

# Projeto Final - Turma PCD

## VISÃO GERAL

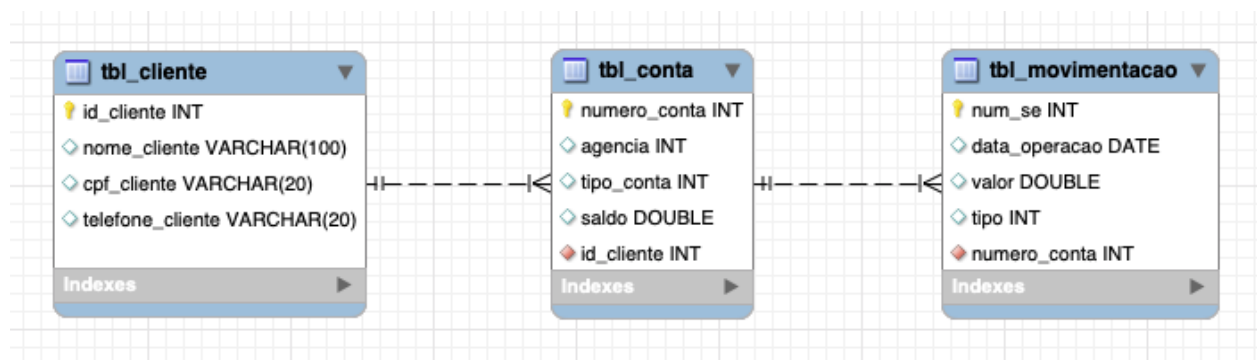
O Banco Itaú quer automatizar algumas contas correntes de antigos clientes para que sejam migradas para um novo sistema diferente dos demais. Alguns destes clientes possuem características bem específicas e precisam de uma conta que seja a mais simples possível. Dessa forma vocês foram escalados na SQUAD que vai implementar esta solução

## OBJETIVOS

1. Seu projeto tem por objetivo implementar uma API que permita:
  - a. cadastrar clientes
  - b. cadastrar contas de clientes
  - c. registrar movimentações de contas
2. Vocês como equipe devem trabalhar de forma organizada e ordenada para conseguirem implementar os requisitos do sistema de forma ampla

## ESPECIFICAÇÕES

Aqui temos as seguintes especificações do modelo de dados (Banco de dados). Teremos também uma descrição da figura a seguir. Observe a figura abaixo:



---

Neste modelo temos 3 tabelas, basicamente que serão descritas a seguir, com seus respectivos tipos de dados

- **tbl\_cliente** - Tabela que armazena dados do cliente
  - id\_cliente: integer auto\_increment e chave\_primaria
  - nome\_cliente: varchar(100)
  - cpf\_cliente: varchar(20) unique
  - telefone\_cliente: varchar(20) unique
- **tbl\_conta** - Tabela que armazena as contas correntes do cliente
  - numero\_conta: integer auto\_increment e chave primária
  - agencia: inteiro
  - tipo\_conta: inteiro cujos valores podem ser
    - 1 - conta pessoa fisica
    - 2 - conta estudantil
    - 3 - conta salario
  - saldo: double
  - id\_cliente: o respectivo cliente proprietário da conta
- **tbl\_movimentacao** - Tabela que registra as movimentações da conta
  - num\_seq: integer not null auto\_increment e chave primária (uma forma de registrar essas operações)
  - data\_operacao: date (a data que foi feita a operação)
  - valor: double
  - tipo\_operacao: integer (1 para credito e - 2 para debito)
  - descricao: varchar(255) - para descrever a operação
  - numero\_conta: o número da conta corrente para operação

Aqui as relações entre as tabelas são:

- 1 cliente possui N contas (mas 1 conta é de apenas 1 cliente - não temos o recurso de contas conjuntas)
- 1 conta possui N movimentações, ou seja, é possível ter várias movimentações de débito (retirada) ou crédito (depósito) na mesma conta. Cada movimentação refere-se apenas a uma única conta

---

## DEFINIÇÕES DO PROJETO

Você deverá ter os seguintes componentes na sua API

- model (para mapear as tabelas para classes Java)
- repo (para manipular esses dados no banco)
- service (para ter os serviços de manipulação e implementação das regras de negócio)
- controller (para expor estas funcionalidades na web)

## DEFINIÇÕES DOS SERVIÇOS

Precisaremos ter os seguintes serviços para manipular seus dados, descritos a seguir:

- **ClienteService** (para manipular o cliente)
  - **cadastrarCliente(Cliente c)**: que recebe como entrada um objeto do tipo cliente e retorna-o completo caso ele tenha sido cadastrado com sucesso
  - **recuperarTodos()** - que retorna uma lista com todos os clientes cadastrados (apenas os clientes)
  - **recuperarPeloID(int id)**: que recebe um ID de cliente e retorna seus dados
- **ContaService** (para manipular os dados da conta)
  - **adicionarConta(Conta c)**: que recebe uma conta (inclusive com o ID do seu respectivo cliente e armazena no banco)
  - **recuperarPeloNumero(int numero)**: que recebe um número de conta e recupera seu conteúdo
  - **alterarDados(Conta c)**: que altera os dados da conta (será muito usada para a alteração do saldo)
  - **recuperarContasPeloCliente(int idCliente)**: que retorna uma lista de contas a partir do ID do cliente
- **MovimentacaoService** (para manipular as movimentações)
  - **cadastrarMovimentacao(Movimentacao m)** - para cadastrar
  - **recuperarTodas(int conta)** - para recuperar todas as movimentações de uma determinada conta

---

## DEFINIÇÕES DA API

Nesta seção falaremos de quais endpoints serão necessários para esta API e quais serviços devem ser chamados

### ClienteController

- `/clientes` (GET) - chama o serviço `recuperarTodos`
- `/clientes/{id}` (GET) - chama o serviço `recuperarPeloId` e pode retornar status 200 ou 404 se o cliente não existir
- `/clientes` (POST) - chama o serviço `cadastrarCliente` e pode retornar status 201 ou 400.

### ContaController

- `/contas/{id}` (GET) - chama o serviço `recuperarPeloNumero`, podendo retornar 200 ou 404
- `/contas/cliente/{id}` (GET) - chama o serviço `recuperarContasPeloCliente`, podendo retornar status 200 ou 404
- `/contas` (POST) - para cadastrar uma nova conta, chamando o serviço `adicionarConta`, podendo retornar 201 ou 400

\* o serviço de alteração de dados de conta não é acessível via URL, apenas através da inclusão de uma movimentação

### MovimentacaoController

- `/movimentacao` (POST) - chama o serviço `cadastrarMovimentacao`, podendo retornar 201 ou 400
- `/movimentacao/{id}` (GET) - chama o serviço `recuperarTodas`

---

## BONUS

- Incluir um serviço e um endpoint para **Transferencia de valores**
  - Serviço `TransferirValores(int contaOrigem, int ContaDestino, double valor)`
    - retorna boolean (se a operação for bem sucedida, em caso de saldo disponível na conta origem)
  - Endpoint `/transferencia` passando os seguintes parâmetros na URL  
(@RequestParam)
    - `contaOrigem`
    - `contaDestino`
    - `valor`