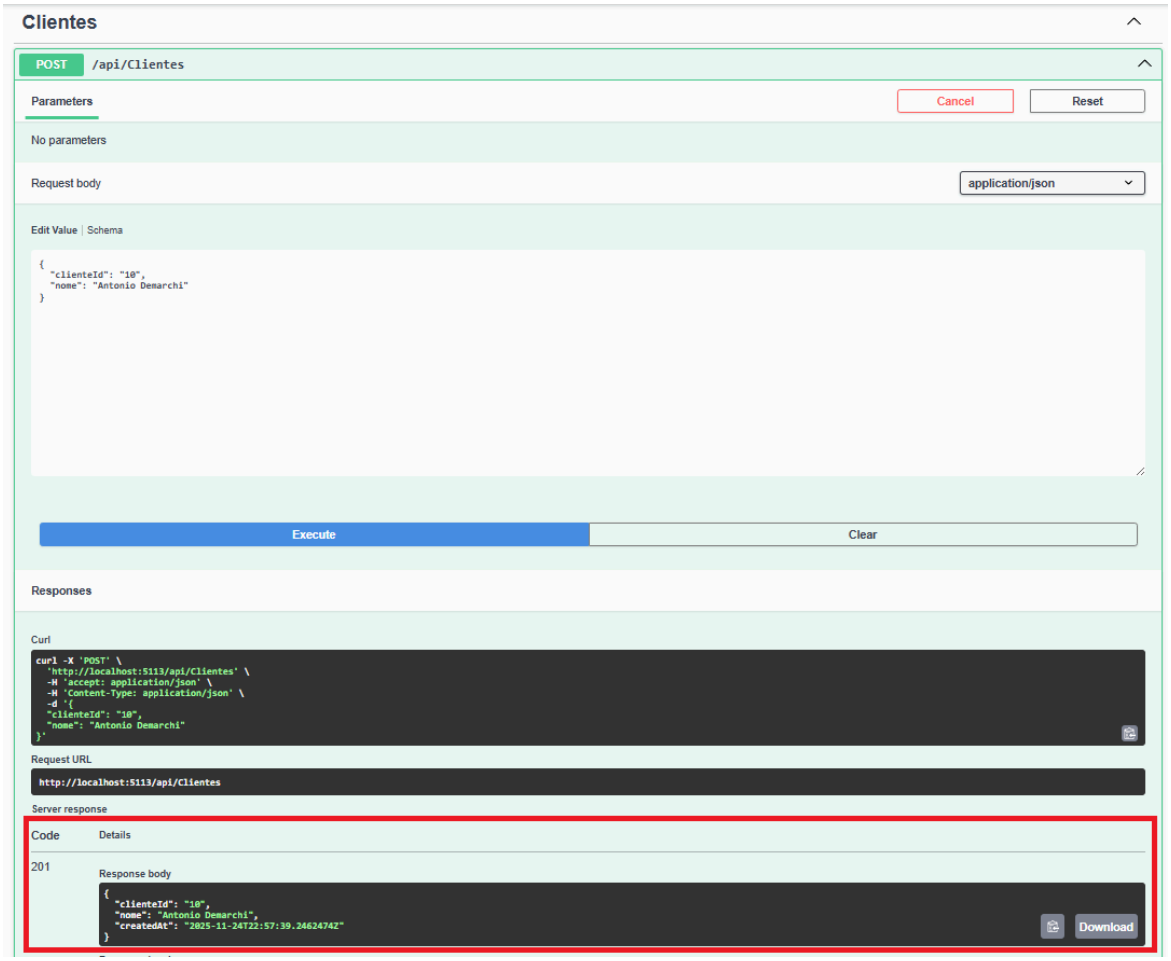
Testa falha em captura excedendo reservado, lançando exceção de domínio.

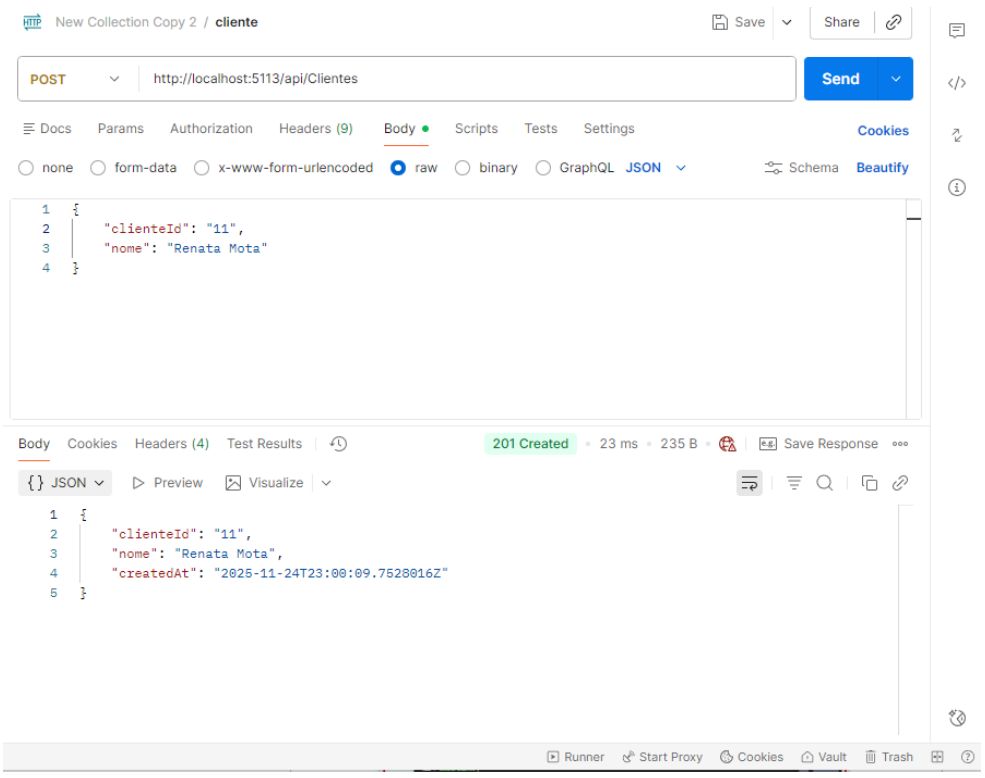
Arrange: Crédito 1000 + Reserva 100.
Act: Capture(200, "TX-FAIL").
Assert: Lança DomainException com mensagem específica.

Dependências Principais

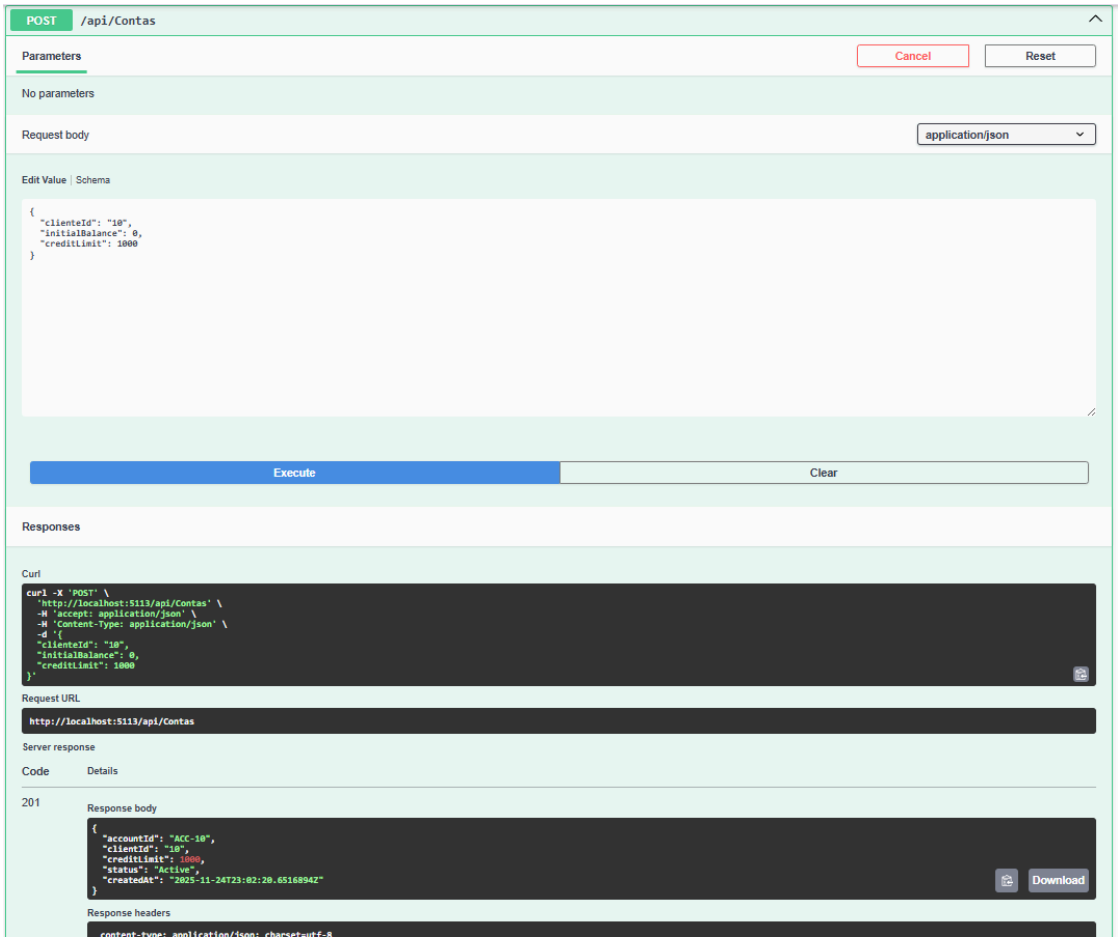
Pacote/Biblioteca	**Propósito**
MediatR	Dispatch de comandos.
MassTransit	Publicação de eventos (Outbox).
Entity Framework Core	ORM Persistência (PostgreSQL).
Microsoft.AspNetCore	API REST
Serilog	Logging estruturado

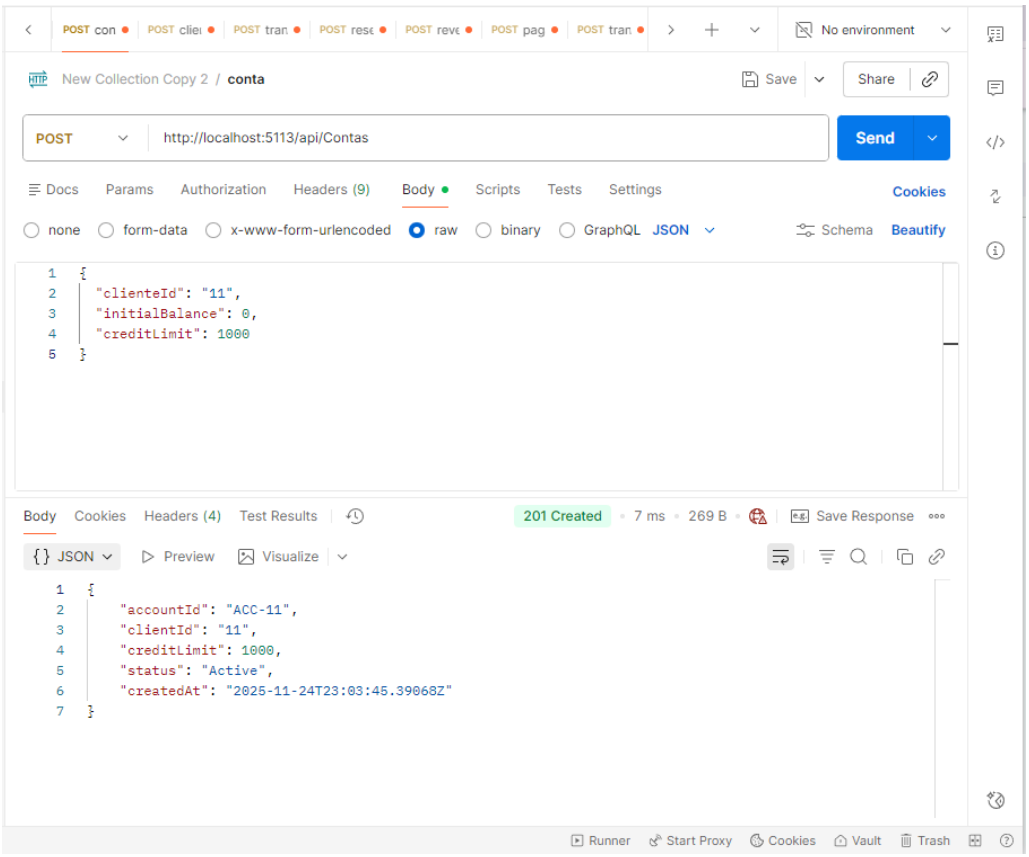
Execuções: Criar Cliente



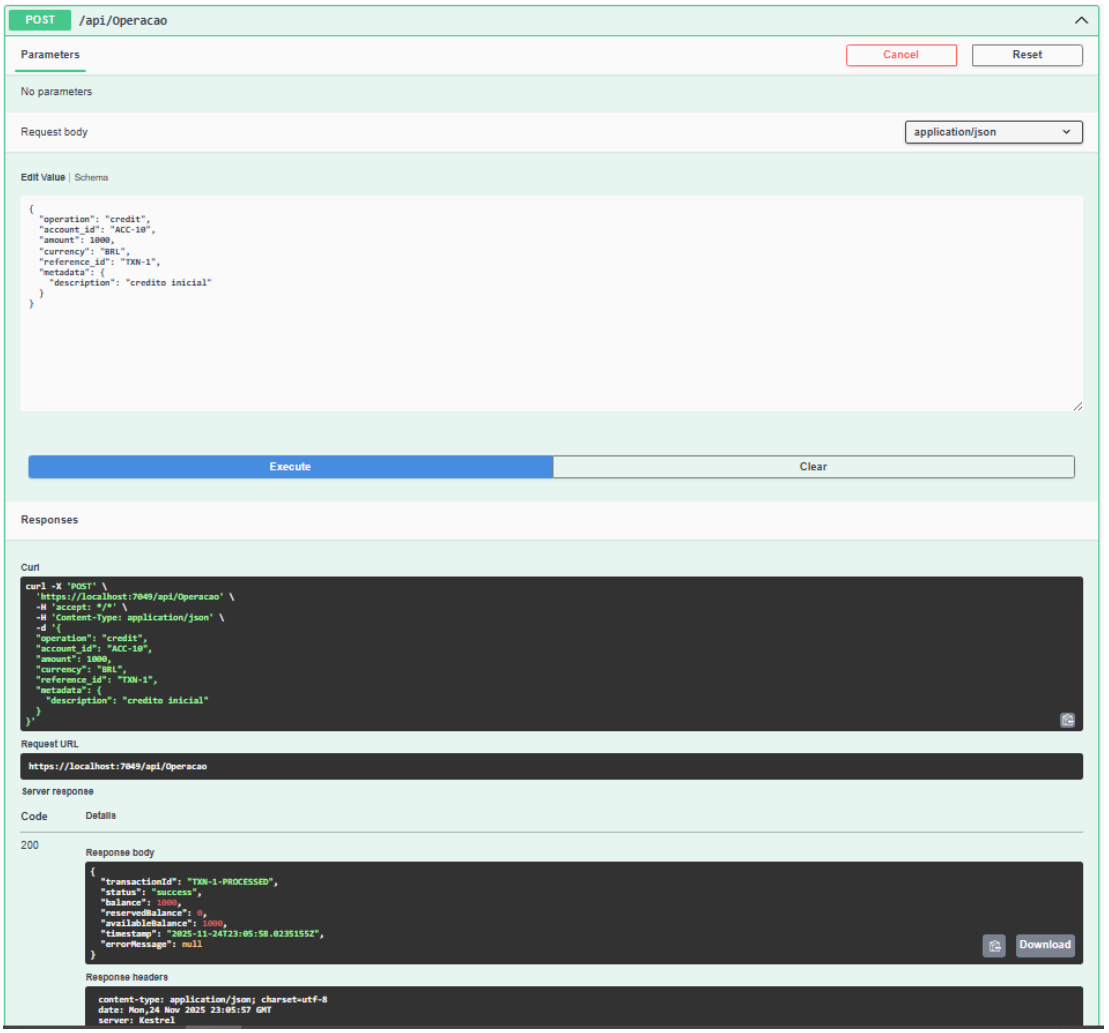


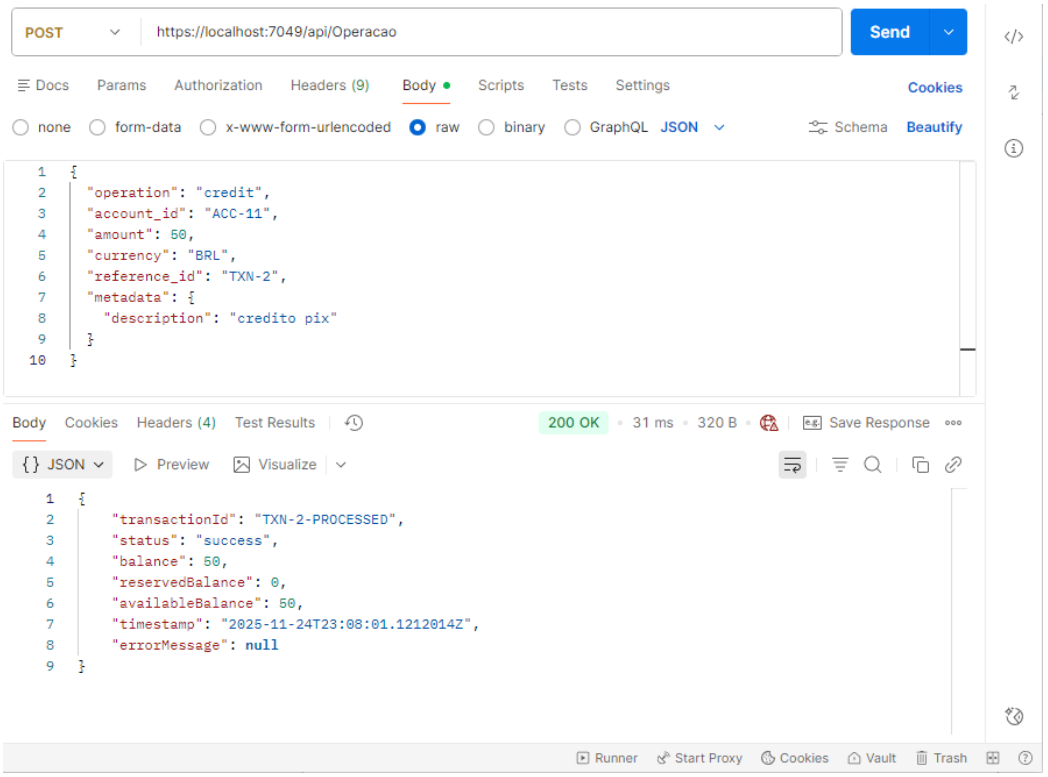
Criar Conta (múltiplos clientes se necessário)



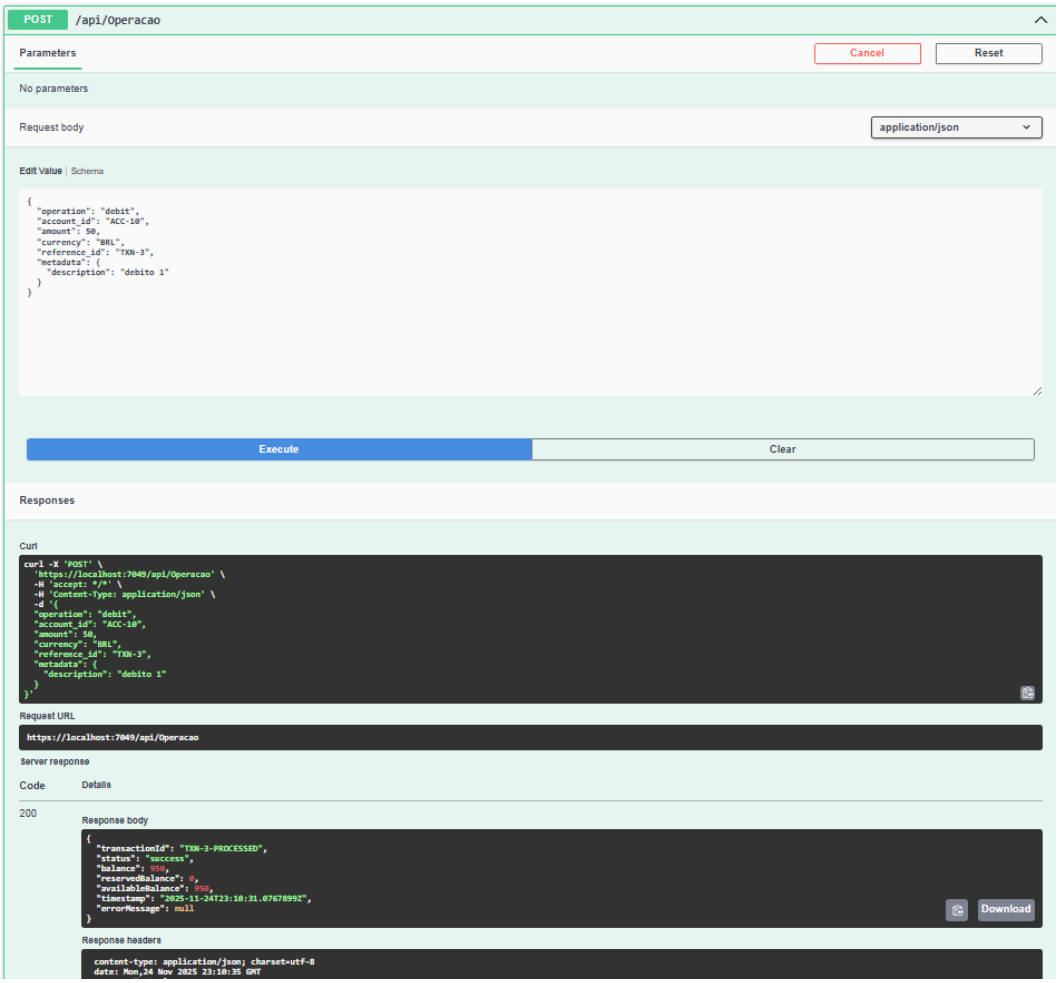


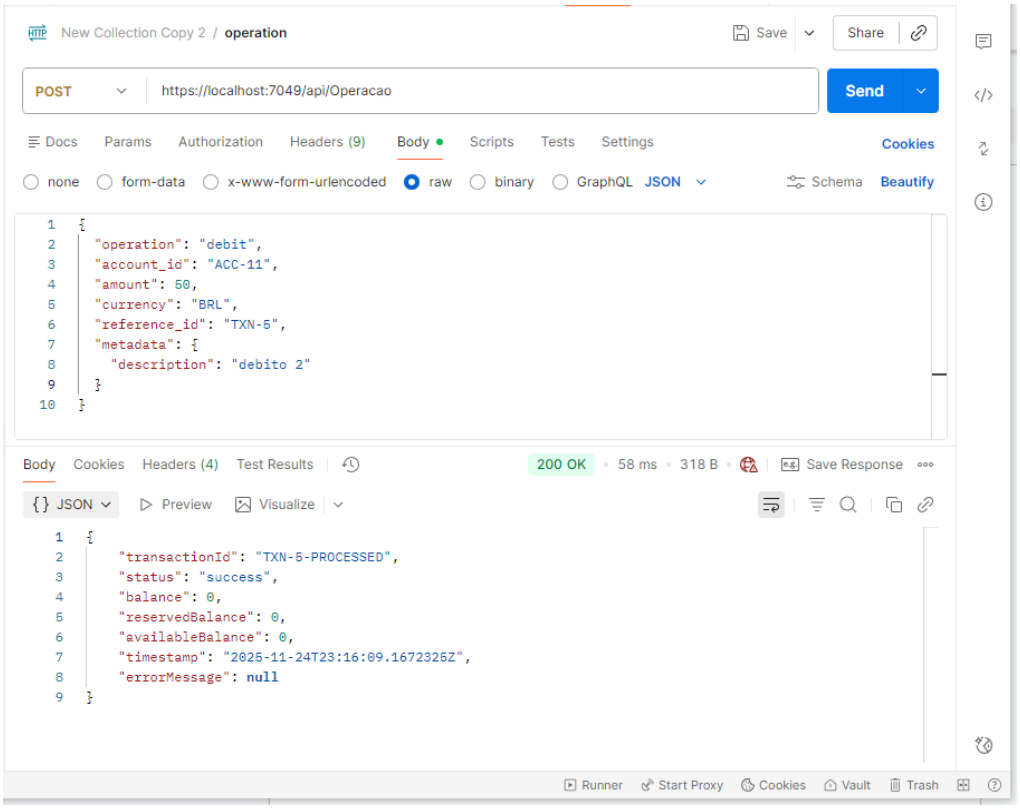
Operação Crédito



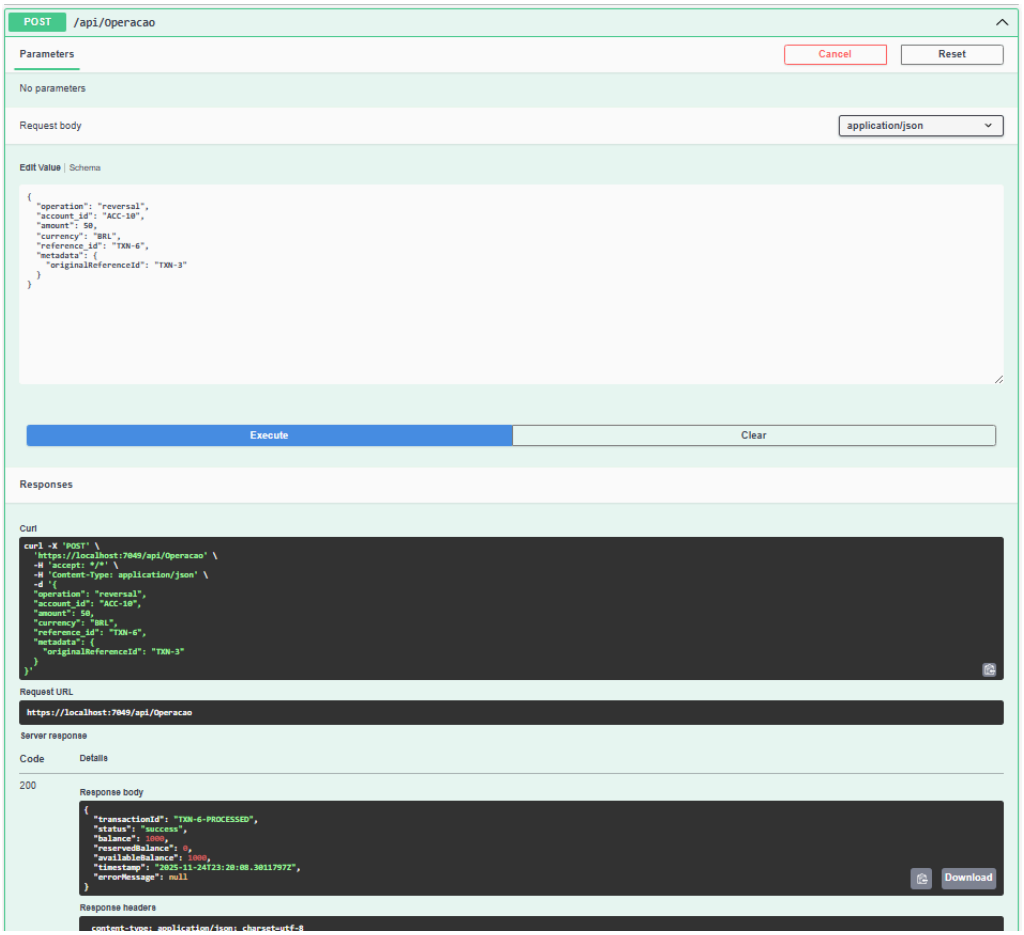


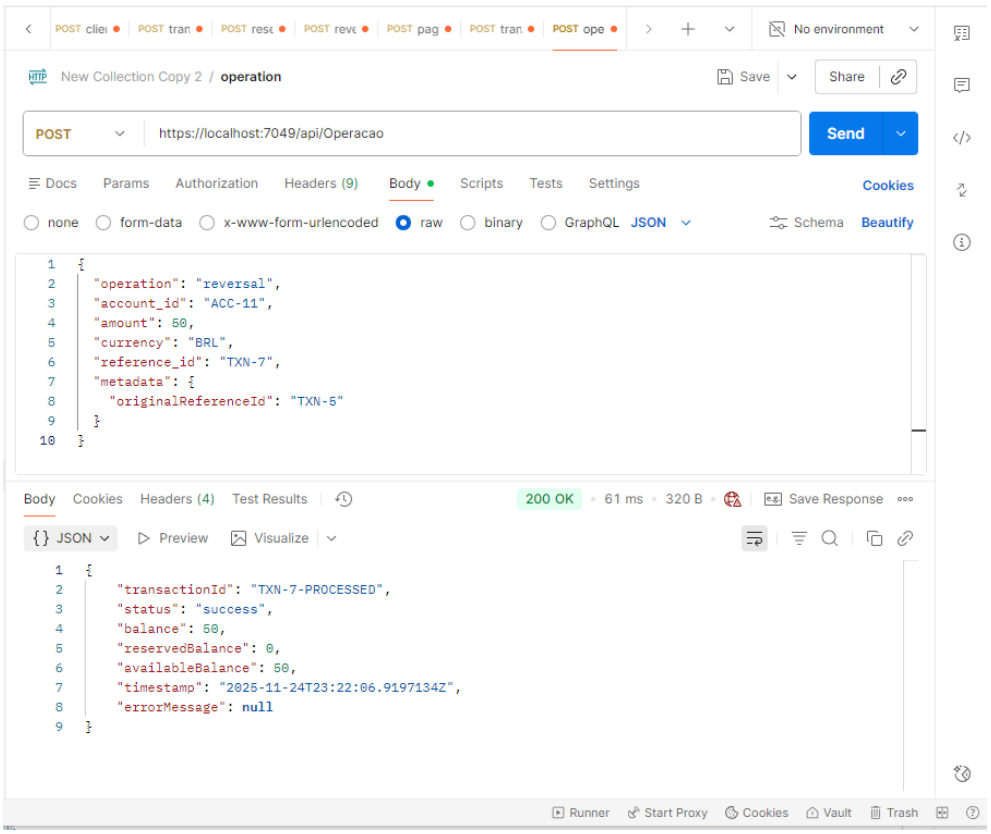
Operação Débito



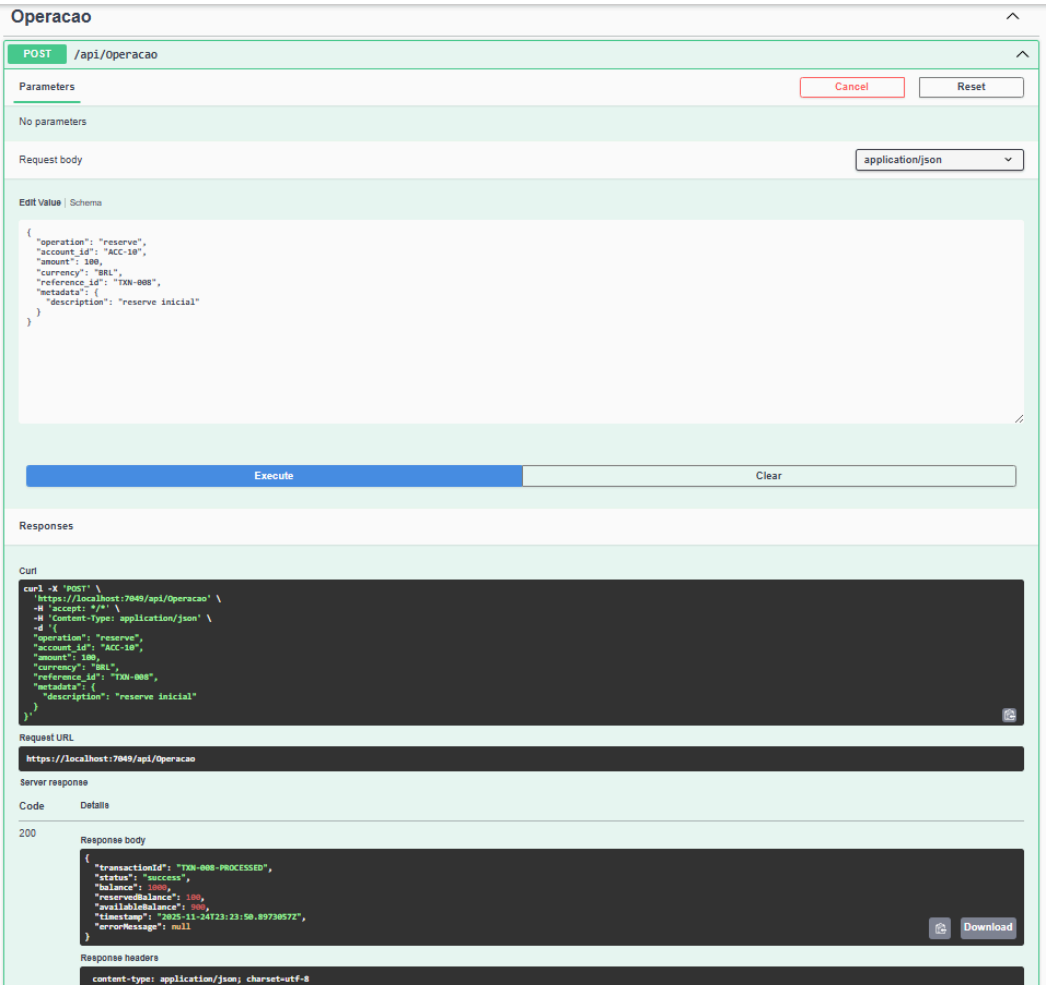


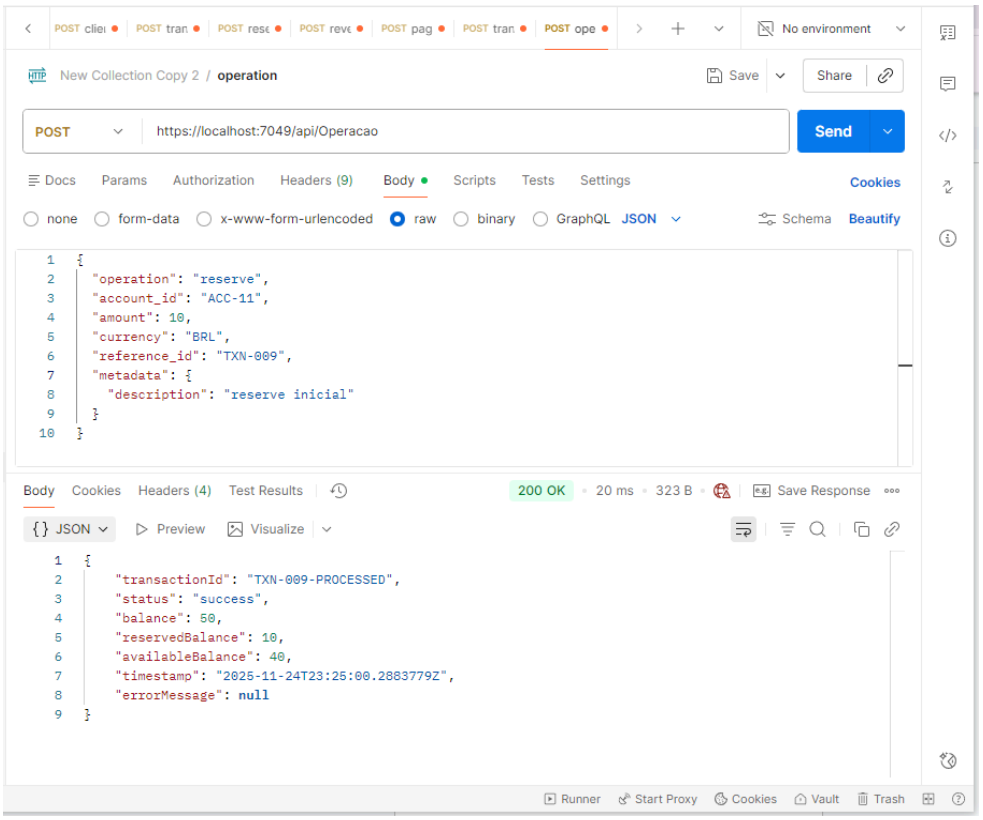
Reverter (Operação)


POST
/api/Operacao
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"operation": "reversal",
"account_id": "ACC-10",
"amount": 50,
"currency": "BRL",
"reference_id": "TXN-6",
"metadata": {
"originalReferenceId": "TXN-3"
}
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'https://localhost:7049/api/Operacao' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"operation": "reversal",
"account_id": "ACC-10",
"amount": 50,
"currency": "BRL",
"reference_id": "TXN-6",
"metadata": {
"originalReferenceId": "TXN-3"
}
}'
Request URL
https://localhost:7049/api/Operacao
Server response
Code
Details
200
Response body
{
"transactionId": "TXN-6-PROCESSED",
"status": "success",
"balance": 1000,
"reservedBalance": 0,
"availableBalance": 1000,
"timestamp": "2025-11-24T23:20:08.30117972",
"errorMessage": null
}
Download
Response headers
content-type: application/json; charset=utf-8

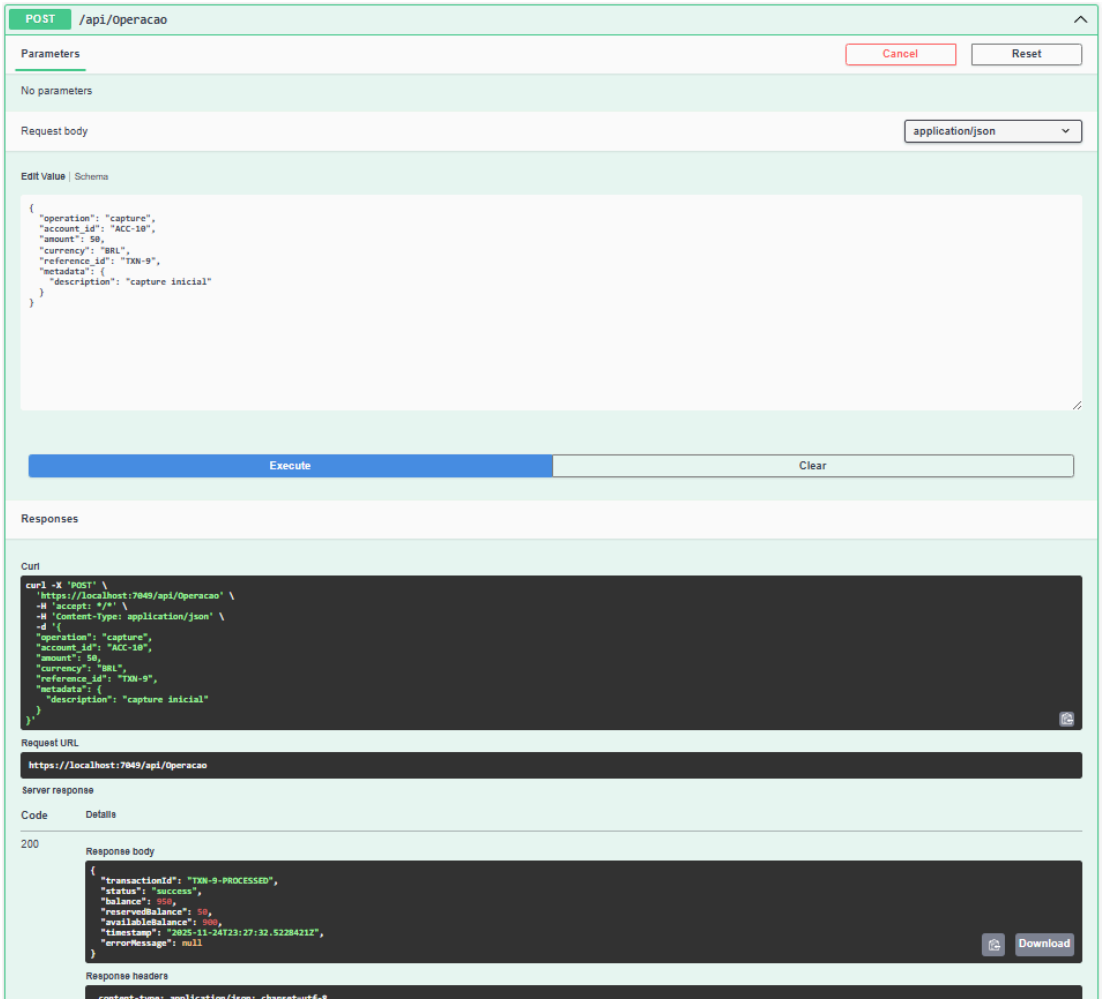

No environment
New Collection Copy 2 / operation
Save
Share
POST
https://localhost:7049/api/Operacao
Send
Docs
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"operation": "reversal",
"account_id": "ACC-11",
"amount": 50,
"currency": "BRL",
"reference_id": "TXN-7",
"metadata": {
"originalReferenceId": "TXN-5"
}
}
Body
Cookies
Headers (4)
Test Results
200 OK
61 ms
320 B
Save Response
JSON
Preview
Visualize
{
"transactionId": "TXN-7-PROCESSED",
"status": "success",
"balance": 50,
"reservedBalance": 0,
"availableBalance": 50,
"timestamp": "2025-11-24T23:22:06.9197134Z",
"errorMessage": null
}
Runner
Start Proxy
Cookies
Vault
Trash

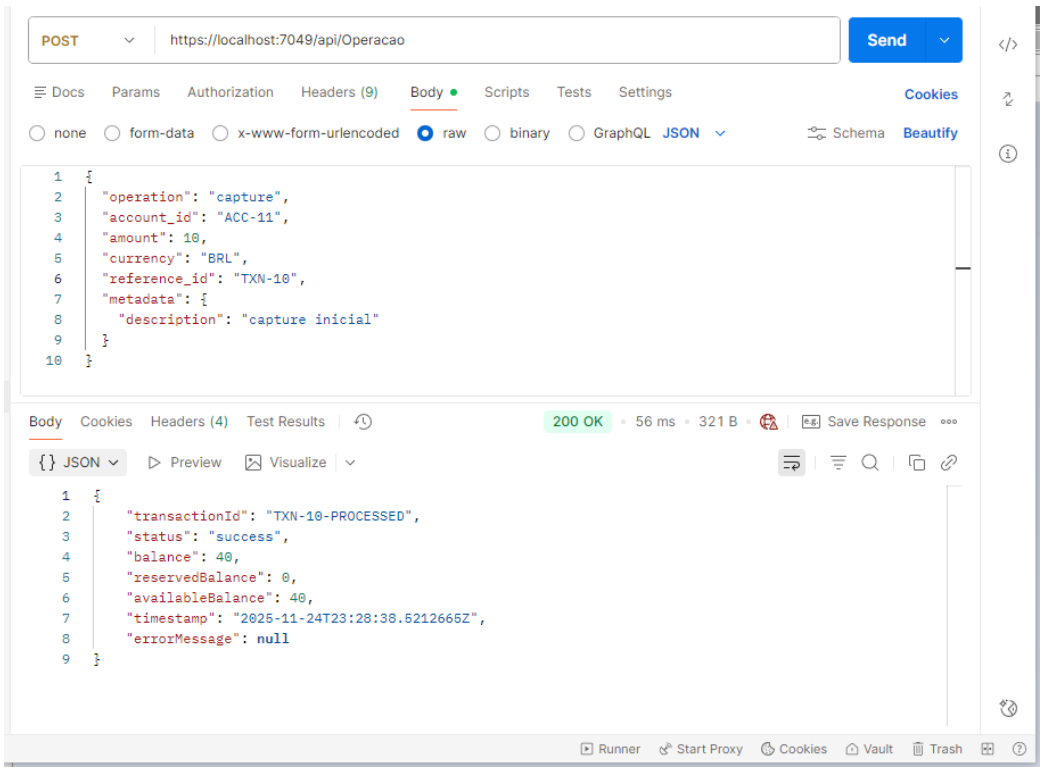
Reserva


Operacao
POST
/api/Operacao
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"operation": "reserve",
"account_id": "ACC-10",
"amount": 100,
"currency": "BRL",
"reference_id": "TXN-008",
"metadata": {
"description": "reserve inicial"
}
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'https://localhost:7049/api/Operacao' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"operation": "reserve",
"account_id": "ACC-10",
"amount": 100,
"currency": "BRL",
"reference_id": "TXN-008",
"metadata": {
"description": "reserve inicial"
}
}'
Request URL
https://localhost:7049/api/Operacao
Server response
Code
Details
200
Response body
{
"transactionId": "TXN-008-PROCESSED",
"status": "success",
"balance": 1000,
"reservedBalance": 100,
"availableBalance": 900,
"timestamp": "2025-11-24T23:23:50.8973057Z",
"errorMessage": null
}
Download
Response headers
content-type: application/json; charset=utf-8

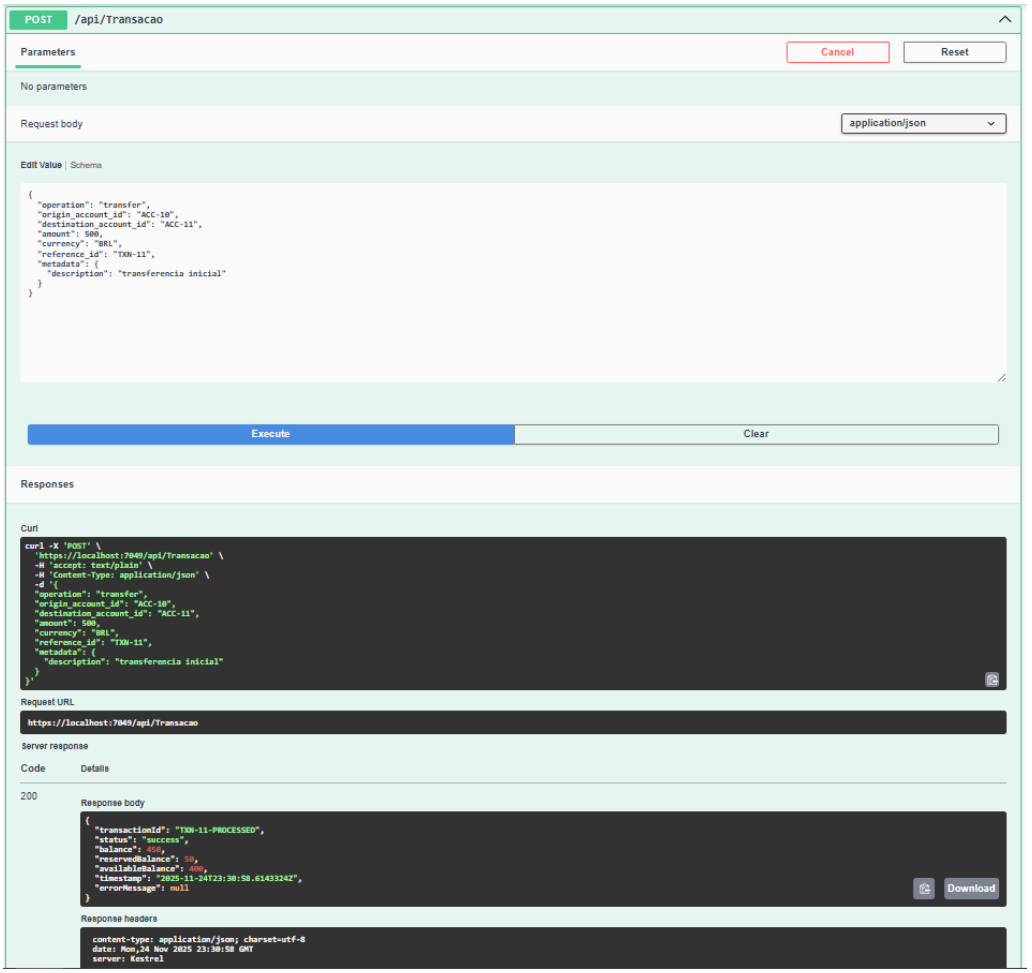

No environment
New Collection Copy 2 / operation
Save
Share
POST
https://localhost:7049/api/Operacao
Send
Docs
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"operation": "reserve",
"account_id": "ACC-11",
"amount": 10,
"currency": "BRL",
"reference_id": "TXN-009",
"metadata": {
"description": "reserve inicial"
}
}
Body
Cookies
Headers (4)
Test Results
200 OK
20 ms
323 B
Save Response
JSON
Preview
Visualize
{
"transactionId": "TXN-009-PROCESSED",
"status": "success",
"balance": 50,
"reservedBalance": 10,
"availableBalance": 40,
"timestamp": "2025-11-24T23:25:00.2883779Z",
"errorMessage": null
}
Runner
Start Proxy
Cookies
Vault
Trash

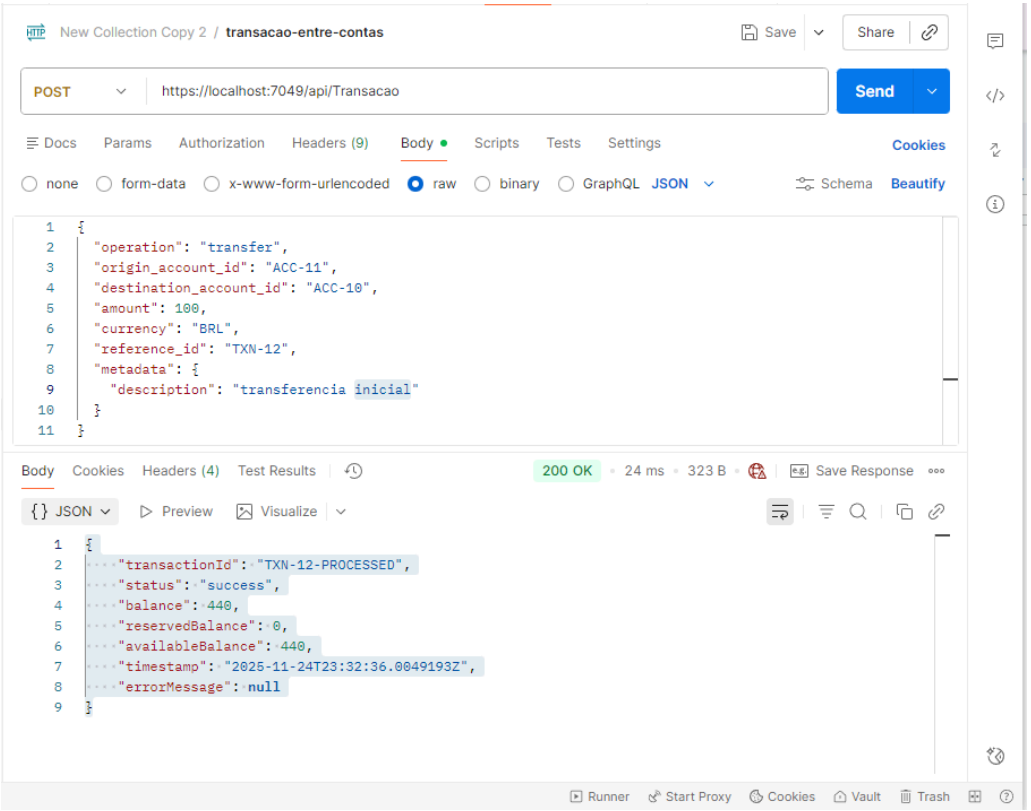
Capture





Transferência





Não tive tempo para fazer o melhor trabalho possível devido aos

compromissos do dia a dia, gostaria de ter feito um projeto melhor porém foi o que tive como entregar dentro do prazo estipulado. Gostaria de uma oportunidade para demonstrar meu esforço e minha dedicação na equipe de vocês.

Att.
Antonio Eduardo Silveira Demarchi