

 **marceloandrade** Update README.md a685aab on Aug 18

1 contributor

125 lines (104 sloc) 6.44 KB

# Rotina de pagamento

A Pismo oferece aos seus clientes uma plataforma para o processamento de pagamentos. Dentre as muitas das responsabilidades de uma plataforma de pagamento, uma das mais críticas é o controle dos lançamentos de débito e crédito e seus vencimentos nas faturas dos clientes. O desafio técnico a seguir explora o seguinte cenário.

Cada portador de cartão (cliente) possui uma conta e a cada operação realizada pelo cliente um lançamento é criado e associado a sua respectiva conta. Cada lançamento possui um tipo (compra a vista, compra parcelada, saque, etc.), um valor, uma data de criação e uma data de vencimento. Segue abaixo uma estrutura de dados sugerida:

Operations Types

OperationType_ID	Description	ChargeOrder
1	COMPRA A VISTA	2
2	COMPRA PARCELADA	1
3	SAQUE	0
4	PAGAMENTO	0

Accounts

Account_ID	AvailableCreditLimit	AvailableWithdrawalLimit
1	5000.0	5000.0

Transactions

Transaction_ID	Account_ID	OperationType_ID	Amount	Balance	EventDate	DueDate
1	1	1	-50.0	-50.0	2017-04-05	2017-05-10
2	1	1	-23.5	-23.5	2017-04-10	2017-05-10
3	1	1	-18.7	-18.7	2017-04-30	2017-06-10

Quando ocorre um pagamento o sistema deve utilizar seu valor para abater o saldo de cada transação levando em consideração primeiro o ChargeOrder do tipo de operação associado a transação e em seguida o EventDate. Em outras palavras, as transações com ChargeOrder mais baixo devem ser abatidas primeiro. Se duas transações possuem o mesmo ChargeOrder, a mais antiga deve ser abatida primeiro. Observe que um pagamento pode ter um valor insuficiente para abater o saldo total da conta, ou pode ter um saldo maior.

Pagamento = R\$ 70,00

#### Transactions

Transaction_ID	Account_ID	OperationType_ID	Amount	Balance	EventDate	DueDate
1	1	3	-50.0	-50.0 + 50.0 = 0	2017-04-05	2017-05-10
2	1	1	-23.5	-23.5 + 20.0 = 0	2017-04-10	2017-05-10
3	1	1	-18.7	-18.7	2017-04-30	2017-06-10
4	1	4	70.0	0.0	2017-04-30	2017-06-10

A linha 4 mostra o lançamento de pagamento com saldo igual a 0. Se o pagamento fosse maior do que o saldo total das transações o saldo do pagamento mostraria um valor maior do que 0 indicando que o cliente possui crédito.

Todo valor abatido deve ser devolvido ao limite disponível na conta do cliente (ver coluna AvailableCreditLimit e AvailableWithdrawalLimit na tabela Accounts).

O sistema deve rastrear que lançamento de pagamento abateu qual lançamento de débito e o valor abatido. Veja o exemplo abaixo:

#### PaymentsTracking

PaymentTracking_ID	CreditTransaction_ID	DebitTransaction_ID	Amount
1	4	1	50.0
2	4	2	20.0

## Solução

Desenvolver duas APIs como microserviços: AccountsAPI e TransactionsAPI. Cada API deve ter no mínimo os seguintes endpoints.

### AccountsAPI

```

PATCH /v1/accounts/<id>
{
  "available_credit_limit": {
    "amount": 123.45 // enviar valor negativo para subtrair
  },
  "available_withdrawal_limit": {
    "amount": 123.45 // enviar valor negativo para subtrair
  }
}

GET /v1/accounts/limits

```

### TransactionsAPI

```

POST /v1/transactions
{
  "account_id": 1,
  "operation_type_id": 1,

```

```
        "amount": 123.45
    }

    POST /v1/payments (pode enviar multiplos pagamentos)
    [
        {
            "account_id": 1,
            "amount": 123.45
        },
        {
            "account_id": 1,
            "amount": 456.78
        }
    ]
```

## Observações

---

- Desenvolver utilizando Java ou Groovy.
- Cada microserviço deve ser uma aplicação standalone.
- Idealmente, cada API deve utilizar bancos separados (pode ser apenas schemas diferentes).
- Será avaliada a proficiência do candidato na utilização dos frameworks escolhidos, qualidade do código, utilização de testes unitários e de integração e a sofisticação na escrita da solução.
- A solução de ser publicada no github juntamente com um readme com instruções para execução.
- Bônus: utilização de docker e AWS.