Sumário

**Introdução1**

Visão Geral do Projeto2

Objetivo do Documento3

**Justificativa da Arquitetura**3

Razões para a Escolha3

Benefícios4

**Padrões e Princípios de Design**5

**Tecnologias e Ferramentas Utilizadas**7

**Melhorias Futuras4**

**1 . Introdução**

**Visão Geral do Projeto**

Desenvolver uma arquitetura de software escalável e resiliente, garantindo alta disponibilidade, segurança e desempenho. Isso inclui a seleção adequada de padrões arquiteturais, integração de tecnologias e frameworks, além de otimização de requisitos não-funcionais. Deve abranger aspectos importantes, como design, integração, segurança e desempenho.

* **Escalabilidade:** Garanta que a arquitetura possa lidar com o aumento da carga de trabalho sem degradação significativa do desempenho. Considere dimensionamento horizontal, balanceamento de carga e estratégias de cache. Resiliência: Projete para a recuperação de falhas. Isso inclui redundância, failover, monitoramento proativo e estratégias de recuperação.
* **Segurança:** Proteja os dados e sistemas contra ameaças. Implemente autenticação, autorização, criptografia e mecanismos de proteção contra ataques.
* **Padrões Arquiteturais:** Escolha padrões adequados, como microsserviços, monolitos, SOA ou serverless. Considere trade-offs entre simplicidade e flexibilidade.
* **Integração:** Defina como os componentes se comunicarão. Avalie protocolos, formatos de mensagem e ferramentas de integração.
* **Requisitos Não-Funcionais:** Otimize para desempenho, disponibilidade e confiabilidade. Defina métricas e metas claras.
* **Documentação:** Registre decisões arquiteturais, diagramas e fluxos de dados. Isso facilita a comunicação e a manutenção

**Objetivo do Documento**

Decidi construir o sistema **CashFlowApp** em uma **arquitetura monolítica** utilizando .NET 8 e orientado por princípios como SOLID e Domain-DrivenDesign (DDD), com um banco de dados MongoDB, além de integrar tecnologias de Redis Cache, Circuit Breaker e Retry Policies.

Essa abordagem foi cuidadosamente selecionada para atender a exigências como escalabilidade, resiliência, segurança e desempenho, considerando os desafios específicos do projeto e a necessidade de otimizar requisitos não funcionais de maneira eficiente e econômica.

O documento proposto visa esclarecer de modo decisivo a tomada de decisão e justificar sua ação. Mostrando o fluxograma e a explicação de cada conceito e tecnologia empregada no sistema e seus benefícios para atingir o objetivo do desafio.

**2. Justificativa da Arquitetura**

**Razões para a Escolha**

A decisão de optar por uma arquitetura monolítica para o CashFlowApp, ao invés de uma abordagem baseada em microsserviços, foi baseada em fatores de simplicidade, custo e controle no estágio inicial do projeto. A arquitetura monolítica permite consolidar todas as funcionalidades do sistema em uma única aplicação, o que facilita o desenvolvimento inicial, reduzindo a necessidade de configuração complexa de múltiplos serviços e orquestração. Em um projeto inicial, onde o volume de usuários e a carga de trabalho são ainda previsíveis e limitados, essa escolha ajuda a concentrar os esforços em funcionalidades e requisitos específicos, sem a sobrecarga de gerenciar a complexidade de uma estrutura distribuída.

Em termos de custos, a manutenção de um sistema monolítico exige menor infraestrutura, pois requer apenas um único ambiente de execução, além de simplificar o processo de desenvolvimento e testes, onde todos os componentes residem no mesmo sistema. Por fim, a centralização do controle possibilita maior facilidade para a implementação de políticas de segurança, integração de módulos e gerenciamento de atualizações, o que é vantajoso para um projeto em fase de crescimento e amadurecimento.

**Benefícios da Arquitetura Monolítica**

A arquitetura monolítica traz benefícios significativos em termos de simplicidade de desenvolvimento, manutenção e escalabilidade em seu estágio inicial. Com uma única base de código, os desenvolvedores podem trabalhar de forma mais integrada, evitando a sobrecarga de gerenciar dependências interserviços ou lidar com comunicações entre diferentes microsserviços. Isso facilita a implementação de novas funcionalidades e a correção de bugs, uma vez que todas as partes do sistema estão centralizadas e acessíveis em um único projeto.

Além disso, a arquitetura monolítica favorece a manutenção e a eficiência dos testes, já que as interações e dependências estão concentradas em uma aplicação única. Em relação à escalabilidade, um sistema monolítico permite a implementação de estratégias de escalabilidade vertical, aumentando a capacidade dos servidores conforme necessário.

Embora a escalabilidade horizontal seja mais associada a microsserviços, o uso de cache, como o Redis, pode mitigar a carga sobre o banco de dados, oferecendo desempenho otimizado sem comprometer a simplicidade estrutural do sistema monolítico.

Isso torna a arquitetura monolítica uma opção prática e eficiente, especialmente durante as fases iniciais e intermediárias do ciclo de vida do CashFlowApp.

**3. Padrões e Princípios de Design**

|  |
| --- |
| Representação CashFlowApp |

Representação CashFlowApp

**SOLID e DDD**

A aplicação dos princípios SOLID e do Domain-Driven Design (DDD) no CashFlowApp proporciona uma estrutura de código altamente modular e orientada a objetos, melhorando significativamente a manutenção e a extensibilidade do sistema.

* **Princípios SOLID**: Esses cinco princípios (Responsabilidade Única, Aberto/Fechado, Substituição de Liskov, Segregação de Interface, e Inversão de Dependência) guiam o design de classes e interfaces, tornando o código mais modular, flexível e de fácil manutenção. A responsabilidade de cada classe é limitada ao seu domínio específico, o que facilita a adição de novas funcionalidades sem a necessidade de alterar o código existente. Por exemplo, o princípio de Inversão de Dependência (D) permite a injeção de dependências, facilitando o teste de unidades e a substituição de implementações.
* **Domain-Driven Design (DDD)**: Com o DDD, a aplicação é dividida em domínios e subdomínios claramente definidos que correspondem às necessidades do negócio. No CashFlowApp, conceitos como “Transação” e “Saldo Diário” estão organizados em domínios que refletem as operações e fluxos financeiros da aplicação. Essa abordagem fortalece a organização do código e melhora a comunicação entre as equipes de desenvolvimento e de negócios, pois todos utilizam a mesma linguagem e compreensão do domínio. O DDD também traz clareza ao separar as responsabilidades de negócio (Domínio) das de infraestrutura e persistência (Infraestrutura).

**Organização por Camadas**

A organização do CashFlowApp é estruturada em três principais camadas: Domínio, Aplicação e Infraestrutura, cada uma com responsabilidades bem definidas que promovem a separação de preocupações, a manutenibilidade e a escalabilidade.

* **Camada de Domínio**: Contém as regras de negócio e entidades principais do sistema. É onde o comportamento dos objetos de domínio é definido, aplicando-se os conceitos do DDD, como agregados, entidades e serviços de domínio. Essa camada é independente de frameworks e tecnologias de persistência, o que facilita a evolução das regras de negócio sem impacto direto nas outras camadas.
* **Camada de Aplicação**: Atua como intermediária entre o Domínio e a Infraestrutura, gerenciando fluxos de uso e orquestrando operações complexas entre diferentes domínios. A camada de Aplicação implementa serviços de aplicação que não contêm lógica de negócio em si, mas coordenam chamadas para as entidades e serviços do domínio, e aplicam políticas de segurança e validação de dados, encapsulando a complexidade do sistema para os usuários e clientes externos.
* **Camada de Infraestrutura**: Responsável pela persistência dos dados (como integração com MongoDB), comunicação com APIs externas e implementação de políticas de resiliência, como Cache Redis, Circuit Breaker e Retry Policies. Nesta camada, são implementados os detalhes de persistência e tecnologia, que podem ser modificados ou substituídos sem afetar a lógica de negócio da camada de Domínio.
* Essa arquitetura em camadas permite que cada parte do sistema evolua de maneira independente, com responsabilidades bem delimitadas que facilitam a manutenção e a expansão do sistema à medida que novas funcionalidades ou tecnologias são introduzidas.

**4. Tecnologias e Ferramentas Utilizadas**

**.NET 8**

A escolha do .NET 8 para o desenvolvimento do CashFlowApp traz diversas vantagens, especialmente em termos de performance, segurança e atualizações contínuas. .NET 8 oferece melhorias substanciais na execução de código, resultando em um desempenho otimizado, com menores tempos de resposta e maior eficiência no uso de recursos do sistema. Esse desempenho superior é crucial para um sistema que requer alta responsividade e é submetido a grande volume de transações financeiras. Além disso, o .NET 8 inclui atualizações robustas de segurança, com recursos aprimorados para gerenciamento de autenticação e proteção contra ameaças, o que ajuda a proteger dados financeiros sensíveis. A escolha do .NET 8 também garante acesso a uma comunidade ativa e suporte de longo prazo, essencial para a manutenção e atualização contínua do CashFlowApp.

**MongoDB**

Optamos pelo MongoDB como banco de dados NoSQL por sua flexibilidade em armazenar dados em documentos JSON/BSON. Esse tipo de banco permite uma estrutura de dados mais flexível e facilmente ajustável, adaptando-se ao crescimento e à evolução do modelo de dados sem grandes refatorações. A estrutura não-relacional do MongoDB suporta a escalabilidade horizontal, permitindo o aumento de capacidade de armazenamento e processamento com a adição de novos servidores. Para o CashFlowApp, que lida com dados financeiros em crescimento constante, a escalabilidade do MongoDB facilita a expansão, enquanto a flexibilidade da estrutura de dados permite atender a novos requisitos e alterações de maneira ágil.

**Redis Cache**

O Redis Cache foi escolhido para armazenar em cache informações frequentemente acessadas, o que ajuda a reduzir a carga no banco de dados MongoDB e melhora a resposta do sistema em operações intensivas. A introdução do Redis diminui o número de consultas repetidas ao banco de dados, proporcionando maior eficiência no processamento e menor latência no acesso a dados críticos. Esse mecanismo é especialmente útil para consultas em tempo real, garantindo que informações acessadas com frequência, como saldos diários ou transações recentes, sejam recuperadas rapidamente, otimizando a experiência do usuário e a eficiência do CashFlowApp.

**Resiliência e Disponibilidade**

* **Circuit Breaker e Retry Policies**: Implementamos esses padrões para aumentar a resiliência do sistema em casos de falhas temporárias, como problemas de conectividade com o banco de dados ou serviços externos. O Circuit Breaker interrompe temporariamente chamadas para um serviço em falha, protegendo o sistema de sobrecarga. Ao mesmo tempo, as Retry Policies permitem a repetição de tentativas de conexão, ajudando a evitar falhas em chamadas que poderiam ser bem-sucedidas com uma nova tentativa. Esses padrões minimizam interrupções no funcionamento do CashFlowApp e garantem que ele consiga se recuperar rapidamente de falhas.
* **Mecanismos de Recuperação e Failover**: Para garantir a continuidade do sistema em situações de falha, foram implementados mecanismos de failover e recuperação, que redirecionam as operações para instâncias alternativas ou backups. Essas técnicas de recuperação, combinadas com monitoramento ativo e alertas, asseguram que o CashFlowApp continue operando com o mínimo de interrupções e que qualquer falha seja identificada e solucionada rapidamente. A capacidade de recuperação rápida é vital para manter a confiabilidade e a disponibilidade, especialmente em um sistema financeiro que demanda alta integridade e acessibilidade dos dados.

**Ferramentas Utilizadas**

O sistema pode ser executando utilizando **Visual Studio 2022** ou caso prefira pode utilizar Postman. Para o Redis foi utilizado o **Redis Cloud** (<https://cloud.redis.io>) na criação do database e para executar e visualizar o conteúdo do database foi usado o **Redis Insight**

5. Melhorias futuras

O CashFlowApp está projetado para crescer e evoluir, com uma arquitetura que permite futuras melhorias para atender a requisitos de segurança, escalabilidade e extensibilidade. Entre as evoluções planejadas, a implementação de um sistema de autenticação robusto está em destaque, com suporte para tokens JWT, autenticação de dois fatores e controle de acesso baseado em perfis de usuário. Esses recursos proporcionarão uma camada adicional de segurança, garantindo que o acesso ao sistema seja seguro e personalizado conforme as permissões do usuário.

Além disso, a introdução de novas abstrações com o uso de interfaces tornará o sistema mais flexível, facilitando a adição de novas funcionalidades e a integração de serviços externos. Isso permite que futuras features sejam incorporadas sem alterar o núcleo da aplicação, reforçando o compromisso com os princípios SOLID e promovendo uma arquitetura de fácil manutenção.

Para atender a requisitos de escalabilidade e processamento de eventos em tempo real, também está prevista a implementação de mensageria com RabbitMQ. Esse mecanismo permitirá o controle de eventos e a integração de serviços assíncronos, aumentando a eficiência e a resiliência do sistema à medida que cresce. Essas melhorias futuras reforçam o compromisso do CashFlowApp em ser uma solução completa, segura e adaptável para as necessidades de gestão financeira de seus usuários.