# **Documento de Arquitetura de Microserviços: Controle de Fluxo de Caixa**

### 1. Introdução

Este documento descreve a arquitetura proposta para o sistema de Controle de Fluxo de Caixa utilizando microserviços. A solução foi desenvolvida com base nos requisitos funcionais e não funcionais apresentados, utilizando tecnologias .NET para os serviços, React com Tailwind CSS para o frontend, e Docker para facilitar a implantação.

### 2. Visão Geral

### 2.1 Design de Arquitetura

### Diagrama Descrição gerada automaticamente

* **Tipo de Arquitetura**: Microserviços (simples)
* **Containers**: Docker
* **Mensageria**: RabbitMQ
* **Serviços**: .NET Core
* **Banco de Dados**: PostgreSQL
* **Frontend**: React.JS + Tailwind CSS
* **Escalabilidade**: Horizontal

## 2.2. Microserviços

## **Serviço de Controle de Lançamentos:**

* Responsável pelo registro de débitos e créditos.
* Utiliza ASP.NET Core e Entity Framework para interação com o banco de dados.
* Dockerizado para facilitar a implantação.

**Serviço de Consolidado Diário:**

* Calcula o saldo diário consolidado com base nos lançamentos.
* Utiliza cache para otimizar consultas.
* Dockerizado para fácil implementação.

## 2.2. Frontend

* Desenvolvido em React com Tailwind CSS.
* Consumirá os serviços de microserviços.
* Implementação de testes de unidade e integração.

## 2.3. Infraestrutura

* Banco de Dados: Microsoft SQL Server para armazenamento dos lançamentos.
* Message Broker: RabbitMQ para comunicação assíncrona entre serviços.
* Orquestração: Kubernetes para gerenciamento e orquestração de contêineres.
* Balanceamento de Carga: Utilização de um serviço para distribuir o tráfego.

## 2.4. Ferramentas Adicionais

* Logging: ELK Stack (Elasticsearch, Logstash, Kibana).
* Monitoramento: Application Insights para métricas de desempenho.

### 3. Detalhes Técnicos

## 3.1.Serviço de Controle de Lançamentos

* 1. Endpoint CRUD para lançamentos: **/api/lancamentos**
  2. Banco de dados: Entity Framework com Microsoft SQL Server.

## 3.2. Serviço de Consolidado Diário

* 1. Endpoint para saldo consolidado: **/api/consolidado**
  2. Cache para otimização de consultas.
  3. Utilização de RabbitMQ para comunicação assíncrona.

## 3.3. Frontend

* 1. Implementação em React com Tailwind CSS para o design.
  2. Consumo de APIs dos microserviços.
  3. Testes unitários e de integração com Jest e React Testing Library.

## 4. Segurança

* Implementação de autenticação e autorização usando IdentityServer ou solução equivalente.
* Comunicação segura utilizando HTTPS.

### 5. Docker e Orquestração

* Dockerização de todos os serviços.
* Utilização do Docker Compose para facilitar a execução multi-container.
* Implantação e orquestração no Kubernetes.

### 6. Monitoramento e Logging

* Logging centralizado com ELK Stack.
* Monitoramento de métricas de desempenho com Application Insights.

### 7. Readme e Documentação

* Instruções detalhadas para configuração, construção e execução local.
* Requisitos de sistema, dependências e variáveis de ambiente documentadas.

### 8. GitHub

* Código-fonte hospedado em repositório público no GitHub.
* Informações sobre contribuições, problemas e solicitações de recursos disponíveis.

### 9. Considerações Finais

Esta arquitetura foi projetada para atender aos requisitos apresentados, proporcionando uma solução robusta, escalável e de fácil manutenção. Certifique-se de seguir as instruções fornecidas no readme para configurar e executar o sistema localmente. Qualquer dúvida ou sugestão pode ser encaminhada por meio do sistema de issues no GitHub.