Documentação do Sistema

de Fluxo de Caixa

Sumário

[Mapeamento de Domínio 3](#_Toc183074287)

[Domínio Funcional 3](#_Toc183074288)

[Requisitos Funcionais 4](#_Toc183074289)

[Requisitos Não Funcionais 5](#_Toc183074290)

[Justificativa das Ferramentas e Tecnologias 6](#_Toc183074291)

[Desenho da Arquitetura da Solução 7](#_Toc183074292)

## Mapeamento de Domínio

* **Registrar Lançamentos Financeiros:** Permite o registro de transações financeiras, com informações como tipo (crédito ou débito), valor, data e descrição.
* **Consultar Lançamentos Financeiros:** Permite a recuperação dos lançamentos registrados, com suas informações detalhadas.
* **Remover Lançamentos Financeiros:** Possibilita a exclusão de lançamentos existentes, identificados pelo seu ID.
* **Consultar Consolidado Diário:** Oferece um resumo diário das transações, apresentando a data e o saldo do dia.

## Domínio Funcional

**Capacidades de Negócio:**

* **Cadastro de Lançamentos:** A classe LancamentoBusiness possui o método AddLancamentoAsync, que permite adicionar novos lançamentos financeiros ao sistema. O método valida se o ID do lançamento foi fornecido, gerando um novo se necessário. As informações necessárias para um lançamento são geridas pelo Lancamento (tipo, valor, data e descrição).
* **Consulta de Lançamentos:** O método GetAllLancamentosAsync permite recuperar todos os lançamentos registrados. Há também um método GetLancamentoByIdAsync para consultar um lançamento específico pelo seu ID.
* **Remoção de Lançamentos:** A capacidade de remover lançamentos é implementada pelo método RemoveLancamentoAsync. Este método busca o lançamento pelo ID e o remove do repositório. Trata também o caso em que o lançamento não é encontrado, lançando uma exceção.
* **Relatório Consolidado Diário:** O método GetConsolidadoDiarioAsync sugere a geração de um relatório consolidado diário, possivelmente agregando os lançamentos por dia para calcular saldos.

**Mapeamento Detalhado:**

|  |  |  |
| --- | --- | --- |
| **Capacidade de Negócio** | **Método na LancamentoBusiness** | **Observações** |
| Cadastro de Lançamentos | AddLancamentoAsync | Validação básica de ID, geração de ID se não informado. |
| Consulta de Lançamentos | GetAllLancamentosAsync, GetLancamentoByIdAsync | Permite consulta completa ou por ID. |
| Remoção de Lançamentos | RemoveLancamentoAsync | Lança exceção se o lançamento não existir. |
| Relatório Consolidado Diário | GetConsolidadoDiarioAsync | A complexidade do relatório não está visível no código. |

## Requisitos Funcionais

Os requisitos funcionais descrevem o que o sistema deve fazer. Baseando-se nos arquivos fornecidos, temos:

* **Cadastro de Lançamentos:** O sistema deve permitir o cadastro de novos lançamentos financeiros, incluindo tipo (crédito ou débito), valor, data e descrição. O sistema deve gerar um ID único para cada lançamento.
* **Consulta de Lançamentos:** O sistema deve permitir a consulta de lançamentos financeiros, possibilitando a busca de todos os lançamentos ou de um lançamento específico por meio do seu ID.
* **Remoção de Lançamentos:** O sistema deve permitir a remoção de lançamentos financeiros existentes, identificados por seu ID. O sistema deve tratar o caso em que o lançamento não for encontrado, retornando uma mensagem de erro apropriada.
* **Relatório Consolidado Diário:** O sistema deve gerar um relatório consolidado diário, mostrando o saldo para cada dia. A forma como o saldo é calculado (somatório dos valores de cada lançamento do dia, considerando tipo crédito/débito) precisa ser definida mais precisamente.
* **Validação de Dados:** O sistema deve validar os dados de entrada para garantir a integridade dos dados. Isto inclui (mas não se limita a): valor numérico válido, data válida, e tipo de lançamento válido ('C' ou 'D', por exemplo). Este requisito não está explícito, mas é implicitamente necessário.

## Requisitos Não Funcionais

1. **Dois Serviços Independentes:**

A solução mais direta e robusta é separar completamente os serviços de controle de lançamentos e o serviço de consolidado diário em dois serviços distintos. Isso garante que a falha de um não afete o outro.

* **Serviço de** **Lançamentos:** Responsável por todas as operações CRUD (Criar, Ler, Atualizar, Deletar) de lançamentos financeiros.
* **Serviço de Consolidado Diário:** Responsável por gerar e fornecer o consolidado diário. Este serviço pode ser executado como um processo separado (ex: agendado ou com base em eventos) e pode ser mais tolerante a falhas.

1. **Garantir o Desempenho do Serviço de Lançamentos:**

A meta de desempenho (50 requisições por segundo com no máximo 5% de perda) exige otimização:

* **Escalabilidade:** O serviço de consolidado diário precisa ser escalável, permitindo adicionar mais instâncias para atender a demanda em picos. O uso de containers (Docker) e orquestração (Kubernetes) facilita a escalabilidade horizontal.
* **Cache:** Implementar um cache (ex: Redis) para armazenar o consolidado diário. Isso reduz a carga no banco de dados e melhora o tempo de resposta.
* **Monitoramento:** Implementar monitoramento detalhado para identificar gargalos de desempenho e garantir que a meta de 5% de perda de requisições seja atendida.

## Justificativa das Ferramentas e Tecnologias

* **.NET:** A escolha do .NET como framework foi por conta de ser uma plataforma robusta, madura, com ampla comunidade e suporte para desenvolvimento de APIs. O .NET oferece um bom desempenho e integração com diversas ferramentas.
* **GraphQL:** O uso do GraphQL para a API sugere a necessidade de uma forma eficiente e flexível de expor dados. GraphQL permite aos clientes solicitar exatamente os dados que precisam, evitando o *overfetching* (receber mais dados do que o necessário) ou *underfetching* (ter que fazer múltiplas requisições para obter todos os dados). Isso contribui para um melhor desempenho e otimização de banda.
* **Entity Framework Core (EF Core):** O EF Core simplifica o acesso e manipulação de dados no banco de dados relacional, reduzindo a quantidade de código e abstraindo a interação com o banco de dados.
* **PostgreSQL:** A utilização do PostgreSQL como banco de dados indica a necessidade de um banco de dados robusto, escalável e com suporte a funcionalidades avançadas.
* **Docker:** O uso do Docker para a execução do PostgreSQL e da API facilita o desenvolvimento, teste e deploy do projeto. Docker garante consistência do ambiente e simplifica o processo de implantação em diferentes plataformas.

## Desenho da Arquitetura da Solução

Diagrama

Descrição gerada automaticamente