**README - CashFlowTracker**

**Tecnologias Utilizadas**

* **.NET na versão X**: Escolhido pela robustez e suporte extensivo para aplicações empresariais.
* **SQL Server**: Utilizado como banco de dados principal devido à sua confiabilidade e integração eficiente com o .NET.
* **RabbitMQ**: Adotado para gerenciamento de filas de processamento de relatórios, garantindo a entrega confiável de mensagens e suporte a picos de tráfego.

**Desenho da Solução/Arquitetura**

A aplicação CashFlowTracker é estruturada seguindo a Clean Architecture, facilitando a manutenção e a testabilidade. Implementamos o padrão CQRS (Command Query Responsibility Segregation) utilizando MediatR para separar as operações de leitura e escrita, garantindo maior clareza e eficiência.

**Executando o Projeto**

Para executar o projeto, use o seguinte comando:

**docker-compose up --build**

Isto irá iniciar todos os serviços necessários, incluindo a API, workers e bancos de dados.

**Considerações Finais/Observações**

* **Escalabilidade**: A arquitetura foi projetada para escalar horizontalmente, permitindo lidar com aumentos de carga.
* **Manutenção**: O uso de Clean Architecture e princípios SOLID facilita a manutenção e futuras expansões do sistema.

**Pontos Importantes**

**Controle de Lançamentos e Serviço do Consolidado Diário**

* **Controle de Lançamentos**: O sistema processa transações financeiras de forma assíncrona, garantindo alta disponibilidade.
* **Serviço de Consolidado Diário**: Implementado para gerar relatórios diários, mesmo sob alta carga.

**Estrutura do Projeto**

* **Arquitetura Limpa e Ports and Adapters**: Facilita testes e manutenção, promove desacoplamento e flexibilidade.

**Tecnologias Adicionais**

* **FluentValidation**: Para validação de dados.
* **xUnit**: Utilizado para testes unitários.

**Execução da Aplicação**

* **Docker Compose**: Simplifica a execução e garante consistência entre ambientes.
* **Postman Collection**: Disponível para testar a API.

**Casos de Uso da Aplicação**

* **Transações Financeiras**: Registros e relatórios diários.
* **Resiliência**: O sistema continua operacional mesmo com falhas parciais.

**Observações Finais**

* **Segurança**: Importante verificar autorizações e autenticações em fases futuras.
* **Arquiteturas Alternativas**: Microserviços poderiam ser explorados para casos de uso específicos.