🏆 Desafio de DevTrail: Sistema de Banco Digital (BankDevTrail)

1. Objetivo

Desenvolver um sistema de simulação de banco digital (BankSim) que gerencie **Clientes** e suas **Contas**, realize operações financeiras e mantenha um histórico imutável (Ledger) de todas as transações, demonstrando o domínio dos conceitos de POO e modelagem de dados (relacionamentos).

2. Requisitos Funcionais Obrigatórios

2.1. Entidades Principais e Relacionamentos

Entidade	Descrição	Atributos Mínimos	Relacionamento
Cliente (Client)	Pessoa que possui uma ou mais contas no banco.	id (único), nome, cpf (único), ata_nascimednto.	**1 Cliente** pode ter **N Contas**
Conta (Account)	Conta bancária de um cliente.	numero (único), saldo (decimal), cliente_id (chave estrangeira), status (ativa/inativa), tipo (enum).	**1 Conta** pode ter **N Transações**
Transação (Transaction)	Movimentação financeira imutável.	id (único), tipo, valor, data_hora, conta_origem_id (FK), conta_destino_id (FK).	-

2.2. Operações Obrigatórias

O sistema deve permitir as seguintes operações através da camada de serviços:

1. **Cadastro de Cliente:** Registrar um novo cliente (CPF deve ser único).
2. **Abertura de Conta:** Abrir uma nova Conta para um Cliente existente.
3. **Depósito (deposit):** Adicionar valor a uma conta. Deve gerar uma Transação.
4. **Saque (withdraw):** Retirar valor, com validação de saldo. Deve gerar uma Transação.
5. **Transferência (transfer):** Transferir valor entre duas contas. Deve ser atômica (falha em uma, falha em ambas) e registrada como **duas Transações**.

6. **Consulta:**
    - Consultar **todas as contas** de um cliente (buscando pelo Cliente.id ou CPF).
    - Retornar o **Extrato** (todas as Transações) de uma Conta específica.

3. Requisitos Técnicos e Arquiteturais (Foco no Roadmap)

1. **Programação Orientada a Objetos (POO):**
    - Use classes e interfaces (Semana 2) para modelar as entidades e os serviços.
    - **Sugestão de Herança:** Crie uma classe base Conta e classes derivadas como ContaPoupanca ou ContaCorrente (Polimorfismo).
2. **Estruturas e Consultas (Semana 3):**
    - Use coleções (List, Dictionary) para gerenciar as contas (inicialmente, em memória).
    - Utilize **LINQ** para implementar as consultas, como buscar o cliente pelo CPF e filtrar as transações no extrato.
3. **Arquitetura Base (Semana 4 & 6):**
    - Implemente a lógica de negócio (saque, transferência) em uma **Camada de Serviço** dedicada (ex: BankService ou TransactionService).
    - Aplique o princípio da **Injeção de Dependência (DI)** para conectar a Camada de Serviço à camada que a utiliza API Controller.
4. **Tratamento de Exceções:** Implementar validações e tratamento de erros (saldo insuficiente, cliente não encontrado, etc.).

4. Ponto de Inovação (Opcional, mas Altamente Encorajado)

Inovação	Descrição
Taxas de Operação Variáveis	Adicionar uma regra de negócio que aplica uma taxa de 0.5% sobre o valor transferido, gerando uma nova Transação de débito na conta de origem (aplicação de lógica de negócio no Service Layer).

5. Estrutura de Entrega

O resultado final deve ser um projeto .NET C# que inclua:

1. **Camada de Domínio/Modelos:** Classes Client, Account, Transaction.
2. **Camada de Serviços:** Classe(s) com a lógica de negócio (DI - dependency injection, é obrigatória nesta camada).
3. **Aplicação Principal:** API Controller com Swagger que demonstra todas as operações obrigatórias.
4. **Camada de Infraestrutura:** camada de Dados