O Banco Digital da Ana

A Ana é fundadora de uma fintech chamada BankMore. Ela contratou uma equipe de desenvolvedores para criar uma plataforma baseada em microsserviços, com foco em escalabilidade e segurança.

Requisitos:
Ela quer que o sistema inicial tenha três funcionalidades principais:

- Cadastro e autenticação de usuários
- Realização de movimentações na conta corrente (depósitos e saques)
- Transferências entre contas da mesma instituição
- Consulta de saldo

Ana organizou uma série de reuniões com os times de Arquitetura, Segurança da Informação, Infraestrutura e produtos para alinhar os padrões e exigências técnicas da empresa e cada time trouxe pontos importantes:

Time de Arquitetura:

- Todos os microsserviços adotem os padrões de DDD (Domain-Driven Design)
- A arquitetura de cada serviço adote o Pattern CQRS (Command Query Responsibility Segregation)

Time de Segurança:

- Todas as APIs devem ser protegidas com autenticação via token (JWT). Nenhum endpoint pode ser acessado sem um token válido;
- Dados sensíveis como CPF ou número da conta não podem transitar entre os microsserviços ou ser armazenados fora do microsserviço de Usuário.

Time de Qualidade:

- Todas as APIs devem conter um projeto de testes automatizados.

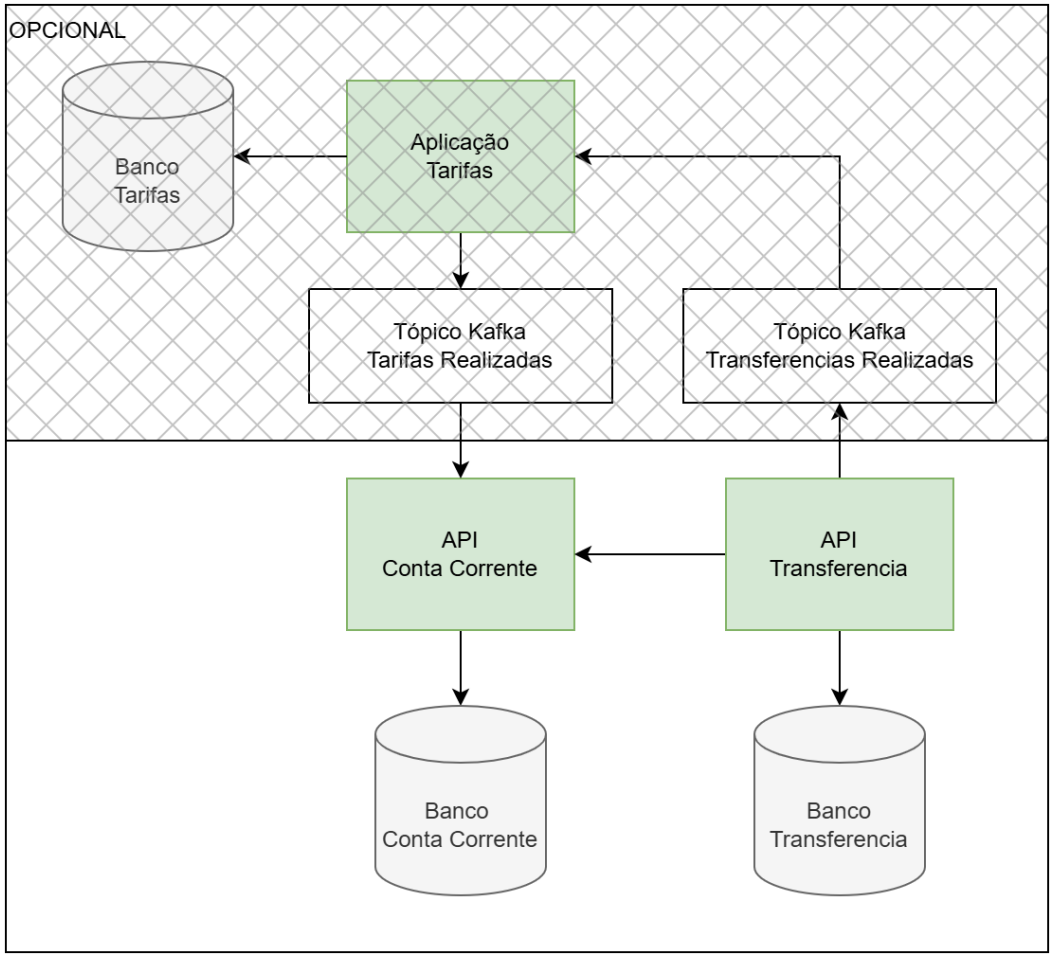
Time de Infraestrutura (Para produção):

- O sistema deve ser executado em ambiente de nuvem, com orquestração via Kubernetes.
- Cada microsserviço precisa rodar com pelo menos 2 réplicas (instâncias)

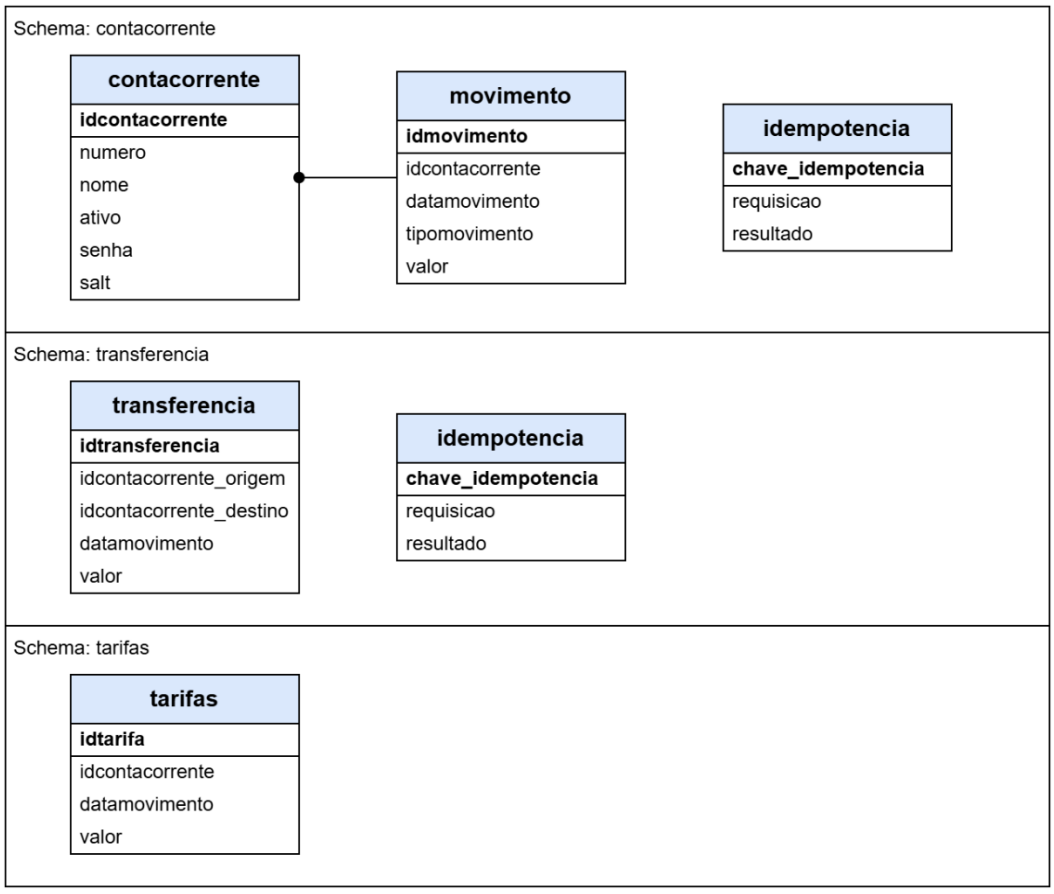
Time de Crédito:

- É importante que a API seja resiliente a falhas, pois o aplicativo que utiliza a API pode perder a conexão com a API antes de receber a resposta e então nestes casos o comportamento é repetir a mesma requisição até que o aplicativo receba um retorno. Por isso os serviços devem ser idempotentes.
- (OPCIONAL) Seria interessante se fosse possível implementar a API de Tarifas no sistema inicial.

Desenho de Arquitetura



Modelo ER



API Conta Corrente:

Cadastrar conta corrente deve:

- Receber o CPF do usuário e uma senha;
- Validar o CPF;
- Persistir na tabela CONTACORRENTE;
- Retornar o número da conta;
- Retornar HTTP 400 caso o CPF seja inválido. Neste caso deve ser retornado no body uma mensagem descritiva e o tipo de falha INVALID_DOCUMENT.

Efetuar Login deve:

- Receber o número da conta ou o CPF do usuário. Deve também receber a senha;
- Validar a senha com o registro no banco de dados;
- Retornar token JWT contendo a identificação da conta corrente.
- Retornar HTTP 401 em caso de erro. Neste caso deve ser retornado no body uma mensagem descritiva e o tipo de falha USER_UNAUTHORIZED.

Inativar conta corrente deve:

- Receber a senha da conta corrente;
- Receber o Token de autenticação no header da requisição;
- Retornar 403 caso o Token esteja inválido ou expirado;
- Validar:
    - Apenas contas correntes cadastradas podem receber movimentação; TIPO: INVALID_ACCOUNT;
    - Validar a senha com o registro no banco de dados;
- Persistir o campo ATIVO na tabela CONTACORRENTE para 0;
- Retornar HTTP 204 em caso de sucesso;

Movimentação na conta corrente deve:

- Receber a identificação da requisição, o número da conta corrente, o valor a ser movimentado, e o tipo de movimento (C = Crédito, D = Débito).
- Receber o Token de autenticação no header da requisição;
- Retornar 403 caso o Token esteja inválido ou expirado;
- O número da conta corrente é opcional e caso não receba esse campo, deve utilizar a identificação da conta que está no token.
- Validar:
    - Apenas contas correntes cadastradas podem receber movimentação; TIPO: INVALID_ACCOUNT;
    - Apenas contas correntes ativas podem receber movimentação; TIPO: INACTIVE_ACCOUNT;
    - Apenas valores positivos podem ser recebidos; TIPO: INVALID_VALUE;
    - Apenas os tipos “débito” ou “crédito” podem ser aceitos; TIPO: INVALID_TYPE;
    - Apenas o tipo “crédito” pode ser aceito caso o número da conta seja diferente do usuário logado; TIPO: INVALID_TYPE.
- Persistir na tabela MOVIMENTO;
- Retornar HTTP 204 em caso de sucesso;
- Retornar HTTP 400 caso os dados estejam inconsistentes. Neste caso deve ser retornado no body uma mensagem descritiva de qual foi a falha e o tipo de falha.

Saldo da conta corrente deve:

- Receber o Token de autenticação no header da requisição;
- Retornar HTTP 403 caso o Token esteja inválido ou expirado;
- Validar:
    - Apenas contas correntes cadastradas podem consultar o saldo; TIPO: INVALID_ACCOUNT.
    - Apenas contas correntes ativas podem consultar o saldo; TIPO: INACTIVE_ACCOUNT.
- Calcular o Saldo: Soma dos créditos menos a soma dos débitos;
- Retornar HTTP 200 com body:
    - Número da conta corrente
    - Nome do titular da conta corrente
    - Data e hora da resposta da consulta
    - Valor do Saldo atual
        - Retornar valor "0,00" caso não houver movimentações;
- Retornar HTTP 400 caso os dados estejam inconsistentes. Neste caso deve ser retornado no body uma mensagem descritiva de qual foi a falha e o tipo de falha.

API Transferência:

Efetuar transferência entre contas da mesma instituição deve:

- Receber a identificação da requisição, o número da conta de destino e o valor a ser transferido;
- Receber o Token de autenticação no header da requisição;
- Retornar 403 caso o Token esteja inválido ou expirado;
- Validar:
    - Apenas contas correntes cadastradas podem realizar transferências; **TIPO: INVALID_ACCOUNT**;
    - Apenas contas correntes ativas podem realizar transferências; **TIPO: INACTIVE_ACCOUNT**;
    - Apenas valores positivos podem ser recebidos: **TIPO: INVALID_VALUE**.
- Realizar chamada para a API Conta Corrente para realizar um débito;
    - Repassar o token;
    - Enviar a identificação da requisição e o valor a ser movimentado;
- Realizar chamada para a API Conta Corrente para realizar um crédito;
    - Repassar o token;
    - Enviar a identificação da requisição, o número da conta corrente de destino e o valor a ser movimentado;
    - Em caso de falhas deve ser feito um estorno na conta logada:
        - Realizar chamada para a API Conta Corrente para realizar um débito;
        - Repassar o token;
        - Enviar a identificação da requisição e o valor a ser movimentado.
- Persistir na tabela TRANSFERENCIA;
- Retornar HTTP 204 em caso de sucesso;
- Retornar HTTP 400 caso os dados estejam inconsistentes ou alguma das requisições falharem. Neste caso deve ser retornado no body uma mensagem descritiva de qual foi a falha e o tipo de falha.

OPCIONAL

Criar uma Aplicação para Tarifa. A aplicação deve:

- Consumir mensagens de um tópico Kafka de transferências realizadas.
- Ter parametrizado no arquivo "appsettings.json" da aplicação o valor da tarifa de transferências (Exemplo 2 reais).
- Ao consumir a mensagem Kafka, deve registrar no banco de dados que houve uma tarifação
    - Persistir:
        - A identificação da conta corrente
        - O valor tarifado
        - A data e hora do momento da tarifação
- Enviar para um tópico de tarifações realizadas.
    - Enviando no body:
        - A identificação da conta corrente
        - O valor tarifado

Neste caso deve ser adicionada mais uma etapa (após o sucesso das duas chamadas para conta corrente) na ***API Transferência*** para produzir uma mensagem neste tópico de ***transferências realizadas***, enviando a identificação da requisição e a identificação da conta corrente logada. Além disso, também será preciso implementar o consumo do tópico Kafka de ***tarifações realizadas*** na ***API Conta Corrente*** implementando o mesmo funcionamento do serviço ***movimentação na conta corrente*** (Sempre debitando o valor tarifado).

Algumas informações da Ailos:

As APIs da Ailos geralmente utilizam:

- **Dapper:** Componente para conexão com o banco de dados;

- **Mediator:** Padrão de projeto comportamental que permite que você reduza as dependências caóticas entre objetos;
- **Swagger:** Todos os serviços são documentados usando Swagger, todos os atributos são documentados, todos as requisições e retornos possíveis são documentados e com exemplos;
- **KafkaFlow:** Biblioteca dotnet para trabalhar com o Kafka;
- **Oracle:** Banco de dados padrão utilizado na empresa.
- **Dotnet:** Versão utilizada na Ailos é a 8.

Diferenciais desejados:

- Utilizar Kafka em alguma comunicação entre APIs trazendo alguma assincronicidade;
- Testes de integração além dos unitários;
- Utilizar cache;

Outras informações:

- Não é preciso entregar a solução em Kubernetes. A informação do Time de Infraestrutura é apenas informativa, mas deve ser levada em conta na maneira como a aplicação será desenvolvida;
- Entretanto, a solução deve ser entregue com os arquivos docker-compose.yaml prontos para podermos subir as aplicações em containers Docker. Um container para cada API, um para o kafka e um para o banco de dados;
- Recomendamos utilizar o SQLite. Pode ser preciso ajustar os scripts de banco;
- Alguns pontos, como criptografia e cache, foram omitidos propositalmente para avaliar as decisões do desenvolvedor durante a implementação;