Desafio Técnico

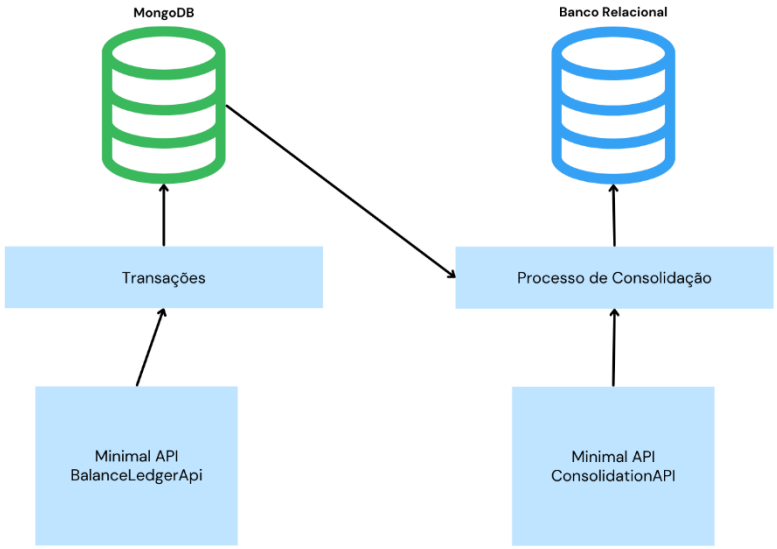
BalanceLedgerApi: https://github.com/gustavorborba/VerityBalanceLedgerApi

ConsolidationAPI: https://github.com/gustavorborba/VerityConsolidationApi

O projeto consiste em duas Minimal APIs escritas utilizando o framework .NET 9 e C#13. Se utilizando da arquitetura de microserviços, cada API tem uma responsabilidade e está desacoplada da outra.

BalanceLedgerApi -> responsável por processar requisições de transações salvando em um banco não relacional, no caso foi usado MongoDB.

ConsolidationApi -> responsável por realizar a consolidação das transações em batchs os salvando em um banco relacional.



Foi utilizado repository pattern em conjunto com uma camada de serviço, tendo assim o encapsulamento do acesso ao banco em repositórios e a centralização das regras de negócio dentro de serviços. Essa decisão vai além de apenas ter uma estrutura modular e organizada, com ela temos a separação de responsabilidades. Evitando códigos duplicados e mantendo a concentração das regras de negócio em um único lugar, facilitando a manutenção e evitando imprevisibilidade com regras espalhadas pelo código. E claro, a escrita de testes unitários também é facilitada.

Me atentei a manter o código o mais limpo na medida do possível, além de ser claro para entendimento de outros desenvolvedores, seguindo KISS e princípios de boas práticas. Organizando o código e seguindo SOLID sempre que faz sentido.

Redundância e Paralelismo

Os sistemas foram pensados para terem resistência a falhas até um certo nível, porém entendo que ainda caberia aprofundamento nesse sentido.

Ambas APIs funcionam de forma assíncrona. A Consolidation exclusivamente possuí maior tratamento para execuções paralelas. Para o processamento dos dados, foi

pensado em execuções através de pacotes de tamanho parametrizável, evitando grandes alocações em memória, esses pacotes deixei configurado para serem processados até 5 por vez para evitar sobrecargas com múltiplas tasks.

Para ambientes finais

BalanceLedgerApi: pode ser escalada horizontalmente, tendo mais instancias, seja em nuvem ou máquinas virtuais. Isso implicaria também em ter um Load Balancer para distribuição das requisições. Pensando é claro, em um fluxo alto de dados.

ConsolidationApi: O MongoDB possui o sharding para escalonamento horizontal. Creio que a integração com banco relacional não seja um problema aqui.

CI/CD para ambas APIs, configuração de pipelines para garantir a qualidade e robustez. Poderia ser pensado steps com sonarqube, análise de vulnerabilidades, execução de testes e utilização de algum serviço para orquestrar a escalabilidade e o balanceamento.

Escolhas tecnológicas:

MongoDB: Foi escolhido pela possibilidade de leitura e escrita visando alto desempenho tanto para salvar as transações, quanto para o processamento desses dados, permitindo também alta escalabilidade. Se tratando também de dados financeiros posso configurar facilmente replicas dele.

Nesse caso aqui também avaliei usar Redis, ele me ofereceria algumas funcionalidades interessantes, porém, pensando na segurança de salvar transações financeiras, escalabilidade para o desafio e em apenas processar os dados no final do dia, na minha visão técnica, MongoDB acaba tendo vantagem. Sem contar que, com redis eu teria que em algum momento persistir esses dados num banco relacional ou não.

SQLite: Esse foi usado única e exclusivamente para simular uso de banco relacional. De forma fácil e simples pude criar uma "POC" sem grandes dependências. Escolheria um banco relacional pela integridade e consistência dos dados consolidados.

Entity Core: Por não usar grandes estruturas de dados e não efetuar consultas complexas. Garante facilidade e eficiência. Em outros casos, poderia avaliar o uso de Dapper ou ADO pensando em alta performance e/ou execução de querys mais pesadas.

Quartz: Me permite configurar Cron Jobs sem grandes esforços. Em um ambiente de produção eu avaliaria outras opções, Hangfire, ou principalmente algo com cloud, usando por exemplo AWS EventBridge ou até algo usando Kubernates.

TODOs, pendencias:

Melhor desenvolvimento de segurança. A autorização feita cabe ainda aprofundamento com controle de usuários, criptografia de senhas e melhor controle entre as Apis.

Cobertura maior de testes unitários e de integração.

Redundância maior para falhas. O Job irá executar novamente caso alguma falha aconteça, porém acredito que poderia haver outros controles para ele e além dele.

Controle de status para saber o que foi consolidado ou não.

Validação para entrada de dados. Acredito que um validator caberia bem.

Tratamento adequado para as CommonResponses, hoje os métodos ficaram com a responsabilidade de tratar exceptions e success, a ideia era centralizar e evitar isso.

Unificar execuções via docker