**INSTITUTO FEDERAL DE EDUCAÇÃO,
CIÊNCIA E TECNOLOGIA DA BAHIA
Campus Santo Antônio de Jesus**

TRABALHO DE CONCLUSÃO DE CURSO

PROPOSTA DE ARQUITETURA DE MICROSSERVIÇOS PARA UM SISTEMA ERP CONTÁBIL

Lucas Oliveira Sampaio Silva

**Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas**

**Santo Antônio de Jesus - Bahia
20 de julho de 2023**

IFBA/SAJ-ADS-2023

LUCAS OLIVEIRA SAMPAIO SILVA

PROPOSTA DE ARQUITETURA DE MICROSSERVIÇOS PARA UM SISTEMA ERP CONTÁBIL

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia da Bahia, campus Santo Antônio de Jesus, como requisito parcial para obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Msc. Leandro Costa Souza.

Santo Antônio de Jesus - Bahia
20 de julho de 2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

S586r Silva, lucas Oliveira Sampaio

Proposta de arquitetura de microsserviços para um sistema ERP contábil. / lucas Oliveira Sampaio Silva. – Santo Antônio de Jesus, 2023.
70 f.: il. algumas color.

Orientador: Prof. Msc. Leandro Costa Souza

Monografia (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia da Bahia. Campus Santo Antônio de Jesus, 2023.

1. Engenharia de Software 2. Arquitetura de Software. 3. Microsserviços. 4. Santo Antônio de Jesus (BA). I. Souza, Leandro Costa (Orientador). II. Instituto Federal da Bahia. III. Título.

CDU 004.41

Elaborado por Alessandro Santana de Souza – CRB-5/1512
Sistema Integrado de Bibliotecas – Instituto Federal da Bahia (SIB-IFBA)
Biblioteca IFBA Campus Santo Amaro

LUCAS OLIVEIRA SAMPAIO SILVA

PROPOSTA DE ARQUITETURA DE MICROSSERVIÇOS PARA UM SISTEMA ERP CONTÁBIL

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas e Aprovado em sua forma final pelo Instituto Federal da Bahia, *campus* Santo Antônio de Jesus.

Santo Antônio de Jesus – BA, 08, de agosto de 2023.

_____

Prof. Dr. Cleber Jorge Lira de Santana
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

_____

Prof. Msc. Leandro Costa Souza
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

_____

Prof. Esp. Yuri Silva Bispo
Instituto Federal de Educação, Ciência e Tecnologia da Bahia

Dedicatória

Dedico este trabalho a toda a minha família, minha base e minha fonte de motivação para continuar. Vocês estiveram sempre ao meu lado, apoiando e incentivando cada passo desta jornada acadêmica. Agradeço por todo o amor, apoio incondicional e sacrifícios que fizeram por mim. Este trabalho é dedicado a vocês, que são o alicerce da minha vida.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, minha fonte de sabedoria e inspiração, por guiar meus passos e iluminar meu caminho ao longo desta jornada acadêmica.

Aos meus pais, cujo amor, apoio incondicional e sacrifícios ao longo de todos esses anos fizeram com que eu me tornasse a pessoa que sou hoje. Suas palavras de encorajamento e incentivo foram fundamentais para que eu pudesse enfrentar todos os desafios e obstáculos ao longo deste trabalho.

À minha querida avó Conça, minha gratidão não tem fim. Sua crença em mim, o incentivo constante, os cuidados que sempre me proporcionou e sua paciência para esperar por mim ao final de cada aula noturna foram um verdadeiro porto seguro.

À minha namorada Lorena, por estar ao meu lado, compartilhando os altos e baixos desta jornada acadêmica, e por seu amor, paciência e compreensão. Seu apoio constante e motivação foram fundamentais para que eu não desistisse e persistisse em busca dos meus objetivos.

Aos meus colegas de curso, que se tornaram amigos ao longo desses anos, agradeço pela parceria, colaboração, troca de conhecimentos e momentos de descontração que partilhamos. Vocês foram uma fonte de motivação e apoio durante todo o processo.

Aos meus estimados professores, que com dedicação, paciência e conhecimento transmitiram ensinamentos valiosos ao longo desta jornada acadêmica, agradeço por sua orientação, feedback e por despertarem em mim o desejo constante de aprender mais.

Em especial, agradeço ao professor Ikaro Campos, que no início de tudo se tornou minha grande inspiração para abraçar as oportunidades de aprendizado e os desafios na área de desenvolvimento de software. Agradeço por cada conselho e ensinamento.

De igual modo, sou grato ao professor Leandro Costa pela paciência demonstrada ao me ensinar e pelas conversas estimulantes que sempre me incentivaram a explorar mais profundamente os conceitos tecnológicos.

Agradeço a todos que estiveram presentes em minha vida, seja de forma direta ou indireta, e que contribuíram imensamente para o meu crescimento pessoal e acadêmico.

RESUMO

Sampaio, Lucas Oliveira. **Proposta de Arquitetura de Microsserviços Para Um Sistema ERP Contábil.** Orientador: Prof. Msc. Leandro Costa Souza. 2023. PG f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação Ciência e Tecnologia da Bahia, Santo Antônio de Jesus – BA, 2023.

Este estudo apresenta uma proposta de arquitetura de microsserviços para um sistema ERP (Enterprise Resource Planning), voltado para a contabilidade de pequenas empresas. A contabilidade, quando aliada a um sistema ERP, pode proporcionar um conhecimento aprofundado do domínio do negócio, auxiliando na gestão eficiente dos recursos e na manutenção da integridade operacional. A arquitetura de microsserviços proposta neste trabalho visa oferecer a flexibilidade necessária para se adaptar às novas demandas do mercado e diferentes segmentos de negócios. Sua correta definição é de grande importância, pois influencia diretamente no sucesso e desempenho do produto final, proporcionando uma base sólida para sua construção e evolução. A pesquisa abrange o estudo de conceitos teóricos relacionados à arquitetura de microsserviços e a análise de requisitos específicos no contexto contábil de pequenas empresas. Concluindo, motivada pela constante evolução e diversidade no contexto empresarial, destaca-se como resultado a proposta de uma arquitetura de microsserviços para um sistema ERP contábil, visando integrar as rotinas contábeis com as operações empresariais, reforçando a necessidade e relevância desta integração para a eficiência operacional das pequenas empresas.

Palavras-chave: Engenharia de software; Arquitetura de software; Microsserviços.

ABSTRACT

Sampaio, Lucas Oliveira. **Proposta de Arquitetura de Microsserviços Para Um Sistema ERP Contábil.** Orientador: Prof. Msc. Leandro Costa Souza. 2023. PG f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação Ciência e Tecnologia da Bahia, Santo Antônio de Jesus – BA, 2023.

This study presents a proposal for a microservices architecture for an ERP (Enterprise Resource Planning) system, focused on the accounting of small businesses. Accounting, when combined with an ERP system, can provide in-depth knowledge of the business domain, assisting in the efficient management of resources and the maintenance of operational integrity. The proposed microservices architecture in this work aims to offer the necessary flexibility to adapt to the changing demands of the market and different business segments. Its proper definition is of great importance, as it directly influences the success and performance of the final product, providing a solid foundation for its construction and evolution. The research encompasses the study of theoretical concepts related to microservices architecture and the analysis of specific requirements in the accounting context of small businesses. In conclusion, motivated by the constant evolution and diversity in the business context, the proposal of a microservices architecture for an accounting ERP system is highlighted as a result, aiming to integrate accounting routines with business operations, reinforcing the need and relevance of this integration for the operational efficiency of small businesses.

Keywords: Software engineering; Software architecture; Microservices.

LISTA DE FIGURAS

Figura 2.1 – Diagrama esquemático de um sistema MVC. **20**
Figura 2.2 – Modelo Cliente Servidor. **21**
Figura 2.3 – Arquitetura em Camadas. **22**
Figura 2.4 – Modelo SOA. **24**
Figura 2.5 – Arquitetura de Microsserviços. **26**
Figura 2.6 – Agregador de Pedidos. **34**
Figura 2.7 – Ecossistema de Microsserviços. **36**
Figura 3.1 – Diagrama das Fronteiras do SERPC. **54**
Figura 3.2 – Diagrama das Fronteiras da zona fiscal do SERPC. **57**
Figura 3.3 – Diagrama dos Componentes do SERPC. **58**
Figura 3.4 – Diagrama do SERPC com suas tecnologias. **59**
Figura 4.1 – Composição de modulos do SERPC. **62**
Figura 4.2 – Solução do Visual Studio. **63**
Figura 4.3 – interface do Docker Desktop. **64**
Figura 4.4 – Interface de Gerenciamento do RabbitMQ. **65**
Figura 4.5 – Interface de gerenciamento do Consul. **66**
Figura 4.6 – Interface gráfica que consome os endpoints externos do SERPC**.** **67**

LISTA DE QUADROS

Quadro 2.1 – Lista setorial de atividades inclusas no Simples Nacional. **43**
Quadro 3.1 – Requisitos funcionais do SERPC. **49**
Quadro 3.2 – Requisitos essenciais para o ciclo de vida do SERPC. **50**
Quadro 3.3 – Serviços componentes do SERPC. **51**

LISTA DE SIGLAS E ABREVIATURAS

API	*Application Programming Interface*
CTe	Conhecimento de Transporte Eletrônico
DAS	Documento de Arrecadação do Simples
DEFIS	Declaração de Informações Socioeconômicas e Fiscais
deSTDA	Declaração de Substituição Tributária, Diferencial de Alíquota e Antecipação
DF-e	Documentos Fiscais Eletrônicos
DDD	*Domain-Driven Design*
DIR	Declaração do Imposto de Renda
DIRF	Declaração do Imposto de Retido na Fonte
DIRPF	Declaração do Imposto de Renda Pessoa Física
DIFAL	Diferencial de Alíquota
DRE	Demonstração do resultado do exercício
DNS	*Domain Name System*
ERP	*Enterprise Resource Planning*
FGTS	Fundo de Garantia do Tempo de Serviço
HTTP	*Hypertext Transfer Protocol*
ICMS	Imposto sobre Circulação de Mercadorias e Serviços
INSS	Instituto Nacional do Seguro Social
IPI	Imposto sobre Produtos Industrializados
IPTU	Imposto sobre a Propriedade Predial e Territorial Urbana
ISSQN	Imposto sobre Serviços de Qualquer Natureza
MDF-e	Manifesto Eletrônico de Documentos Fiscais
MVC	*Model-View-Controller*
NFC-e	Nota Fiscal do Consumidor Eletrônica
NF-e	Nota Fiscal Eletrônica
NFS-e	Nota Fiscal de Serviço Eletrônica
PGDAS	Programa Gerador do Documento de Arrecadação do Simples Nacional
RAIS	Relação Anual de Informações Sociais
REST	*Representational State Transfer*
RFC	*Remote procedure call*
SERPC	Sistema de Planejamento de Recursos Empresariais Contábil
SOA	*Service-oriented architecture*

SUMÁRIO

1. INTRODUÇÃO 15

1.1. OBJETIVOS 16

1.2. JUSTIFICATIVA 16

1.3. METODOLOGIA E ORGANIZAÇÃO DO TRABALHO 17

2. FUNDAMENTAÇÃO TEÓRICA 18

2.1. ARQUITETURA DE SOFTWARE 18

2.2. PADRÕES DE ARQUITETURA 19

2.2.1. Arquitetura Monolítica 19

2.2.2. Arquitetura Cliente-Servidor 21

2.2.3. Arquitetura em Camadas 22

2.2.4. Arquitetura SOA 23

2.3. ARQUITETURA DE MICROSSERVIÇOS 24

2.3.1. Características dos microsserviços 26

2.3.2. Modelagem de Microsserviços 29

2.2.3. Design Orientado ao Domínio Aplicado aos Microsserviços 32

2.3.4. Ecossistema de Microsserviços 35

2.4. CONCEITOS BÁSICOS SOBRE A CONTABILIDADE 41

2.4.1. Departamentos da Contabilidade 41

2.4.2. Simples nacional 42

2.4.4. Setorização das Empresas 44

2.4.5. Organização do Estado Brasileiro 45

2.4.6. ERP 45

3. PROPOSTA DE ARQUITETURA DE MICROSSERVIÇOS 47

3.1. OBJETIVOS DO SISTEMA ERP CONTÁBIL 47

3.2. ANÁLISE E LEVANTAMENTO DOS REQUISITOS FUNCIONAIS 48

3.3. CLASSIFICAÇÃO E CONFIGURAÇÃO DOS SERVIÇOS 51

3.4. DEFINIÇÃO DAS FRONTEIRAS 53

3.5. COMUNICAÇÃO E INTEGRAÇÃO ENTRE OS MICROSSERVIÇOS 54

3.5.1. Comunicação Interna 55

3.5.2. Comunicação Externa 55

3.5.3. Descoberta de Serviços *56*
3.6. DEFINIÇÃO DE ARQUITETURA DE MICROSSERVIÇOS 56
3.7. TECNOLOGIAS PARA O ECOSSISTEMA DE MICROSSERVIÇOS 58
3.7.1. .NET *59*
3.7.2. Ocelot *59*
3.7.3. Duende Identity Server *60*
3.7.4. Consul *60*
3.7.5. RabbitMQ *60*
3.7.6. Docker *61*
3.7.8. Considerações Finais Acerca das Tecnologias *61*
4. IMPLEMENTAÇÃO DA PROVA DE CONCEITO 62
5. DISCUSSÃO E CONSIDERAÇÕES FINAIS 68
6. TRABALHOS FUTUROS 70
7. REFERÊNCIAS BIBLIOGRÁFICAS 71

1. INTRODUÇÃO

Nos últimos anos, a arquitetura de microsserviços tem se destacado como uma abordagem revolucionária no desenvolvimento de sistemas de software. Sua eficiência e flexibilidade têm atraído a atenção de empresas de tecnologia em todo o mundo, impulsionando a adoção em larga escala devido às suas características únicas, que proporcionam escalabilidade, flexibilidade e agilidade no desenvolvimento de aplicações.

> "*Microservices*" - mais um novo termo nas ruas movimentadas da arquitetura de software. Embora nossa inclinação natural seja ignorar tais coisas com um olhar desdenhoso, esta pequena terminologia descreve um estilo de sistemas de software que estamos achando cada vez mais atraente. Vimos muitos projetos usarem esse estilo nos últimos anos, e os resultados até agora têm sido positivos, tanto que para muitos de nossos colegas esse está se tornando o estilo padrão para criar aplicativos corporativos (FOWLER e LEWIS, 2014).

No contexto da globalização acelerada pela evolução tecnológica é cada vez mais exigido que os sistemas de software possam se adaptar facilmente às mudanças de demanda, crescer de forma escalável e se integrar harmoniosamente com outras aplicações. E isso é exatamente o que a arquitetura de microsserviços se propõe a oferecer. Seu foco em componentes independentes e altamente especializados permite que as empresas desenvolvam sistemas modulares e distribuídos, capazes de atender às necessidades específicas de seus clientes de forma ágil e eficiente.

No entanto, não se pode ignorar que a implementação desse tipo de arquitetura também pode trazer desafios e complexidades. Nesse contexto, a arquitetura de software desempenha um papel crucial. Ela fornece as bases sólidas para estabelecer regras, princípios e práticas que garantem o sucesso do sistema proposto. Neste trabalho, serão explorados os princípios e as boas práticas relacionadas ao estilo arquitetural de microsserviços, com foco em aplicações de ERP (Enterprise Resource Planning), mais especificamente um sistema ERP Contábil voltado para pequenas empresas.

A utilização de sistemas ERP é uma ótima alternativa para superar desafios gerenciais. Esses sistemas oferecem soluções integradas que permitem gerenciar diversas áreas do negócio, como contabilidade, finanças, estoque, vendas, recursos humanos, produção e cadeia de suprimentos, além de automatizar processos. A fácil observação da modularidade desses sistemas torna a arquitetura de microsserviços uma escolha atraente para sua implementação.

A arquitetura proposta para o ERP Contábil deste trabalho tem como objetivo oferecer a flexibilidade necessária para se adaptar aos diferentes segmentos de mercado, além de integrar a contabilidade com a empresa cliente, facilitando a comunicação, a troca de informações e o conhecimento aprofundado do domínio do negócio.

1.1. Objetivos

O propósito central deste trabalho é explorar como a arquitetura de microsserviços pode ser empregada para desenvolver um software ERP Contábil que seja capaz, não apenas de se adaptar e evoluir de acordo com a diversidade de setores e as transformações dinâmicas do mercado empresarial, mas também de lidar com a complexidade e a variação das regras e tributos entre diferentes estados e municípios, atendendo inicialmente ao escopo de empresas optantes ao regime tributário do Simples Nacional. O principal objetivo é entender a aplicabilidade da arquitetura de microsserviços nesse contexto, objetivando a criação de um sistema flexível que possa ser atualizado e expandido com facilidade ao longo do tempo. Com a execução desta pesquisa, espera-se oferecer uma contribuição significativa ao conhecimento existente sobre as possibilidades da arquitetura de microsserviços para o desenvolvimento de softwares ERP, focando principalmente na capacidade de adaptação e evolução do sistema para atender aos requisitos dinâmicos do ambiente empresarial.

1.2. Justificativa

A adoção da arquitetura de microsserviços no desenvolvimento de um ERP contábil representa uma evolução significativa em relação aos modelos arquiteturais mais tradicionais. Este estudo se justifica pela importância de compreender e implementar, de maneira eficaz, os princípios e melhores práticas associadas à arquitetura de microsserviços no contexto específico de um sistema ERP Contábil.

Uma das principais razões para adotar a arquitetura de microsserviços nesse contexto é a busca pela flexibilização e facilidade na adição de novos módulos. Através dessa abordagem, o sistema ERP pode ser dividido em serviços independentes e menores, o que permite o desenvolvimento e a manutenção individualizada de cada módulo. Essa característica fornecida pelos microsserviços facilita a personalização

do sistema para atender às necessidades específicas de cada segmento empresarial, proporcionando maior flexibilidade e adaptabilidade.

A compreensão dos padrões e melhores práticas no desenvolvimento de microsserviços é essencial para orientar as decisões arquiteturais desse projeto. Isso contribui para a entrega de um sistema de qualidade, capaz de evoluir de acordo com as necessidades do negócio. Portanto, a justificativa para este trabalho está fundamentada na busca pelos padrões que proporcionam agilidade, flexibilidade e adaptabilidade no desenvolvimento de sistemas baseados em microsserviços, visando a proposta de uma arquitetura de microsserviços que seja capaz de evoluir com as demandas do mercado.

1.3. Metodologia e Organização do Trabalho

Como proposta metodológica, este trabalho propõe um aprofundamento de conhecimentos teóricos sobre a arquitetura de microsserviços, explorando seus princípios e boas práticas. O objetivo é compreender os conceitos únicos dessa temática e como essa arquitetura pode ser aplicada de forma eficiente e eficaz no contexto de um sistema ERP Contábil.

Ao final deste trabalho, será desenvolvida uma proposta de arquitetura tomando como base os conceitos estudados, seus princípios e boas práticas, de forma que essa modelagem arquitetural possa proporcionar benefícios tanto para as empresas de tecnologia quanto para os clientes finais.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado os conceitos fundamentais estudados para a elaboração do sistema ERP Contábil

2.1. Arquitetura de Software

A arquitetura de software pode ser compreendida como um conjunto de decisões e padrões que estabelecem a estrutura essencial de um sistema. Ela abrange a organização dos componentes, a forma como se comunicam e a distribuição de suas funcionalidades. Ralph Johnson em uma troca de e-mails com Martin Fowler falou que "A Arquitetura diz respeito às coisas importantes. O que quer que sejam elas.".

Essa troca de e-mail entre Ralph e Fowler mais tarde se tornou um artigo escrito por Fowler e pode ser usado para evidenciar a forma abstrata do entendimento conceitual acerca da arquitetura de software. Ela destaca que a arquitetura diz respeito aos aspectos mais relevantes e significativos de um sistema, independentemente de sua natureza. Essa abstração permite que a arquitetura seja aplicável a uma diversidade de contextos e domínios, se adaptando às necessidades específicas de cada projeto.

A arquitetura de software vai além da definição de estruturas e componentes. Ela envolve decisões fundamentais que orientam a criação do sistema, levando em consideração seus requisitos funcionais e não funcionais, suas restrições e objetivos. Ao estabelecer esse conjunto de padrões e diretrizes, a arquitetura de software define o formato do sistema, fornecendo uma base sólida para o seu desenvolvimento, manutenção e evolução. Ainda referenciando a troca de e-mails, Ralph afirma que:

> *Na maioria dos projetos de softwares de sucesso os desenvolvedores mais experientes que trabalham nesse projeto têm um entendimento compartilhado acerca do Design do sistema. Esse entendimento compartilhado é chamado de 'arquitetura'. Ele inclui como o sistema é dividido em Componentes e como os componentes interagem por meio de interfaces. Esses componentes geralmente são compostos de componentes menores, mas a arquitetura inclui somente os componentes e as interfaces que são conhecidas por todos os desenvolvedores.* (FOWLER, 2003).

Dessa forma, é possível entender que a arquitetura se trata da tradução dos objetivos e desafios vivenciados pela equipe desenvolvedora do software em questão e pode ser vista como uma construção social, pois não se limita apenas ao software

que está sendo construído, mas leva em consideração todo o contexto de desenvolvimento.

2.2. Estilos de Arquitetura

Newman (2021) Fala que em essência os microsserviços são apenas outra forma de decomposição modular, o que significa que podemos contar com muitas das técnicas anteriores da área de software modular e de programação estruturada para ajudar na definição das fronteiras de negócio.

Este capítulo apresenta uma abordagem evolutiva acerca da modularidade dos estilos de arquitetura de software, destacando como essas abordagens evoluíram ao serem impulsionadas pela busca contínua por flexibilidade no desenvolvimento de software.

É importante destacar que esses estilos não são mutuamente exclusivos, podendo ser combinados e complementados de acordo com as necessidades específicas de cada projeto.

2.2.1. Arquitetura Monolítica

A abordagem da arquitetura monolítica consiste na criação de um sistema como um único bloco indivisível, no qual todas as funcionalidades e responsabilidades só podem ser implantadas em conjunto, sendo inicializadas por um único processo. Nesse estilo arquitetural, o sistema aloca todos os recursos necessários para sua execução, centralizando a aplicação em uma única plataforma.

> Um aplicativo monolítico é aquele que é totalmente autossuficiente, em termos de comportamento. Ele pode interagir com outros serviços ou armazenamentos de dados durante a execução de suas operações, mas o núcleo de seu comportamento é executado em seu próprio processo e o aplicativo inteiro normalmente é implantado como uma única unidade. Se um aplicativo desse tipo precisar ser dimensionado horizontalmente, em geral, o aplicativo inteiro será duplicado em vários servidores ou máquinas virtuais (SMITH, 2022, p. 16).

Esse estilo de design apresenta seus próprios benefícios e desvantagens. Uma característica comum nesse estilo é a utilização de uma base de código em comum para toda a aplicação, o que pode facilitar os processos de desenvolvimento e implan-

tação do sistema. A centralização dos recursos simplifica as rotinas de desenvolvimento e depuração, uma vez que todo o sistema é mantido em um único contexto, facilitando a compreensão. Além disso, a comunicação entre os componentes do sistema geralmente é realizada por chamadas de função ou métodos internos, sem as complexidades adicionais de comunicação em rede. Isso pode resultar em um melhor desempenho em certos cenários.

Para obter modularidade em sistemas monolíticos, geralmente as diferentes partes do sistema, como a interface do usuário, a lógica de negócios e o acesso a dados, podem ser organizadas em módulos ou camadas lógicas, construindo uma espécie de sistema monolítico modular. onde um processo é dividido em módulos lógicos conforme mostra a Figura 2.1 ao evidenciar o padrão arquitetural MVC. No entanto, todas essas camadas permanecem fortemente acopladas e compartilham o mesmo contexto de execução.

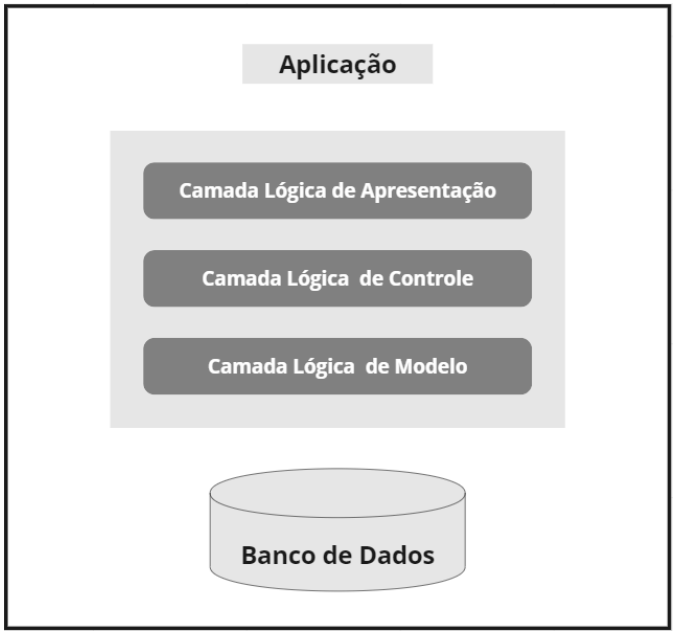

Figura 2.1 – Diagrama esquemático de um sistema MVC.
Fonte: (Elaborado pelo Autor, 2023)

Embora a arquitetura monolítica ainda seja encontrada em muitos sistemas atuais, especialmente quando a complexidade do projeto ou os requisitos do sistema não justificam a adoção de uma arquitetura mais distribuída, ela apresenta limitações quando o projeto requer alta modularidade, escalabilidade e integração.

2.2.2. Arquitetura Cliente-Servidor

A arquitetura cliente-servidor introduziu a separação física das funcionalidades na forma como os sistemas de software são projetados e organizados. Nesse modelo, as funcionalidades do sistema são divididas em duas partes principais: o cliente e o servidor. O cliente fornece uma maneira conveniente para os usuários interagirem com o sistema. Ele envia solicitações ao servidor, como solicitações de dados, processamento ou serviços específicos. O servidor, por sua vez, é responsável por processar essas solicitações, realizar as operações necessárias e retornar os resultados ao cliente. A figura 2.2 demonstra a arquitetura cliente-servidor e sua separação física.

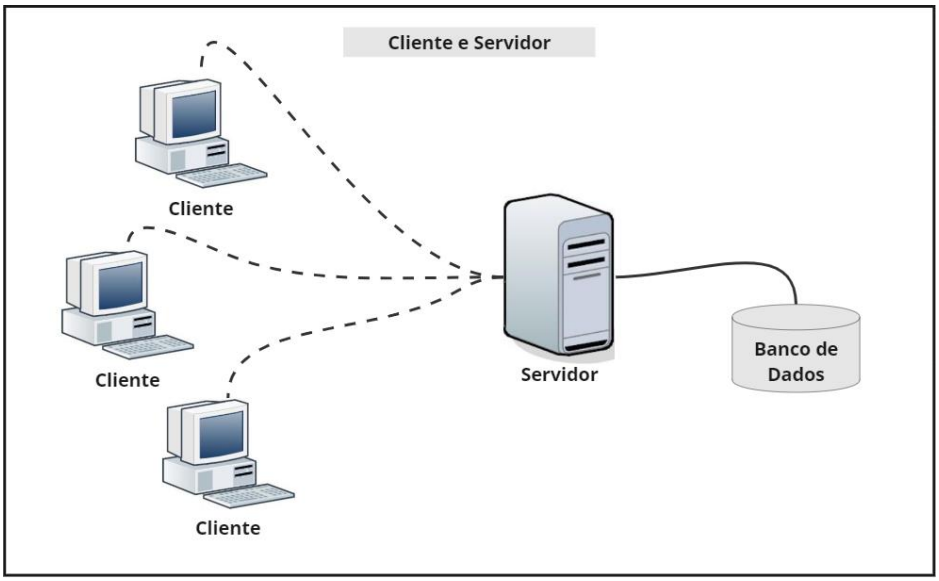

Figura 2.2 – Modelo Cliente Servidor.
Fonte: (Elaborado pelo Autor, 2023)

Essa arquitetura traz vantagens ao permitir o compartilhamento de recursos e a separação de responsabilidades. A lógica de negócios, o processamento de dados e os recursos são centralizados no servidor, enquanto a interface do usuário é tratada pelo cliente. Essa divisão permite uma maior modularidade no projeto do sistema, facilitando a manutenção, a escalabilidade e as atualizações. Alterações no servidor podem ser realizadas sem afetar diretamente o cliente, o que proporciona flexibilidade e facilidade na evolução do sistema.

A comunicação entre o cliente e o servidor é estabelecida por meio de protocolos de rede. Esses protocolos permitem que o cliente envie solicitações ao servidor e receba as respostas correspondentes. Dependendo das necessidades do sistema, a

comunicação pode ocorrer de forma síncrona, em que o cliente aguarda a resposta antes de continuar sua execução, ou de forma assíncrona, em que o cliente continua sua execução enquanto espera pela resposta do servidor.

No entanto, a arquitetura cliente-servidor também possui desvantagens. A dependência do servidor pode resultar em um único ponto de falha. Se o servidor ficar indisponível, todos os clientes serão afetados. Além disso, a comunicação constante entre o cliente e o servidor pode gerar latência na rede, impactando o desempenho do sistema. A complexidade adicional do sistema inserida pela adição de protocolos de comunicação em rede também influencia no projeto.

2.2.3. Arquitetura em Camadas

A arquitetura em camadas introduz mais um conceito de modularidade nas aplicações ao propor a separação das funcionalidades em diferentes camadas, cada uma responsável por um conjunto específico de tarefas. A evolução para essa abordagem evidencia a constante requisição por modularidade e distribuição nas aplicações, principalmente no contexto da web. A Figura 2.3 evidencia a arquitetura em camadas e sua divisão de funcionalidades.

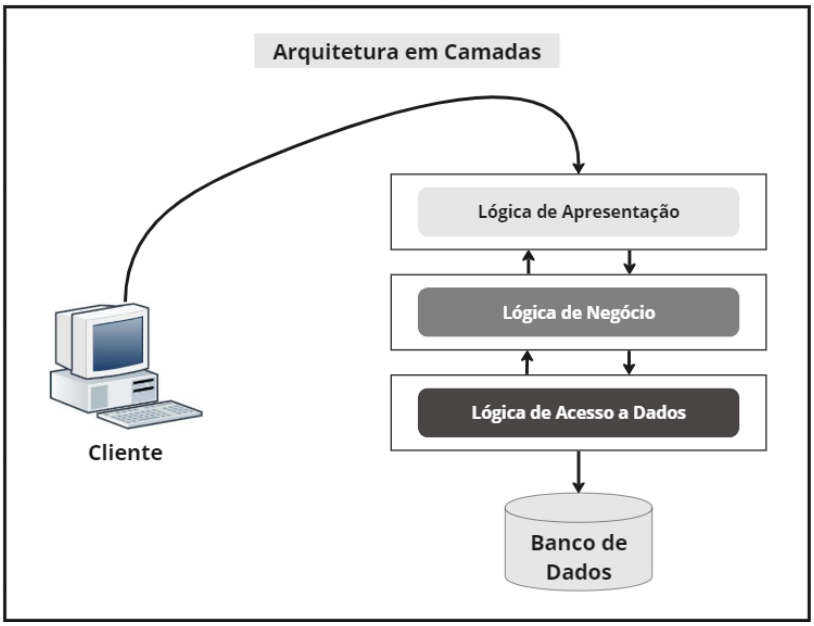

Figura 2.3 – Arquitetura em Camadas.
Fonte: (Elaborado pelo Autor, 2023)

Nesse modelo, cada camada possui uma responsabilidade clara e limitada, o que facilita a compreensão, teste e alterações em partes específicas da aplicação sem afetar as outras camadas. Além disso, essas camadas podem ser distribuídas em servidores separados, trazendo benefícios de otimização e escalabilidade.

A arquitetura em camadas pode ser considerada um aprimoramento da arquitetura cliente-servidor que busca uma divisão mais granular das funcionalidades em diferentes camadas. Essa abordagem trouxe benefícios significativos quanto a modularidade, escalabilidade e facilidade de manutenção, porém também trouxe desafios adicionais em respeito a coordenação e comunicação entre camadas distribuídas.

2.2.4. Arquitetura SOA

A arquitetura SOA (Arquitetura Orientada a Serviços) é um modelo de construção de software que enfatiza a orientação a serviços. Nessa arquitetura, os serviços são os principais componentes do sistema e são projetados para serem distribuídos, autônomos e independentes.

Em SOA, as funcionalidades do software são implementadas como serviços separados e independentes. Esses serviços podem ser combinados para criar aplicações mais complexas. Eles encapsulam e expõem uma interface bem definida, geralmente por meio de serviços web, permitindo a comunicação e interação com outros serviços. A Figura 2.4 mostra um exemplo da Arquitetura Orientada a Serviços e seus serviços componentes.

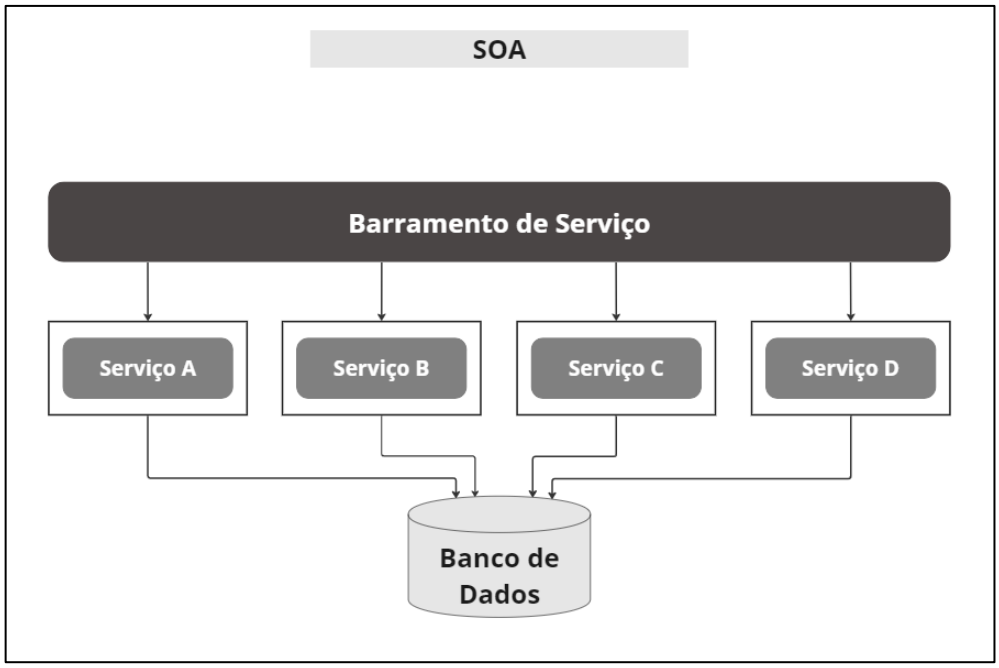

Figura 2.4 – Modelo SOA.
Fonte: (Elaborado pelo Autor, 2023)

Os serviços devem ser autônomos, o que permite que sejam implantados e executados de forma independente. Além disso, eles devem ser interoperáveis, possibilitando o acesso e consumo por diferentes sistemas e aplicativos, independentemente da plataforma ou tecnologia utilizada.

SOA é um estilo de arquitetura que promove a modularidade, reutilização e integração de serviços independentes. Essa abordagem permite a criação de sistemas flexíveis, escaláveis e adaptáveis, facilitando a evolução dos sistemas de software. No entanto, a gestão e governança dos serviços, bem como a segurança da comunicação, são desafios que devem ser considerados ao adotar esse estilo arquitetural.

2.3. Arquitetura de Microsserviços

A Arquitetura de Microsserviços pode ser percebida como um mecanismo eficaz na resposta à crescente demanda por modularidade, escalabilidade e agilidade no ciclo de desenvolvimento de aplicações modernas. Através dessa abordagem arquitetural, sistemas monolíticos complexos obtém a capacidade de ser decompostos em múltiplas unidades distribuídas, constituindo um conjunto de serviços independen-

tes voltados para seus contextos e lógicas de negócios específicos. A adoção de microsserviços representa uma significativa evolução na prática do design de software distribuído, que fomenta não somente a modularidade nas aplicações, mas também a especialização e a eficiente distribuição das equipes de desenvolvimento.

Em tradução literal das palavras de Fowler e Lewis (2014):

> Resumindo, o estilo de arquitetura de microsserviços é uma abordagem para desenvolver um único aplicativo como um conjunto de pequenos serviços, cada um executando em seu próprio processo e se comunicando com mecanismos leves, geralmente uma API de recurso HTTP. Esses serviços são construídos em torno de capacidades de negócios e implantados de forma independente por máquinas de implantação totalmente automatizadas. Há um mínimo de gerenciamento centralizado desses serviços, que podem ser escritos em diferentes linguagens de programação e usar diferentes tecnologias de armazenamento de dados (FOWLER e LEWIS, 2014).

Essa descrição destaca a natureza autônoma e diversificada dos microsserviços, apontando para a flexibilidade oferecida por essa arquitetura que eleva a modularidade um passo adiante. Tornando cada componente um serviço completamente autônomo que pode evoluir de forma independente. A característica modular dos microsserviços implica que cada serviço pode ser desenvolvido, testado e implantado de forma independente dos outros. Isso facilita o trabalho em paralelo, acelerando a velocidade e a eficiência do ciclo de desenvolvimento do software. Além disso, permite que as equipes se concentrem em um domínio ou serviço específico do negócio, promovendo assim a especialização das equipes de desenvolvimento. A figura 2.5 evidencia um exemplo de arquitetura de Microsserviços e sua divisão em componentes autônomos.

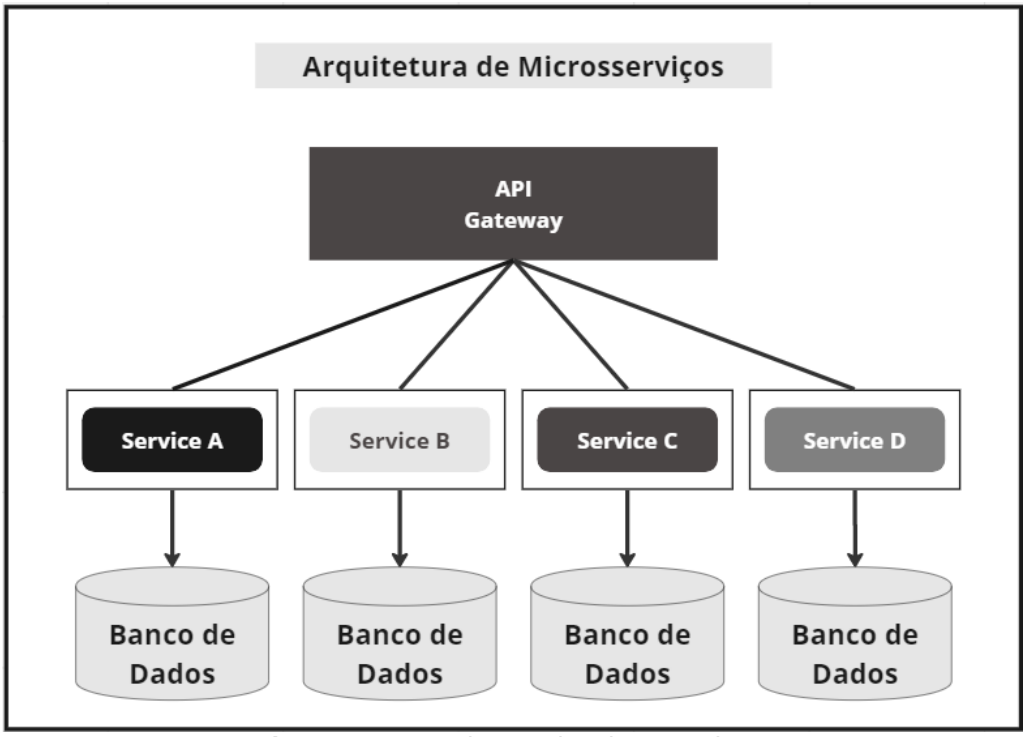

Figura 2.5 – Arquitetura de Microsserviços.
Fonte: (Elaborado pelo Autor, 2023)

No entanto, a arquitetura de microsserviços também apresenta suas próprias complexidades e desafios, e não deve ser considerada como a solução imediata para um projeto. Este ponto é evidenciado por Newman (2021):

> As arquiteturas de microsserviços podem proporcionar muita flexibilidade para escolher tecnologias, lidar com robustez e estabilidade, organizar equipes, e outras tarefas. Essa flexibilidade é, parcialmente, o motivo pelo qual muitas pessoas estão adotando as arquiteturas de microsserviços. No entanto, os microsserviços carregam consigo um grau significativo de complexidade, e você precisa garantir que essa complexidade se justifique. Para muitas pessoas, essas arquiteturas se tornaram a opção padrão, a ser usada praticamente em qualquer situação. Entretanto, ainda acho que elas são uma opção de arquitetura, cujo uso deve ser justificado pelos problemas que você estiver tentando resolver; muitas vezes, abordagens mais simples podem funcionar mais facilmente. (NEWMAN, 2021, p. 59).

A Arquitetura de Microsserviços oferece grande flexibilidade para um sistema de software em constante evolução, porém essa flexibilidade tem um custo que deve ser considerado ao decidir como o sistema será construído.

2.3.1. Características dos microsserviços

Os microsserviços têm várias características distintas que os definem e diferenciam de outras arquiteturas de software. Nesta seção será explorada algumas dessas características.

2.3.1.1. Composição através de Serviços

A principal forma de composição em uma arquitetura de microsserviços é dividir o software em serviços independentes, cada um executando em seu próprio processo e se comunicando com os demais por meio de mecanismos de rede ou chamadas de procedimento remoto, conforme descrito por Fowler e Lewis (2014). Ao contrário das bibliotecas, que são componentes vinculados a um programa e acionados por chamadas de função em memória, os serviços são componentes externos ao processo que se comunicam por meio de mecanismos como solicitações de serviço da Web ou chamadas de procedimento remoto.

A decomposição do aplicativo em vários serviços independentes oferece a vantagem de permitir que a maioria das alterações em um único serviço exija apenas a reimplantação desse serviço específico. Embora algumas alterações possam exigir a modificação das interfaces de serviço e, consequentemente, a coordenação entre os serviços, uma arquitetura de microsserviços bem projetada visa minimizar essas ocorrências por meio de limites de serviço coesos e mecanismos de evolução nos contratos de serviço.

No entanto, essa abordagem não está livre de desvantagens. Uma das principais desvantagens é que as chamadas remotas são mais custosas do que as chamadas locais, resultando em latência adicional nas aplicações de microsserviços. A latência pode ser um problema significativo em aplicações de alto desempenho. Para essas situações, existem algumas estratégias de otimização que podem ser utilizadas para minimizar a latência, como o uso de protocolos de rede mais rápidos, a implementação de caches de dados, a localização física de serviços que se comunicam com frequência em servidores próximos uns aos outros ou em um mesmo servidor, além da adoção de técnicas de processamento assíncrono.

2.3.1.2. Organização que reflete o negócio

A arquitetura de microsserviços é projetada com o objetivo de refletir diretamente as capacidades de negócio da organização. Segundo Fowler e Lewis (2014), cada microsserviço é desenvolvido em torno de uma capacidade de negócio específica, o que proporciona a flexibilidade necessária para atender às demandas de negócios de forma eficiente e eficaz.

Um princípio fundamental desta abordagem é o de que a estrutura de um sistema, em uma visão ampla, é geralmente um espelho da estrutura de comunicação da organização que o projeta. Esta ideia, inicialmente proposta por Conway em 1968, é frequentemente referida como a Lei de Conway. De acordo com essa lei, se uma organização é dividida em equipes com diferentes áreas de responsabilidade, a arquitetura do sistema que essa organização desenvolve provavelmente refletirá essa divisão.

Na arquitetura de microsserviços, esse princípio se traduz em equipes multidisciplinares responsáveis por um microsserviço específico, ou um conjunto de microsserviços relacionados. Cada equipe tem autonomia para desenvolver, implantar e escalar seus microsserviços de forma independente. Essa autonomia se estende a todos os aspectos do microsserviço, incluindo sua lógica de negócios, banco de dados e interfaces de comunicação. Cada microsserviço é, portanto, auto suficiente e não depende de outros serviços para funcionar.

2.3.1.3. Governança Descentralizada

Na arquitetura de microsserviços, a governança descentralizada surge como uma peça fundamental para impulsionar a inovação e a flexibilidade. Conforme destacado por Fowler e Lewis (2014), essa abordagem permite que cada serviço se beneficie da liberdade de escolher a tecnologia mais adequada às necessidades específicas do projeto, sem impor uma plataforma tecnológica única ou padrões centralizados.

No contexto dos microsserviços, a governança descentralizada vai além da seleção de tecnologia. As equipes têm autonomia para desenvolver suas próprias ferramentas e soluções, as quais podem ser compartilhadas e adotadas por outras equipes conforme necessário. Além disso, as equipes são responsáveis por todos os aspectos do software que constroem, desde o desenvolvimento até a operação contínua.

A governança descentralizada é uma abordagem que favorece a flexibilidade, a inovação e a responsabilidade, tornando-se uma característica essencial da arquitetura de microsserviços. Ao atribuir a responsabilidade direta pelo funcionamento do software às equipes, incentiva-se a qualidade do código e a responsabilidade dos desenvolvedores. Essa responsabilidade direta promove a melhoria da qualidade do código e a especialização dos desenvolvedores.

2.3.1.4. Design Evolutivo

A flexibilidade proporcionada pelos microsserviços promove sua evolução. À medida que as necessidades do negócio evoluem, os microsserviços oferecem a liberdade de serem facilmente adicionados, atualizados ou substituídos sem causar impacto negativo nos demais serviços do sistema. Para isso, seu design é construído observando o padrão de mudanças entre os módulos da aplicação. Essa ideia é ressaltada por Fowler e Lewis (2014):

> Essa ênfase na capacidade de substituição é um caso especial de um princípio mais geral de design modular, que é conduzir a modularidade através do padrão de mudança. Você deseja manter as coisas que mudam ao mesmo tempo no mesmo módulo. Partes de um sistema que raramente mudam devem estar em serviços diferentes daqueles que estão passando por muita rotatividade. Se você estiver trocando repetidamente dois serviços juntos, isso é um sinal de que eles devem ser mesclados (FOWLER e LEWIS, 2014).

Nesse contexto, é destacado um princípio que visa conduzir a modularidade através do padrão de mudança. Em outras palavras, módulos que sofrem alterações com a mesma frequência devem estar agrupados. Partes de um sistema que raramente sofrem alterações devem estar em serviços diferentes daqueles que passam por mudanças frequentes. Caso seja observado que dois serviços são frequentemente modificados em conjunto, isso pode ser um indicativo de que eles devem ser combinados.

2.3.2. Modelagem de Microsserviços

A modelagem adequada de microsserviços é uma etapa crucial para a implementação bem-sucedida de uma arquitetura de microsserviços. Ela influencia diretamente a delimitação de fronteiras de negócio e determina como os serviços irão interagir uns com os outros. Newman (2021) Fala que em essência os microsserviços são

apenas outra forma de decomposição modular, o que significa que podemos contar com muitas das técnicas anteriores da área de software modular e de programação estruturada para ajudar na definição das fronteiras de negócio. Com isso em mente ele evidencia que um bom modelo de microsserviços leva em conta três princípios básicos: a ocultação de informações, a coesão e a redução de acoplamentos. Cada um desses princípios desempenha um papel significativo na formação de uma arquitetura de microsserviços eficaz e sustentável.

2.3.2.1 Ocultação de informações

A ocultação de informações é um princípio de engenharia de software desenvolvido por David Parnas, conforme mencionado por Newman (2021). Esse princípio destaca a importância de minimizar a interdependência entre os módulos de um software. Na arquitetura de microsserviços, esse princípio é aplicado por meio da criação de interfaces bem definidas que escondem os detalhes de implementação de cada serviço.

Na arquitetura de microsserviços, a ocultação de informações desempenha um papel fundamental na garantia da segurança e da integridade dos dados. Esse princípio envolve proteger as informações internas de um serviço e permitir que sejam acessadas apenas por meio de interfaces claramente definidas. Dessa forma, cada serviço pode manter seu próprio estado interno e se comunicar com outros serviços somente quando necessário.

A ocultação de informações também contribui para a escalabilidade e a manutenção dos microsserviços. Com a clara definição das interfaces, os serviços podem ser atualizados ou escalados de forma independente, sem afetar negativamente outros serviços. Isso facilita a evolução do sistema, permitindo que cada serviço seja modificado ou aprimorado sem a necessidade de alterar ou impactar os demais serviços.

2.3.2.2 Coesão: organização harmônica dos microsserviços

A Coesão, em uma Arquitetura de Microsserviços, descreve a harmonia e a interdependência funcional dentro de cada microsserviço. Um microsserviço alta-

mente coeso é aquele que cumpre um conjunto restrito e específico de tarefas intimamente relacionadas. Isso contribui para um sistema mais fácil de entender, gerenciar e evoluir, pois a lógica de cada serviço é bem definida e limitada a um escopo claro.

Newman (2021) destaca a importância da coesão na arquitetura de microsserviços, afirmando:

> Queremos encontrar fronteiras no domínio de nosso problema que ajudem a garantir que comportamentos relacionados estejam em um só lugar e se comuniquem com outras fronteiras o mínimo possível. Se as funcionalidades relacionadas estiverem espalhadas pelo sistema, dizemos que a coesão é fraca - em nossas arquiteturas de microsserviços, nosso objetivo é ter uma forte coesão (NEWMAN, 2021, p.63).

A coesão se refere ao grau em que os elementos de um serviço estão interligados e cooperam para atingir um objetivo comum. Um serviço mais coeso se traduz em uma funcionalidade mais eficiente e clara. Na arquitetura de microsserviços, é crucial garantir que cada serviço seja altamente coeso e tenha uma função de negócios bem definida. Isso não apenas maximiza a eficácia do serviço, mas também facilita a compreensão do papel de cada serviço dentro da arquitetura geral, permitindo que desenvolvedores, stakeholders e outros atores do sistema compreendam e mantenham a arquitetura de forma mais eficaz.

2.3.2.3 Acoplamento: redução da dependência entre microsserviços

O acoplamento se refere à interdependência entre diferentes serviços. Em uma arquitetura de microsserviços, o objetivo é projetar uma estrutura com o mínimo de acoplamento possível, garantindo que cada serviço seja autônomo e independente dos demais. De acordo com Newman (2021), quando funcionalidades relacionadas estão dispersas pela aplicação, as alterações nessas funcionalidades tendem a se espalhar por várias fronteiras, aumentando o acoplamento entre os serviços.

Para sintetizar os objetivos de acoplamento em uma arquitetura de microsserviços, Newman faz referência à Lei de Constantine, que afirma que uma estrutura será estável quando a coesão for alta e o acoplamento for baixo. Isso significa que os componentes dentro de uma entidade devem estar fortemente relacionados, enquanto as dependências entre diferentes entidades devem ser minimizadas.

É importante destacar que um certo grau de acoplamento entre serviços é inevitável. No entanto, é crucial equilibrar essa independência com uma comunicação

adequada entre os serviços, garantindo a colaboração eficaz e a funcionalidade integral do sistema. Mesmo com um acoplamento mínimo, os microsserviços ainda precisam interagir entre si para formar um sistema de software completo e eficiente buscando a flexibilidade.

2.2.3. Design Orientado ao Domínio Aplicado aos Microsserviços

O Domain-Driven Design (DDD), ou Design Orientado ao Domínio, é uma filosofia de desenvolvimento de software que prioriza o domínio e a lógica do domínio. O DDD é uma abordagem extremamente útil na arquitetura de microsserviços, pois propõe uma modelagem baseada no entendimento profundo do domínio de negócio. Esse foco na realidade do negócio ajuda a criar microsserviços que refletem com maior precisão as funcionalidades e os processos da organização.

Algumas ideias centrais do DDD podem ser de grande valor para a arquitetura de microsserviços, auxiliando no desenho de serviços que espelham com mais fidelidade as operações do mundo real. Newman (2021) destaca a importância de alguns conceitos fundamentais extraídos do livro Domain-Driven Design de Eric Evans, que incluem a linguagem ubíqua, agregados e contextos delimitados.

2.2.3.1. Linguagem ubíqua

A linguagem ubíqua desempenha um papel fundamental no contexto do Domain-Driven Design (DDD), ao estabelecer um vocabulário comum entre desenvolvedores e usuários finais. Essa linguagem tem o objetivo de unificar o modelo de design com o modelo de negócios, permitindo a construção de um software que reflita com precisão as funcionalidades e os conceitos do negócio.

Ao adotar uma linguagem ubíqua, é possível criar um entendimento compartilhado que conecta desenvolvedores, especialistas de negócios e usuários finais. O uso dessa linguagem comum ajuda a garantir que todos os envolvidos tenham a mesma compreensão dos conceitos, reduzindo ambiguidades e mal-entendidos. Conforme explicado por Newman (2021):

> Uma linguagem ubíqua se refere à ideia de que devemos nos esforçar para empregar os mesmos termos utilizados pelos usuários em nosso código. A ideia é que ter uma linguagem comum entre a equipe de desenvolvimento e

> as pessoas facilitará modelar o domínio do mundo real, além de melhorar a comunicação (NEWMAN, 2021, p.79).

Isso significa traduzir para o software a linguagem utilizada no domínio do negócio. A linguagem ubíqua facilita a modelagem do mundo real no sistema, melhorando a comunicação entre todas as partes envolvidas. Na arquitetura de microsserviços, a adoção de uma linguagem ubíqua implica refletir essa linguagem comum em todos os serviços. Cada microsserviço deve utilizar os termos compreendidos e utilizados pelo negócio, o que contribui para a clareza e eficácia do design e funcionamento do sistema como um todo. Além disso, é importante mencionar que uma mesma entidade pode ter nomes diferentes dependendo do contexto em que está sendo usada, o que preserva o significado da entidade para cada contexto e mantém o entendimento comum entre desenvolvedores e especialistas de negócio.

A linguagem ubíqua auxilia na construção de microsserviços que atendam precisamente às necessidades do negócio, garantindo que o software desenvolvido esteja alinhado com a lógica e os processos do domínio. Isso resulta em uma compreensão compartilhada das funcionalidades e conceitos do domínio por todos os envolvidos, desde a concepção até a implantação do software.

2.2.3.2. Agregadores

No Design Orientado ao Domínio (DDD), o conceito de agregados é fundamental para a organização e manipulação de objetos de domínio. Um agregado é uma coleção de objetos que são tratados como uma única unidade para fins de operações de domínio. Essa unidade é sempre garantida a ser consistente em termos de regras de negócios e lógica de domínio.

Os agregados são compostos por uma ou mais entidades, sendo uma delas a entidade raiz. A entidade raiz é o único objeto acessível externamente e garante a consistência interna do agregado, ou seja, a entidade raiz controla o acesso e modificações às entidades que compõem o agregado. A figura 2.6 mostra um agregador de pedidos que gerencia o acesso e as modificações das entidades que compõem esse agregado.

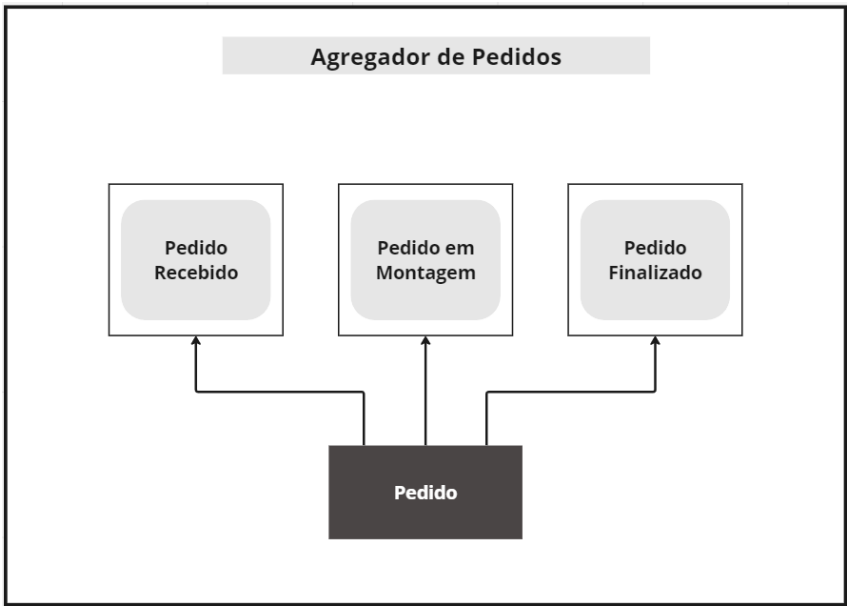

Figura 2.6 – Agregador de Pedidos.
Fonte: (Elaborado pelo Autor, 2023)

De acordo com Newman (2021), os agregados podem ser vistos como máquinas de estado, onde cada estado representa uma condição ou situação específica em que o agregado pode existir. As transições entre esses estados são controladas pelas operações de domínio aplicadas ao agregado.

Na arquitetura de microsserviços, os agregados são de grande importância pois ajudam a manter a integridade e a consistência dos dados ao agrupar entidades relacionadas que precisam ser consistentes como uma unidade. Assim, os agregados auxiliam na criação de limites claros entre diferentes partes do sistema, facilitando o gerenciamento e a manutenção dos serviços. Além disso, como cada agregado é tratado como uma única unidade, isso permite que diferentes microsserviços manipulem diferentes agregados de maneira independente, favorecendo o paralelismo e a distribuição do sistema.

2.2.3.3. Contexto Delimitado

O Contexto Delimitado é um conceito fundamental no Domain-Driven Design, que se refere à definição de limites claros nos quais um modelo de domínio específico é aplicado. Esses limites podem ser estabelecidos com base em diferentes critérios, como funcionalidades específicas do sistema, conjuntos de regras de negócio, subdomínios específicos ou mesmo departamentos de negócio. Dentro de cada Contexto

Delimitado, existe um modelo de domínio único composto por um conjunto de conceitos, definições e regras que representam aspectos específicos desse contexto.

Segundo Newman (2021), os contextos delimitados funcionam como um mecanismo de ocultação, escondendo detalhes de implementação que não são relevantes para outras partes do sistema. Isso é especialmente importante para detalhes internos que não têm impacto ou interesse para aqueles que estão fora do contexto. Ele argumenta que essas preocupações internas devem permanecer ocultas para o mundo externo, pois não é necessário nem benéfico que sejam conhecidas por todos.

Na arquitetura de microsserviços, cada microsserviço pode ser visto como um Contexto Delimitado. Ele encapsula um modelo de domínio específico e oferece uma interface claramente definida para interagir com outros microsserviços. Isso facilita o desenvolvimento, teste e evolução independentes de cada microsserviço, garantindo ao mesmo tempo a integridade e a consistência do modelo de domínio dentro do seu próprio Contexto Delimitado.

Os Contextos Delimitados ajudam a gerenciar a complexidade de um sistema, dividindo-o em partes menores e mais gerenciáveis, cada uma com sua própria lógica e regras de domínio. Isso facilita a compreensão do sistema como um todo, pois cada parte pode ser compreendida em seus próprios termos, sem a necessidade de entender o sistema como um todo de uma só vez.

2.3.4. Ecossistema de Microsserviços

A construção de uma arquitetura de microsserviços é um trabalho complexo que vai além da simples criação de microsserviços. Ela envolve um conjunto de elementos adicionais, que juntos formam o que pode ser chamado de "ecossistema de microsserviços". Segundo Fowler (2017):

> Em ecossistemas de microsserviços bem projetados e sustentáveis, os microsserviços são separados de toda a infraestrutura. Eles são separados do *hardware*, separados das redes, separados do pipeline de *build* e *deployment*, separados da descoberta de serviço (service discovery) e do balanceamento de carga. Isso tudo faz parte da infraestrutura do ecossistema de microsserviços, e construir, padronizar e manter esta infraestrutura de maneira estável, escalável, tolerante a falhas e confiável são fatores essenciais para uma operação bem sucedida dos microsserviços (FOWLER, 2017, p. 34).

O ecossistema de microsserviços é composto por várias ferramentas e padrões que dão suporte ao funcionamento completo dos microsserviços. Isso vai desde a

automação da infraestrutura até a descoberta de serviços, a gestão de falhas, a comunicação entre os serviços e a capacidade de rastrear eventos e atividades, cada uma dessas partes é fundamental para manter os microsserviços funcionando corretamente.

Cada um desses elementos ajuda a melhorar o funcionamento e a manutenção dos microsserviços, aumentando a eficiência do ecossistema como um todo. Fowler (2017) fala que esse ecossistema pode ser dividido em 4 partes, a Figura 2.7 mostra essa divisão em camadas sendo elas, camada de hardware, camada de comunicação, camada de plataforma de aplicação e camada de microsserviços.

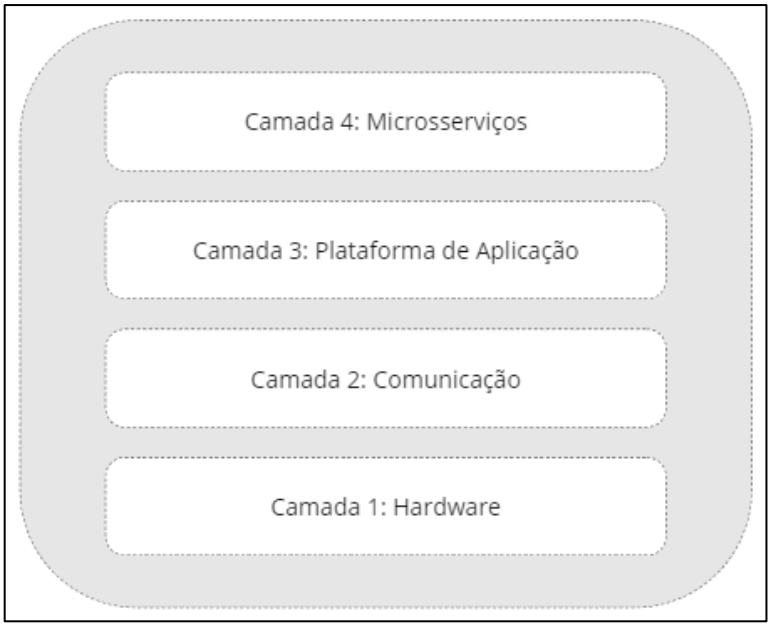

Figura 2.7 – Ecossistema de Microsserviços.
Fonte: (Elaborado pelo Autor, 2023)

2.3.4.1. Camada de Hardware

Na base de todo ecossistema de microsserviços está a camada de hardware. Essa é a camada que inclui as máquinas físicas - os computadores - onde todos os microsserviços e ferramentas internas são executados. Fowler (2017) destaca que o hardware fornece a fundação sobre a qual todo o sistema é construído.

Esta camada abrange não apenas os servidores físicos, que podem ser propriedade direta da empresa ou alugados por meio de serviços de nuvem, mas também bancos de dados dedicados ou compartilhados, sistemas operacionais e diversas ferramentas que operam ao nível do servidor.

Garantir a segurança na arquitetura de microsserviços implica em prestar atenção especial à infraestrutura de hardware. Alguns dos aspectos que devem ser considerados incluem o gerenciamento de configurações, que garante que as configurações corretas sejam implementadas em todos os sistemas; o isolamento e abstração de recursos de hardware, que ajuda a proteger os dados e as operações de cada microsserviço; e o monitoramento em nível de servidor, que permite acompanhar o desempenho e identificar e resolver problemas antes que eles afetem o sistema como um todo.

A camada de hardware não é apenas a base física do sistema, mas também um componente crucial na manutenção da segurança e da eficiência operacional no ecossistema de microsserviços.

2.3.4.2. Camada de Comunicação

Posicionada acima da camada de hardware, a camada de comunicação assume um papel vital ao gerenciar a troca de informações entre os diversos microsserviços. Fowler (2017) destaca que as fronteiras entre essa camada e as demais não são rigidamente definidas, embora os componentes que a compõem sejam facilmente identificáveis. Eles incluem rede, DNS, Chamadas Remotas de Procedimento (RPCs), pontos de contato de API (endpoints de API), descoberta de serviços, registro de serviços e balanceamento de carga.

Os microsserviços oferecem várias maneiras de se comunicar entre si. Uma delas é por meio de RPCs, uma tecnologia que permite que um programa de computador chame um procedimento em outro servidor, como se estivessem no mesmo ambiente. Além disso, há a opção de usar endpoints de API, que funcionam como pontos de conexão para a comunicação entre diferentes serviços.

Além disso, a comunicação pode ocorrer por meio de brokers de mensagens, plataformas responsáveis por gerenciar a troca de mensagens entre serviços. Essa prática aumenta a segurança e a confiabilidade da comunicação, promovendo uma

comunicação desacoplada que fortalece a resiliência e a escalabilidade do sistema. Exemplos populares de sistemas de mensagens incluem RabbitMQ[1] e Kafka[2].

Nesse contexto, Fowler (2017) evidencia também que os métodos de comunicação podem ser classificados como síncronos ou assíncronos. A comunicação síncrona ocorre quando um serviço espera uma resposta imediata de outro serviço (bloqueante), enquanto a comunicação assíncrona (não bloqueante) permite que um serviço envie uma solicitação a outro e continue suas operações sem esperar uma resposta imediata.

Dentro da mesma camada, a descoberta e o registro de serviços, juntamente com o balanceamento de carga, são componentes essenciais. A descoberta de serviços permite que um serviço encontre e se comunique eficientemente com outros serviços, mesmo que estejam em servidores ou regiões diferentes. Isso é facilitado por ferramentas como Consul[3] ou Eureka[4]. Por sua vez, o registro de serviços permite que um serviço se anuncie para outros serviços, possibilitando sua descoberta mútua.

O balanceamento de carga garante a distribuição uniforme do fluxo de informações entre diferentes servidores ou instâncias de serviço. Essa prática otimiza o uso de recursos, reduz os tempos de resposta e evita sobrecarga em qualquer servidor. Dessa forma, a camada de comunicação estabelece a interação entre os microsserviços que compõem o ecossistema.

2.3.4.3. Camada de Plataforma de Aplicação

---

[1] RabbitMQ é um broker de mensagens que suporta vários protocolos de mensagens e streaming. E pode ser implantado em configurações distribuídas e federadas para atender aos requisitos de alta escala e alta disponibilidade <https://www.rabbitmq.com>.

[2] Kafka é uma plataforma de streaming de eventos distribuída, capaz de lidar com requisitos de dados de alto desempenho, análise de streaming, integração de dados e aplicativos de missão crítica <https://kafka.apache.org>.

[3] Consul é uma solução de rede de serviços moderna que fornece descoberta de serviço, automatização de tarefas de rede e segurança <https://www.consul.io>.

[4] Eureka é um serviço de descoberta para aplicativos em nuvem da Netflix que também fornece balanceamento de carga <https://github.com/Netflix/eureka>.

A terceira camada no ecossistema de microsserviços é a camada de plataforma de aplicação. Essa camada é composta por ferramentas e serviços centralizados aplicáveis a todo o ambiente. Fowler (2017) indica que a construção dessa camada deve incentivar as equipes de microsserviços a focarem inteiramente no desenvolvimento e na manutenção de seus respectivos serviços, minimizando preocupações suplementares. Uma plataforma de aplicação eficaz deve englobar ferramentas internas de autosserviço para desenvolvedores, um processo de desenvolvimento padronizado, um sistema centralizado e automatizado de build e release, testes automatizados, uma solução de deployment centralizada e padronizada, além de monitoramento e logging centralizados a nível de microsserviço.

As ferramentas internas de autosserviço desempenham um papel crucial ao oferecer aos desenvolvedores uma estrutura simplificada de trabalho. Essas ferramentas viabilizam o acesso centralizado a serviços que tornam mais simples as tarefas de design, desenvolvimento e manutenção dos microsserviços. Ao disponibilizar essas facilidades, a plataforma de aplicação permite que os desenvolvedores se concentrem em suas atividades principais, potencializando a eficiência e a produtividade. Além disso, com o desenvolvimento dessas plataformas, o gerenciamento das camadas fica centralizado nas respectivas equipes responsáveis, evitando alterações que causem interrupções.

A padronização do ciclo de desenvolvimento traz vantagens significativas, incluindo a promoção de um ambiente de trabalho consistente, a facilitação do entendimento do código e a diminuição de erros. Segundo Fowler (2017), fornecer um sistema de controle de versão centralizado é benéfico, pois facilita o rastreamento de alterações e suporta o trabalho colaborativo. Além disso, é de suma importância fornecer um ambiente de desenvolvimento estável e eficiente, que sirva como espelho do ambiente de produção e inclua todas as dependências. Isso reduz a probabilidade de problemas imprevistos na produção.

Na sequência do ciclo de desenvolvimento, temos a etapa de teste, build, empacotamento e liberação. Como destacado por Fowler (2017), uma vez que qualquer alteração de código tenha sido submetida ao repositório, todos os testes necessários devem ser executados, construindo e empacotando automaticamente as novas versões. Ferramentas de integração contínua, como o Jenkins, facilitam esse processo ao automatizar tarefas e minimizar o espaço para erros humanos.

O processo de pipeline e deployment ocorre após a etapa de teste, build, empacotamento e liberação. Nesse ponto, o novo código é implementado nos servidores de produção. O ecossistema de microsserviços possui uma complexidade característica, principalmente quando levamos em consideração que centenas de implantações diárias são comuns. Portanto, torna-se imperativo construir ferramentas de implantação robustas e padronizar as práticas de implantação para todas as equipes de desenvolvimento.

Por fim, o logging e o monitoramento são componentes essenciais da Camada de Plataforma de Aplicação. Devido à alta velocidade de desenvolvimento de microsserviços, muitas vezes é impraticável reproduzir bugs no código, uma vez que é quase impossível recriar o estado do sistema no momento da falha. O logging em nível de microsserviço fornece aos desenvolvedores as informações necessárias para compreender completamente o estado do seu serviço em um determinado momento, seja ele passado ou presente. Isso facilita a identificação de problemas e a implementação de soluções, aumentando a resiliência e a eficácia do sistema como um todo.

2.3.4.4. Camada de Microsserviços

A camada de microsserviços fica no topo do ecossistema de microsserviços, como explicado por Fowler (2017). Essa camada é o lugar onde ficam os microsserviços e tudo que é específico para eles. Ela está separada de tudo o que está abaixo dela, como a infraestrutura, o hardware, a forma como os serviços se encontram, como a carga de trabalho é dividida e como eles se comunicam. O que não está separado da camada de microsserviços são as configurações específicas de cada serviço.

Na criação de softwares, geralmente, todas as configurações das aplicações ficam em um único lugar. Porém, conforme o sistema cresce e mais microsserviços são adicionados, fica complicado gerenciar todas essas configurações. Para resolver isso, as configurações de cada microsserviço podem ficar em lugares diferentes, cada um sendo cuidado pela equipe que é responsável por aquele microsserviço. Assim, cada equipe cuida das configurações do seu microsserviço, tendo um controle maior sobre elas.

Essa maneira descentralizada de cuidar das configurações permite que as equipes de microsserviços tenham liberdade e rapidez para configurar e atualizar seus

serviços. Além disso, evita a sobrecarga de um único lugar com todas as configurações, tornando o gerenciamento mais eficiente e expansível conforme o número de microsserviços aumenta.

Mesmo que as configurações sejam gerenciadas de maneira descentralizada, é muito importante ter padrões e práticas comuns para garantir que as configurações sejam consistentes e seguras em todo o ecossistema de microsserviços. O trabalho conjunto das equipes e regras claras para o gerenciamento de configurações são essenciais para que os microsserviços funcionem bem e se integrem na camada de microsserviços do ecossistema.

2.4. Conceitos Básicos Sobre a Contabilidade

A contabilidade desempenha um papel fundamental na gestão das empresas, fornecendo informações financeiras essenciais para auxiliar na tomada de decisões estratégicas. Através do registro e análise de transações financeiras, a contabilidade oferece uma visão abrangente do desempenho financeiro da empresa, permitindo que os gestores avaliem sua saúde financeira, identifiquem áreas de melhoria e tomem medidas corretivas quando necessário.

2.4.1. Departamentos da Contabilidade

No contexto da contabilidade, existem três principais departamentos que desempenham funções específicas. O departamento fiscal é responsável por lidar com todas as questões relacionadas aos impostos. Este papel abrange a preparação e o envio de declarações de impostos, a garantia de conformidade com as leis fiscais em vigor, a implementação de estratégias de planejamento tributário para minimizar a carga tributária da empresa e o atendimento a auditorias ou investigações fiscais. Além disso, o departamento fiscal deve lidar com as constantes mudanças na legislação tributária, bem como as variações nas alíquotas e taxas impostas pelas diferentes unidades federativas. É crucial que esse departamento acompanhe as mudanças nas leis fiscais e regulamentos para garantir que a empresa esteja em conformidade e evite problemas legais.

O departamento pessoal tem a responsabilidade de gerenciar todas as questões relacionadas aos funcionários da empresa. Isso inclui a manutenção dos registros dos funcionários, o processamento da folha de pagamento, o monitoramento de horas trabalhadas, a organização de férias e licenças, além do cálculo de benefícios e descontos. O departamento também é responsável por garantir que a empresa esteja em conformidade com as leis trabalhistas, como as leis de contratação, demissão e segurança no trabalho, considerando as diferentes categorias de funcionários e acordos coletivos. É essencial que esse departamento mantenha registros precisos e atualizados dos funcionários, garantindo o cumprimento das obrigações trabalhistas e o bem-estar dos colaboradores.

O departamento contábil é responsável pela gestão dos registros financeiros da empresa. Isso envolve a manutenção de um sistema contábil eficiente e preciso, a preparação de demonstrações financeiras como o balanço patrimonial, a demonstração de resultados e o fluxo de caixa, além da análise e interpretação dessas informações para fornecer insights sobre o desempenho financeiro da empresa. O departamento contábil também pode ser responsável pela elaboração de orçamentos, análise de custos e auxílio na tomada de decisões financeiras estratégicas.

Esses três departamentos desempenham um papel fundamental para garantir que as empresas operem em conformidade com as leis e regulamentos empresariais, além de proporcionar uma visão clara de suas situações patrimoniais e financeiras. Os três departamentos contábeis fornecem informações confiáveis e relevantes que apoiam as empresas para tomar decisões estratégicas, gerir seus recursos de forma eficiente e assegurar a integridade de suas operações financeiras.

2.4.2. Simples nacional

O Simples Nacional é um regime compartilhado de arrecadação, cobrança e fiscalização de tributos aplicável às Microempresas e Empresas de Pequeno Porte. Esse regime tributário abrange uma ampla gama de atividades econômicas pertencentes a diferentes grupos setoriais. É fundamental observar a diversidade de atividades econômicas das empresas. Mesmo no escopo limitado do Simples Nacional, o Quadro 2.1 evidencia essa diversidade ao listar a divisão setorial de atividades permitidas por esse regime de arrecadação.

Quadro 2.1 – Lista setorial de atividades inclusas no Simples Nacional.

Setores de Atividades
Administração pública, defesa e seguridade social
Água, esgoto, atividades de gestão de resíduos e descontaminação
Agricultura, pecuária, produção florestal, pesca e aquicultura
Alojamento e alimentação
Artes, cultura, esporte e recreação
Atividades financeiras, de seguros e serviços relacionados
Atividades imobiliárias
Atividades profissionais, científicas e técnicas
Atividades administrativas e serviços complementares
Construção
Comércio; reparação de veículos automotores e motocicletas
Educação
Eletricidade e gás
Informação e comunicação
Indústrias extrativas
Indústrias de transformação
Organismos internacionais e outras instituições extraterritoriais
Outras atividades de serviços
Transporte, armazenagem e correio
Saúde humana e serviços sociais
Serviços domésticos

Fonte: (MARQUES, 2022)

Esse regime engloba a participação conjunta de todos os entes federativos, o que inclui União, Estados, Distrito Federal e Municípios. Sua administração é de responsabilidade de um Comitê Gestor, composto por oito integrantes: quatro deles pertencentes à Secretaria da Receita Federal do Brasil (RFB), dois representando os Estados e o Distrito Federal e outros dois atuando em nome dos Municípios. As empresas que desejam ingressar no Simples Nacional devem cumprir alguns critérios: enquadrar-se como Microempresa ou Empresa de Pequeno Porte, satisfazer os requisitos impostos pela legislação e efetuar a opção formal pelo regime simplificado.

O Simples Nacional se apresenta como uma alternativa vantajosa para microempresas e empresas de pequeno porte, em razão de suas características relevantes. Entre elas, a centralização de diversos tributos e o seu pagamento através de um único documento de arrecadação, o DAS.

Além disso, o regime simplificado também proporciona a apresentação de uma declaração única e simplificada de informações socioeconômicas e fiscais, a DEFIS, que agiliza o processo de prestação de contas das empresas.

2.4.4. Setorização das Empresas

A segmentação das empresas desempenha um papel crucial no entendimento da complexidade dos processos contábeis e financeiros, cada setor da economia possui características únicas, desafios e oportunidades distintos. A diversidade de atuação das empresas, abrangem setores que vão desde o comércio de bens de consumo até serviços especializados, resultando em uma série de especificidades na condução das atividades contábeis.

Nos setores de varejo e manufatura, por exemplo, os processos de gestão de estoque e precificação de produtos são de extrema importância. Além disso, existem leis e impostos específicos, como IPI e ICMS, e desafios como depreciação de equipamentos, custeio de produtos e reconhecimento de receitas e despesas, que devem ser efetivamente gerenciados.

As empresas do setor de serviços, por sua vez, se deparam com uma realidade contábil diferenciada. A gestão de contratos de prestação de serviços, a alocação de custos diretos e indiretos, a incidência do ISSQN, a necessidade de seguir regulamentações setoriais, além do reconhecimento de receitas, são pontos cruciais neste contexto.

Além disso, independentemente do setor, as empresas devem se adequar a uma série de regulamentações e normativas específicas, abrangendo questões que vão desde a esfera fiscal e trabalhista até a observância de regulamentos ambientais e de segurança. Para evitar sanções ou penalidades, é crucial que essas organizações implementem práticas contábeis alinhadas ao seu campo de atuação.

Vale ressaltar ainda que a variabilidade dos fluxos de trabalho e das demandas de cada setor impõe a necessidade de sistemas e processos adaptáveis às exigências particulares de cada tipo de negócio. A diversidade de setores nos quais as pequenas empresas estão inseridas exerce influência significativa na condução de suas operações contábeis. O entendimento e a administração adequada dessas diferenças são fundamentais para assegurar a eficiência e a conformidade dos processos contábeis dessas empresas.

2.4.5. Organização do Estado Brasileiro

A organização política e administrativa do Estado brasileiro, como estabelecido pela Constituição Federal de 1988, tem impacto significativo nas operações das empresas, especialmente no que se refere às questões contábeis e fiscais. Em seu Artigo 18, a Constituição delineia a federação brasileira, que compreende a União, os Estados, o Distrito Federal e os Municípios, todos autônomos, desde que estejam em conformidade com as normas constitucionais.

Essa organização distribui o poder político e as competências legislativas entre as diferentes partes que compõem o Estado Federal. Cada ente federativo tem a liberdade e a autonomia de estabelecer e regulamentar suas próprias normas e leis, desde que estas não violem as regras constitucionais estabelecidas. Com 26 estados, o Distrito Federal e 5.568 municípios, o Brasil conta com um total de 5.570 entidades federativas, cada uma com seu próprio conjunto de regulamentações e leis fiscais.

Para as empresas, isso significa que elas devem não apenas estar em conformidade com as normas e leis fiscais federais, mas também devem levar em consideração as legislações estaduais e municipais pertinentes. Isso pode resultar em uma complexidade adicional, pois as regulamentações podem variar significativamente de um estado ou município para outro.

Por exemplo, a cobrança de impostos, como o ICMS (Imposto sobre Circulação de Mercadorias e Serviços), pode variar de acordo com a legislação estadual, enquanto o ISSQN (Imposto Sobre Serviços) é determinado pelas leis municipais. Da mesma forma, as empresas devem levar em consideração quaisquer incentivos fiscais ou benefícios que possam ser oferecidos por determinados estados ou municípios.

A organização do Estado brasileiro implica que as empresas devem ter uma compreensão profunda das leis e regulamentações fiscais em todos os níveis de governo. Esta é uma tarefa complexa, mas essencial para a operação eficaz e a conformidade legal dos negócios.

2.4.6. ERP

Os Sistemas de Planejamento de Recursos Empresariais (ERP) são soluções de software desenvolvidas para integrar os diversos departamentos de uma empresa, permitindo a gestão eficaz e otimizada dos recursos empresariais. Esses sistemas reúnem processos como vendas, compras, estoque, produção, finanças, entre outros, em um único lugar, facilitando a tomada de decisão e o planejamento estratégico.

A adoção de um sistema ERP pode trazer inúmeras vantagens para as empresas. Esses sistemas oferecem funcionalidades como automatização e simplificação de processos empresariais, eliminando a necessidade de sistemas separados para cada departamento e reduzindo a probabilidade de erros humanos. Isso resulta em maior eficiência operacional e economia de tempo e recursos.

Esses sistemas oferecem visibilidade e controle completos sobre as operações de negócios, proporcionando uma visão unificada de todos os processos empresariais. Isso facilita a identificação de gargalos operacionais, o monitoramento do desempenho em tempo real e a tomada de decisões baseada em dados. Além disso, um sistema ERP pode melhorar significativamente a conformidade regulatória e a precisão dos relatórios financeiros, uma vez que todos os dados são centralizados em um único sistema, tornando mais fácil rastrear transações, gerar relatórios precisos e manter a conformidade com as regulamentações fiscais e contábeis.

Além disso, os sistemas ERP oferecem visibilidade e controle completos sobre as operações de negócios, proporcionando uma visão unificada de todos os processos empresariais. Isso facilita a identificação de gargalos operacionais, o monitoramento do desempenho em tempo real e a tomada de decisões baseada em dados. A integração da contabilidade com o ERP é bastante vantajosa, pois permite que todas as informações financeiras sejam centralizadas e atualizadas em tempo real. Isso proporciona uma visão precisa e atualizada da situação financeira da empresa, outra vantagem é a melhoria na conformidade regulatória. Com todos os dados contábeis centralizados em um único lugar, fica mais fácil garantir a conformidade com as regulamentações fiscais e contábeis, gerando relatórios precisos para atender as obrigações legais.

3. PROPOSTA DE ARQUITETURA DE MICROSSERVIÇOS

Neste capítulo, é apresentada uma sugestão de arquitetura para o sistema ERP Contábil. levando em consideração as necessidades e o contexto em que ele vai operar. Serão expostos os componentes do sistema, as fronteiras entre eles e a descrição de como essa arquitetura poderia funcionar. Serão utilizados alguns conceitos do Design Orientado a Domínio, uma vez que eles são muito eficazes quando aplicados a microsserviços e possuem uma interação natural com eles. Além disso, serão consideradas as melhores práticas no desenvolvimento de microsserviços, visando criar uma proposta eficiente e escalável.

3.1. Objetivos do Sistema ERP Contábil

Antes de propor uma arquitetura, é bastante importante evidenciar e contextualizar o cenário em que a solução de software busca atuar. É preciso levantar os pontos cruciais para o projeto, buscando entender os objetivos, desafios e necessidades que se pretende resolver. O SERPC (Sistema ERP Contábil) busca oferecer a contabilidade como serviço para pequenas e médias empresas aderentes ao regime tributário do Simples Nacional.

O contexto empresarial abrange uma diversidade de atividades econômicas. Cada um desses possui características próprias e, em alguns casos, exigências contábeis distintas. Além disso, as empresas estão em constante evolução, se especializando e buscando novas formas de se destacar no mercado. Portanto, o sistema precisa acompanhar esse avanço e oferecer recursos específicos que permitam uma contabilidade mais eficiente e adequada às particularidades de cada segmento.

Visto essa variedade de regras, funcionalidades e módulos que o sistema deverá implementar, é necessário também observar a realidade, objetivos e dimensionamento do processo de desenvolvimento desse sistema, que, para lidar com essa gama de atividades, deverá ser construído de forma gradual e terá sua evolução a longo prazo. As regras de negócios para esse sistema estarão em constante mudança, e para controlar isso será preciso garantir a qualidade na manutenção ao longo do processo de construção.

A arquitetura proposta deve possibilitar a modularização das funcionalidades contábeis, facilitando a oferta da contabilidade de forma eficiente e integrada. Destacando a diversidade nas atividades econômicas das empresas, a necessidade de adaptação e evolução do sistema para atender aos diferentes setores e regras federativas que podem sofrer alterações ao longo do tempo. Esses pontos ressaltam a relevância de uma arquitetura flexível e modular para atender às demandas contábeis específicas das empresas e oferecer uma solução adaptável, eficiente e personalizada.

3.2. Análise e Levantamento dos Requisitos Funcionais

Após compreender o contexto e os objetivos do SERPC, torna-se evidente a importância de uma arquitetura robusta e flexível para atender às necessidades das pequenas e médias empresas que buscam serviços contábeis. Essas empresas estão inseridas em um ambiente empresarial diversificado, no qual cada setor possui particularidades e exigências contábeis distintas.

Por esse motivo, a arquitetura do sistema deve ser adaptável e capaz de evoluir para se adequar às particularidades de cada setor. É essencial que o SERPC seja capaz de oferecer recursos específicos que permitam uma contabilidade eficiente e personalizada, levando em consideração as diferentes demandas e exigências contábeis dos setores. Além disso, é importante reconhecer que o cenário contábil está em constante mudança. As empresas estão sempre buscando novas formas de se destacar no mercado e se especializando em suas áreas de atuação. Portanto, a arquitetura do SERPC precisa acompanhar essa evolução e oferecer recursos que possibilitem uma contabilidade atualizada e alinhada com as necessidades em constante transformação.

Para atender a esses desafios, a arquitetura do SERPC deve ser modular, permitindo a incorporação gradual de módulos específicos para cada setor, além dos módulos comuns a todas as empresas. Essa abordagem modular possibilita a personalização do sistema de acordo com as exigências contábeis de cada setor, ao mesmo tempo em que mantém uma base sólida de funcionalidades essenciais para todas as empresas, como o registro de transações financeiras, controle de estoque, emissão de notas fiscais e apuração de impostos.

Nesta seção, serão realizados uma análise aprofundada e o levantamento dos requisitos arquiteturais necessários para o SERPC. O objetivo é obter uma visão clara dos requisitos que nortearão o desenvolvimento do sistema. Essa análise fornecerá as bases necessárias para desenvolver o sistema de forma a se adequar aos requisitos específicos aos quais será exposto. O Quadro 3.1 evidencia os requisitos funcionais do sistema com foco inicial nas rotinas comuns aos diferentes segmentos empresariais aderentes ao Simples Nacional.

Quadro 3.2 – Requisitos funcionais do SERPC.

Requisito	**Descrição**
Pessoas	Cadastro, gerenciamento e manutenção
Empresas	Cadastro, gerenciamento e manutenção
Produtos	Cadastro, gerenciamento e manutenção
Fornecedores	Cadastro, gerenciamento e manutenção
Estoque	Gerenciamento de Estoque e Inventário
Emissão Fiscal de DFe	Documentos Fiscais Eletrônicos (NFe, NFCe, MDFe, DIFAL, CTe)
Recolhimento de Notas	Acompanhamento e Recolhimento de Documentos Fiscais Emitidos e Destinados.
Situação Cadastral	Monitoramento da situação cadastral nos 3 entes federativos.
PGDAS-D	Declaração eletrônica que calcula, transmite e gera o DAS (guia de recolhimento do imposto).
DIRF	Declaração do Imposto de Renda Retido na Fonte. Devem ser gerados os informes de rendimento das Pessoas Físicas Vinculadas.
DIRPF	Declaração de Imposto de Renda Pessoa Física.
DEFIS	Declaração de Informações Socioeconômicas e Fiscais.
RAIS	Relação Anual de Informações Sociais;
SEFIP / GFIP	Sistema Empresa de Recolhimento do FGTS. Repasse e informações do FGTS.
GIA / GNRE	Recolhimento do ICMS fora do Simples, para empresas que extrapolaram o teto de R$3,6 milhões de faturamento anual.
DeSTDA	Declaração de Substituição Tributária, Diferencial de Alíquota e Antecipação
SINTEGRA	Sistema Integrado de Informações sobre Operações Interestaduais com Mercadorias e Serviços
Folha de Pagamento	Crédito de Salários e Pró-labore
Admissão e Demissão	Envio e cadastro da documentação e procedimentos.
Repasse INSS	Repasse da contribuição de INSS (Funcionários e Sócios)
Repasse IR	Repasse do Imposto Retido (Funcionários e Sócios)
Livro Caixa	Movimentação financeira da empresa (Caixa e Bancos).

Livro Registro de Entrada	Entrada de Mercadorias na empresa.
Livro Registro de Saída	Saída de Mercadorias da empresa.
Livro Registro de Inventário	Saldo remanescente do Estoque.
Livro Registro dos Serviços Prestados	Registro dos Serviços Prestados e respectivos documentos fiscais.
Relatórios Contábeis	Balanço, DRE e outros.
Alvarás e Impostos no âmbito municipal	Impostos e Taxas diversos (IPTU, Taxa de Incêndio, Vigilância Sanitária, etc.)

Fonte: (Elaborado pelo autor, 2023)

Este quadro apresenta os requisitos funcionais básicos para o atendimento das empresas do Simples Nacional, abrangendo desde os cadastros de pessoas, empresas, produtos e fornecedores até funções específicas, como emissão de documentos fiscais eletrônicos, acompanhamento e recolhimento de notas fiscais, monitoramento da situação cadastral, declarações obrigatórias, etc. Esses requisitos são fundamentais para garantir a eficiência e a conformidade das atividades contábeis das empresas. Além disso é fundamental que o sistema possua também outros requisitos que não estão atrelados a contabilidade, mas que são essenciais para a garantia da eficiência, segurança e desempenho do sistema. O quadro 3.2 evidencia esses requisitos.

Quadro 3.1 – Requisitos essenciais para o ciclo de vida do SERPC.

Requisito	Descrição
Segurança	Garantir a segurança dos dados sensíveis das empresas, implementar mecanismos de autenticação e controle de acesso.
Desempenho	Deve ser capaz de lidar com uma grande quantidade de solicitações para atender vários usuários simultaneamente.
Confiabilidade	Disponibilidade contínua, minimizando a ocorrência de falhas, erros e interrupções no funcionamento do sistema.
Usabilidade	Interface intuitiva de fácil utilização e compreensão pelos usuários
Adaptabilidade	A arquitetura deve ser flexível e permitir a integração com outros sistemas.
Manutenibilidade	A arquitetura deve permitir atualizações, correções e implementações de novas funcionalidades de maneira ágil.

Integridade	Deve assegurar a qualidade das informações registradas, evitando a ocorrência de inconsistências ou corrupção das informações.
Escalabili-dade	O sistema deve ser capaz de adaptar-se ao crescimento da sua base de utilizadores, suportando esse aumento sem comprometer o desempenho.

Fonte: (Elaborado pelo autor, 2023)

Esses requisitos são essenciais para garantir o bom funcionamento, a confiabilidade e a eficiência do sistema, proporcionando uma experiência de uso satisfatória para as empresas e usuários envolvidos em todo o processo da aplicação.

3.3. Classificação e Configuração dos Serviços

Com base nos requisitos arquiteturais identificados, será realizada uma análise detalhada para agrupar os requisitos que visam a mesma funcionalidade. Segundo Newman (2021):

> [...] uma definição sucinta para a coesão em microsserviços é "Códigos que mudam juntos devem permanecer juntos" (NEWMAN, 2021, p. 63).

Com o objetivo de criar um sistema coeso, serão identificados requisitos que compõem uma mesma funcionalidade ou que possuam cooperação harmônica entre si, afim de delimita-los e nomeá-los.

Para promover a coesão do sistema será adotado o princípio da Linguagem Ubíqua, originário do DDD (Domain-Driven Design), para nomear os serviços que comporão o sistema. Esse princípio busca garantir que as referências utilizadas no desenvolvimento do software sejam consistentes com o modelo do domínio do negócio, evitando ambiguidades e mal-entendidos durante o processo de desenvolvimento.

Dessa forma, o SERPC será estruturado como um conjunto de microsserviços, em que cada um corresponderá a aspectos distintos das funcionalidades do sistema. Os microsserviços serão nomeados de acordo com o contexto e a funcionalidade que representam no domínio do negócio. O Quadro 3.3 destaca alguns dos serviços componentes do SERPC, todos nomeados e organizados seguindo a abordagem descrita anteriormente.

Quadro 3.2 – Serviços componentes do SERPC.

Nome do Serviço	Descrição

Contratação	Este componente, será responsável por gerenciar processos de admissão, férias e demissão de funcionários.
Repasse	Este componente lidará com repasses obrigatórios para entidades governamentais, incluindo INSS, FGTS e Imposto de Renda.
Folha de Pagamento	Este componente gerenciará a folha de pagamento, que inclui o cálculo de salários, deduções e benefícios para cada funcionário.
RAIS	Este componente será responsável pelo envio da Relação Anual de Informações Sociais.
Livros	Este componente será responsável pela manutenção dos livros contábeis, que inclui o livro caixa, o registro de entradas e saídas, o registro de inventário e o registro de serviços prestados.
Relatórios	Este componente gerenciará a geração de relatórios contábeis, que são necessários para fins de revisão interna e auditoria.
Auditoria	Este componente será responsável pela auditoria dos livros e relatórios contábeis.
DFe	Este componente será responsável pela emissão de vários tipos de documentos fiscais eletrônicos, incluindo NFe, NFCe, NFSe, MDFe, CTe e DIFAL.
Recolhimento de Notas	Este componente fará o recolhimento de notas fiscais emitidas e destinadas.
Simples Nacional	Este componente será responsável pela gestão do regime tributário Simples Nacional, incluindo rotinas como PGDAS, DEFIS, deSTDA, e o recolhimento de ICMS fora do Simples Nacional.
DIR	Este componente será responsável pela emissão das declarações de impostos DIRF e DIRPF.

Fonte: (Elaborado pelo autor, 2023)

Esses componentes representam a estrutura básica do SERPC para atender às empresas do Simples Nacional. À medida que o sistema evolui e as necessidades das empresas mudam, novos serviços podem ser adicionados para atender a essas demandas. A abordagem de microsserviços proporciona uma arquitetura modular e flexível, sendo uma das principais vantagens dessa abordagem de construção de software, o que permitirá que o sistema seja adaptado e evoluído para atender às necessidades em constante mudança do ambiente empresarial. Cada um desses componentes será desenvolvido como um microsserviço independente, possibilitando uma maior agilidade na implementação de mudanças e atualizações com o mínimo de impacto em outros sistemas. Além disso, cada microsserviço será projetado para ser escalável, garantindo um alto desempenho mesmo ao lidar com uma grande quantidade de solicitações simultâneas.

3.4. Definição das Fronteiras

Na arquitetura de microsserviços, é consensual que o acoplamento deve ser minimizado. Entretanto, isso não implica sua total inexistência. Para estabelecer limites claros e objetivos para cada microsserviço, o SERPC empregará o princípio do DDD de contexto delimitado (bounded context). San Newman em seu livro “Criando Microsserviços” fala que:

> Um contexto delimitado (*bounded context*) em geral representa uma fronteira organizacional maior. No escopo dessa fronteira, responsabilidades explícitas devem ser colocadas em prática. [...] Os contextos delimitados contêm um ou mais agregados. Alguns agregados podem ser expostos para fora do contexto delimitado; outros poderão estar ocultos internamente. Como no caso dos agregados, os contextos delimitados podem ter relacionamentos com outros contextos delimitados – quando são mapeados aos serviços, essas dependências se tornam dependências entre serviços. (NEWMAN, 2021, p. 82, 83)

A implementação deste princípio na arquitetura do SERPC significa que cada microsserviço será projetado para atender a uma funcionalidade específica, garantindo assim a clareza e a eficiência das operações. Porém, por mais que o acoplamento seja evitado, existirá circunstâncias em que será aceitável, ou até mesmo necessário, a interdependência entre microsserviços. Nesses casos será visto o acoplamento de domínio que poderá ocorrer entre microsserviços que atuam em um objetivo comum.

> Um acoplamento de domínio descreve uma situação na qual um microsserviço deve interagir com outro microsserviço porque o primeiro microsserviço precisa utilizar a funcionalidade que o outro microsserviço oferece (NEWMAN, 2021, p. 67).

O SERPC adotará a mesma divisão de responsabilidades em departamentos que a contabilidade utiliza, sendo eles: departamento pessoal, fiscal e contábil. Essa divisão será implementada no sistema com o objetivo de alinhar-se aos objetivos de negócio e garantir uma estrutura organizada e eficiente para lidar com as diferentes áreas da contabilidade. Dessa forma, o SERPC utilizará a Lei de Conway para atribuir suas fronteiras, criando microsserviços que refletirão a estrutura organizacional e as responsabilidades da contabilidade. Na figura 3.1 é evidenciada a divisão de fronteiras do SERPC.

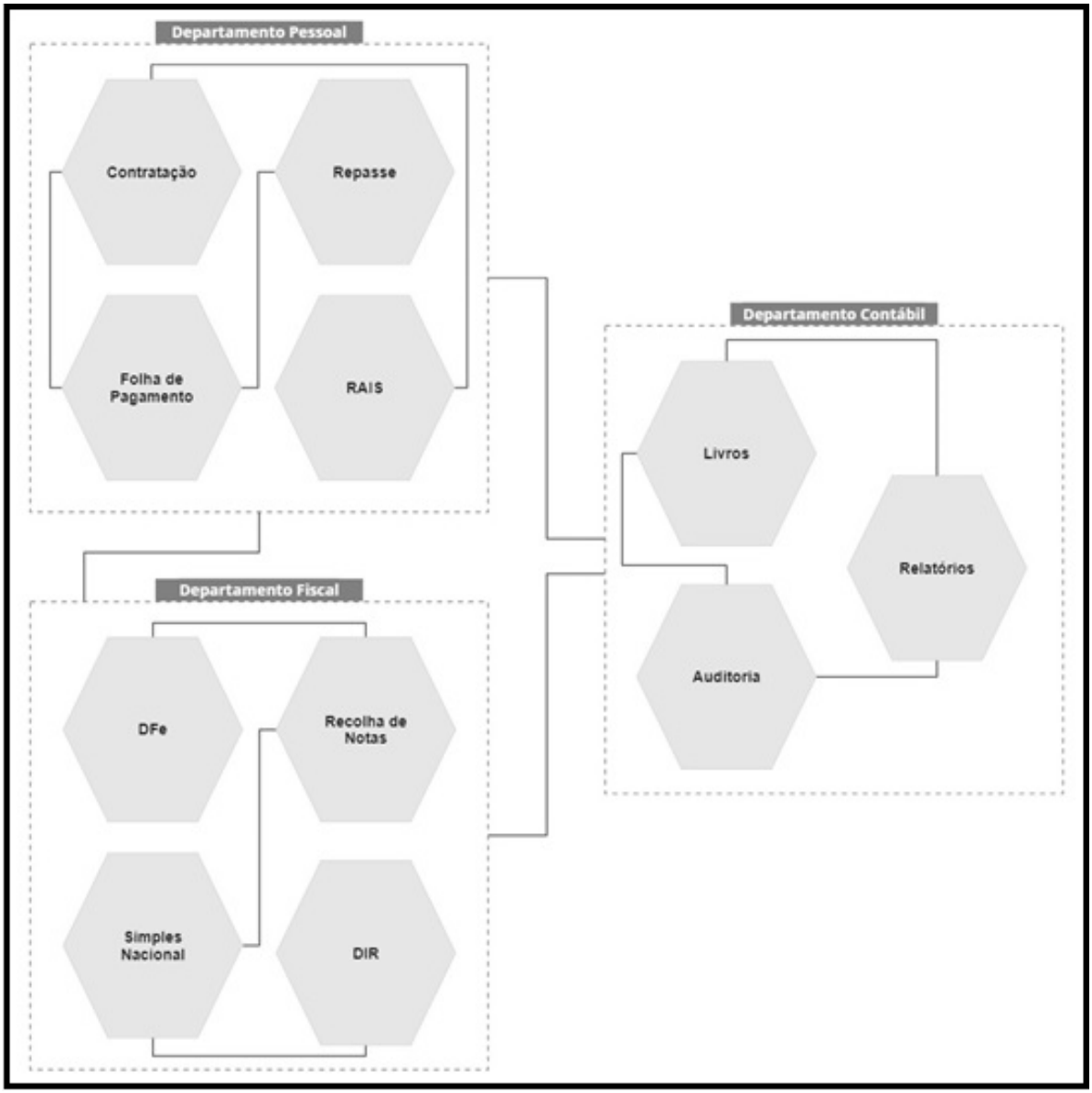

Figura 3.1 – Diagrama das Fronteiras do SERPC.
Fonte: (Elaborado pelo Autor, 2023)

A adoção desses conceitos fortalecerá a arquitetura do SERPC, permitindo que ela seja modular e eficiente, com microsserviços que podem funcionar de forma autônoma, mas ainda assim colaborar quando necessário para alcançar objetivos comuns. Isso facilitará a manutenção, a escalabilidade e a evolução do sistema ao longo do tempo.

3.5. Comunicação e Integração Entre os Microsserviços

Uma das principais características de um sistema baseado em microsserviços é a sua capacidade de permitir a comunicação e a integração entre os diversos serviços. Para o SERPC, a abordagem escolhida prioriza o uso de mensageria para a comunicação interna sempre que possível, proporcionando uma série de benefícios, incluindo desacoplamento, escalabilidade e tolerância a falhas.

3.5.1. Comunicação Interna

A comunicação interna entre os microsserviços do SERPC será preferencialmente realizada através de um sistema de mensageria utilizando um broker de mensagens. Este sistema permitirá a troca de informações de maneira eficiente e desacoplada, onde os microsserviços produzem e consomem mensagens de forma independente. Essa abordagem favorece a resiliência e a escalabilidade do sistema, pois os microsserviços não precisam estar disponíveis ao mesmo tempo para a comunicação ser efetiva.

> [...]Basicamente, os brokers oferecem alguns recursos que podem ser muito convenientes em uma comunicação assíncrona. As propriedades que eles oferecem variam, mas o recurso mais interessante é garantia de entrega, algo que todos os Brokers amplamente utilizados proporcionam de alguma forma. uma garantia de entrega descreve o compromisso do broker em garantir que uma mensagem seja entregue. Do ponto de vista do microsserviço que envia a mensagem, isso pode ser muito útil. O fato de destinatário *downstream* está indisponível não será um problema - o Broker armazenará a mensagem até que ela possa ser entregue. Com isso, podemos reduzir o número de itens com os quais os microsserviços *upstream* precisam se preocupar (NEWMAN, 2021, p. 169).

Entretanto, existirá situações onde a comunicação direta será necessária, especialmente em contextos onde o processamento de um serviço é diretamente depende do resultado de outro. Para esses casos, o SERPC poderá utilizará APIs REST para a comunicação síncrona entre os serviços.

3.5.2. Comunicação Externa

Para a comunicação com sistemas externos, o SERPC inicialmente adotará o padrão API Gateway. Este padrão atuará como um único ponto de entrada para todas as comunicações externas, oferecendo uma interface unificada para os vários microsserviços que compõem o sistema. A utilização do API Gateway simplifica a integração

com outros sistemas e oculta a complexidade inerente à distribuição dos microsserviços Newman (2021) fala que:

> [...]Ter algum tipo de gateway de agregação pode reduzir o número de chamadas que o cliente externo precisa fazer reduzindo a quantidade de dados a ser enviada de volta. Isso pode resultar em vantagens significativas quanto à redução no uso de bando e a melhoria na latência da aplicação (NEWMAN, 2021, p. 555).

A utilização do API Gateway no SERPC ajudará a reduzir a complexidade para os clientes externos, que não precisarão estar cientes de todos os microsserviços individuais, seus endereços ou detalhes de implementação. Em vez disso, eles irão interagir com um único ponto de entrada que gerenciará e direcionará as solicitações de forma transparente.

Além disso, a implementação do API Gateway oferece benefícios adicionais em termos de segurança e desempenho. Este padrão permite a implementação de recursos como criptografia, balanceamento de carga e cache, melhorando a eficiência e a segurança da comunicação entre a aplicação e os clientes externos.

3.5.3. Descoberta de Serviços

A fim de permitir que os microsserviços localizem uns aos outros de forma eficiente, o SERPC implementará um sistema de descoberta de serviços. Este sistema manterá um registro atualizado de todos os microsserviços e seus endereços, facilitando a comunicação dinâmica entre eles.

3.6. Definição de Arquitetura de Microsserviços

Este capítulo aborda a definição da arquitetura de microsserviços do Sistema, discorrendo sobre os princípios e padrões arquiteturais que orientarão sua estrutura e design. O SERPC será projetado como um ecossistema de microsserviços, constituído por um conjunto coeso de serviços autônomos que, embora funcionem de forma independente, compõem em conjunto um sistema completo.

Os princípios que orientarão a arquitetura do SERPC são fundamentados no *Domain-Driven Design*, uma abordagem que prioriza o alinhamento da implementação do software com seus respectivos domínios de negócio. Isso significa que cada mi-

crosserviço será projetado para se especializar em um contexto específico, o que permitirá que ele evolua de forma independente e seja otimizado de acordo com suas responsabilidades específicas. Isso não apenas promove a agilidade e a resiliência do sistema, mas também garante que cada microsserviço seja a melhor versão de si mesmo, atendendo às suas responsabilidades de forma eficiente. A Figura 3.2 exemplifica essa aplicação dos conceitos do DDD na definição dos microsserviços, focando no departamento fiscal, um dos domínios de negócios abordados pelo SERPC.

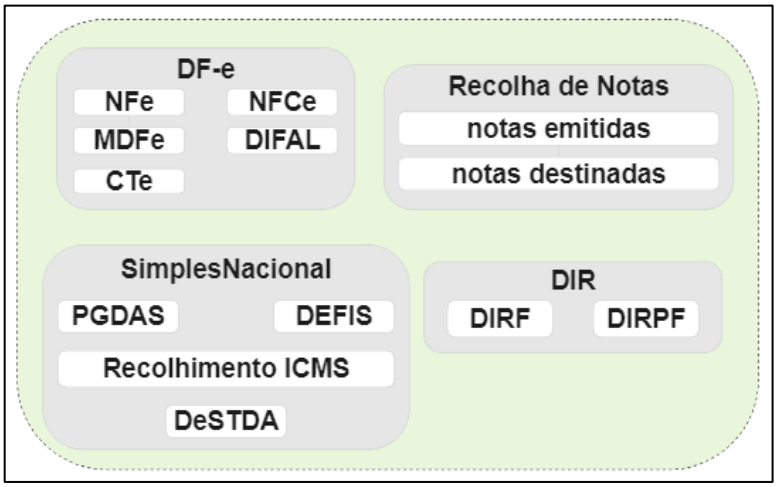

Figura 3.2 – Diagrama das Fronteiras da zona fiscal do SERPC.
Fonte: (Elaborado pelo Autor, 2023)

Na arquitetura proposta, a autonomia de cada microsserviço estende-se além das fronteiras de negócio, alcançando a escolha de tecnologias e linguagens de programação que permitam sua execução e otimização de forma mais eficaz, de acordo com as características específicas dos processamentos que realiza.

A fim de promover o desacoplamento e facilitar a interoperabilidade entre os serviços, será adotado o uso de interfaces genéricas que proporcionem a ocultação de informações detalhadas sobre a implementação dos serviços.

Em termos de comunicação, será dada preferência para a interação assíncrona por meio de um barramento de mensagens, favorecendo a resiliência, controle e rastreabilidade das operações. O Padrão API Gateway será adotado para fornecer um único ponto de entrada para todas as comunicações externas com o sistema.

Para a comunicação interna entre os microsserviços do SERPC, a estratégia predominante será a utilização de um sistema de mensageria, porém poderão ser utilizadas APIs REST para a comunicação síncrona entre os serviços quando necessá-

rio. Esta abordagem permitirá um alto grau de flexibilidade e adaptabilidade na interação entre os microsserviços. A Figura 3.3 evidencia a organização da Arquitetura de Microsserviços proposta, contendo os seus principais componentes.

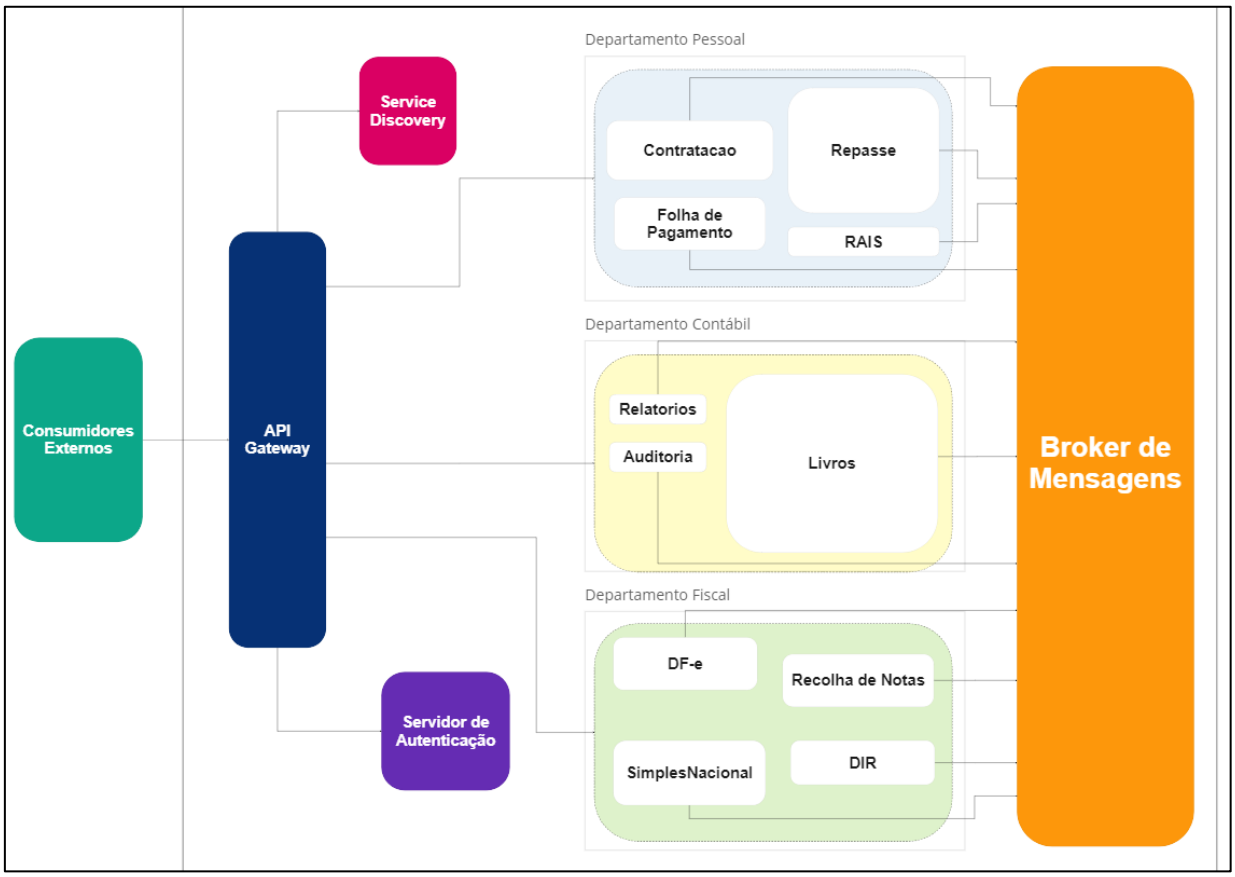

Figura 3.3 – Diagrama dos Componentes do SERPC.
Fonte: (Elaborado pelo Autor, 2023)

3.7. Tecnologias Para o Ecossistema de Microsserviços

A escolha das tecnologias é um ponto muito importante para o projeto do SERPC. Essa decisão deve levar em conta o que mais se adequa ao contexto do sistema e que favoreça a implementação da arquitetura de microsserviços, deve ser observado o contexto da equipe, quais as suas especialidades, tamanho e experiências. A figura 3.4 mostra os componentes do SERPC e suas tecnologias:

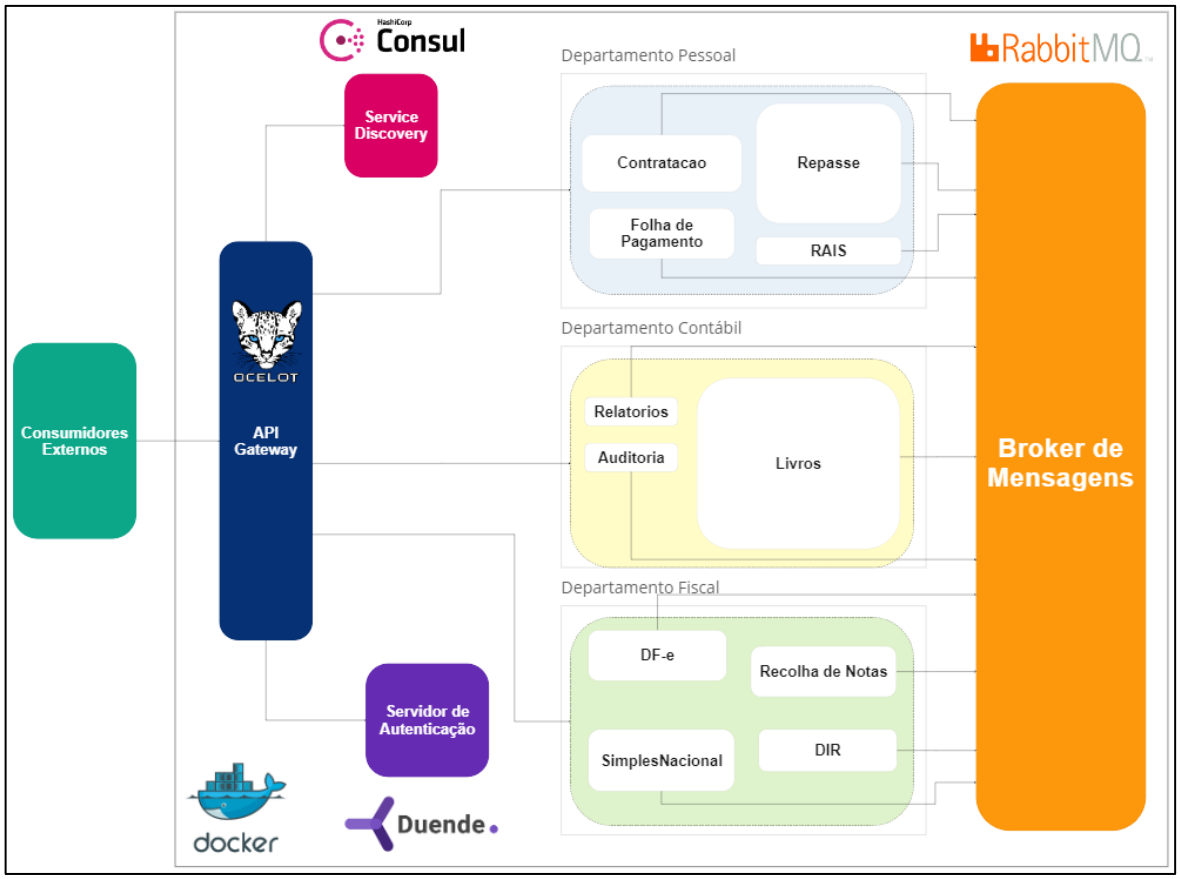

Figura 3.4 – Diagrama do SERPC com suas tecnologias.
Fonte: (Elaborado pelo Autor, 2023)

3.7.1. .NET

Para a construção do SERPC, visando proporcionar uma padronização inicial nos microsserviços, garantir um maior controle de qualidade e facilidade na construção dos módulos, será adotado o ecossistema .NET para a criação desses microsserviços. O ecossistema de microsserviços será composto por tecnologias específicas que desempenham funções essenciais na arquitetura.

O .NET foi escolhido para a construção do SERPC levando em conta as especialidades da equipe, focando em benefícios como a facilidade na construção dos módulos e a padronização inicial. A sua maturidade e suporte pela Microsoft também foram fatores decisivos.

3.7.2. Ocelot

O API Gateway Ocelot será adotado para controle de acessos externos. Esta ferramenta atuará como um "porteiro eletrônico", recebendo todas as solicitações dos

clientes e as encaminhando ao microsserviço correspondente. Esse recurso ajudará a manter o sistema organizado, simplificará sua manutenção e permitirá sua expansão de maneira escalável.

O Ocelot é um API Gateway maduro e altamente personalizável, e foi escolhido para controlar os acessos externos devido à sua facilidade de configuração e excelente integração com o .NET.

3.7.3. Duende Identity Server

A autenticação e autorização dos usuários serão geridas pelo Duende Identity Server. Esta solução será responsável por verificar a identidade dos usuários e controlar o acesso aos microsserviços, assegurando a segurança do sistema e permitindo que apenas indivíduos autorizados acessem as funcionalidades.

O Duende Identity Server foi escolhido para gerir a autenticação e autorização dos usuários, considerando sua segurança comprovada, flexibilidade e excelente integração com o ecossistema .NET.

3.7.4. Consul

O Consul será empregado para descoberta e registro de serviços. Este recurso funciona como uma agenda telefônica para os microsserviços, permitindo que se descubram e se comuniquem de forma eficiente. Isso é crucial para manter o sistema robusto e capaz de suportar o crescimento da aplicação sem grandes obstáculos.

O Consul foi selecionado para descoberta e registro de serviços devido à sua robustez, facilidade de uso e capacidade de se adaptar a ambientes de alta disponibilidade.

3.7.5. RabbitMQ

Para a comunicação entre microsserviços, será utilizado o RabbitMQ como broker de mensagens. Essa tecnologia atua como um intermediário entre os microsserviços, possibilitando uma comunicação eficaz, resiliente e confiável entre eles, mesmo que não estejam sendo executados ao mesmo tempo.

O RabbitMQ foi selecionado como broker de mensagens pela sua capacidade de assegurar uma comunicação eficaz, resiliente e confiável entre os microsserviços, dada a sua maturidade e robustez comprovada.

3.7.6. Docker

Para a criação de ambientes de desenvolvimento, serão utilizados contêineres Docker. O Docker permite empacotar o software e todas as suas dependências em um "contêiner", facilitando a transferência do software de um ambiente para outro e assegurando sua execução de maneira consistente e confiável em qualquer ambiente.

O Docker foi escolhido para a criação de ambientes de desenvolvimento, levando em conta as especialidades da equipe, sua popularidade e pela sua capacidade de empacotar o software e todas as suas dependências de maneira consistente e confiável.

3.7.8. Considerações Finais Acerca das Tecnologias

Com a escolha dessas tecnologias, espera-se criar um ambiente robusto e escalável para o desenvolvimento do SERPC, garantindo a eficiência, a segurança e a adaptabilidade necessárias para atender às demandas de um sistema de planejamento de recursos empresariais contábil.

4. IMPLEMENTAÇÃO DA PROVA DE CONCEITO

Este capítulo tem como objetivo descrever o estado atual da implementação do SERPC.

No início, foi estabelecida a configuração preliminar do projeto, responsável por delinear a estrutura que proporcionará o ambiente adequado para a realização do projeto. Posteriormente, o ambiente de desenvolvimento foi apropriadamente configurado, com a instalação do broker de mensagens e do servidor de descoberta.

Após a configuração inicial, procedeu-se à criação dos módulos, configurando assim a base para o desenvolvimento dos vários serviços que constituem o software ERP Contábil. Cada módulo foi desenhado para corresponder a uma funcionalidade específica do sistema, favorecendo assim a organização e a manutenção otimizada do código. Essa estratégia também permite que cada módulo seja desenvolvido, atualizado ou escalonado de forma independente. A Figura 4.1 mostra a estrutura do SERPC seus componentes e microsserviços.

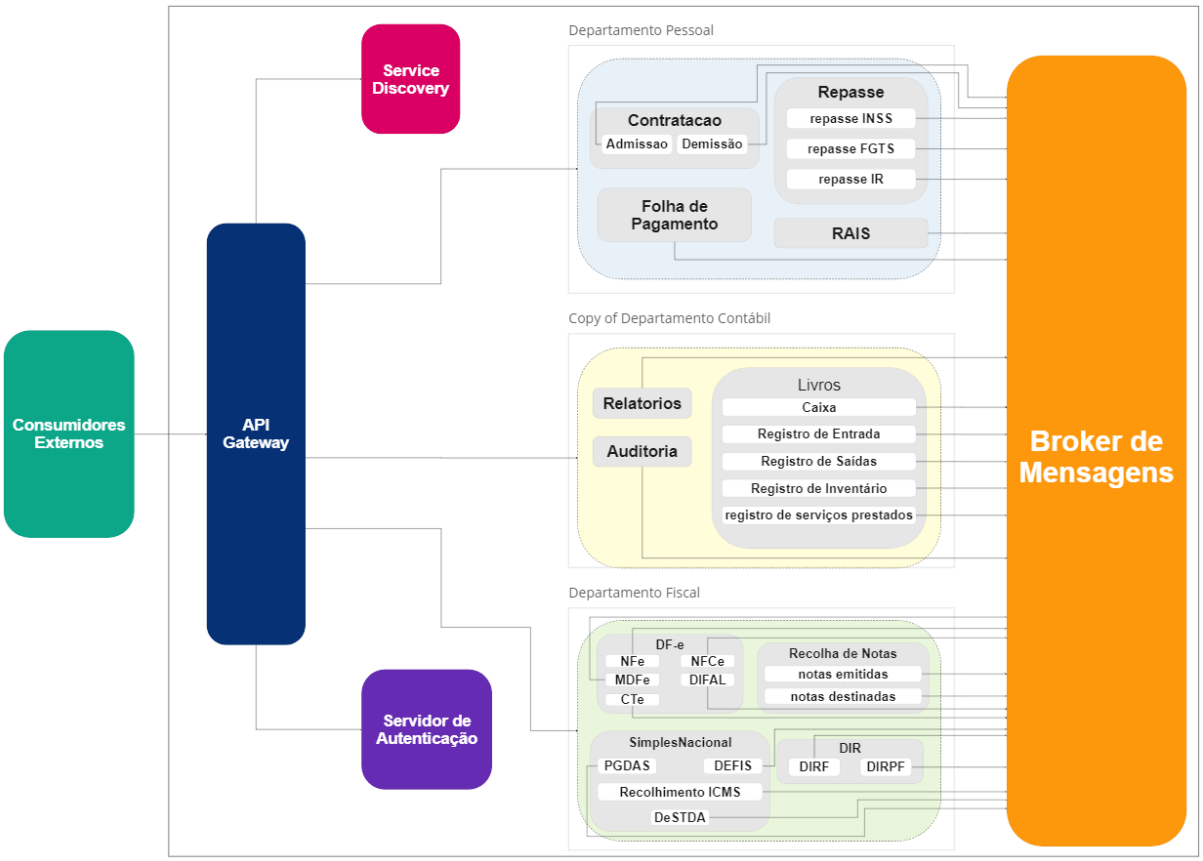

Figura 4.1 – Composição de modulos do SERPC.
Fonte: (Elaborado pelo Autor, 2023)

O sistema está sendo desenvolvido utilizando a IDE Visual Studio, uma ferramenta poderosa e completa da Microsoft para a criação, desenvolvimento e teste de aplicativos em diversas plataformas e linguagens de programação. No contexto do Visual Studio, uma "solução" é um contêiner que reúne múltiplos projetos relacionados, permitindo que sejam gerenciados e desenvolvidos em conjunto.

Neste projeto, a solução do Visual Studio foi utilizada para estruturar as fronteiras do sistema e criar os projetos para todos os respectivos microsserviços. A figura 4.2 ilustra a disposição dos projetos de microsserviços dentro da solução.

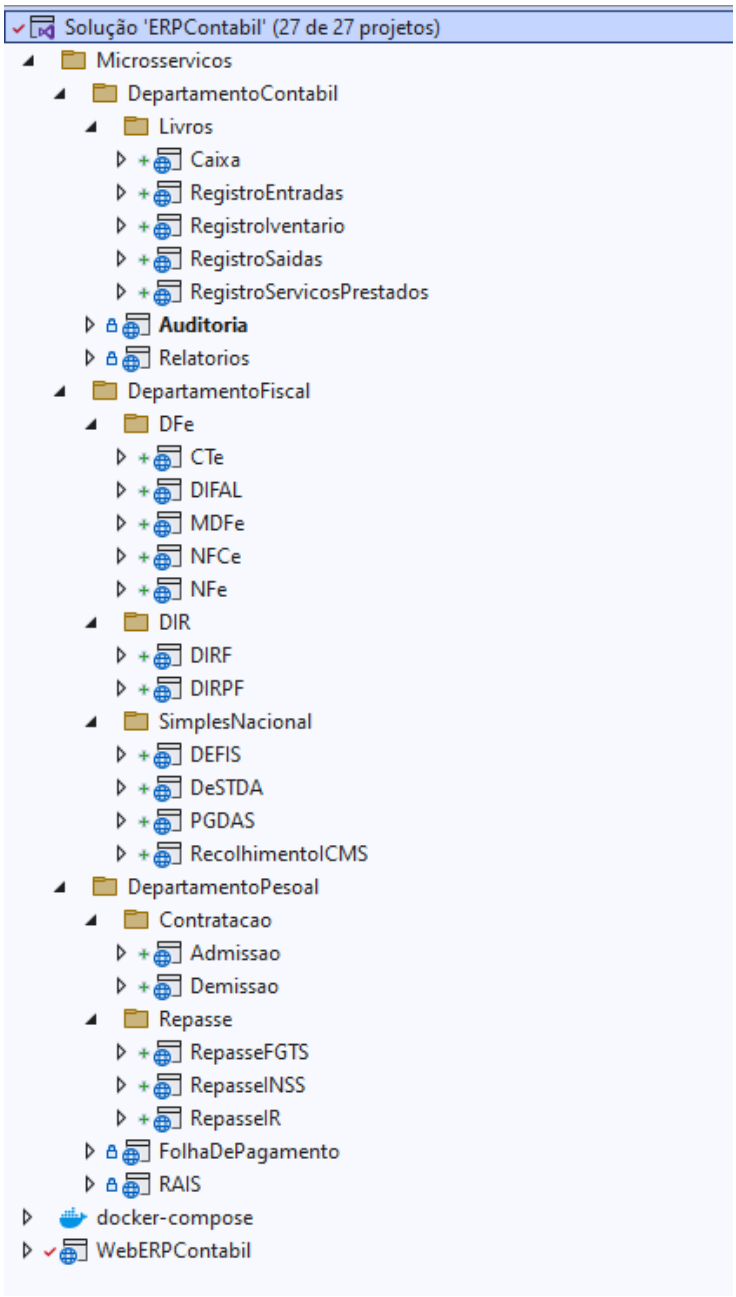

Figura 4.2 – Solução do Visual Studio.
Fonte: (Elaborado pelo Autor, 2023)

É importante ressaltar que, cada um desses projetos, apesar de estarem reunidos em uma única solução, podem ser executados e desenvolvidos de forma independente. Isso ressalta uma das principais vantagens da arquitetura de microsserviços, que é a possibilidade de desenvolver e implantar serviços de forma independente, proporcionando flexibilidade e agilidade no desenvolvimento do sistema.

Para provisionar os recursos de infraestrutura, está sendo implementado o uso de containers Docker. A adoção dessa estratégia não visa apenas facilitar o desenvolvimento, mas também possibilitar a simulação de um ambiente de produção de maneira rápida e confiável. A Figura 4.3 apresenta a interface do Docker Desktop, ferramenta que contribui para a visualização e controle dos containers em execução.

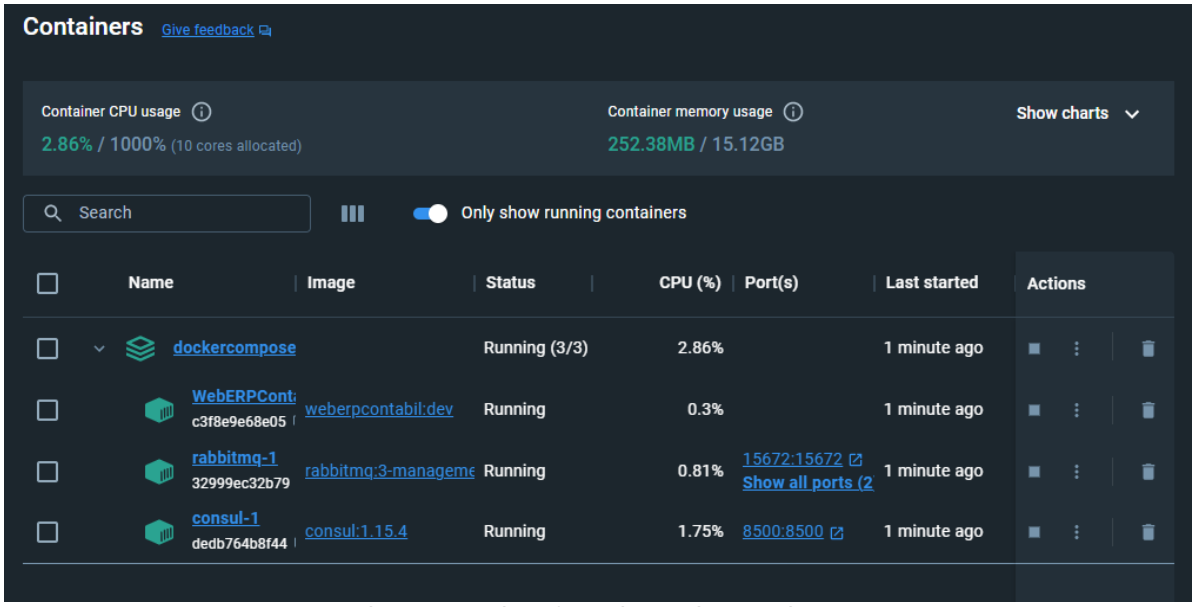

Figura 4.3 – interface do Docker Desktop.
Fonte: (Elaborado pelo Autor, 2023)

Para possibilitar a comunicação assíncrona entre os serviços, está sendo configurada a utilização do broker de mensagens RabbitMQ. Esse mecanismo irá permitir a troca de mensagens eficiente entre os microsserviços, facilitando a sincronização de dados e a coordenação das atividades do sistema. A figura 4.4 apresenta a interface do RabbitMQ.

Figura 4.4 – Interface de gerenciamento do RabbitMQ.
Fonte: (Elaborado pelo Autor, 2023)

Também está sendo realizada a configuração da descoberta automática de serviços utilizando o Consul, essa implementação vai possibilitar que cada serviço se registre autonomamente no catálogo de serviços. Sendo usada para garantir a visibilidade de todos os serviços integrados ao sistema ERP Contábil, permitindo monitorar seu estado e desempenho em tempo real. A figura 4.5 exibe a interface de gerenciamento do Consul, destacando alguns dos serviços do SERPC em operação.

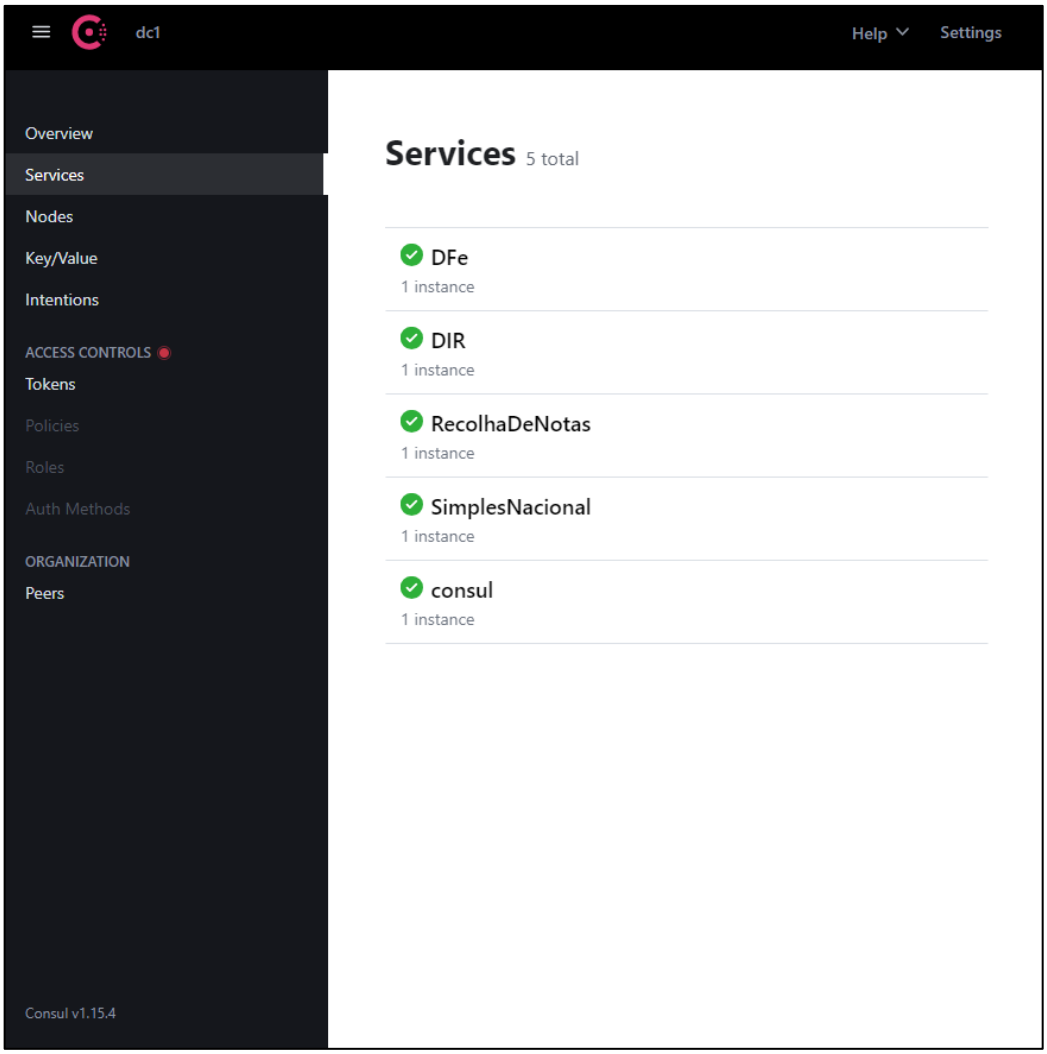

Figura 4.5 – Interface de gerenciamento do Consul.
Fonte: (Elaborado pelo Autor, 2023)

Além disso, foi desenvolvida uma versão inicial simplificada de uma interface de usuário para o sistema ERP Contábil. O objetivo desta é simular o acesso externo aos endpoints de acesso do API Gateway, possibilitando que os usuários interajam com o sistema de maneira intuitiva e eficiente. A figura 4.6 mostra uma das telas da interface que foi criada.

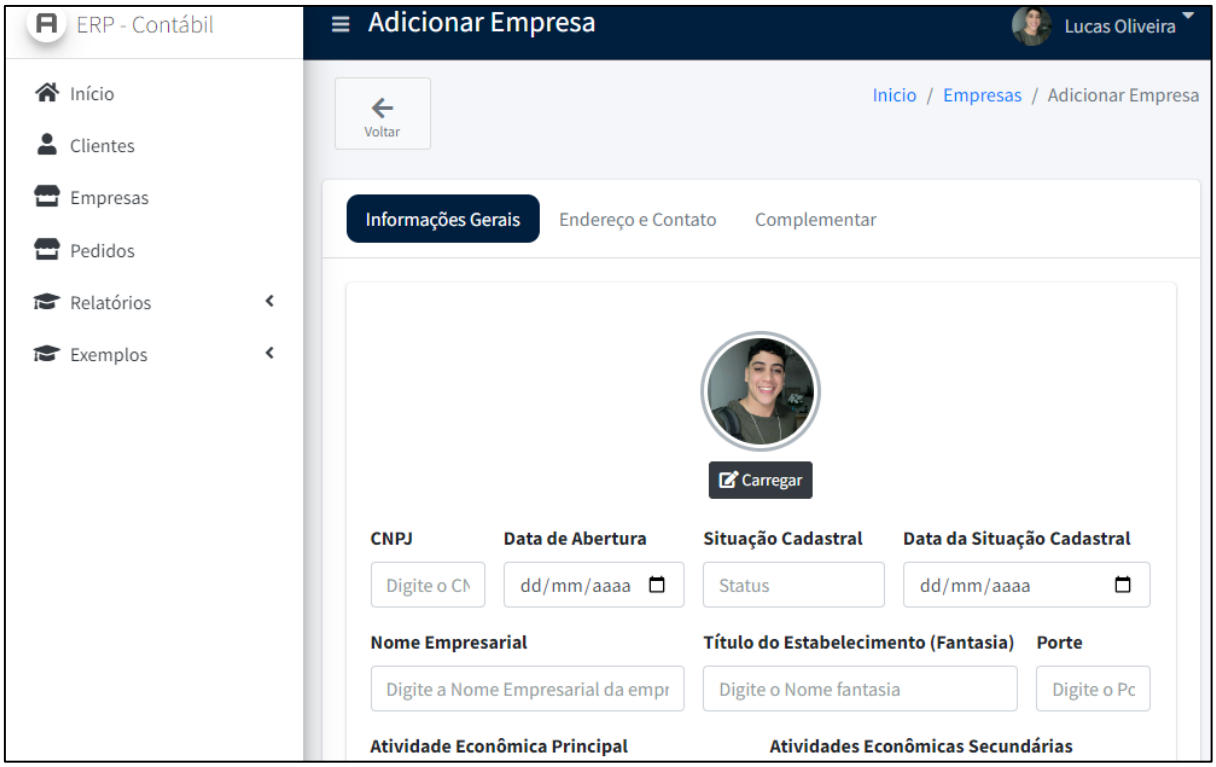

Figura 4.6 – interface gráfica que consome os endpoints externos do SERPC.
Fonte: (Elaborado pelo Autor, 2023)

É importante salientar que este trabalho é um projeto em constante evolução e constitui uma aplicação prática dos conceitos anteriormente apresentados. À medida que o desenvolvimento avança, o sistema continuará sendo expandido e aperfeiçoado, com o propósito de sempre aprimorar a eficiência e a usabilidade do software.

Todo o código-fonte deste projeto estará acessível publicamente para análise e contribuições. para isso o Git será adotado para controle de versões. Essa ferramenta registra todas as alterações feitas no código-fonte, facilitando o rastreamento e o gerenciamento das mudanças ao longo do tempo, além de viabilizar o trabalho colaborativo entre os desenvolvedores. O código-fonte do projeto estará acessível no seguinte repositório do GitHub: https://github.com/kas-oss/ERPContabil.git.

5. DISCUSSÃO E CONSIDERAÇÕES FINAIS

A análise detalhada sobre a arquitetura de microsserviços proporcionou contribuições significativas ao entendimento das práticas eficientes para o desenvolvimento de aplicações baseadas nessa abordagem. Identificamos pontos relevantes, trazendo à tona uma compreensão mais aprofundada dos benefícios e desafios associados a essa metodologia. Com o estudo dos requisitos específicos do contexto contábil em pequenas empresas, foi possível reconhecer as necessidades e desafios particulares desse setor, possibilitando uma abordagem mais adequada e focada.

Este estudo possibilitou a observação da grande quantidade de ferramentas e tecnologias já desenvolvidas para monitorar, controlar e criar ambientes propícios ao desenvolvimento de aplicações que utilizam a arquitetura de microsserviços. Mesmo que o conceito de microsserviços não seja recente, sua aplicabilidade se mantém contemporânea e relevante no contexto de desenvolvimento de aplicações, dada a sua característica principal: a flexibilidade, que se encaixa perfeitamente no dinamismo dos negócios atuais.

A aplicação dos conceitos estudados para criar uma proposta de arquitetura para um sistema ERP Contábil, baseado em microsserviços, demonstrou ser promissora. O potencial desse padrão arquitetural para oferecer benefícios significativos em termos de escalabilidade, resiliência e agilidade na entrega de novas funcionalidades é evidente.

A abordagem também se mostra como uma solução adequada para lidar com as demandas variáveis do mercado e os diferentes segmentos de negócios. A flexibilidade proporcionada pela arquitetura de microsserviços permite uma rápida e eficiente adaptação do sistema às mudanças nas necessidades e requisitos das empresas, assegurando uma maior agilidade e capacidade de resposta. Ao observar a divisão da contabilidade em diferentes serviços, fica evidente que cada departamento possui demandas e necessidades únicas. Portanto, a adoção da arquitetura de microsserviços em um sistema ERP contábil surge como uma estratégia promissora para a integração efetiva das rotinas contábeis com as operações empresariais.

No entanto, é crucial enfatizar que a implementação bem sucedida dessa arquitetura exige um planejamento cuidadoso, levando em conta as particularidades de cada empresa e a adoção das melhores práticas relacionadas aos microsserviços. Embora para esse contexto, a proposta de arquitetura seja promissora, cada negócio

possui suas peculiaridades e, portanto, os resultados podem variar. A implementação de uma arquitetura de microsserviços envolve um nível de complexidade e um investimento inicial em tempo e recursos que não devem ser subestimados.

Assim, este estudo serve como um ponto de partida para futuros desenvolvimentos de sistemas ERP focados em pequenas empresas, incentivando a busca por soluções que promovam a eficiência operacional e o crescimento sustentável dessas empresas através da integração contábil. Tal abordagem traz vantagens significativas para a gestão eficaz de recursos e a manutenção da integridade operacional.

6. TRABALHOS FUTUROS

Os trabalhos futuros nesta área de pesquisa focarão na continuidade do estudo da arquitetura de microsserviços, com ênfase no aprofundamento de conceitos fundamentais para o desenvolvimento e manutenção de sistemas de microsserviços. Há vários tópicos relacionados que ainda requerem aprofundamento e que prometem oferecer perspectivas interessantes para aprimorar a arquitetura do ERP Contábil.

Entre esses tópicos, destacam-se conceitos importantes como Command Query Responsibility Segregation (CQRS), SAGAs, Integração Contínua e Entrega Contínua (CI/CD), além dos princípios de orquestração e coreografia. Cada um desses elementos pode contribuir significativamente para a evolução da arquitetura dos microsserviços, oferecendo novas maneiras de melhorar a eficiência, a escalabilidade e a flexibilidade do sistema.

Além disso, pretende-se explorar o uso do Kubernetes, uma ferramenta de orquestração de contêineres de código aberto que permite uma maior escalabilidade e disponibilidade do sistema. O estudo aprofundado do Kubernetes pode oferecer um melhor entendimento de como gerenciar e escalar aplicativos baseados em microsserviços, promovendo um desempenho mais eficiente e robusto do sistema.

Com a continuação desses estudos, pretende-se não apenas aprofundar o entendimento desses conceitos, mas também aplicar esses conhecimentos de maneira prática para evoluir continuamente a arquitetura do ERP Contábil. Dessa forma, busca-se contribuir de forma mais abrangente para a evolução das melhores práticas na área de arquitetura de microsserviços.

7. REFERÊNCIAS BIBLIOGRÁFICAS

BALDISSERA, Olívia. **Quais são os tipos de arquitetura de software e como escolher o melhor para seu projeto**. mar. 2021. Disponível em: <https://posdigital.pucpr.br/blog/tipos-de-arquitetura-de-software>. Acesso em: 20 jul. 2023.

FOWLER, M; LEWIS, J. **Microservices**. mar. 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 20 jul. 2023.

FOWLER, M; LEWIS, J. **Who Needs an Architect?** ago. 2003. Disponível em: <https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf>. Acesso em: 20 jul. 2023.

FOWLER, M. **Software Architecture Guide**. ago. 2019. Disponível em: <https://martinfowler.com/architecture/>. Acesso em: 20 jul. 2023.

FOWLER, Susan J. **Microsserviços Prontos Para a Produção construindo sistemas padronizados em uma organização de engenharia de software**: São Paulo: Novatec, 2017.

JUNIOR, E. **Manual do Arquiteto de Software**. mar. 2022. Disponível em: <https://arquiteturadesoftware.online>. Acesso em: 20 jul. 2023.

LISBOA, Flávio. **Arquitetura de software distribuído Boas práticas para um mundo de microsserviços**: São Paulo: Casa do Código, 2022.

MARQUES, Vanessa. **Simples Nacional, Confira A Tabela CNAE Completa**. dez. 2022 Disponível em: <https://www.jornalcontabil.com.br/simples-nacional-confira-a-tabela-cnae-completa>. Acesso em: 20 jul. 2023.

NEWMAN, Sam. **Criando Microsserviços Projetando sistemas com componentes menores e mais especializados**: São Paulo: Novatec, 2021.

RICHARDSON, C. **What are microservices?** mar. 2014 Disponível em: <https://microservices.io/index.html>. Acesso em: 20 jul. 2023.

RICHARDSON, C. **Introducing Assemblage - a microservice architecture definition process**. fev. 2023 Disponível em: <https://microservices.io/post/architecture/2023/02/09/assemblage-architecture-definition-process.html>. Acesso em: 20 jul. 2023.

SMITH, Steve. **Architecting Modern Web Applications with ASP.NET Core and Azure**: Washington: Microsoft Corporation, 2021.

VALENTE, Marco Tulio. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**, Editora: Independente, 2020.