UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
INSTITUTO DE GEOCIÊNCIAS E CIÊNCIAS EXATAS – RIO CLARO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# MICRO4DELPHI: UM PROCESSO PARA APOIAR A MODERNIZAÇÃO DE SISTEMAS LEGADOS EM DELPHI PARA ARQUITETURA DE MICROSSERVIÇOS

LUCAS FERNANDO FÁVERO

RIO CLARO - SP

2025

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

Lucas Fernando Fávero

# Micro4Delphi: Um Processo para apoiar a Modernização de Sistemas Legados em Delphi para Arquitetura de Microsserviços

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro – SP, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Frank José Affonso

Rio Claro – SP
2025

F273m

Fávero, Lucas Fernando
Micro4Delphi : um processo para apoiar a modernização de sistemas legados em Delphi para a arquitetura de microsserviços / Lucas Fernando Fávero. -- Rio Claro, 2025
138 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Instituto de Geociências e Ciências Exatas, Rio Claro
Orientador: Frank José Affonso

1. Microsserviços. 2. Modernização. 3. Sistemas distribuídos. 4. Delphi. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

# IMPACTO POTENCIAL DESTA PESQUISA

Esta pesquisa propõe o processo Micro4Delphi, voltado à modernização de sistemas legados em Delphi para arquiteturas de microsserviços. O trabalho realizado contribui para a ciência e engenharia de software ao preencher uma lacuna na literatura e oferecer uma metodologia aplicável em cenários reais. Seu impacto se estende ao campo técnico e econômico, ao possibilitar redução de custos e aumento da eficiência na manutenção e evolução de sistemas legados. Do ponto de vista social e educacional, preserva o patrimônio tecnológico baseado em Delphi, ainda amplamente utilizado, e oferece subsídios para formação e capacitação de profissionais. Além disso, o processo favorece a inovação e a internacionalização, ao alinhar práticas locais a tendências globais de modernização de software, promovendo maior sustentabilidade tecnológica e relevância no cenário atual.

# POTENTIAL IMPACT OF THIS RESEARCH

This research proposes the Micro4Delphi process, aimed at modernizing legacy Delphi systems to microservice architectures. The conducted work contributes to software science and engineering by filling a gap in the literature and offering a methodology applicable to real-world scenarios. Its impact extends to the technical and economic areas, enabling cost reductions and increased efficiency in the maintenance and evolution of legacy systems. From a social and educational perspective, it preserves the Delphi-based technological heritage, still widely used, and provides support for professional development and training. Furthermore, the process fosters innovation and internationalization by aligning local practices with global software modernization trends, promoting greater technological sustainability and relevance in the current scenario.

UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"
Instituto de Geociências e Ciências Exatas
Câmpus de Rio Claro

Lucas Fernando Fávero

# Micro4Delphi: Um Processo para apoiar a Modernização de Sistemas Legados em Delphi para Arquitetura de Microsserviços

Dissertação de Mestrado apresentada ao Instituto de Geociências e Ciências Exatas do Câmpus de Rio Claro – SP, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Comissão Examinadora

- Prof. Dr. Frank José Affonso (Orientador)
  Departamento de Estatística, Matemática Aplicada e Computação
  Universidade Estadual Paulista "Júlio de Mesquita Filho" – Câmpus de Rio Claro
- Profa. Dra. Elisa Yumi Nakagawa
  Departamento de Sistemas de Computação
  Universidade de São Paulo – Câmpus de São Carlos
- Profa. Dra. Rogéria Cristiane Gratão de Souza
  Departamento de Ciências de Computação e Estatística
  Universidade Estadual Paulista "Júlio de Mesquita Filho" – Câmpus de São José do Rio Preto

Conceito: Aprovado.

Rio Claro (SP), 25 de agosto de 2025

*Este trabalho é dedicado a minha esposa Isabela, que sempre me apoiou durante o desenvolvimento deste trabalho.*

# AGRADECIMENTOS

Agradeço a Deus que sempre me abençoou com saúde, disposição e me deu forças para concluir essa longa jornada acadêmica.

Agradeço a minha esposa, Isabela, pela paciência, companheirismo, apoio e suporte emocional em todos os momentos do desenvolvimento deste trabalho.

Agradeço aos meus pais, Osmar e Maria, por toda a estrutura familiar fornecida, que tornou possível a minha formação pessoal.

Agradeço ao meu orientador Prof. Dr. Frank José Afonso, por proporcionar essa oportunidade, por sua imensa contribuição durante os últimos anos, pela amizade conquistada e principalmente pela paciência, atenção e suporte que me foi dado durante todo o desenvolvimento deste trabalho.

Agradeço a Integrativa de Catanduva, empresa onde atuo a 18 anos como profissional de tecnologia da informação pela oportunidade de me afastar em determinados horários de minhas atividades profissionais, o que viabilizou dedicação a este trabalho de mestrado.

Por fim, agradeço aos verdadeiros amigos e familiares que ajudaram e me incentivaram em busca dessa conquista acadêmica.

# RESUMO

Os sistemas de informação desempenham um papel estratégico nas organizações modernas, sendo fundamentais para a coleta, processamento e distribuição de dados que subsidiam a tomada de decisões. Tradicionalmente, muitos desses sistemas foram construídos com base em arquiteturas monolíticas, caracterizadas por uma execução centralizada e pela forte interdependência entre seus componentes. Com a evolução tecnológica, surgiram novas abordagens arquiteturais, como a arquitetura orientada a serviços e, mais recentemente, a arquitetura de microsserviços. A modernização de sistemas legados para a arquitetura de microsserviços tem se tornado uma alternativa viável para empresas que buscam maior flexibilidade, desempenho e adaptabilidade, visando anteder novas tendências de desenvolvimento e execução de sistemas. Baseado no cenário exposto, uma investigação da literatura foi conduzida sobre a modernização de sistemas legados para arquitetura de microsserviços. Essa visão permitiu estabelecer um panorama detalhado referente ao estágio desse tema de pesquisa, suas soluções, domínios de aplicação que têm se beneficiado de tais soluções, estratégias de avaliação utilizadas, atributos de qualidade, e, por fim, as principais atividades relacionadas ao processo de modernização como um todo. Embasado nos resultados dessa investigação, o objetivo deste projeto de mestrado acadêmico foi a elaboração do Micro4Delphi, um processo de modernização de sistemas legados em Delphi para a arquitetura de microsserviços. A opção por Delphi é embasada em duas constatações: (i) quantidade expressiva de sistemas desenvolvidos nesse ambiente de desenvolvimento e pela necessidade de modernizá-los para um modelo de arquitetura mais atual; e (ii) inexistência de iniciativas/soluções voltadas para apoiar esse tipo de modernização. No que respeito ao desenvolvimento, vale ressaltar ainda que a elaboração do Micro4Delphi teve contribuição direta de profissionais especialistas em Delphi de duas empresas, que auxiliaram no refinamento das atividades do processo, bem como experimentação prática da *stack* tecnológica para preservar o sistema modernizado no mesmo ambiente de desenvolvimento (Delphi). Diante do exposto, pode-se dizer que projeto apresenta duas contribuições relevantes: (i) o processo Micro4Delphi, que possui um conjunto de passos bem definido para condução da modernização de software; (ii) um mapeamento da literatura, pois acredita-se que esse mapeamento possa oferecer aos interessados (ou seja, profissionais e pesquisadores) uma visão geral que pode auxiliar no aprimoramento e/ou desenvolvimento de novas soluções/iniciativas. Por fim, vale destacar que o processo Micro4Dephi foi avaliado através da condução de um estudo de caso para um sistema de gestão escolar chamado Avance, sendo possível mostrar a aplicação das principais etapas do processo.

**Palavras-chave:** microsserviços, modernização, sistemas distribuídos, Delphi.

# ABSTRACT

Information systems have played a strategic role in modern organizations, being fundamental for the collection, processing, and distribution of data that supports decision-making. Traditionally, many of these systems were built using monolithic architectures, characterized by centralized execution and strong interdependence among their components. With technological evolution, new architectural approaches have emerged, such as service-oriented architecture and, more recently, microservices architecture. Modernizing legacy systems to microservice architecture has become a viable alternative for companies seeking greater flexibility, performance, and adaptability, aiming to anticipate new trends in system development and execution. Based on the above scenario, a literature review was conducted on the modernization of legacy systems to microservice architecture. This overview enables us to establish a detailed overview of the status of this research topic, its solutions, application domains that have benefited from such solutions, evaluation strategies used, quality attributes, and the main activities related to the modernization process as a whole. Based on the results of this investigation, the objective of this academic master's project was to develop Micro4Delphi, a process for modernizing legacy Delphi systems to microservice architecture. The choice of Delphi is based on two findings: (i) the significant number of systems developed in this development environment and the need to modernize them to a more current architectural model; and (ii) the lack of initiatives/solutions aimed at supporting this type of modernization. Regarding development, it is also worth noting that the development of Micro4Delphi benefited from direct contributions from Delphi specialists from two companies, who assisted in refining the process activities, as well as practical experimentation with the technology stack to maintain the modernized system within the same development environment (Delphi). In light of the above, it can be said that the project presents two relevant contributions: (i) the Micro4Delphi process, which has a well-defined set of steps for conducting software modernization; (ii) a literature review, as it is believed that this review can provide stakeholders (i.e., professionals and researchers) with an overview that can assist in the improvement and/or development of new solutions/initiatives. Finally, it is worth noting that the Micro4Dephi process was evaluated through a case study for a school management system called Avance, demonstrating the application of the main steps of the process.

**Keywords:** microservices, modernization, distributed systems, Delphi.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | *Application Programming Interface* |
| CD | *Continuous Delivery* |
| CI | *Continuous Integration* |
| DevOps | *Development and Operations* |
| DOM | *Document Object Model* |
| HTTP | *Hypertext Transfer Protocol* |
| IDE | *Integrated Development Environment* |
| JSON | *Javascript Object Notation* |
| MSA | *Microservices Architecture* |
| PPGCC/UNESP | *Programa de Pós-Graduação em Ciência da Computação da UNESP* |
| QP | *Questão de Pesquisa* |
| REST | *Representational State Transfer* |
| SMS | *Systematic Mapping Study* |
| SOA | *Service Oriented Architecture* |
| SOAP | *Simple Object Access Protocol* |
| UDDI | *Universal Description, Discovery and Integration* |
| WSDL | *Web Services Description Language* |
| XML | *Extensible Markup Language* |

# SUMÁRIO

# 1 INTRODUÇÃO

## 1.1 Contextualização

Nas últimas décadas, os sistemas distribuídos têm evoluído da arquitetura monolítica para modelos mais flexíveis, como a arquitetura orientada a serviços (SOA) e, mais recentemente, para a arquitetura de microsserviços. Essa transição é impulsionada, principalmente, pelo aumento da complexidade dos sistemas, pela necessidade de maior agilidade na entrega de software e pela demanda por aplicações escaláveis, resilientes e com maior capacidade de evolução tecnológica contínua (LEWIS; FOWLER, 2014; SOLDANI; TAMBURRI; HEUVEL, 2021). A arquitetura monolítica, embora inicialmente simples de desenvolver, testar e implantar, apresenta limitações significativas à medida que a base de código cresce. O acoplamento forte entre os componentes, a dificuldade de manutenção e atualização, além da complexidade crescente do sistema, tornam esse modelo pouco adequado para aplicações de grande porte ou com alta taxa de mudança (DRAGONI *et al.*, 2017; NEWMAN, 2019; NADAREISHVILI *et al.*, 2021). Em resposta a essas limitações, a arquitetura orientada a serviços (SOA) emergiu como uma abordagem para decompor aplicações em serviços fracamente acoplados, com foco na reutilização e interoperabilidade. No entanto, a adoção da SOA trouxe novos desafios, como a complexidade de gerenciamento do barramento de serviços, sobrecarga na governança e dificuldades na evolução dos serviços diante de mudanças frequentes nos requisitos (XIAO; WIJEGUNARATNE; QIANG, 2016; TAIBI; LENARDUZZI, 2018). A arquitetura de microsserviços representa uma evolução desse conceito (SOA), propondo a construção de sistemas como um conjunto de pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Essa abordagem permite maior escalabilidade, independência na implantação, facilidade de manutenção e resiliência a falhas. Além disso, possibilita a adoção de diferentes tecnologias por serviço e a organização de equipes menores e autônomas (NEWMAN, 2019; TSERPES, 2019; ALSHUQAYRAN; ALI; EVANS, 2021).

Apesar das vantagens, a adoção da arquitetura de microsserviços também impõe desafios, como o aumento da complexidade na comunicação entre serviços, na gestão de dados distribuídos, testes, observabilidade, segurança e na curva de aprendizado para desenvolvedores. Além dos aspectos técnicos, a modernização para a arquitetura de microsserviços requer transformações culturais e organizacionais, como mudanças nas práticas de desenvolvimento, cultura DevOps (do inglês, *Development and Operations*) e maior colaboração entre equipes multidisciplinares (EVANS, 2003; BOGNER *et al.*, 2021). Mesmo diante desse cenário, muitas organizações têm optado por migrar seus sistemas legados monolíticos para a arquitetura de

microsserviços com o objetivo de obter maior disponibilidade, escalabilidade, reutilização de componentes, uso eficiente de recursos como *containers* e mensageiras, e liberdade tecnológica. A modularização dos sistemas permite uma divisão mais clara de responsabilidades entre as equipes, reduzindo a sobrecarga de comunicação e promovendo maior agilidade na entrega de valor (RICHARDSON, 2019; ZIMMERMANN, 2022).

## 1.2 Motivação

Diante do contexto apresentado e motivado pela necessidade em entender os aspectos relacionados ao tema de pesquisa deste projeto de mestrado acadêmico, uma investigação da literatura (ou seja, mapeamento sistemático) foi conduzida com base nas diretrizes propostas por Petersen *et al.* (2008), Petersen, Vakkalanka e Kuzniarz (2015). O mapeamento sistemático (do inglês, *Systematic Mapping Study* – SMS) é uma técnica de investigação que permite aos interessados estabelecer uma visão completa e justa sobre um tópico de pesquisa através de uma metodologia confiável, rigorosa, auditável e isenta de influências pessoais. Para isso, um protocolo de pesquisa foi elaborado e estritamente seguido (veja Apêndice A), cujos interesses de investigação podem ser sintetizados da seguinte maneira: (i) características demográficas dos estudos selecionados neste mapeamento; (ii) soluções existentes na literatura que têm apoiado a modernização de sistemas legados para arquitetura de microsserviços; (iii) técnicas utilizadas para avaliação de tais soluções; (iv) atributos de qualidade utilizados em tais soluções; e (v) as principais atividades para que a modernização possa ser conduzida.

Ao analisar os estudos selecionados na referida investigação, observa-se que exitem diversas motivações que levam as organizações a modernizarem seus sistemas (considerados legados) para a arquitetura de microsserviços. Em primeiro lugar, esses sistemas apresentam dificuldades significativas de manutenção, uma vez que o acoplamento entre os módulos e a complexidade do código tornam custosas as atividades de modificação, teste e implantação. Além disso, aplicações monolíticas geralmente possuem tempo de inicialização elevado, o que compromete a agilidade nos ciclos de entrega contínua (NEWMAN, 2019; SOLDANI; TAMBURRI; Van Den Heuvel, 2018; SOLDANI; TAMBURRI; HEUVEL, 2021). Outro fator relevante é a necessidade crescente de alta disponibilidade e escalabilidade dinâmica, especialmente em ambientes que lidam com variações de carga e picos de acesso. Nesse cenário, os microsserviços oferecem uma abordagem vantajosa ao permitir o dimensionamento independente dos componentes, possibilitando que apenas os serviços mais exigidos sejam replicados, otimizando recursos e desempenho (RICHARDSON, 2019; LEWIS; FOWLER, 2019). Além disso, a arquitetura de microsserviços favorece a adoção de diferentes tecnologias, permitindo a utilização de múltiplas linguagens de programação, bancos de dados e *stacks* tecnológicos dentro do mesmo ecossistema. Essa flexibilidade é especialmente valiosa em empresas de médio e

grande porte, que frequentemente operam com sistemas diversos e heterogêneos para a gestão de produtos, serviços e processos internos. A possibilidade de construir serviços especializados com tecnologias adequadas ao seu domínio contribui significativamente para a agilidade no desenvolvimento, integração e evolução dos sistemas corporativos (DRAGONI *et al.*, 2017; NEWMAN, 2019; RICHARDSON, 2019).

Além dos aspectos técnicos relacionados ao software, a investigação conduzida também revelou uma carência de iniciativas que apoiem a modernização de sistemas legados para arquitetura de microsserviço como uma solução mais holística. Analisando os estudos dessa investigação, notou-se que as soluções e/ou iniciativas selecionadas se concentram em algumas atividades e/ou interesses pontuais relacionados à modernização (SOLDANI; TAMBURRI; HEUVEL, 2021; BOGNER *et al.*, 2021; LENARDUZZI; TAIBI; JANES, 2022). No que tange aos passos para a transição do sistema legado para a arquitetura de microsserviços, vale destacar que a investigação conduzida revelou que a modernização tem sido explorada/investigada com base em seis etapas, a saber: planejamento, análise, decomposição, desenvolvimento, integração e monitoramento. Fazendo um paralelo entre as atividades de modernização e o ambiente de desenvolvimento Delphi (EMBARCADERO, 2025), embora esse ambiente de desenvolvimento tenha evoluído desde seu lançamento, correspondendo a um período de 30 anos, contatou-se que não existe nenhuma iniciativa existente na literatura que apoie a modernização de sistemas legados em Delphi para a arquitetura de microsserviços (veja Capítulo 3). No que diz respeito especificamente ao Delphi, pode-se afirmar que esse ambiente de desenvolvimento é aderente aos recentes conceitos e práticas de engenharia, permitindo o desenvolvimento de aplicações voltadas para diferentes segmentos, a saber: *desktop*, Web, mobile, serviços (EMBARCADERO, 2025). Portanto, a combinação dessas evidências com a existência de uma quantidade importante de sistemas de software desenvolvido com base nesse ambiente de desenvolvimento pode ser considerada como um fator de motivação para a realização deste projeto de mestrado acadêmico.

## 1.3 Objetivos

Diante do cenário apresentado sobre a modernização de sistemas legados para a arquitetura de microsserviços e sobre a carência desse tipo de iniciativa para sistemas desenvolvidos em Delphi, o objetivo deste projeto de mestrado acadêmico é propor um processo sistemático que possa apoiar esse tipo de atividade. Embasado pelos resultados da investigação da literatura conduzida e apresentada no Capítulo 3, foi proposto um processo chamado Micro4Delphi, o qual foi organizado em seis macro atividades, sendo que cada atividade possui um conjunto de passos para que o propósito de cada atividade seja executado. Sobre essas atividades/passos, vale ressaltar que elas/eles abrangem tanto aspectos técnicos quanto organizacionais, com o objetivo de reduzir os riscos associados à migração e aumentar a viabilidade prática da transfor-

mação arquitetural. Em síntese, esse processo tem como foco principal apoiar organizações e equipes técnicas na transição gradual de sistemas monolíticos desenvolvidos em Delphi para a arquitetura de microsserviços. Em outras palavras, o Micro4Delphi foi concebido para permitir que a modernização seja realizada mantendo o ambiente Delphi como base tecnológica, promovendo o desacoplamento incremental dos componentes legados, sem a necessidade imediata de reescrita completa do sistema. Ainda sobre a concepção do processo Micro4Delphi, deve destacar o envolvimento de profissionais da indústria de software de duas empresas distintas, sendo uma sediada em Catanduva (Integrativa) e outra em Rio Claro (SuperSoft). Vale destacar que a interação entre os envolvidos foi realizada de maneira voluntária e sem a transferência de conhecimento dessas empresas para os envolvidos neste projeto de mestrado acadêmico.

Conforme mencionado neste capítulo, uma investigação da literatura (SMS) foi conduzida visando atingir dois propósitos: (i) estabelecer um panorama geral sobre a modernização de software para a arquitetura de microsserviços; e (ii) assegurar a originalidade do processo proposto neste projeto de mestrado acadêmico. Portanto, outro objetivo desta dissertação consiste na apresentação desse panorama de modo detalhado para que os interesses investigados possam nortear os interessados (ou seja, pesquisadores e/ou os profissionais da indústria) no tema de pesquisa deste projeto na proposição de novas iniciativas/soluções e/ou melhorias das existentes. De modo similar ao que ocorreu neste projeto de mestrado acadêmico com o ambiente de desenvolvimento Delphi, os resultados dessa investigação podem servir de referência para elaboração de outros processos para outras linguagens de programação (suportadas por outros ambientes de desenvolvimento).

## 1.4 Organização da dissertação

Como forma de apresentar o trabalho realizado, esta dissertação de mestrado acadêmico foi organizada como segue. No Capítulo 2 é apresentada a fundamentação teórica, onde são reportados os conceitos sobre arquitetura de software, arquitetura monolítica, arquitetura orientada a serviços, arquitetura de microsserviços, as principais diferenças entre a arquitetura de microsserviços e arquiteturas monolíticas, e uma visão geral sobre o ambiente de desenvolvimento Delphi. Um mapeamento sobre modernização de sistemas legados para a arquitetura de microsserviços, descrevendo como essa pesquisa foi conduzida com base em 43 estudos primários, é apresentado no Capítulo 3. Como forma de estabelecer um panorama sobre esse tema de pesquisa, são relatadas nesse capítulo as principais descobertas desse mapeamento e as análises realizadas. No Capítulo 4 são reportados os detalhes do Micro4Delphi, um processo que visa apoiar a modernização de sistemas legados desenvolvidos em Delphi para a arquitetura de microsserviços. No Capítulo 5 é apresentado o estudo de caso que foi desenvolvido para avaliar a viabilidade, os pontos fortes e fracos do processo Micro4Delphi. Por fim, no Capítulo 6 são

apresentadas as conclusões e contribuições oriundas deste trabalho, bem como as perspectivas para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

## 2.1 Considerações Iniciais

Este capítulo apresenta definições e conceitos que serviram como base para o desenvolvimento deste trabalho. Inicialmente, a Seção 2.2 apresenta os conceitos sobre arquitetura de software. Em seguida, na Seção 2.3 são detalhadas as principais características da arquitetura monolítica. Já na Seção 2.4 são reportadas as principais características da arquitetura orientada a serviços. Uma descrição detalhada sobre a arquitetura de microsserviços é apresentada na Seção 2.5, destacando as características dessa arquitetura, os benefícios em adotar esse estilo arquitetural, os desafios que devem ser levados em consideração ao se adotar esse estilo arquitetural, as vantagens e desvantagens e as melhores práticas que se deve adotar para trabalhar com microsserviços. Na Seção 2.6 são apresentadas as diferenças entre a arquitetura de microsserviços e arquiteturas monolíticas. Na Seção 2.7 são apresentados os principais conceitos do Delphi. Por fim, na Seção 2.8 são reportadas as considerações finais deste capítulo.

## 2.2 Arquitetura de Software

A arquitetura é a parte da Engenharia de Software que se concentra na estruturação e organização do software, a fim de garantir qualidade, escalabilidade e manutenibilidade do mesmo. O projeto de arquitetura pode ser considerado como uma abordagem sistemática e holística que considera não apenas as partes individuais do sistema, mas também as interações e interdependências entre essas partes para projetar, construir, manter e evoluir software de forma eficiente e escalável. Em resumo, a arquitetura é importante para garantir que o software seja projetado e construído de maneira eficiente, escalável e flexível, facilitando a atividade de manutenção/evolução ao longo do tempo (FOWLER, 2002).

Na visão de Pressman (2016), a arquitetura de software define a estrutura de componentes do software, suas interações e as políticas de gerenciamento da informação, visando a criação de sistemas de software confiáveis, escaláveis e fáceis de manter. É uma parte fundamental do processo de desenvolvimento de software, pois permite que as equipes de desenvolvimento considerem as questões de *design* e escolham as melhores abordagens para resolvê-las. A arquitetura de software afeta diretamente a manutenibilidade, escalabilidade, performance e segurança de um sistema, devendo ser cuidadosamente planejada e implementada para garantir o sucesso do projeto. Os autores afirmam que o objetivo principal da arquitetura é garantir que o software seja construído de forma coerente, consistente e alinhada às expectativas do usuário

e do negócio. Para isso, é necessário estabelecer padrões de projeto, definir as interfaces entre componentes e avaliar o impacto de mudanças no sistema.

Fazendo uma análise desse assunto na literatura, nota-se que os autores têm descrito a arquitetura de software de diferentes maneiras. Para Booch (1994), "arquitetura de software é a parte visível do sistema que fornece uma base sólida para a construção do software". Na visão de Fowler (2002), "arquitetura de software é a estrutura de um sistema de software, composta de vários componentes inter-relacionados, destinada a atender aos requisitos do sistema". Já Garlan e Shaw (1993) apresentam uma definição sobre arquitetura de software como "a estrutura dos componentes de um sistema, seus relacionamentos, princípios e boas práticas que governam o projeto e sua evolução ao longo do tempo". Clements *et al.* (2010) definem a arquitetura de software como sendo um conjunto de estruturas necessárias para compreender os elementos de software, as relações entre eles e suas propriedades. Para Bass, Clements e Kazman (2012), a arquitetura de software pode ser definida como "a estrutura de um programa que compreende os seus componentes, as propriedades visíveis externamente, e os relacionamentos entre tais componentes". Por fim, Rocha (2018) comenta que a arquitetura de software de um sistema consiste na definição dos componentes de software, suas propriedades externas e seus relacionamentos com outros softwares. Abrange a forma como suas partes são organizadas, incluindo questões como o comportamento dessa estrutura e quais componentes são responsáveis por realizar um conjunto específico de funções. Em resumo, a arquitetura de software é responsável por definir os componentes que farão parte de um projeto, suas características, funções e a maneira como devem interagir entre si e com outros softwares.

Diante do exposto, pode-se dizer que arquitetura de software pode ser considerada como o desenho geral e a estrutura lógica de um sistema de software, incluindo as relações e interações entre seus componentes, bem como suas propriedades funcionais e não funcionais. Em outras palavras, a arquitetura define como o sistema será organizado, otimizado e escalado para atender às demandas do usuário e do negócio, garantindo a qualidade, segurança e eficiência do software. A arquitetura de software é fundamental para a construção de sistemas de software robustos, escaláveis e de alta qualidade.

Embora Pressman (2016) e Sommerville (2018) tenham um entendimento comum sobre a importância dos estilos de arquitetura para o desenvolvimento de software, suas abordagens e ênfases podem ser diferentes. Pressman (2016) descreve os estilos arquiteturais como "padrões de organização de subsistemas que fornecem uma estrutura de alto nível para as aplicações". Ele apresenta exemplos de vários estilos, incluindo a arquitetura em camadas, a arquitetura cliente-servidor, a arquitetura baseada em eventos, a arquitetura orientada a serviços (do inglês, *Service-Oriented Architecture* – SOA), a arquitetura baseada em microsserviços, entre outras. Por outro lado, Sommerville (2018) apresenta a arquitetura como uma estrutura fundamental da aplicação e descreve os estilos arquiteturais como "abstrações de alto nível que podem ser usadas

para orientar a organização de um sistema de software". O autor também apresenta exemplos de estilos, mas sua ênfase concentra-se na importância da escolha de um estilo arquitetural adequado em relação às necessidades do sistema em questão. Por fim, vale destacar que ambos os autores concordam que a escolha do estilo arquitetural correto pode resultar em impactos na qualidade e na manutenção do software.

De acordo com o conteúdo apresentado nesta seção, diferentes tipos de arquitetura de software podem ser encontrados na literatura. Como este trabalho está relacionado à modernização de sistemas legados para microsserviços, optou-se por apresentar conceitos e definições sobre os tipos de arquitetura relacionados a esse tema de pesquisa, a saber: arquitetura monolítica; arquitetura orientada a serviços; e arquitetura de microsserviços.

## 2.3 Arquitetura Monolítica

A arquitetura monolítica é aquela em que os componentes são geralmente separados em camadas lógicas e, por esse motivo, ficou conhecida como arquitetura em camadas (do inglês, *layered architecture*). Pode-se dizer que é um tipo de arquitetura simples e fácil de compreender, mas que pode se tornar complexa e difícil de manter à medida que o sistema cresce e aumenta de tamanho significativamente. Além disso, esse tipo de arquitetura tem-se mostrado difícil de escalar e adaptar a novos requisitos, pois todas as mudanças precisam ser feitas no mesmo módulo integrado (DRAGONI *et al.*, 2017).

A arquitetura monolítica pode ser definida como uma abordagem tradicional na construção de sistemas de software, onde o sistema é visto como uma única entidade integrada e interdependente. Nessa arquitetura, todas as funcionalidades do sistema são incorporadas em um único módulo de software que é compilado e executado como uma única entidade. A comunicação entre as diferentes partes do sistema é feita por chamadas de funções e mensagens internas (DRAGONI *et al.*, 2017).

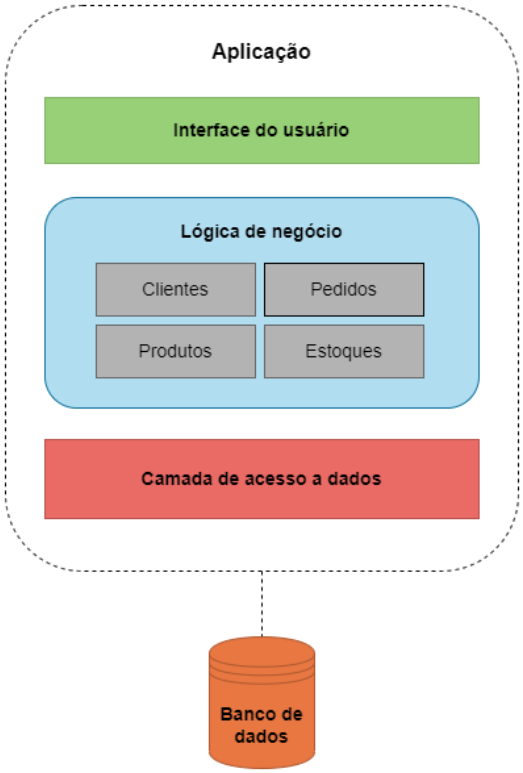
Para Rocha (2018), a arquitetura de um software monolítico tem a característica de compartilhar os recursos de uma mesma máquina, como memória, banco de dados e arquivos. Nesse tipo de arquitetura não existe um isolamento de uso de recurso por um determinado módulo da aplicação, sendo que um recurso pode ser utilizado por qualquer módulo, ou até mesmo, por todos os módulos do sistema.

Fazendo um paralelo entre as definições encontradas na literatura sobre arquitetura de software monolítico, Dragoni *et al.* (2017) comentam que um "monolítico é uma aplicação de software cujo os módulos não podem ser executados independentemente". Já Carvalho (2017) afirmou que a arquitetura monolítica ainda é o padrão mais utilizado para o desenvolvimento de aplicações corporativas, sendo sua principal característica a inserção de todas as funcionalidades

em um único executável. Para Newman (2019), as desvantagens da arquitetura monolítica podem ser sintetizadas em: (i) alto acoplamento no código e na implantação, (ii) problemas de concorrência quando muitos desenvolvedores ou times estão trabalhando na mesma base de código; (iii) baixa coesão de código; (iv) alto impacto de alterações; e (v) implantação do sistema a cada alteração de código.

Diante do exposto, pode-se dizer que a arquitetura monolítica é um modelo de arquitetura de software no qual todos os componentes do sistema são combinados em uma única unidade ou módulo, que funciona como um todo integrado. Nesse modelo, todas as funcionalidades são executadas dentro da mesma aplicação e compartilham o mesmo processo e espaço de memória. A arquitetura monolítica é comumente utilizada em aplicações menores ou sistemas simples, onde há pouca complexidade e interação entre as funcionalidades. Na Figura 1 é ilustrada a arquitetura de uma aplicação monolítica.

Figura 1 – Arquitetura Monolítica




Fonte: Elaborada pelo autor.

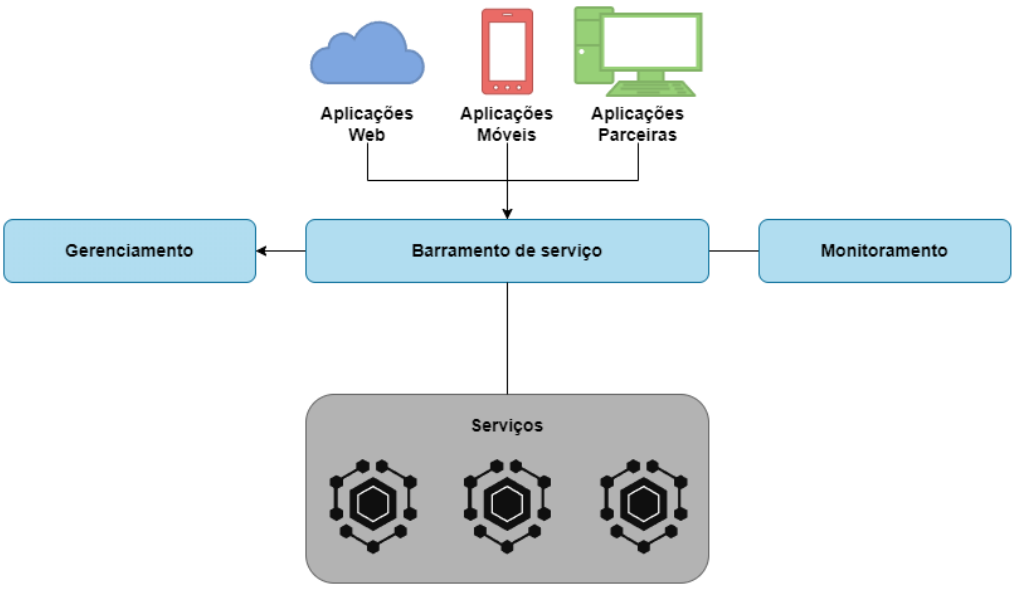
## 2.4 Arquitetura Orientada a Serviços

A arquitetura orientada a serviços é um modelo de arquitetura de software que se baseia na divisão de tarefas em pequenos serviços independentes, com a finalidade de aumentar a flexibilidade, escalabilidade e reusabilidade do sistema. Essa arquitetura permite a integração de

aplicações através de serviços Web, conhecidos como *Web Services*, que podem ser acessados por diferentes tipos de clientes, tais como aplicações *desktop* ou dispositivos móveis. Em geral, esses serviços são implementados como componentes independentes que possuem sua própria lógica de negócios. Esses serviços são expostos por meio de interfaces bem definidas e podem ser combinados de diversas maneiras para atender as diferentes necessidades de negócios. A arquitetura SOA é um modelo de arquitetura de software baseada em padrões, que são amplamente utilizados em aplicações empresariais para integrar diferentes sistemas e aplicativos (SOMMERVILLE, 2018).

Fazendo uma breve síntese da literatura, Xiao, Wijegunaratne e Qiang (2016) comentam que SOA surgiu da necessidade de uma arquitetura distribuída baseada em componentes e maior agilidade para atender as necessidades de negócio e a natureza evolutiva do mesmo. Já MacKenzie *et al.* (2016) descreveram SOA como um paradigma para organizar e utilizar capacidades distribuídas sob controle de diferentes domínios. A IBM (2023) define SOA como um *framework* de aplicação que permite quebrar aplicações em funções e processos de negócio individuais chamados de serviços. De acordo com Rocha (2018), SOA fornece uma maneira uniforme para oferecer, descobrir, interagir e utilizar capacidades para produzir efeitos desejados consistentes com pré-condições e expectativas mensuráveis. Por fim, para Silva, Justino e Adachi (2022) a arquitetura orientada a serviços pode ser considerada um estilo de decomposição de um software em serviços fracamente acoplados, escaláveis e interoperáveis. Em outras palavras, a SOA integra os componentes de software que foram implantados e são mantidos separadamente, permitindo que se comuniquem e trabalhem juntos para formar aplicações que funcionam em sistemas diferentes. Na Figura 2 é ilustrada a arquitetura de uma aplicação orientada a serviços.

Figura 2 – Arquitetura Orientada a Serviços




Fonte: Elaborada pelo autor.

## 2.5 Arquitetura de Microsserviços

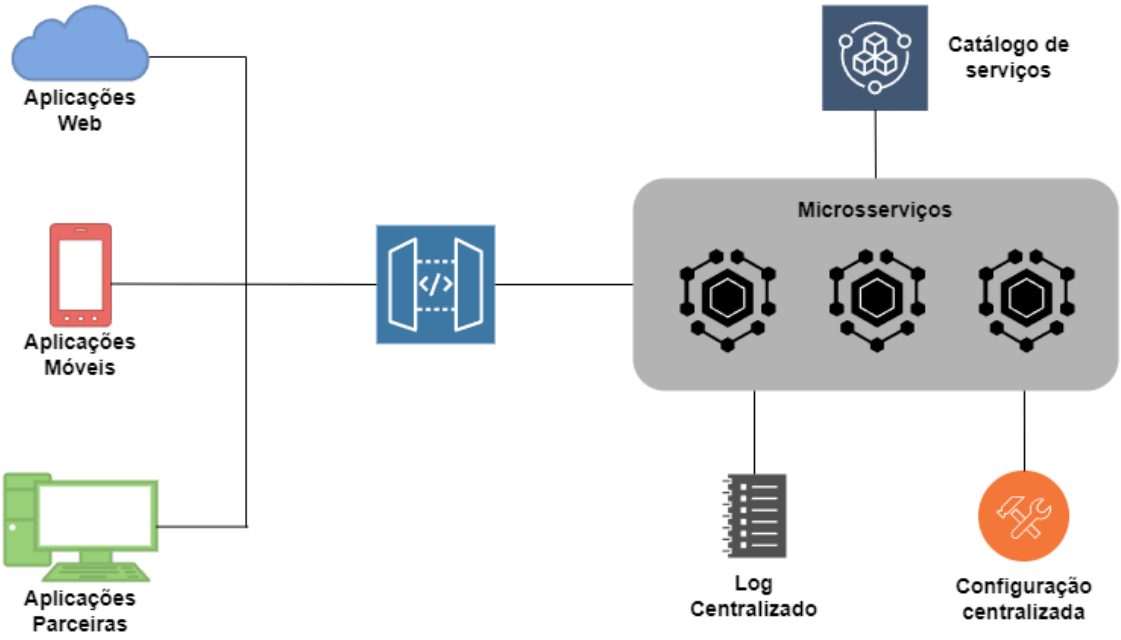
A arquitetura de microsserviços (do inglês, *Microservices Architecture* – MSA) representa uma modalidade de desenvolvimento de software que envolve a construção de uma aplicação como uma coleção de pequenos serviços independentes, sendo cada um com sua própria responsabilidade e comunicação através de uma interface bem definida. Cada serviço é desenvolvido, testado, implantado e mantido de maneira independente, o que permite uma maior agilidade e flexibilidade na evolução da aplicação. Além disso, a arquitetura de microsserviços permite a utilização de diferentes tecnologias para cada serviço, o que aumenta a capacidade de adaptação às mudanças no ambiente de negócio, tornando o software mais fácil de ser atualizado e mantido ao longo do tempo (LEWIS; FOWLER, 2014; TSERPES, 2019).

De acordo com Lewis e Fowler (2014), o estilo arquitetural de microsserviços é uma abordagem de desenvolvimento de uma única aplicação como uma suíte de pequenos serviços, cada um rodando em seu próprio processo e se comunicando por meio de mecanismos leves. Esses serviços normalmente são construídos em torno de uma capacidade de negócio e são implantados de forma independente e automática. Para Tserpes (2019), o termo "microsserviços" se refere a um conceito de arquitetura de software pelo qual a funcionalidade de uma aplicação é decomposta em funções menores e independentes residindo em recursos próprios de infraestrutura. Newman (2015) reforça essa ideia descrevendo que os microsserviços devem ser pequenos, focados em fazer uma tarefa de maneira eficiente e autônoma. Já a IBM (2023) define microsserviços como um estilo arquitetural em que grandes e complexas aplicações são compostas por um ou mais serviços. Na prática, microsserviços envolvem diferentes conceitos que emergiram de boas práticas e da comunidade de desenvolvedores ágeis, estando fortemente conectados à abordagem DevOps (do inglês, *Development and Operations*). Diante do exposto, pode-se dizer que as arquiteturas de microsserviços facilitam a escalabilidade e agilizam o desenvolvimento de aplicativos, habilitando a inovação e acelerando o tempo de introdução de novos recursos no mercado (AMUNDSEN *et al.*, 2016).

Na Figura 3 é ilustrada a arquitetura de microsserviços e seus principais componentes. A API (do inglês, *Application Programming Interface*) *Gateway* é responsável por ser o ponto de entrada do sistema e realizar o roteamento das funções para os microsserviços, o catálogo de serviços é responsável pelo registro e descoberta dos microsserviços, o *log* centralizado é responsável pela persistência dos *logs* de todos os microsserviços e a configuração centralizada tem a função de manter toda a configuração da aplicação. Os microsserviços podem ser implantados de forma automática e independente, encapsulando todas as regras e funcionalidades de negócio do sistema.

No que diz respeito aos **benefícios e características**, Sharma (2017) comentou que a arquitetura de microsserviços possui muitas características em comum com SOA, como por

Figura 3 – Arquitetura de Microsserviços




Fonte: Elaborada pelo autor.

exemplo o foco em serviços e o modo como um serviço é desacoplado de outro. A Arquitetura SOA evoluiu a integração entre aplicações monolíticas através da utilização de APIs baseadas em SOAP, já na arquitetura de microsserviços isso é mais simples, uma vez que não possui um barramento de serviços para centralizar a composição dos serviços. Para Carvalho (2017), a abordagem de microsserviços facilita o entendimento de cada serviço por serem focados em uma única capacidade de negócio, tornando a utilização da IDE (do inglês, *Integrated Development Environment*) mais rápida e a escalabilidade do sistema menos dolorosa, facilitando a integração e entrega contínua.

Segundo a AWS (2023) os microsserviços têm como principais características serem autônomos, onde cada serviço pode ser desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços. Além disso, os microsserviços não precisam compartilhar nenhum código ou implementação com os outros e todas as comunicações entre componentes individuais ocorrem por meio de APIs bem definidas e especializadas. Em geral, são projetados para ter um conjunto de recursos dedicados à solução de um problema específico, e caso a complexidade de um serviço aumente, este poderá ser dividido em outros serviços menores. Já na visão de Jambunathan e Kalpana (2016), a maioria dos microsserviços compartilham um conjunto de características, tais como:

- ***Design* orientado a domínio**: A decomposição funcional do aplicativo com base no domínio e a equipe responsável pela criação de funções são responsáveis pela criação de microsserviços com uma única pilha de tecnologia;

- **Princípio de responsabilidade única**: Todo microsserviço deve abordar uma única funcionalidade e deve ter a responsabilidade de resolvê-la completamente;
- **Interface explicitamente publicada**: Cada microsserviço terá sua própria interface publicada e exposta, portanto o lado consumidor apenas possui a visualização dessa interface para interagir e não possui nenhuma dependência de tempo em execução;
- **Implantação independente**: Microsserviços devem ser implantados independentes do sistema geral sem causar impactos;
- **Ocultação de detalhes**: Os detalhes de implementação são ocultos e não são expostos ao mundo externo, permitindo que sejam dissociados completamente um do outro, auxiliando as equipes na criação dos microsserviços em diferentes aspectos como decisão sobre seus idiomas, ferramentas, persistência de dados, entre outros; e
- **Protocolos de comunicação leves**: Microsserviços se comunicarão utilizando os protocolos REST e HTTP.

Para Newman (2015), a adoção da MSA propicia sete importantes benefícios, a saber: heterogeneidade tecnológica, resiliência, escalabilidade, fácil implantação, alinhamento organizacional, componibilidade e otimização para substituição de serviços. Richardson (2018) também descreveu uma série de benefícios ao adotar a MSA, a saber:

- **Serviços pequenos e fáceis de manter**: Por serem de tamanho menor não sobrecarregam a IDE na fase de desenvolvimento e facilitam o entendimento por parte dos desenvolvedores, além de serem inicializados mais rapidamente, aumentando a produtividade e tornando as implantações mais fáceis;
- **Serviços escaláveis de maneira independente**: Cada serviço pode ser escalado de maneira independente e pode ser implantado em um hardware diferente, sendo essa uma grande diferença em relação à arquitetura monolítica, onde todos os componentes são implantados juntos no mesmo hardware;
- **Isolamento de falhas**: As falhas afetam somente aquele serviço e os demais continuam operando normalmente. Diferentemente das aplicações monolíticas, quando uma falha ocorre em um componente, todo o sistema pode ficar indisponível;
- **Entrega contínua e implantação de aplicações grandes e complexas**: Como os serviços são pequenos, se torna mais fácil criar testes automatizados e testar cada serviço, pois o tempo de execução dos testes é menor. Além disso, cada serviço pode ser implantado de maneira independente desde que não exista alteração no contrato, o que resulta em atualizações com intervalos menores de tempo; e

- **Adoção de novas tecnologias**: Ao criar um novo microsserviço, os desenvolvedores são livres para escolher a tecnologia que mais se adequa ao contexto, eliminando assim o compromisso de longo prazo com uma tecnologia específica. Em grandes organizações é normal que as opções sejam mais restritas, destacando o fato de que o tamanho dos microsserviços facilita a reescrita do código em uma nova linguagem, caso seja necessário.

Embora a arquitetura de microsserviços tenha vários benefícios, esse estilo de arquitetura também apresenta **desafios** significativos que precisam ser enfrentados. Segundo Newman (2015) os desafios de trabalhar com microsserviços são vários, tais como a complexidade do gerenciamento de várias peças de software independentes, o aumento da complexidade da comunicação entre os serviços e a necessidade de lidar com a escalabilidade de cada serviço individualmente. O autor também destaca a importância de manter uma cultura forte de colaboração e comunicação entre equipes para lidar com esses desafios. Lewis e Fowler (2014) destacam que os microsserviços introduzem um novo nível de complexidade em termos de infraestrutura e implantação, sendo necessário um maior investimento em ferramentas e automação de processos. Para os autores operar um sistema distribuído é muito mais difícil do que operar um sistema centralizado, testar um sistema distribuído é muito mais difícil do que testar um sistema centralizado, mudanças de dados entre serviços podem ser problemáticas, necessidade de um bom suporte para implantação automatizada e a complexidade de um sistema de microsserviços pode ser maior do que a de um monolítico. Já Richardson (2019) ressalta a importância de lidar com a complexidade na integração entre os serviços, a necessidade de implementar mecanismos de monitoramento e gerenciamento de erros distribuídos, e a necessidade de manter a consistência dos dados em diferentes serviços. O autor também destaca a importância de se pensar na arquitetura como um todo, em vez de apenas nos serviços individualmente, a fim de garantir a escalabilidade e a resiliência do sistema como um todo. Para Evans (2003), um dos principais desafios é a necessidade de manter a consistência do domínio em sistemas distribuídos, garantindo que todas as partes do sistema estejam de acordo com as regras de negócio e que os dados estejam em sincronia. Nessa direção, esse último autor reporta os seguintes desafios:

- **Comunicação e coordenação entre serviços distribuídos**: Para que os microsserviços funcionem corretamente é preciso garantir que todos eles estejam se comunicando adequadamente e coordenando suas atividades;
- **Desafios de gerenciamento de dados**: Cada microsserviço tem seu próprio banco de dados, o que pode tornar o gerenciamento de dados complexo, especialmente quando há necessidade de manter consistência entre diferentes serviços;
- **Complexidade do ambiente de execução**: Um ambiente de microsserviços pode ser composto por muitos serviços em execução em diferentes plataformas e tecnologias, o

que pode aumentar a complexidade da infraestrutura de TI;

- **Desafios de segurança**: Como os microsserviços são distribuídos, é preciso garantir a segurança da comunicação e autenticação entre os serviços; e
- **Desafios de teste e depuração**: Testar e depurar microsserviços pode ser mais complexo do que testar e depurar um software monolítico, já que envolve testar a interação entre diferentes serviços.

Em resumo, trabalhar com microsserviços apresenta desafios significativos que envolvem a complexidade do gerenciamento de sistemas distribuídos, a garantia da integridade dos dados, a arquitetura de resiliência e a manutenção da consistência do domínio. Cada autor reporta um olhar específico sobre esses desafios, mas todos destacam a importância de investir em ferramentas, estratégias e processos que permitam lidar de maneira efetiva com as complexidades inerentes aos sistemas distribuídos.

Embora o uso de microsserviços tem ganhado cada vez mais destaque no desenvolvimento de software, entender suas **vantagens e desvantagens** pode ser um importante diferencial para adoção desse estilo arquitetural. Segundo Lewis e Fowler (2014), os microsserviços oferecem vantagens como maior modularidade, escalabilidade independente, possibilidade de escolha de diferentes tecnologias para cada serviço, e melhorias na velocidade de desenvolvimento e implantação. Newman (2015) destaca a possibilidade de escolha de tecnologias específicas para cada serviço e de escalar individualmente cada componente. Além disso, o autor ressalta que os microsserviços podem proporcionar melhorias na tolerância a falhas e na recuperação de desastres, já que cada serviço pode ter um processo de recuperação próprio. Evans (2003) enfatiza a redução do acoplamento e aumento da coesão, que possibilita mudanças com menor impacto em outros serviços. Já Richardson (2019) destaca diversas vantagens em se trabalhar com microsserviços, a saber:

- **Escalabilidade**: É mais fácil escalar microsserviços individualmente, permitindo que uma aplicação seja escalada de forma mais eficiente e granular do que uma aplicação monolítica;
- **Flexibilidade**: Microsserviços são independentes uns dos outros, o que permite que cada serviço possa ser desenvolvido, testado, implantado e atualizado de forma independente, sem afetar os outros serviços;
- **Resiliência**: Quando um serviço falha, os outros serviços podem continuar funcionando, minimizando o impacto na aplicação como um todo;

- **Heterogeneidade**: Microsserviços podem ser desenvolvidos usando diferentes tecnologias e linguagens de programação, o que permite que as equipes de desenvolvimento escolham as melhores ferramentas para cada serviço; e
- **Evolução**: Microsserviços são mais fáceis de evoluir do que uma aplicação monolítica, uma vez que cada serviço pode ser atualizado individualmente, sem exigir que toda a aplicação seja atualizada de uma só vez.

Embora inúmeras vantagens possam ser citadas, ao optar pelo desenvolvimento baseado em microsserviços, algumas desvantagens podem ser observadas. Lewis e Fowler (2014) destacam que uma das principais desvantagens é a complexidade operacional que surge ao trabalhar com vários serviços independentes. Os autores afirmam que o gerenciamento e a coordenação entre os serviços podem se tornar bastante desafiadores, especialmente quando se trata de lidar com transações distribuídas. Além disso, a abordagem de microsserviços exige um alto nível de automação e boas práticas de DevOps para garantir uma entrega contínua e confiável. Richardson (2019) destaca outra desvantagem que é a necessidade de ter uma arquitetura orientada a eventos para garantir que os serviços sejam notificados sobre as mudanças em outros serviços. Isso aumenta a complexidade do sistema e pode exigir mais esforço na implementação de uma infraestrutura de mensageria confiável. Evans (2003) também menciona que a abordagem de microsserviços pode aumentar a complexidade do *design* e a sobrecarga de comunicação. Além disso, a divisão em serviços pequenos pode aumentar a complexidade da integração entre os serviços e exigir uma arquitetura mais sofisticada. Por fim, Newman (2015) destaca que os microsserviços também apresentam desafios em relação ao gerenciamento de dados. O autor afirma que, com várias bases de dados independentes, a consistência dos dados pode ser difícil de manter. Além disso, as mudanças nos esquemas dos bancos de dados podem ser mais complexas de gerenciar quando vários serviços estão envolvidos.

Os microsserviços têm se tornado cada vez mais populares na indústria de software nos últimos anos, especialmente em ambientes de nuvem e aplicações distribuídas. Diversos autores da literatura têm se dedicado a estudar e aprofundar esse tema, trazendo contribuições importantes sobre as **melhores práticas**.

De acordo com Newman (2015), um dos pontos mais importantes para se trabalhar com microsserviços é o conceito de "*bounded contexts*" ou contextos delimitados, que consiste em identificar e separar as diferentes partes do sistema que possuem lógicas distintas e podem ser independentes. Isso permite que os microsserviços sejam criados de maneira mais autônoma, facilitando a escalabilidade e a manutenção. O autor defende que cada serviço deve ser capaz de executar suas funções sem depender de outros serviços ou componentes. Para isso, é necessário que os microsserviços sejam autônomos, possam ser implantados e escalados de maneira independente, e que tenham uma interface bem definida. Outra prática recomendada

é o uso de APIs RESTful para a comunicação entre serviços. Esse tipo de API permite uma integração mais simples e flexível, além de permitir que cada serviço seja desenvolvido em uma linguagem e tecnologia diferente, desde que possam se comunicar usando protocolos comuns. Outro ponto enfatizado pelo autor é a importância do monitoramento e observabilidade dos serviços por meio de ferramentas de monitoramento e *logging*, que são capazes de identificar e solucionar problemas de forma ágil. Para isso, é importante que os microsserviços gerem *logs* com informações detalhadas sobre seu comportamento. O autor também destaca a importância da automação em todas as fases do ciclo de vida do serviço, desde o desenvolvimento até a implantação e operação, enfatizando que a automação pode ajudar a reduzir a complexidade e os custos de manutenção, além de permitir que as equipes de desenvolvimento se concentrem em criar novas funcionalidades. Por fim, é destacada a importância da cultura DevOps para o sucesso dos microsserviços, onde as equipes de desenvolvimento e operações devem trabalhar em conjunto para garantir a implantação e operação dos microsserviços. Além disso, vale destacar que a comunicação e colaboração são fundamentais para o sucesso dessa arquitetura.

Segundo Evans (2003), para implementar microsserviços de maneira eficaz, é importante manter uma comunicação estreita entre as equipes de desenvolvimento e de negócios, a fim de entender os requisitos do domínio e projetar serviços que atendam a esses requisitos. O autor também enfatiza a importância de definir fronteiras bem definidas entre os serviços para evitar a interdependência excessiva e garantir que cada serviço possa ser implantado, atualizado e escalado independentemente dos outros. Outra prática recomendada é o uso de contratos bem definidos entre serviços para garantir a compatibilidade das interfaces e minimizar o acoplamento. Além disso, o referido autor também destaca alguns aspectos importantes: (i) monitorar os microsserviços para detectar problemas e garantir que estejam funcionando corretamente, (ii) definir uma estratégia clara para a separação dos microsserviços para que possam ser gerenciados e testados de forma independente; e (iii) manter um foco no domínio do negócio, e não somente em tecnologias, para que os microsserviços possam ser desenvolvidos de forma mais efetiva.

Lewis e Fowler (2014) enfatizam que a abordagem de microsserviços não é uma bala de prata e que pode ter desvantagens, como a complexidade de gerenciar várias peças menores em vez de um grande monolítico. No entanto, os autores também argumentam que os microsserviços podem oferecer benefícios significativos, como melhorias na escalabilidade, resiliência e velocidade de desenvolvimento. Dentre as melhores práticas recomendadas para se trabalhar com microsserviços, pode-se destacar as seguintes:

- **Modelagem de domínio**: A criação de microsserviços deve ser baseada na modelagem de domínio da organização, permitindo a identificação dos limites dos serviços e como eles se relacionam entre si;

- **Autonomia**: Cada microsserviço deve ser autônomo e independente, com sua própria base de código, equipe e infraestrutura, gerenciando seus próprios dados e regras de negócios. Isso permite que as equipes possam trabalhar de maneira independente, sem afetar o restante do sistema;
- **Comunicação leve**: Os microsserviços devem se comunicar de maneira leve e com acoplamento fraco, preferencialmente usando uma API HTTP ou outro mecanismo simples;
- **Interface explícita**: A interface de cada microsserviço deve ser explícita, claramente definindo quais dados são necessários para a comunicação com outros serviços;
- **Design para falhas**: Os microsserviços devem ser projetados para falhar de maneira isolada, sem afetar outros serviços ou o aplicativo como um todo;
- **Testes automatizados**: Cada microsserviço deve ser testado por meio de testes automatizados, permitindo que as equipes possam ter mais confiança nas mudanças realizadas, a fim de evitar problemas no sistema como um todo;
- **Gerenciamento de configuração**: Os microsserviços devem ser gerenciados através de ferramentas de gerenciamento de configuração, como o Kubernetes, permitindo o gerenciamento da escalabilidade, disponibilidade e redundância;
- **Monitoramento**: Cada microsserviço deve ter um monitoramento ativo, permitindo o acompanhamento do desempenho e comportamento em tempo real;
- **Documentação**: A documentação de cada microsserviço deve ser clara e detalhada, permitindo que outras equipes possam utilizá-lo sem dificuldades; e
- **Evolução contínua**: Os microsserviços devem ser projetados para evoluir continuamente, permitindo atualizações frequentes e implementações mais rápidas.

Por fim, Richardson (2019) destaca a importância de se pensar em padrões e melhores práticas para o desenvolvimento de microsserviços, incluindo o uso de contêineres para facilitar a implantação, a adoção de práticas de integração contínua (do inglês, *Continuous Integration* – CI) e entrega contínua (do inglês, *Continuous Delivery* – CD), assim como a implementação de mecanismos de monitoramento e registro de *logs*. O autor também destaca que a adoção de microsserviços não é uma solução simples ou de fácil implementação. É necessário um alto nível de maturidade em práticas de DevOps, testes automatizados e integração contínua para garantir a qualidade e a confiabilidade do sistema como um todo. Além disso, é importante ter uma boa compreensão dos limites e responsabilidades de cada serviço, para evitar a criação de acoplamentos desnecessários entre os componentes do sistema. Apesar dos desafios, o referido autor

acredita que a popularidade dos microsserviços continuará crescendo, impulsionada pela necessidade de empresas em se manterem competitivas no mercado de software. No entanto, o autor enfatiza a importância de seguir as melhores práticas e padrões estabelecidos pela comunidade para garantir o sucesso da implementação de uma arquitetura baseada em microsserviços.

## 2.6 Diferenças entre Microsserviços e Arquiteturas Monolíticas

De acordo com a visão de vários autores da literatura que trabalham com arquitetura de software, existem diferenças significativas entre arquiteturas monolíticas e as arquiteturas baseadas em microsserviços. Por exemplo, Lewis e Fowler (2014) afirmam que a principal diferença entre essas arquiteturas é que todos os componentes do sistema estão interconectados e executam no mesmo processo em uma arquitetura monolítica. Em uma arquitetura baseada em microsserviços, cada serviço é independente e se comunica com outros serviços por meio de APIs bem definidas. Em termos práticos isso significa que, em uma arquitetura monolítica, as alterações em um componente podem ter um grande impacto em outros componentes, o que pode tornar a manutenção e a evolução do sistema mais complexa. Já em uma arquitetura de microsserviços, cada serviço pode ser atualizado independentemente, o que permite que o sistema seja mais flexível e escalável. Richardson (2019) também destaca essa diferença, observando que as arquiteturas monolíticas podem facilitar o desenvolvimento, testes e implantação, mas podem se tornar complexas de escalar e evoluir à medida que evoluem ao longo do tempo. Por outro lado, as arquiteturas de microsserviços são mais flexíveis e escaláveis, mas apresentam desafios adicionais, como gerenciamento de rede e comunicação entre serviços.

Newman (2015) enfatiza que uma das principais vantagens da arquitetura de microsserviços é a capacidade de desenvolver e implantar serviços independentes, o que permite a experimentação e a inovação com mais facilidade. No entanto, o autor também observa que essa abordagem requer um investimento significativo em infraestrutura e automação para gerenciar adequadamente a complexidade da rede de serviços. Uma das principais diferenças entre as arquiteturas monolíticas e de microsserviços para o referido autor, está na forma como as funcionalidades são divididas. Em uma arquitetura monolítica, a aplicação é construída como um todo, onde todas as funcionalidades são implementadas dentro de um único código. Já nos microsserviços, as funcionalidades são separadas em serviços independentes e autônomos, onde cada um com sua própria lógica de negócio e responsabilidade bem definida. Outra diferença importante está na forma como os serviços são implementados e escalados. Em uma arquitetura monolítica, a escalabilidade é limitada a uma única instância da aplicação, enquanto que nos microsserviços é possível escalar cada serviço de forma independente, de acordo com a demanda.

Na visão de Evans (2003), a arquitetura baseada em microsserviços é uma alternativa

viável para arquiteturas monolíticas, especialmente em organizações maiores e mais complexas. O autor destaca que a arquitetura baseada em microsserviços pode ajudar a lidar com questões relacionadas à complexidade, escalabilidade, modularidade e desacoplamento. Para o autor, uma das principais diferenças entre microsserviços e arquiteturas monolíticas é que os microsserviços incentivam a separação de preocupações e a especialização de serviços. Com a arquitetura baseada em microsserviços, é possível implementar serviços específicos para atender a necessidades específicas, o que pode melhorar a eficiência e a escalabilidade da aplicação como um todo.

Outra diferença importante destacada é que, em arquiteturas monolíticas, a complexidade tende a crescer à medida que a aplicação cresce, o que pode dificultar a manutenção e o desenvolvimento de novas funcionalidades. Em contrapartida, com a arquitetura baseada em microsserviços, é possível dividir a aplicação em serviços menores e mais gerenciáveis, facilitando a manutenção e evolução de cada um deles. Evans (2003) destaca que a arquitetura baseada em microsserviços pode exigir um investimento maior em infraestrutura e tecnologias para gerenciamento e monitoramento dos serviços, bem como uma mudança cultural na organização para trabalhar de forma mais colaborativa e com maior autonomia entre equipes responsáveis por cada serviço.

Fazendo uma análise do conteúdo reportado nesta seção, pode-se observar que os autores concordam que as arquiteturas baseadas em microsserviços oferecem vantagens significativas em relação às arquiteturas monolíticas em termos de flexibilidade e escalabilidade, mas também apresentam desafios adicionais que precisam ser gerenciados para garantir o sucesso da implementação. Na Tabela 1 é apresentada as principais diferenças entre as arquiteturas monolíticas e as arquiteturas baseadas em microsserviços.

## 2.7 Delphi

De acordo com Embarcadero (2025), Delphi é uma ferramenta de desenvolvimento de aplicações lançada originalmente em 1995 pela Borland, que tem passado por diversas modificações/evoluções para se adequar às novas tendências atuais de mercado/engenharia. Atualmente, essa ferramenta vem sendo mantida Embarcadero Technologies, que fornece aos interessados um ambiente de desenvolvimento integrado baseado na linguagem Object Pascal, que oferece um conjunto robusto de componentes visuais e bibliotecas para o desenvolvimento de aplicações *desktop*, móveis e Web. Uma das principais características dessa ferramenta é a ênfase em produtividade e facilidade de uso, pois possibilita aos desenvolvedores criar interfaces gráficas de maneira rápida e intuitiva, por meio do paradigma de arrastar e soltar componentes visuais, enquanto mantém a capacidade de produzir código eficiente e orientado a objetos. Essa abordagem contribuiu para a ampla adoção do Delphi durante as décadas de 1990 e 2000,

Tabela 1 – Diferenças entre Microsserviços e Arquiteturas Monolíticas

| | **Monolítico** | **Microsserviços** |
|---|---|---|
| Código | Base única de código para toda a aplicação | Múltiplas bases de códigos. Cada microsserviço tem sua própria base de código |
| Facilidade de entender | Frequentemente confuso e difícil de manter | Maior legibilidade e mais fácil de manter |
| Implantação | Complexo com janela de manutenção e interrupção programada | Simples, cada serviço pode ser implantado individualmente, com mínima ou nenhuma interrupção |
| Linguagem | Tipicamente é desenvolvido em uma única linguagem | Cada microsserviço pode ser desenvolvido em uma linguagem diferente |
| Escalabilidade | Exige escalar toda a aplicação ainda que o gargalo seja localizado | Permite escalar apenas serviços com gargalos, sem escalar toda aplicação |

Fonte: Elaborada pelo autor.

especialmente no desenvolvimento de sistemas corporativos e administrativos (TIOBE, 2025).

Atualmente, o Delphi combina os benefícios de linguagens fortemente tipadas com a flexibilidade do desenvolvimento visual, oferecendo suporte a conceitos como programação modular, herança, encapsulamento e polimorfismo, características que permitem o desenvolvimento de aplicações robustas e de fácil manutenção. Além disso, essa ferramenta com bibliotecas para aplicações multiplataformas, possibilitando a criação de softwares para sistemas operacionais como Windows, MacOS, Linux, iOS e Android (EMBARCADERO, 2025; TIOBE, 2025). Apesar de suas vantagens, muitos sistemas desenvolvidos em Delphi acabaram por se tornar legados ao longo do tempo, seja por falta de manutenção contínua, por mudanças tecnológicas no mercado ou pela dificuldade de integração com novas arquiteturas baseadas em serviços e computação em nuvem. Esses sistemas, no entanto, continuam desempenhando papéis críticos em muitas organizações, dada a sua estabilidade e o alto investimento histórico em suas implementações (SOLDANI; TAMBURRI; Van Den Heuvel, 2018; COLANZI *et al.*, 2021a).

Com a crescente adoção de arquiteturas modernas, como a de microsserviços, muitos desenvolvedores e empresas têm buscado estratégias para modernizar aplicações Delphi, de forma a integrar os sistemas legados a ambientes mais flexíveis, escaláveis e distribuídos. Essa modernização geralmente envolve a reestruturação de monólitos Delphi para que suas funcionalidades possam ser expostas como APIs e serviços independentes, sem abrir mão da base de código já consolidada (SOLDANI; TAMBURRI; Van Den Heuvel, 2018; DRAGONI *et al.*, 2017). Dessa forma, compreender as características técnicas e históricas do Delphi é fundamental para planejar e executar a modernização de sistemas legados, respeitando tanto as limitações quanto os pontos fortes dessa tecnologia. O conhecimento do Delphi permite não

apenas preservar o valor dos sistemas existentes, mas também adaptá-los às demandas atuais do mercado por interoperabilidade, escalabilidade e manutenção simplificada (EMBARCADERO, 2025).

## 2.8 Considerações Finais

Neste capítulo foram apresentados os principais conceitos e fundamentos que sustentam o desenvolvimento deste projeto de mestrado acadêmico. Inicialmente, na Seção 2.2, foram apresentados os conceitos sobre arquitetura de software e seus estilos arquiteturais, que fornecem a base de conhecimento necessária para compreender a evolução dos sistemas. Na Seção 2.3, foram abordadas as características da arquitetura monolítica, ponto de partida deste estudo, considerando os desafios associados à manutenção e evolução de sistemas legados. Em continuidade, na Seção 2.4 foi destacada a arquitetura orientada a serviços, considerada um passo intermediário no processo de modernização, pois introduz a extração de funcionalidades do monólito em serviços independentes. Na Seção 2.5 foram apresentados, de maneira detalhada, os princípios, benefícios, desafios e melhores práticas relacionados à arquitetura de microsserviços, a qual representa o objetivo final deste processo de modernização. Já na Seção 2.6, foram discutidas as principais diferenças entre as arquiteturas monolítica e de microsserviços, estabelecendo a base comparativa necessária para compreender a transformação arquitetural. Por fim, na Seção 2.7, foram descritos os conceitos fundamentais do Delphi, tecnologia foco deste projeto, cujo legado foi a motivação sobre a necessidade de modernização. Diante do exposto, pode-se dizer que a fundamentação teórica aqui consolidada fornece o suporte conceitual necessário para a compreensão do processo de modernização de sistemas legados em Delphi para a arquitetura de microsserviços.

# 3 MODERNIZAÇÃO DE SISTEMAS LEGADOS PARA A ARQUITETURA DE MICROSSERVIÇOS

## 3.1 Considerações Iniciais

Neste capítulo, uma visão geral sobre o cenário atual envolvendo a modernização de sistemas legados para microsserviços é apresentada. Pesquisas acadêmicas e práticas da indústria de software têm explorado diversas abordagens para realizar essa modernização. Em resumo, os pesquisadores têm se dedicado a propor soluções e metodologias que definam as atividades necessárias para migrar sistemas legados para MSA. Além disso, eles também têm buscado compreender os benefícios, limitações e desafios associados a essas atividades. Por outro lado, a indústria de software vem enfrentando desafios práticos ao realizar essa transição, uma vez que, em muitos casos, os profissionais carecem de conhecimento sobre técnicas, metodologias ou abordagens bem definidas. Em outras palavras, os desafios estão sendo superados à medida que novas soluções emergem. Com relação a organização deste capítulo, seu conteúdo foi estruturado em três seções. Inicialmente, na Seção 3.2 são apresentados os resultados do mapeamento conduzido com base em 43 estudos primários. Em seguida, na Seção 3.3 foi realizada uma discussão com base nos resultados desse mapeamento, acompanhadas das análises realizadas. Finalmente, na Seção 3.4 são reportadas as considerações finais deste capítulo.

## 3.2 Mapeamento da Literatura

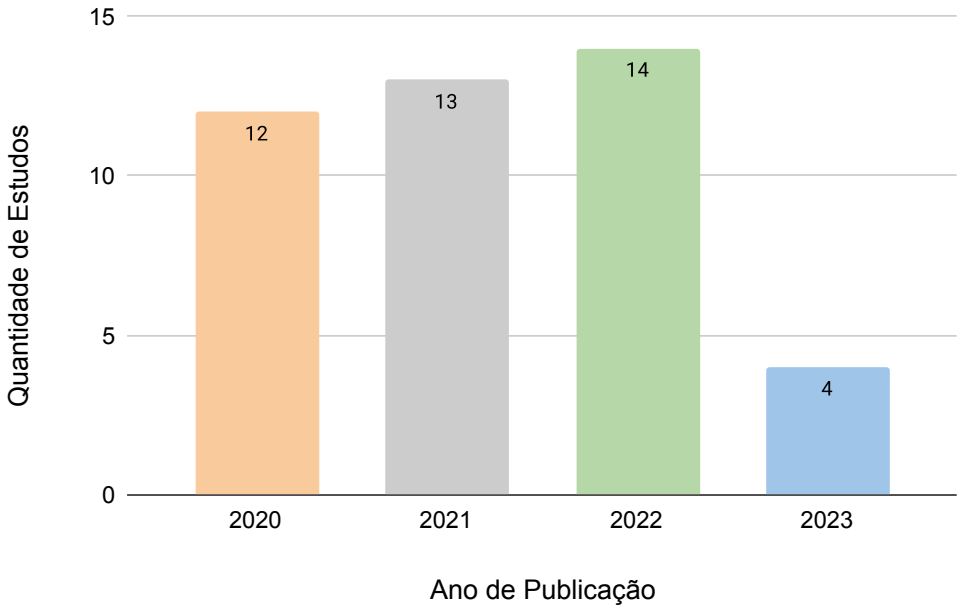
Uma vez que o tema de pesquisa deste projeto de mestrado acadêmico apresenta intersecção entre a academia e a indústria, compreender como esse tema de maneira abrangente é necessário. Isso não apenas contribui para a construção da proposta de projeto em questão, mas também auxilia na delimitação do escopo de trabalho. Assim, para obter um ponto de vista abrangente, a técnica de mapeamento sistemático da literatura (do inglês, *Systematic Mapping Study* – SMS) foi adotada. Em resumo, essa técnica permite a construção de um conhecimento estruturado a partir da síntese, avaliação e interpretação das evidências relacionadas a um tema de interesse (KITCHENHAM; DYBA; JORGENSEN, 2004; KITCHENHAM; CHARTERS, 2007; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Para isso, um protocolo de pesquisa foi elaborado (veja Apêndice A) para selecionar estudos primários que abordam a modernização de sistemas legados e quais soluções foram propostas para lidar com esse assunto. Como resul-

tado, 43 estudos primários foram selecionados para compor o SMS sobre a modernização de sistemas legados, cujas referências podem ser consultadas na Tabela 11 do Apêndice B. A seguir são apresentados os resultados para cada QP (Questão de Pesquisa) do mapeamento conduzido.

### 3.2.1 Características demográficas dos estudos

Esta QP teve como objetivo a identificação de um panorama geral sobre a procedência dos estudos primários levantados por este SMS. O primeiro dado demográfico pode ser visualizado na Figura 4, que ilustra a distribuição da quantidade de estudos por ano de publicação (**QP1.1**). É possível observar que os estudos selecionados cobrem um período de publicações entre os anos de 2020 a 2023, totalizando um intervalo dos últimos 3 anos inteiros de pesquisa e o ano de 2023 fracionado até o mês de setembro. De acordo com o gráfico foram observados 12 estudos no ano de 2020 (27,91%); 13 estudos no ano de 2021 (30,23%); 14 estudos no ano de 2022 (32,56%); e 4 estudos no de 2023 (9,30%). Dentre esses intervalos, o ano de 2022 se destaca com a maior concentração de estudos, porém os anos de 2020 e 2021 chamam bastante atenção, pois contém a segunda e a terceira maior concentração de estudos com números bem próximos, indicando uma tendência atual para publicações científicas relacionadas com a temática do SMS desenvolvido neste projeto de mestrado acadêmico. De modo geral, é possível evidenciar a relevância do tema de pesquisa abordado neste SMS, que vem sendo trabalhado ativamente e com números aproximados de publicações a partir do ano de 2020.

Figura 4 – Distribuição dos estudos por ano de publicação

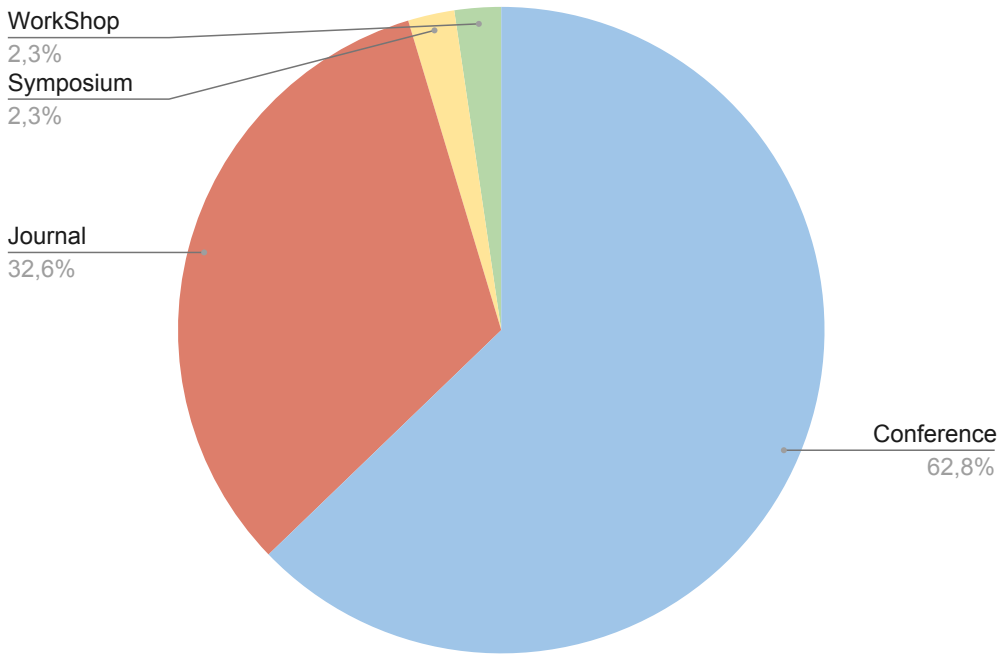



Fonte: Elaborada pelo autor.

O segundo dado demográfico pode ser observado na Figura 5, que ilustra a distribuição

da quantidade de estudos por local de publicação (**QP1.2**). Ao todo, foram identificados quatro locais de publicação, a saber: (i) *Conference*; (ii) *Journal*; (iii) *Symposium*; e (iv) *Workshop*. A maior parte dos estudos está contida em duas categorias principais de publicação, a saber: (1) *Conference*, com 27 estudos (62,80%) publicados em *proceedings* de conferências; e (2) *Journal*, com 14 estudos (32,60%) publicados em periódicos científicos. Para as publicações realizadas em simpósios e *Workshops*, foi encontrado somente 1 estudo (2,30%) para cada um desses tipos de publicação. De acordo com a distribuição ilustrada na Figura 5 é possível observar uma grande concentração de estudos publicados em conferências e jornais (95,40%). Tal constatação pode evidenciar a maturidade dos estudos relacionados com o tema de pesquisa do SMS conduzido neste projeto de mestrado. Como a palavra maturidade pode abranger diferentes aspectos de um trabalho, neste SMS esse termo se refere aos estudos que apresentaram uma solução para modernização de sistemas e algum tipo de avaliação dessa solução proposta.

Figura 5 – Distribuição dos estudos por local de publicação

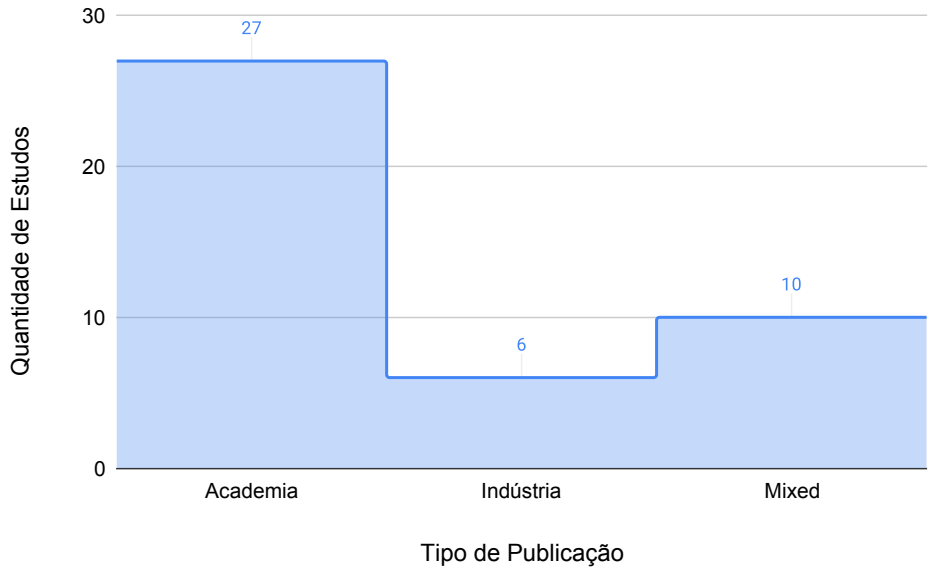



Fonte: Elaborada pelo autor.

Por fim, o terceiro dado demográfico pode ser visualizado na Figura 6, que ilustra a distribuição da quantidade de estudos por tipo de publicação (**QP1.3**). Ao todo foram identificados três tipos de publicações, a saber: (i) Academia; (ii) Indústria; e (iii) *Mixed*. O tipo Academia refere-se aos estudos que desenvolveram seus trabalhos somente no âmbito acadêmico. O tipo Indústria refere-se aos estudos que tiveram envolvimento direto da indústria na proposição e validação de suas propostas. Já o tipo *Mixed* representa os estudos que tiveram envolvimento de ambos, academia e indústria, na proposição e/ou validação de suas propostas. Como pode-se observar, a maior parte dos estudos está contida no tipo Academia, com 27 estudos (62,80%)

publicados; o tipo Indústria conta com 6 estudos (13,95%), e o tipo Mixed conta com 10 estudos (23,25%). Fazendo um *merge* entre as duas últimas categorias tem-se 16 estudos (37,20%) com participação efetiva da indústria, revelando uma boa aproximação entres pesquisadores e profissionais da indústria de software.

Figura 6 – Distribuição dos estudos por tipo de publicação




Fonte: Elaborada pelo autor.

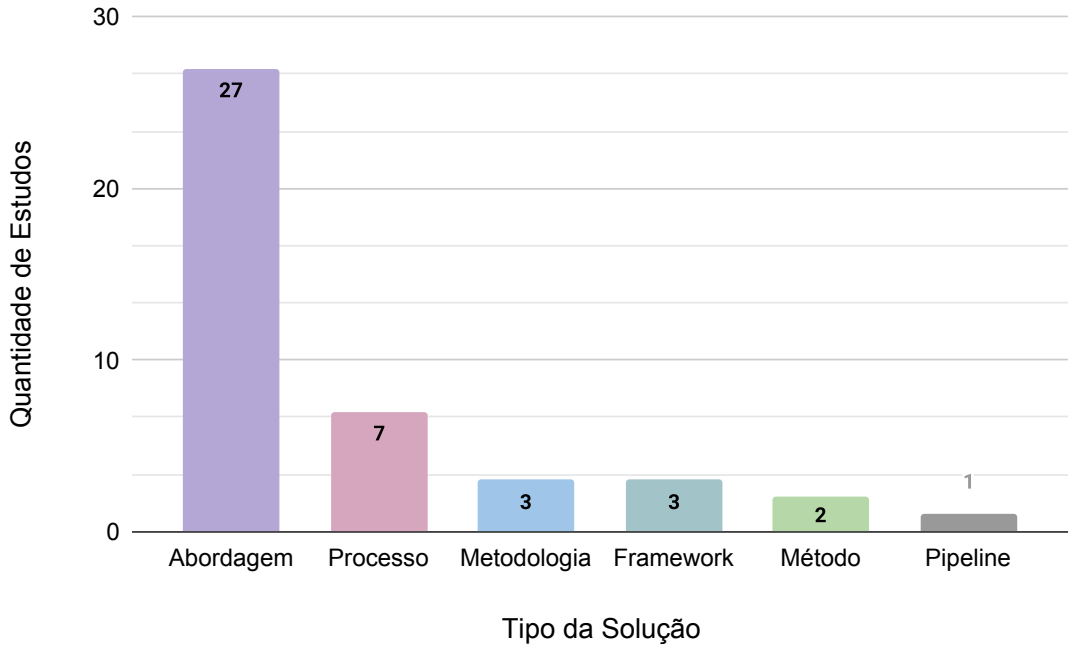
### 3.2.2 Panorama das soluções para modernização

O objetivo delineado para esta QP foi identificar os tipos de soluções existentes na literatura, tais como: (i) Metodologia, (ii) Método, (iii) Processo, (iv) Abordagem, (v) *Pipeline*, e (vi) *Framework*. De acordo com SEVOCAB (2023), Metodologia pode ser definida como uma abordagem sistemática para criar um *design* centrada na aplicação ordenada de uma coleção específica de ferramentas, técnicas e diretrizes. Método refere-se a um item de informação que representa uma série ordenada de etapas para executar um processo, atividade ou tarefa. Já Processo representa um conjunto de atividades inter-relacionadas ou interativas que transformam um conjunto de entradas em um (ou mais) produto(s) esperados. Enquanto que Abordagem é caracterizada pela sua orientação teórica, metodologia de pesquisa, e estratégias utilizadas para investigar, analisar e apresentar resultados. *Pipeline* é uma técnica de projeto de software ou hardware em que a saída de um processo serve como entrada para um segundo, a saída do segundo processo serve como entrada para um terceiro, e assim por diante, muitas vezes com simultaneidade dentro de um único tempo de ciclo. Por fim, *Framework* representa um *design* reutilizável (modelo ou código) que pode ser refinado, especializado e estendido para fornecer alguma parte da funcionalidade geral de muitas aplicações.

Além de identificar a existência das soluções na literatura, esta questão foi dividida em outras subquestões para que os interesses inerentes ao projeto das soluções fossem também investigados, a saber: (i) domínio em que a solução foi projetada (**QP2.1**); (ii) avaliação da solução proposta (**QP2.2**); (iii) evidências de adoção da solução (**QP2.3**); e (iv) presença de atributos de qualidade na solução proposta (**QP2.4**). Como resultado, espera-se que os itens investigados nessa questão (**QP2**) possam compor um panorama de soluções voltadas para a modernização de sistemas legados para MSA.

Na Figura 7 é possível observar a distribuição da quantidade de estudos por tipo de solução (**QP2**). Dentre os seis tipos de soluções identificadas em nosso mapeamento, **Abordagem**, com 27 estudos (62,79%), foi o tipo mais recorrente. Na sequência, tem-se a seguinte distribuição: **Processo** com sete estudos (16,28%), **Metodologia** e ***Framework*** com três estudos (6,98%) cada, **Método** com dois estudos (4,65%), e ***Pipeline*** com um estudo (2,32%). Diante dessa distribuição, nota-se uma predominância de soluções que sistematizaram uma sequência de passos necessários para lidar com a modernização de sistemas legados para MSA. Embora haja uma predominância quanto ao tipo de soluções, a distribuição ilustrada também revela que esse assunto tem sido lidado de diferentes maneiras pelos grupos de pesquisa e profissionais da indústria de software, apresentando alguns vieses de automatização (por exemplo, ***Framework***). Assim, os diferentes tipos de soluções evidenciam que a temática que está sendo investigada neste trabalho pode ser desenvolvida de diferentes maneiras e não existe uma solução padrão amplamente adotada para solucioná-la.

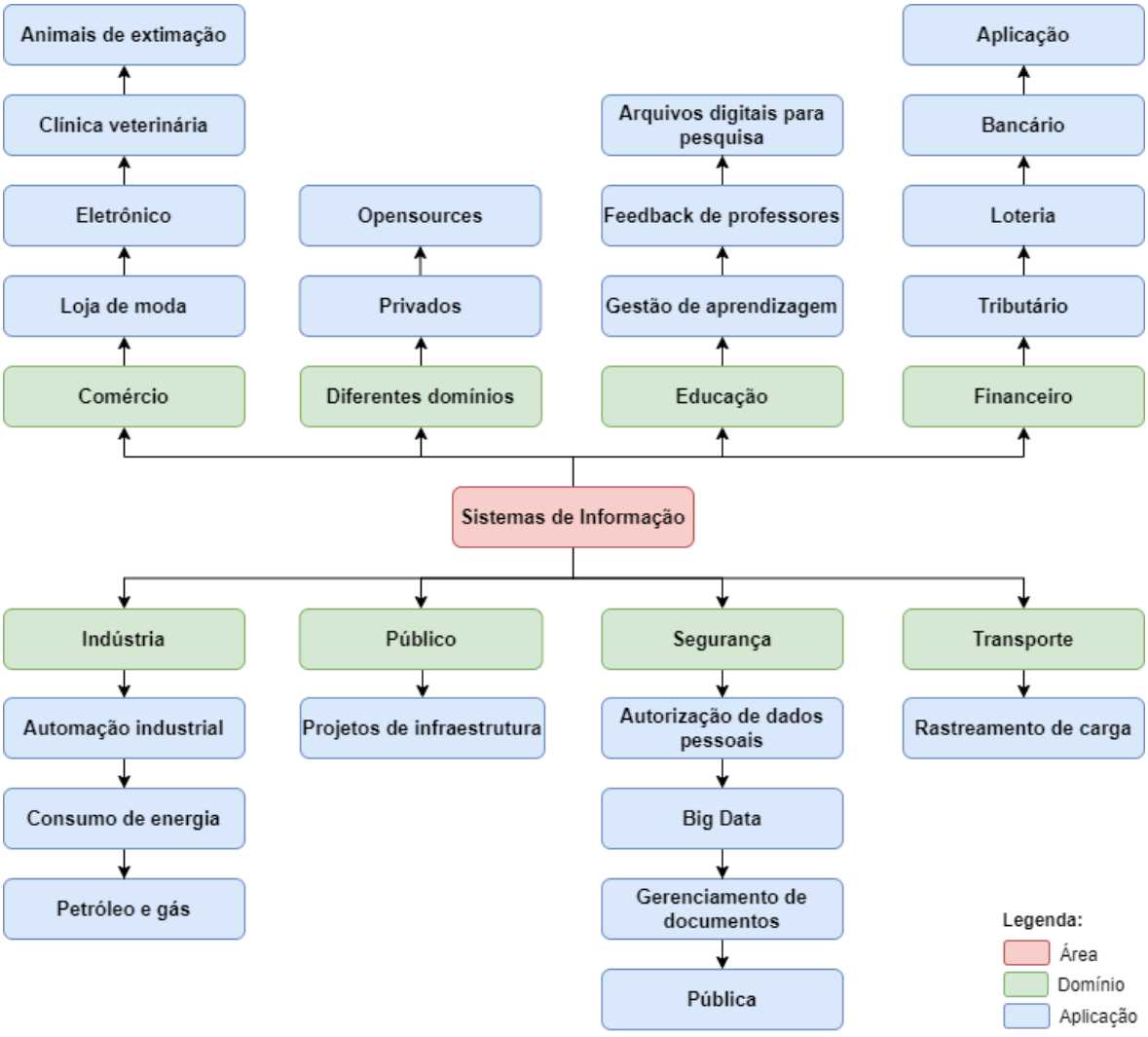
Figura 7 – Distribuição dos estudos por tipo de solução




Fonte: Elaborada pelo autor.

A respeito do domínio em que a solução foi projetada (**QP2.1**), foi analisado o tipo de avaliação adotado em cada estudo primário, resultando em um total de oito domínios e 22 aplicações para validação das soluções propostas, sendo todos voltados para a área de "Sistemas de Informação". Dos 43 estudos selecionados, 33 estudos (ou seja, 76,74%) tiveram suas soluções avaliadas considerando pelo menos um domínio de aplicação, conforme ilustrado na Figura 8. Importante destacar que 10 estudos (ou seja, S3, S4, S10, S11, S14, S17, S19, S22, S29 e S34) tiveram suas soluções propostas avaliadas utilizando sistemas *open souce* de diferentes domínios, dois estudos (ou seja, S7 e S20) tiveram suas propostas avaliadas em sistemas privados de diferentes domínios, e dois estudos (ou seja, S5 e S6) tiveram suas propostas avaliadas em sistemas bancários. Por fim, vale a ressaltar que 10 estudos (ou seja, S15, S16, S18, S23, S26, S28, S31, S35, S41 e S43) não apresentavam explicitamente um domínio de aplicação, não permitindo classificá-los em nosso mapeamento para não introduzir qualquer tipo de viés.

Figura 8 – Distribuição dos estudos por domínio de aplicação




Fonte: Elaborada pelo autor.

Conforme apresentado na Tabela 2, a avaliação (**QP2.2**) das soluções de modernização foram classificadas em quatro categorias, a saber: (i) Estudo de caso, (ii) Experimento, (iii) Prova de conceito, e (iv) Não avaliada. De acordo com Wohlin *et al.* (2012), Estudo de caso é uma abordagem que permite uma exploração profunda e contextualizada de fenômenos específicos, contribuindo para uma compreensão mais rica e detalhada do objeto de estudo. O autor definiu que Experimento é uma investigação sistemática e controlada realizada com o objetivo de testar uma hipótese, validar uma teoria ou adquirir conhecimento sobre um fenômeno específico. Ainda nessa direção, o referido autor comentou que Prova de conceito é uma demonstração prática para verificar a viabilidade e a eficácia de uma ideia, conceito ou método proposto. Por fim, não avaliada representa a categoria em que os estudos não apresentaram avaliação de suas propostas de modernização. Com base nas categorias supracitadas, nota-se que Estudo de caso, com 36 estudos, foi a categoria mais recorrente em nossa investigação. Em seguida, com três e dois estudos, respectivamente, Experimento e Prova de conceito foram as outras categorias identificadas. Dentre os 43 estudos avaliados, somente dois estudos não utilizaram nenhuma técnica de avaliação. Os dados mostram que a grande maioria dos estudos (ou seja, 41) que tratam do tema de pesquisa modernização de sistemas legados para MSA utilizaram algum tipo de avaliação. Nessa direção, notou-se também que esses estudos deixaram novas avaliações como trabalhos futuros, que são novos Estudos de casos, Experimentos e até mesmo casos reais na indústria.

Tabela 2 – Avaliação da solução

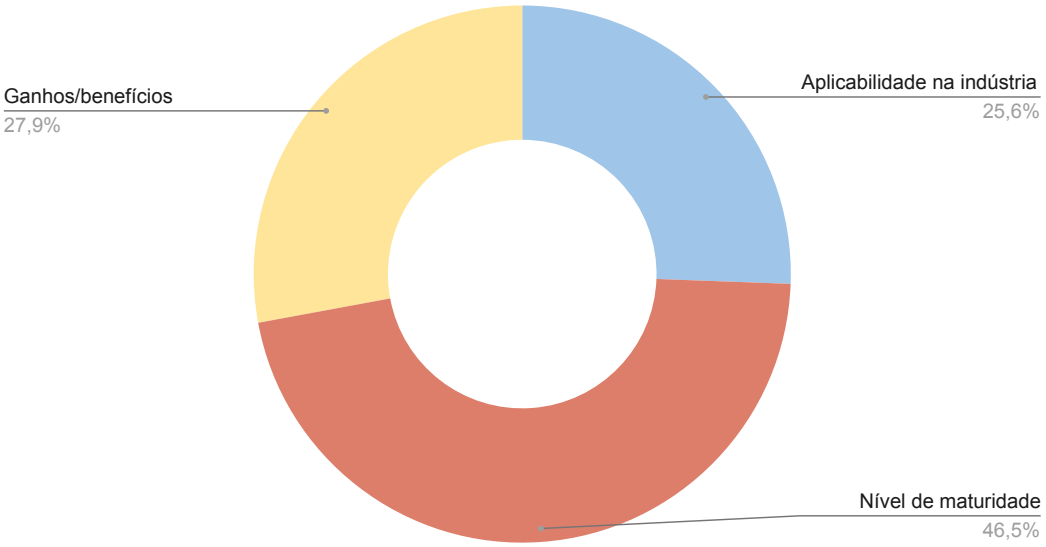
| **Avaliação** | **Estudos** | **Total** |
|---|---|---|
| Estudo de caso | S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S12, S13, S14, S15, S16, S17, S18, S19, S20, S24, S26, S27, S28, S29, S30, S31, S32, S33, S34, S35, S36, S37, S38, S39, S40, S42 | 36 |
| Experimento | S21, S22, S23 | 3 |
| Prova de conceito | S11, S25 | 2 |
| Não avaliada | S41, S43 | 2 |

Fonte: Elaborada pelo autor.

Na Figura 9 é possível observar a distribuição dos estudos por evidências de adoção (**QP2.3**). Três tipos de evidências foram identificadas, a saber: (i) Aplicabilidade na indústria, (ii) Ganhos/benefícios, e (iii) Nível de maturidade. De acordo com as evidências de nossa investigação, a primeira refere-se aos estudos que se destacaram por validarem suas propostas na indústria, ou seja, em um ambiente real de trabalho. Já a segunda contempla os estudos que se destacaram por apresentar ganhos e benefícios relevantes para a academia. E por fim, a terceira refere-se a estudos que validaram suas propostas na academia e na indústria, apresentando

um nível de maturidade satisfatório com enorme potencial de desenvolvimento para trabalhos futuros. Dentre os três tipos de evidências que foram identificadas, "Nível de maturidade" se destacou com 20 estudos (46,50%). Na sequência, "Ganhos/benefícios" com 12 estudos (27,90%) e "Aplicabilidade na indústria" com 11 estudos (25,60%) fecham as evidências identificadas em nosso mapeamento. Diante desse cenário, observa-se uma predominância pelo Nível de maturidade da solução por se tratar de um tema de pesquisa que lida tanto com a acadêmica (por exemplo, sistematização das etapas para modernização de sistemas legados para MSA) quanto com a indústria (por exemplo, aplicação em cenários reais de execução).

Figura 9 – Distribuição dos estudos por evidências de adoção

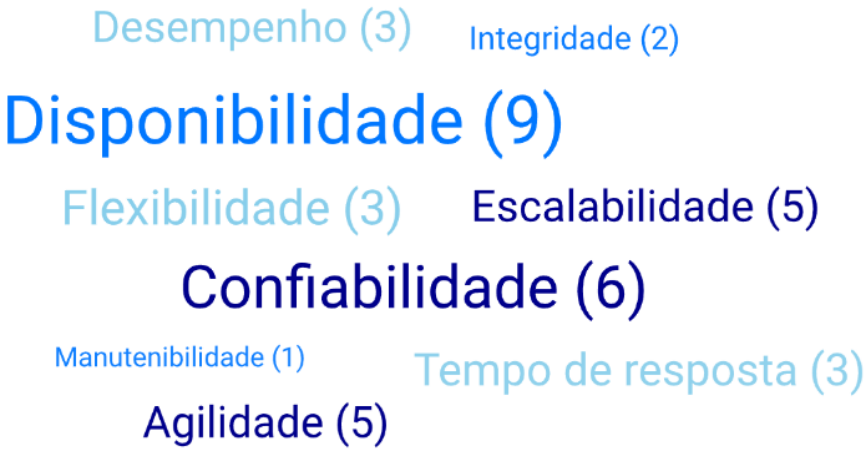



Fonte: Elaborada pelo autor.

Por fim, para analisar os atributos de qualidade (**QP2.4**) mais recorrentes no processo de modernização de sistemas legados para MSA, foi observado o número de ocorrências de cada atributo nos estudos. Conforme ilustrado na Figura 10, o atributo Disponibilidade se destaca com nove ocorrências, evidenciando assim que o processo de disponibilidade das aplicações baseadas em MSA superam sempre as aplicações monolíticas. Outros dois atributos que também merecem destaque são Confiabilidade com seis ocorrências e Escalabilidade com cinco ocorrências, reforçando o conceito de que sistemas baseados em MSA são mais confiáveis e tendem a ter um processo de escalabilidade mais rápido e fácil, resultando para ambos interessados (ou seja, desenvolvedores e usuários) um resultado final que permite atender as novas exigências de mercado. Os atributos Desempenho, Flexibilidade e Tempo de resposta foram identificados três vezes, seguidos pelo atributo Integridade com duas ocorrências e pelo atributo Manutenibilidade com uma ocorrência. Diante do exposto, pode-se dizer que tais atributos formam uma base de atributos que devem ser considerados quando se pretende modernizar um sistema para MSA.

Ainda em relação aos atributos ilustrados na Figura 10, duas considerações devem ser

Figura 10 – Atributos de qualidade identificados em nosso SMS




Fonte: Elaborada pelo autor.

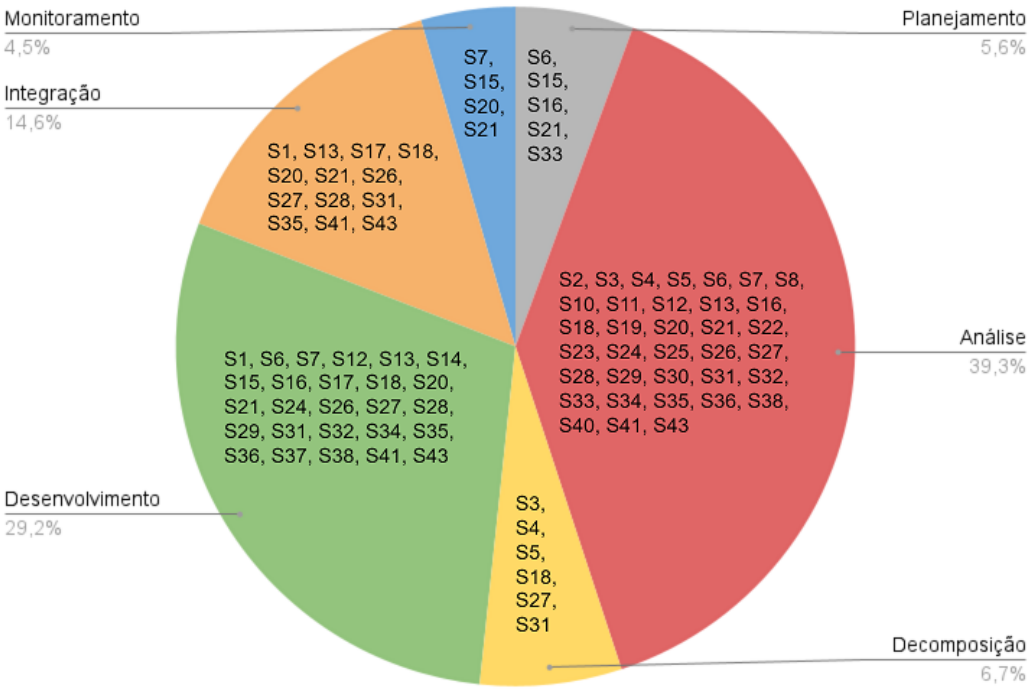
destacadas. A primeira se refere ao número de atributos em uma solução proposta e/ou sistema resultante da modernização. Nota-se que que alguns estudos abordaram mais de um atributo de qualidade, a saber: S1 – Disponibilidade e Flexibilidade; S7 – Escalabilidade, Disponibilidade e Tempo de Resposta; e, S27 – Escalabilidade, Disponibilidade e Integridade. A combinação de atributos é plenamente factível em relação ao desenvolvimento baseado em microsserviços, uma vez que o sistema resultante da modernização está inserido no contexto/domínio de aplicações distribuídas e diferentes atributos devem ser atendidos para que o sistema cumpra seus objetivos. Em contrapartida, a segunda se refere a ausência de atributos de qualidade em 17 estudos do nosso mapeamento (ou seja, S2, S12, S14, S17, S18, S25, S26, S28, S29, S30, S31, S34, S35, S36, S37, S38 e S41). Essa informação revela que grande parte dos estudos analisados não apontaram atributos de qualidade no processo de modernização, direcionando a atenção ao processo de identificação e desenvolvimento dos microsserviços.

### 3.2.3 Atividades de um processo de modernização

Esta QP teve como objetivo a identificação das atividades que devem ser executadas no processo de modernização de sistemas legados para MSA, com base nos estudos primários selecionados neste SMS. Além disso, também foi interesse desta questão entender se essas atividades são executadas manualmente ou apoiadas por processos automatizados. De acordo com as evidências deste SMS, seis atividades foram identificadas, a saber: (i) Planejamento, (ii) Análise, (iii) Decomposição, (iv) Desenvolvimento, (v) Integração, e (vi) Monitoramento. De acordo com Pressman (2016), o "Planejamento" é um processo estruturado e iterativo que envolve definição de objetivos, análise de requisitos, estimativa de recursos, agendamento de atividades, gerenciamento de riscos e comunicação entre as partes interessadas. Para o autor, "Análise" é o processo de compreender e especificar as necessidades dos usuários e

*stakeholders*, bem como os requisitos funcionais e não funcionais do sistema de software que está sendo desenvolvido. Na visão de Sommerville (2018), a "Decomposição" refere-se ao processo de dividir um sistema complexo em partes menores e mais gerenciáveis. O autor destaca a importância da decomposição como uma técnica fundamental para lidar com a complexidade inerente aos sistemas de software. Para o referido autor, "Desenvolvimento" é o processo de construção do software com base nos requisitos identificados durante a fase de análise. O autor enfatiza que o desenvolvimento de software envolve a implementação, teste e integração dos diferentes componentes e funcionalidades do sistema. Já para Newman (2015), a "Integração" refere-se ao processo de combinar e conectar diferentes componentes de software para formar um sistema coeso e funcional. Para o autor a integração é uma parte essencial do desenvolvimento de sistemas distribuídos, como os baseados em arquiteturas de microsserviços. De acordo com Fowler (2010), o "Monitoramento" refere-se ao processo de observação e coleta de métricas e dados operacionais de um sistema em tempo real. O referido autor enfatiza a importância do monitoramento para garantir a saúde, desempenho e confiabilidade de sistemas distribuídos, como os baseados em microsserviços. Conforme ilustrado na Figura 11, as atividades de maior destaque são Análise (39,30%) com 35 estudos e Desenvolvimento (29,20%) com 26 estudos. Na sequência, a atividade Integração (14,60%) com 13 estudos, Decomposição (6,70%) com 6 estudos, Planejamento (5,60%) com 5 estudos, e Monitoramento (4,50%) com 4 estudos completam a distribuição das atividades identificadas em nossa investigação.

Figura 11 – Atividades executadas no processo de modernização




Fonte: Elaborada pelo autor.

Analisando a distribuição ilustrada na Figura 11, nota-se que vários estudos desenvolveram suas propostas de solução com base em mais de uma das atividades mencionadas nesta seção. Por exemplo, o estudo S1 enfatizou o "Desenvolvimento" e "Integração", o estudo S6 o "Planejamento", "Análise" e "Desenvolvimento", o estudo S18 a "Análise", "Decomposição", "Desenvolvimento" e "Integração", e, por fim, o estudo S21 o "Planejamento", "Análise", "Desenvolvimento", "Integração" e "Monitoramento". Outra evidência dessa distribuição é que a maioria dos estudos não direciona seus esforços para as atividades de "Planejamento" e "Decomposição". Isso revela que os estudos tendem a concentrar suas propostas nas atividades de análise do monolítico e no desenvolvimento dos microsserviços. Em paralelo, observa-se o baixo número de estudos (S7, S15, S20 e S21) com atividades relacionadas ao monitoramento dos microsserviços, evidenciando uma carência significativa para esse tipo de aplicação, uma vez que os microsserviços são aplicações distribuídas que necessitam de monitoramento por natureza. Além disso, essa informação corrobora ainda mais com as evidências de nossa investigação, pois os estudos analisados se preocuparam mais com processo de análise para identificação de candidatos a microsserviços e desenvolvimento dos microsserviços do que com o processo de modernização como um todo. Por fim, vale destacar que apenas três estudos (S9, S39 e S42) não apresentaram atividades relacionadas ao processo de modernização de sistemas legados para o estilo MSA em suas propostas.

### 3.2.4 Organização dos microsserviços

Esta questão reúne evidências sobre a organização e/ou reorganização dos microsserviços durante o processo de modernização de uma aplicação legada. Essas evidências permitem estabelecer parâmetros sobre a nova aplicação quanto à aderência de novas *features*, além de outras características inerentes ao estilo arquitetural de microsserviços. Assim, para uma melhor compreensão dos resultados, as informações coletadas para essa questão foram analisadas em dois grupos, a saber: (i) arquitetura, representa as informações referentes as tecnologias utilizadas em uma arquitetura de microsserviços; e (ii) DevOps, representa as tecnologias utilizadas no processo de desenvolvimento dos microsserviços.

Conforme apresentado na Tabela 3, as tecnologias mais utilizadas nos estudos de nosso mapeamento foram: Containers e Docker com cinco estudos cada, e Spring Cloud com quatro estudos. Na sequência, API Gateway, Eureka, Spring Cloud Gateway e Serviço de descoberta foram as tecnologias mencionadas em dois estudos. As demais tecnologias (por exemplo, AWS, RabbitMQ, Clustering, Mensageria, Kubernates, Message Broker, Kafka, entre outras) foram mencionadas em pelo menos um estudo. Por fim, vale destacar que dos 43 estudos de nossa investigação, 27 estudos[1] (S2..S5, S8, S11..S17, S19, S22..S25, S28..S32, S34..S37, S40 e S41) não apresentaram evidências quanto ao uso de tecnologias.

[1] A notação S2..S5 representa um intervalo de estudos.

Tabela 3 – Tecnologias utilizadas na arquitetura de microsserviços

| **Tecnologias** | **Estudos** | **Qtde** |
|---|---|---|
| Containers | S1, S6, S7, S10, S27 | 5 |
| Docker | S18, S20, S21, S27, S39 | 5 |
| Spring Cloud | S9, S20, S21, S39 | 4 |
| API Gateway | S7, S27 | 2 |
| Eureka, Spring Cloud Gateway | S20, S21 | 2 |
| Serviço de descoberta | S7, S43 | 2 |
| API Restfull | S26 | 1 |
| Auth Server, AWS, Configuration Server, RabbitMQ | S27 | 1 |
| Config Repository, Logs/Tracing, Metrics Server | S42 | 1 |
| Clustering, Mensageria | S6 | 1 |
| Jenkins | S20 | 1 |
| Kubernates | S9 | 1 |
| Message Broker | S36 | 1 |
| Kafka, WebSocket Server | S33 | 1 |

Fonte: Elaborada pelo autor.

No que diz respeito às tecnologias utilizadas no processo de desenvolvimento dos microsserviços, apenas seis estudos reportaram evidências sobre esse assunto, como é apresentado na Tabela 4. As ferramentas de gerenciamento de versões GitHub e GitLab e o *framework* Spring Boot foram as tecnologias mais recorrentes com dois estudos cada. As demais (API Java Reflection, Spring AOP, APM Server, Jfrog, SVN, Swager, entre outras) foram detectadas em pelo menos um estudo.

Tabela 4 – Tecnologias utilizadas no desenvolvimento de microsserviços

| **Tecnologias** | **Estudos** | **Qtde** |
|---|---|---|
| GitHub, GitLab | S20, S27 | 2 |
| Spring Boot | S22, S39 | 2 |
| API Java Reflection, Spring AOP | S22 | 1 |
| APM Server, Logs Aggregator | S42 | 1 |
| Bitbucket, Jfrog, SVN | S18 | 1 |
| Nakadi, Open API, Swagger, ZMON | S39 | 1 |

Fonte: Elaborada pelo autor.

Com base nas informações reportadas nesta seção, observa-se que as tecnologias identificadas nas Tabelas 3 e 4 podem ser utilizadas para o desenvolvimento de microsserviços em um processo de modernização, e também na organização dos microsserviços em uma arquitetura de

ambiente distribuído. Nesse sentido, vale ressaltar que as tecnologias identificadas precisam de uma análise e de um estudo mais aprofundado para que se possa ter um melhor entendimento sobre suas funcionalidades e estabelecer assim as melhores escolhas para atender o projeto de um sistema baseado na arquitetura de microsserviços. Newman (2015) fornece *insights* valiosos sobre a organização dos microsserviços em uma arquitetura distribuída. O referido autor defende a abordagem de *design* baseada em domínio, onde os microsserviços são organizados em torno das fronteiras dos domínios de negócios. Para o autor, essa abordagem promove a coesão e a autonomia dos serviços, permitindo que eles sejam desenvolvidos, implantados e escalados de forma independente. Além disso, o autor destaca a necessidade de comunicação assíncrona entre os microsserviços, favorecendo a utilização de padrões de mensageria e eventos para facilitar a integração entre os serviços. Outro ponto importante abordado pelo autor é a necessidade de projetar microsserviços pequenos e focados, cada um com uma única responsabilidade. Em resumo, é enfatizada a importância da modularidade, autonomia e comunicação eficaz entre os microsserviços, onde todos são essenciais para o sucesso de uma arquitetura baseada em microsserviços.

## 3.3 Discussão dos Resultados

Com base nos resultados apresentados pelas questões apresentadas na Seção 3.2, uma análise dos 43 estudos selecionados neste mapeamento é apresentada. Antes de iniciar a discussão dos resultados, é importante salientar que o SMS desenvolvido neste trabalho de mestrado buscou ser fiel aos textos originais dos estudos, interpretando os dados em casos estritamente necessários.

Inicialmente, a **QP1** buscou apresentar um panorama geral sobre os estudos primários levantados por este SMS. Conforme apresentado na Seção 3.2.1, as publicações sobre esse assunto cobrem um intervalo dos últimos 3 anos inteiros de pesquisa e o ano de 2023 fracionado até o mês de setembro (**QP1.1**). Os estudos selecionados por este SMS foram publicados em *Conference*, *Journal*, *Sumposium* e *Workshop* (**QP1.2**), cujas publicações estão distribuídas entre Academia, Indústria e *Mixed* (**QP1.3**). Em seguida, as **QP2** a **QP4** buscaram apresentar um panorama sobre as características das soluções para a modernização de sistemas legados para a arquitetura de microsserviços. A seguir, uma síntese de cada QP é apresentada:

- **QP2**: Esta QP se preocupou com a identificação dos tipos de soluções existentes na literatura, tais como: (i) Metodologia, (ii) Método, (iii) Processo, (iv) Abordagem, (v) Pipeline, e (vi) Framework. A distribuição desses tipos de soluções pode ser vista com mais detalhes na Figura 7;
- **QP2.1**: Nesta QP foram levantados os domínios de aplicação que adotaram as soluções

propostas, sendo importantes para compreender as particularidades envolvidas nos contextos e cenários de utilização das soluções para a modernização de sistemas legados para a arquitetura de microsserviços. Esses domínios podem ser vistos com mais detalhes na Figura 8;

- **QP2.2**: Esta QP ficou responsável por avaliar as soluções de modernização, que foram classificadas como: (i) Estudo de caso, (ii) Experimento, (iii) Prova de conceito, e (iv) Não avaliada. Essa avaliação das soluções pode ser vista com mais detalhes na Tabela 2;

- **QP2.3**: Esta QP foi planejada para verificar os tipos de evidências de adoção nas soluções propostas, a saber: (i) Aplicabilidade na indústria, (ii) Ganhos/benefícios, e (iii) Nível de maturidade. A distribuição dos estudos por evidências de adoção pode ser melhor visualizada na Figura 9;

- **QP2.4**: Esta QP foi responsável por verificar os atributos de qualidade mais recorrentes no processo de modernização de sistemas legados para a arquitetura de microsserviços. Esse tipo de verificação é importante para validar as características de uma arquitetura de microsserviços após a sua modernização. Todos os atributos de qualidade podem ser visualizados na Figura 10;

- **QP3**: Conforme descrito na Seção 3.2.3, nesta QP foram identificadas as atividades que devem ser executados no processo de modernização de sistemas legados para a arquitetura de microsserviços. As atividades podem ser visualizadas na Figura 11, sendo elas: (i) Planejamento, (ii) Análise, (iii) Decomposição, (iv) Desenvolvimento, (v) Integração, e (vi) Monitoramento. Com base nas atividades identificadas, observa-se que os estudos analisados se preocuparam mais com o processo de análise para identificação de candidatos a microsserviços e no Desenvolvimento dos microsserviços; e

- **QP4**: O objetivo desta QP foi entender como os microsserviços são organizados, e também as features utilizadas no estilo arquitetural de microsserviços durante a modernização. Conforme descrito no Seção 3.2.4, dos 43 estudos analisados, somente 16 estudos continham informações relacionadas as tecnologias que foram utilizadas para uma arquitetura de microsserviços e no processo de desenvolvimento dos microsserviços. As tecnologias utilizadas na arquitetura de microsserviços podem ser visualizados na Tabela 3 e as tecnologias utilizadas no desenvolvimento dos microsserviços podem ser visualizadas na Tabela 4.

De modo geral, é possível notar que a modernização de sistemas legados para a arquitetura de microsserviços é uma área de pesquisa que possui grande potencial de exploração em diferentes cenários de aplicação. A partir das informações reportadas na Seção 3.2, foi possível

obter uma visão inicial de como esse assunto tem sido abordado na literatura científica. Além disso, durante o processo de extração e análise dos dados, foram percebidas algumas limitações nos estudos selecionados neste SMS, a saber:

- **Alto número de trabalhos com ausência de atributos de qualidade**: Pode-se dizer que os atributos de qualidade são essenciais para garantir que uma arquitetura baseada em microsserviços atenda aos requisitos necessários para suportar as operações e os objetivos de negócio de uma organização. Richardson (2018) aborda diversos atributos de qualidade dos microsserviços (por exemplo: disponibilidade, desempenho, escalabilidade, segurança e monitoramento). O autor destaca a importância em uma arquitetura de microsserviços e argumenta que eles são essenciais para garantir que os microsserviços atendam às necessidades dos usuários e dos negócios de forma eficaz. Newman (2015) reforça essa ideia, enfatizando que os atributos de qualidade são fundamentais para o sucesso e a eficácia do sistema. O referido autor ressalta que os atributos de qualidade, como disponibilidade, desempenho, escalabilidade, resiliência e segurança, não são apenas desejáveis, mas essenciais para garantir que os microsserviços atendam aos requisitos de negócios e às expectativas dos usuários;

- **Baixo número de trabalhos dedicados a atividades de "Planejamento", "Decomposição" e "Monitoramento"**: Vários autores da literatura fornecem *insights* valiosos sobre a importância dessas fases no contexto da modernização de sistemas legados para microsserviços. Newman (2015) discute amplamente sobre a importância de um planejamento cuidadoso antes de iniciar um processo de modernização e enfatiza a necessidade de alinhamento com os objetivos de negócio, avaliação dos sistemas legados e definição de uma estratégia abrangente para a modernização. O referido autor ressalta que uma decomposição eficaz permite que os sistemas legados sejam divididos em partes menores e mais gerenciáveis, facilitando uma migração gradual para uma arquitetura de microsserviços, permitindo que as equipes se concentrem em áreas específicas do sistema. Por fim, também destaca a importância do monitoramento contínuo para garantir o desempenho, a confiabilidade e a segurança dos microsserviços. Já Fowler (2010), destaca a importância da fase de planejamento como um dos principais pilares para o sucesso na modernização, e discute sobre a necessidade de uma abordagem cuidadosa para identificar microsserviços potenciais e desenvolver uma estratégia clara. O autor enfatiza que uma decomposição correta dos sistemas legados pode levar à criação de microsserviços coesos e com baixo acoplamento, promovendo a escalabilidade e a manutenibilidade do sistema. E ressalta a importância do monitoramento para entender o comportamento e o desempenho dos microsserviços em produção. Por fim, Richardson (2018) aborda a importância do planejamento na modernização de sistemas legados para microsserviços, destacando

a necessidade de uma análise detalhada dos sistemas existentes e uma abordagem estruturada para identificar e decompor funcionalidades em microsserviços. Para o autor, a decomposição é uma etapa fundamental para identificar os limites dos microsserviços. Esse autor enfatiza a importância de ferramentas de monitoramento e observabilidade para rastrear transações, detectar falhas e garantir uma resposta rápida a eventos inesperados. De acordo com as informações, pode-se entender que as fases mencionadas acima são essenciais para obter o sucesso em um processo de modernização de sistemas legados para MSA, portanto, se faz necessário que as atividades sejam melhor analisadas e exploradas em cada fase; e

- **Alto número de estudos que não apresentaram evidências quanto ao uso de tecnologias no processo de modernização e organização dos microsserviços**: A organização dos microsserviços desempenha um papel fundamental na arquitetura de microsserviços, conforme discutido por vários autores da literatura. Richardson (2018) destaca a importância de organizar microsserviços de forma a minimizar o acoplamento e maximizar a coesão. Além disso, ressalta a adoção de padrões de *design* para gerenciar a complexidade e simplificar a integração entre os serviços. Para Newman (2015), a organização dos microsserviços deve refletir a estrutura organizacional da equipe e os limites de contexto do negócio. O autor enfatiza, a necessidade de uma arquitetura modular e flexível, que permita a evolução independente dos serviços e facilite a entrega contínua. Por fim, Fowler (2010) destaca a importância de uma organização baseada em contextos limitados (do inglês, *bounded contexts*) para garantir uma separação clara das responsabilidades e uma evolução incremental dos serviços. O autor aponta a necessidade de equipes autônomas e descentralizadas, responsáveis por serviços específicos e orientadas por princípios de design sólidos. Sendo assim, para que os microsserviços funcionem corretamente é preciso garantir que todos estejam se comunicando adequadamente e coordenando suas atividades em uma arquitetura bem definida. Para isso, a escolha das tecnologias e a organização adequada dos microsserviços, de acordo com o domínio da aplicação, é um processo fundamental para garantir o sucesso da aplicação em um processo de modernização de sistemas legados para uma arquitetura de microsserviços.

Diante do exposto, pode-se dizer que o tema de pesquisa abordado neste projeto tem sido bastante explorado nos últimos anos, principalmente na academia, com foco maior em trabalhos relacionados a abordagens para análise de sistemas monolíticos com a finalidade de identificação de candidatos a microsserviços e desenvolvimento dos microsserviços. Apesar da grande variedade de domínios de aplicação em que os trabalhos tem se dedicado, percebe-se que muitos trabalhos preferem utilizar sistemas *open source* em seus estudos de caso, talvez pela praticidade ou falta de acesso a sistemas utilizados no mercado. Entende-se, que várias atividades

relacionadas as fases de “Planejamento”, “Decomposição” e “Monitoramento” precisam ser mais exploradas, devido a importância dessas fases em um processo de modernização de sistemas legados para MSA. Nota-se também, que os atributos de qualidade dos microsserviços são essenciais para garantir que tal modernização atenda aos requisitos necessários para atingir os objetivos desejados, assim como uma correta organização dos microsserviços em uma arquitetura de ambiente distribuído para garantir autonomia e comunicação eficaz entre os microsserviços.

## 3.4 Considerações Finais

Este capítulo apresentou uma visão abrangente sobre a modernização de sistemas legados para a arquitetura de microsserviços. Na Seção 3.2 foram discutidas as respostas às questões de pesquisa definidas no protocolo metodológico do Apêndice A. Em seguida, na Seção 3.3, foi realizada uma análise dos 43 estudos selecionados no mapeamento sistemático conduzido neste mestrado. A partir dessa investigação, constatou-se que, embora existam diversas iniciativas relevantes, grande parte dos esforços concentram-se nas atividades de análise de sistemas monolíticos e na identificação de candidatos a microsserviços, assim como no desenvolvimento dos próprios serviços. Em outra perspectiva, observou-se que várias etapas essenciais do processo de modernização ainda permanecem pouco exploradas, o que dificulta uma abordagem mais completa e estruturada para alcançar plenamente uma arquitetura baseada em microsserviços. Diante dessas lacunas identificadas, foi concebido o processo Micro4Delphi, proposto neste projeto de mestrado como uma abordagem sistemática para apoiar a modernização de sistemas legados desenvolvidos em Delphi. Em resumo, esse processo (veja Capítulo 4), visa integrar práticas, atividades e orientações capazes de oferecer um caminho mais consistente e gradual para a transformação de sistemas monolíticos para a arquitetura de microsserviços.

# 4 O PROCESSO MICRO4DELPHI

## 4.1 Considerações iniciais

Neste capítulo são reportados os detalhes do projeto do Micro4Delphi desenvolvido neste projeto de mestrado acadêmico. Em resumo, este processo tem por objetivo apoiar a modernização de sistemas legados desenvolvidos em Delphi para a arquitetura de microsserviços. Inicialmente, na Seção 4.2 são descritos os passos para a concepção do processo Micro4Delphi, que teve como origem o mapeamento da literatura apresentado no Apêndice A e posterior interação com profissionais da indústria de software de duas empresas. Em seguida, na Seção 4.3 é apresentado o Micro4Delphi como versão resultante da colaboração entre a academia e a indústria, sendo descritas as atividades do processo, bem como os passos pertinentes a cada atividade. Por fim, na Seção 4.4 são apresentadas as considerações finais deste capítulo.

## 4.2 Concepção do processo

Conforme apresentado na Seção 3.2.3, a investigação da literatura sugere que a atividade de modernização de sistemas legados para a arquitetura de microsserviços tem sido conduzida com base em seis macro atividades, a saber: planejamento, análise, decomposição, desenvolvimento, integração e monitoramento. Embora tais atividades tenham sido identificadas, nota-se que os interessados (ou seja, pesquisadores e profissionais) têm concentrado seus esforços em três atividades: análise, desenvolvimento e integração. Em outra perspectiva, a investigação conduzida na Seção 3.2.3 também revelou que as iniciativas propostas na literatura contemplaram, em sua maioria, uma ou duas atividades (25 estudos). Em contrapartida, somente 15 estudos exploraram três ou mais atividades em suas iniciativas, e nenhum estudo selecionado na referida investigação contemplou as seis atividades. Na Figura 12 é ilustrada a distribuição desses 15 estudos em relação às atividades contempladas em suas iniciativas propostas na literatura. Os dados dessa distribuição revelam a recorrência das atividades na última linha e o número de atividades contempladas em cada estudo. Além disso, nenhum estudo abordou as seis atividades em sua iniciativa e somente o estudo S21 se aproxima da totalidade ao lidar com cinco atividades relacionadas à modernização de sistemas legados para a arquitetura de microsserviços.

**Primeira versão.** Após mapear as atividades essenciais de um processo de modernização (veja Figuras 11 e 12), bem como os estudos mais relevantes da investigação conduzida (S18, S20, **S21**, S27, e S31), a primeira versão do processo Micro4Delphi foi gerada, como ilustrado na Figura 13. Como pode ser observado, o processo foi organizado em seis atividades, tendo

Figura 12 – Distribuição dos estudos que apresentaram maior recorrência quanto ao número de atividades de um processo de modernização. O número um (1) indica que o estudo contempla a atividade. O número zero (0) indica que o estudo não contempla a atividade.

| | Estudo | Planejamento | Análise | Decomposição | Desenvolvimento | Integração | Monitoramento | |
|---|---|---|---|---|---|---|---|---|
| 1 | S6 | 1 | 1 | 0 | 1 | 0 | 0 | 3 |
| 2 | S7 | 0 | 1 | 0 | 1 | 0 | 1 | 3 |
| 3 | S13 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| 4 | S15 | 1 | 0 | 0 | 1 | 0 | 1 | 3 |
| 5 | S16 | 1 | 1 | 0 | 1 | 0 | 0 | 3 |
| 6 | S18 | 0 | 1 | 1 | 1 | 1 | 0 | 4 |
| 7 | S20 | 0 | 1 | 0 | 1 | 1 | 1 | 4 |
| 8 | S21 | 1 | 1 | 0 | 1 | 1 | 1 | 5 |
| 9 | S26 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| 10 | S27 | 0 | 1 | 1 | 1 | 1 | 0 | 4 |
| 11 | S28 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| 12 | S31 | 0 | 1 | 1 | 1 | 1 | 0 | 4 |
| 13 | S35 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| 14 | S41 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| 15 | S43 | 0 | 1 | 0 | 1 | 1 | 0 | 3 |
| | | 4 | 14 | 3 | 15 | 11 | 4 | |

Fonte: Elaborada pelo autor.

início no planejamento até o monitoramento. Além disso, cada atividade é composta por um conjunto de passos que diferem em número e conteúdo para que a execução da atividade seja conduzida de maneira adequada. No que diz respeito ao conteúdo, tanto as atividades quanto os passos nelas contidos foram devidamente documentados de modo a nortear os interessados em conduzir um processo de modernização. Sobre o processo, algumas considerações devem ser destacadas, a saber:

**Natureza cíclica.** Embora o processo ilustrado na Figura 13 tenha aparência matricial, onde as colunas representam as atividades e as linhas os passos, o processo têm natureza incremental e cíclica. De acordo com conceitos da engenharia de software moderna (PRESSMAN, 2016), processos incrementais tendem a apresentar melhor condução e acompanhamento por parte dos times de desenvolvimento e interessados no resultado final. Além disso, as evidências reunidas no Capítulo 3 revelam que o avanço gradativo da modernização também favorece a transição entre o sistema legado e o sistema modernizado, fazendo com que possam atuar de maneira concomitante.

**Ponto de partida.** Como o Micro4Delphi é um processo que visa apoiar a modernização de sistemas legados desenvolvidos em Delphi, o ponto de partida para condução do processo

Figura 13 – Primeira versão do Micro4Delphi

| MICRO4DELPHI | | | | | |
|---|---|---|---|---|---|
| **Planejamento** | **Análise** | **Decomposição** | **Desenvolvimento** | **Integração** | **Monitoramento** |
| Definir escopo | Analisar o modelo de negócio | Identificar funcionalidades candidatas a microsserviços | Reestruturar o repositório de dados | Malha de Serviços e Suporte | Elaborar Dashboard |
| Estruturar a equipe | Analisar o sistema legado | Definir design da decomposição | Definir casos de testes | Integração Contínua | Emitir alertas |
| Cultura organizacional | Analisar o banco de dados | Definir reestruturação dos dados | Implementar microsserviços | Entrega Contínua | Efetuar manutenção |
| Capacitação e treinamentos | Realizar pesquisas UX/UI | Modelar APIs dos microsserviços | Implementar comunicação entre os microsserviços | Deployment Contínuo | Permitir reversão |
| Definir tecnologias | Identificar principais funcionalidades | Priorizar serviços a serem desenvolvidos | Definir controle de versão | Automatizar testes de unidade e integração | Diagnóstico |
| | Modelar estrutura tecnológica | Definir padrão arquitetural | Migrar banco de dados legado | | Lidar com inconsistências devido a replicação |
| | | Definir especificação técnica | Migrar monolítico para microsserviços | | |
| | | QA de Software e Métricas | | | |
| | | Elaborar documentação | | | |

Fonte: Elaborada pelo autor.

pode ser: (i) sistema binário em executável; (ii) sistema binário e seu respectivo repositório de código fonte; ou (iii) conteúdo do segundo item com aporte de artefatos de documentação, manuais, entre outros. Diante do exposto, dependendo do valor do sistema e disponibilidade de artefatos, o tempo de condução do processo pode ser significativamente reduzido, uma vez que tais artefatos podem ser reutilizados no desenvolvimento dos microsserviços.

**Primeira interação.** Após conduzir a elaboração da primeira versão do Micro4Dephi, o processo foi apresentado em uma empresa colaboradora com este projeto de mestrado acadêmico, tendo envolvimento de sete profissionais especialistas em Delphi com diferentes níveis de conhecimento.

No que diz respeito à colaboração, os pesquisadores fizeram uma apresentação geral do processo aos profissionais da referida empresa, sendo destacada cada atividade do processo e seus respectivos passos. Em seguida, o documento do processo foi entregue aos profissionais para

leitura, acompanhado de um questionário contendo 60 questões, conforme pode ser observado no Apêndice C. Esse questionário foi organizado em três seções, sendo a primeira voltada para a detecção de parâmetros relacionados ao perfil do profissional (por exemplo, cargo, tempo de experiência), a segunda voltada para os interesses de modernização e, por fim, a terceira voltada para cada atividade do processo.

Após coleta das informações, os pesquisadores analisaram as respostas relacionadas ao processo de modernização e 15 passos apresentaram problemas quanto à compreensão. Visando sanar os problemas identificados, os pesquisadores atualizaram a documentação do processo Micro4Delphi em duas etapas, a saber: (i) refinamento de redação do texto sem que o número de atividades e passos fosse modificado; e (ii) definição de entradas e saídas para cada atividade e seus respectivos passos. De acordo com as evidências coletadas, esta última pode ser considerada um diferencial importante para entendimento e utilização do processo em um ambiente real de desenvolvimento de software. Diante do exposto, pode-se dizer que o levantamento da literatura revelou um conjunto de atividades e passos coerentes ao cotidiano profissional de uma empresa da área. Embora os resultados nessa primeira interação tenham revelado evidências concretas para readequação do processo (ou seja, remoção ou junção de passos), os pesquisadores optaram por manter o processo em sua versão original e replicar o procedimento em uma segunda empresa.

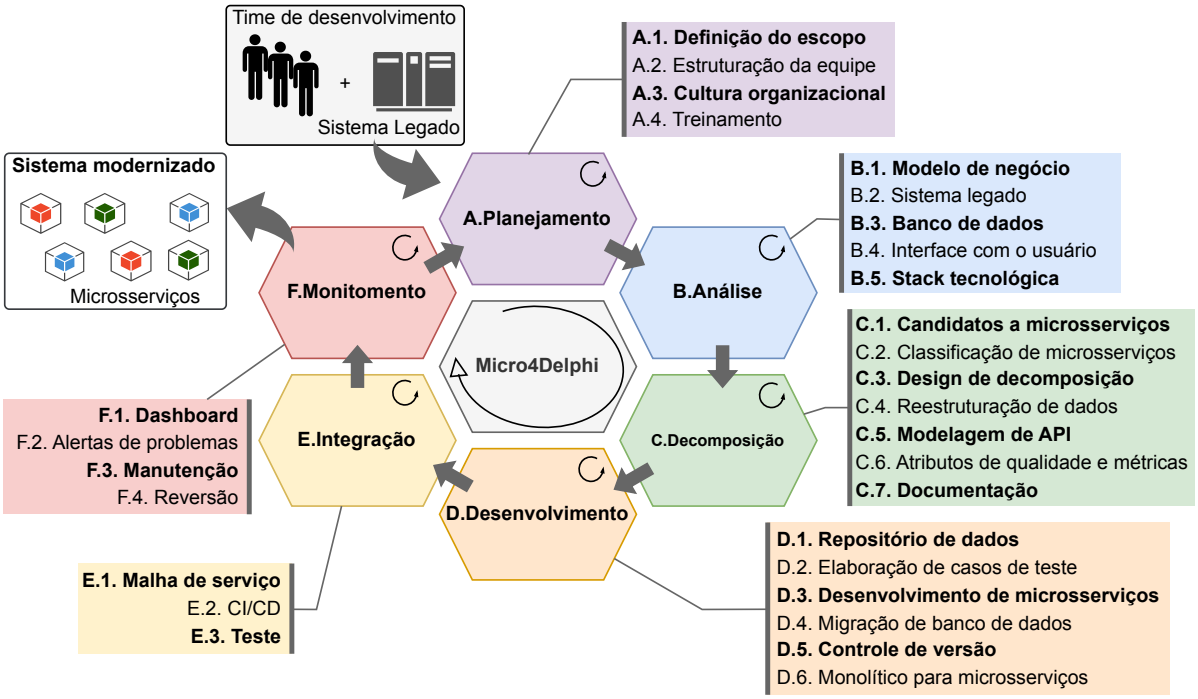
**Segunda interação.** Após refinar a redação do processo, o Micro4Delphi foi submetido para análise por quatro profissionais de outra empresa, com nível de conhecimento e experiência distintos. Para isso, o mesmo procedimento descrito na primeira interação foi adotado, tendo como resultado um número menor de dúvidas ou falta de compreensão quanto aos passos (ou seja, quatro) a serem executados para que a atividade de modernização seja realizada. Após analisar as respostas e evidências coletadas, os pesquisadores optaram por preservar o número de passos em cada atividade pelos seguintes motivos: (i) relatos de experiência da academia; (ii) tamanho das equipes que participaram das interações; e (iii) nível de conhecimento/experiência profissional dos profissionais participantes.

**Terceira interação.** Os pesquisadores, após compilar as evidências fornecidas pelos profissionais de ambas as empresas, optaram por refinar o processo Micro4Delphi. Vale destacar que esse refinamento procurou preservar a essência das atividades e passos identificados com a investigação da literatura apresentada no Capítulo 3. Dessa forma, sem descaracterizar a identidade do processo, alguns passos foram reorganizados e/ou agrupados, visando otimizar o trabalho das equipes encarregadas de conduzir a modernização. Após a readequação, o processo Micro4Delphi passou de 38 para 29 passos, os quais permanecem distribuídos pelo conjunto de atividades identificadas. A seguir, Seção 4.3, é apresentada uma descrição detalhada de cada atividade do processo Micro4Delphi e seus respectivos passos.

## 4.3 Atividades e passos do processo

Como descrito na Seção 4.2, o Processo Micro4Delphi foi elaborado com base na investigação da literatura apresentada no Capítulo 3 e na interação com duas empresas especializadas em desenvolvimento de software em Delphi. Na Figura 14 é ilustrada a versão final do processo Micro4Delphi, o qual é baseado em uma abordagem de desenvolvimento incremental e evolutiva que permite a modernização de um sistema legado de maneira gradativa. Como pode ser observado, o time de desenvolvimento de posse do sistema legado conduz a modernização com base em seis atividades para que os microsserviços sejam elaborados. Sobre o time de desenvolvimento, dois elementos devem ser destacados: (i) membros participantes, são pessoas envolvidas com a atividade de modernização com diferentes tipos de papéis (por exemplo, engenheiros de software, arquitetos de software, desenvolvedores, especialistas de domínio, entre outros); e (ii) tamanho do time, número de participantes interessados/envolvidos definido de acordo com os interesses da organização para que o processo de modernização seja conduzido. O sistema legado "ideal" para o Micro4Delphi pode ser um sistema binário (ou seja, executável em funcionamento), seu repositório de código-fonte, bem como os artefatos de documentação (por exemplo, modelo entidade-relacionamento) gerados durante a etapa de desenvolvimento. Como resultado, espera-se que o sistema modernizado seja desenvolvido de maneira gradativa e com participação ativa dos interessados (ou seja, clientes). A seguir, uma descrição detalhada das atividades e dos respectivos passos é apresentada.

Figura 14 – Visão geral do processo Micro4Delphi




Fonte: Elaborada pelo autor.

## A. Planejamento

O planejamento é uma atividade crucial em qualquer projeto de desenvolvimento de software, incluindo a modernização de sistemas legados para a arquitetura de microsserviços. Em resumo, esta atividade envolve a definição de objetivos, estratégias e recursos necessários para atingir os resultados desejados (ou seja, sistema modernizado). Ao planejar as atividades de modernização, as organizações e times de desenvolvimento aumentam as chances de sucesso, antecipando adversidades e garantindo que os objetivos sejam alcançados dentro dos prazos e recursos definidos. Do ponto de vista da organização, metodologia e infraestrutura adequadas devem ser adotadas para condução das práticas de modernização. No que diz respeito ao desenvolvimento, a adequação de cultura relacionada ao desenvolvimento deve ser introduzida aos times (LI; MA; LU, 2020; AUER *et al.*, 2021). A seguir, uma descrição detalhada dos passos sugeridos para que esta atividade seja conduzida de maneira adequada é apresentada.

### A.1. Definição do escopo

Este passo pode ser considerado fundamental em qualquer projeto de modernização, pois envolve estabelecer claramente os limites e objetivos do projeto, o que inclui as funcionalidades específicas a serem migradas para microsserviços e os requisitos essenciais que devem ser atendidos. Ao definir o escopo de maneira adequada, as equipes passam a ter uma compreensão global do sistema a ser modernizado, uma visão geral sobre as fronteiras desse sistema e perspectivas para incorporação de novas funcionalidades. Como resultado, espera-se que o trabalho de modernização seja realizado de maneira satisfatória, de modo a facilitar o alinhamento de expectativas entre as partes interessadas e com foco na entrega bem-sucedida dos objetivos do projeto (COLANZI *et al.*, 2021b).

De acordo com Auer *et al.* (2021), a definição do escopo passa pelo entendimento do sistema que se pretende modernizar tanto do ponto de vista funcional quanto aos problemas existentes nesse sistema. Para isso, entender os elementos disponíveis de um sistema legado é considerado o ponto de partida para o processo de modernização proposto. De acordo com a investigação conduzida neste projeto de mestrado acadêmico (veja Capítulo 3) e interação com a indústria de software (veja Seção 4.2), esses elementos podem ser: sistema binário, repositório de código-fonte, artefatos de documentação (por exemplo, documento de especificação de requisitos, diagramas, entre outros), ou combinação de ambos. Partindo das informações disponíveis, as seguintes atividades podem colaborar com a definição do escopo, a saber:

**Levantamento de requisitos.** Esta atividade envolve conduzir um levantamento detalhado dos requisitos do sistema legado e das necessidades do negócio é essencial para entender o que deve ser incluído no escopo do projeto. Isso pode envolver entrevistas com *stakehol-*

*ders*, análise de documentação existente, revisão de processos de negócios, entre outras atividades.

**Estabelecimento de limites e metas.** Esta atividade visa definir claramente os limites do projeto, ou seja, quais partes do sistema legado serão incluídas na modernização e quais serão deixadas de fora. Como resultado, as equipes de modernização podem estabelecer metas claras e mensuráveis para o projeto, como prazos de entrega, qualidade do software e benefícios esperados.

**Documentação do escopo.** Esta atividade tem por objetivo documentar o escopo do projeto, visando garantir e/ou facilitar que as partes interessadas na modernização (ou seja, equipes de desenvolvimento e clientes) tenham uma compreensão comum do resultado final (ou seja, sistema modernizado). De maneira resumida, esse documento pode conter uma descrição detalhada das funcionalidades incluídas, exclusões explícitas e critérios de aceitação.

**Revisão e validação do escopo.** Após a definição inicial do escopo, é importante revisar e validar as informações com as partes interessadas para garantir sua precisão e completude, pois evita mal-entendidos e conflitos durante a execução do projeto. Outro ponto importante a ser destacado aqui é a não transferência involuntária de problemas do sistema legado para o sistema modernizado.

Após conduzir as atividades supracitadas, os seguintes artefatos podem ser gerados:

| **Artefatos resultantes do Passo A.1** |
|---|
| **AT1.** Artefatos de documentação do software legado. |
| **AT2.** Modelo Entidade Relacionamento (MER), se não houver. |
| **AT3.** Diagrama de Classes em UML, se não houver. |
| **AT4.** Diagrama de Fluxo de Dados (DFD), se não houver. |

### A.2. Estruturação da equipe

Definir a equipe é um passo essencial para a modernização de software, pois uma equipe deve reunir características que corroboram com a natureza do desenvolvimento baseado em microsserviços, além de apresentar habilidades para lidar com o sistema legado e boas práticas de Engenharia de Software. De acordo com Mazzara *et al.* (2021), uma equipe deve reunir as seguintes características: concisa com relação ao número de membros, multidisciplinar, independente, autônoma e alinhada às práticas DevOps. Nesse sentido, é importante identificar a aderência e/ou cultura da empresa em relação às práticas de DevOps, que requerem infraestrutura para promover: integração contínua, implantação contínua e entrega contínua. Diante

do exposto, pode-se dizer que estruturar a equipe é uma etapa crucial para o sucesso de um projeto de modernização de sistemas legados para arquiteturas de microsserviços. Ao estruturar um grupo de desenvolvedores em uma equipe bem organizada e colaborativa, com expertise (ou seja, competência ou qualidade de especialista) adequada em todos os aspectos do processo de desenvolvimento, assegura-se que o projeto de modernização seja conduzido de maneira eficiente. A seguir, uma síntese sobre as principais características que devem ser levadas em consideração ao se organizar as equipes de modernização é apresentada:

**Identificação de papéis e responsabilidades.** Definir os diferentes papéis dos envolvidos (por exemplo, arquiteto de software, desenvolvedores, engenheiros de qualidade, analistas de negócios, entre outros) em um projeto de modernização é um elemento-chave para o sucesso, pois auxilia na atribuição de responsabilidades de cada membro da equipe e evita sobreposições ou lacunas de trabalho. Para isso, sugere-se, por exemplo, a utilização da matriz RACI (do inglês, *Responsible, Accountable, Consulted, Informed*) para o gerenciamento do projeto.

**Formar uma equipe multidisciplinar.** Uma equipe diversificada, composta por membros com diferentes habilidades e experiências, é essencial para abordar os diversos aspectos do projeto. Nesse sentido, sugere-se incluir especialistas com habilidades específicas em arquitetura de software, desenvolvimento baseado em microsserviços, integração de sistemas, testes de software, e gestão de projetos.

**Estabelecer liderança.** Uma equipe de desenvolvimento de software deve ter um líder responsável por coordenar e supervisionar as atividades de desenvolvimento. Como os projetos de modernização têm natureza eclética do ponto de vista de recursos (por exemplo, linguagens de programação) e conhecimento (por exemplo, estilos e padrões arquiteturais), essa tarefa pode incluir, no mínimo, um líder técnico (responsável pela arquitetura e implementação dos microsserviços) e um gerente de projeto (encarregado de garantir que o projeto seja entregue dentro do prazo e do orçamento).

**Promover colaboração e comunicação.** Visando estabelecer um ambiente de trabalho colaborativo, onde os membros da equipe possam compartilhar ideias, resolver problemas e colaborar efetivamente, as organizações/empresas devem estabelecer canais de comunicação claros e regulares para garantir que todos estejam alinhados com os objetivos e progresso do projeto. Para isso, infraestrutura para comunicação e gestão de projetos deve ser implantada pelas organizações/empresas de modo a facilitar resolução de problemas e impulsionar o controle das atividades.

**Fomentar o aprendizado contínuo.** Visando manter as equipes atualizadas com as tendências e tecnologias atuais, as organizações/empresas devem facilitar o desenvolvimento

profissional por meio de treinamentos, *workshops* técnicos e participação em eventos (indústria e/ou científicos). Evidências da literatura sugerem que uma equipe atualizada com as melhores práticas e tendências em arquitetura de software e desenvolvimento de microsserviços é essencial para o sucesso do projeto.

### A.3. Cultura organizacional

Este passo visa promover a aceitação das mudanças decorrentes da adoção da arquitetura microsserviços e buscar maneiras de institucionalizá-la como cultura organizacional. Em outras palavras, essa mudança na cultura organizacional das empresas requer adequações desde o foco nas capacidades de negócio até a entrega contínua e valiosa de software. Diante do exposto, pode-se dizer que a cultura organizacional desempenha um papel fundamental na modernização de sistemas legados para arquiteturas de microsserviços (MAZZARA *et al.*, 2021; COLANZI *et al.*, 2021b).

Ao promover uma cultura organizacional que valoriza o aprendizado, a experimentação, a colaboração e a resiliência, as organizações podem criar um ambiente propício para o sucesso na modernização, assegurando que a transição seja realizada de maneira eficaz e que os benefícios desse novo estilo arquitetural sejam maximizados. A seguir são destacadas algumas considerações sobre como a cultura organizacional podem influenciar o processo de modernização, a saber (MAZZARA *et al.*, 2021; COLANZI *et al.*, 2021b):

- **Mentalidade orientada para o aprendizado.** Promover a mentalidade de aprendizado contínuo é essencial para incorporar mudanças e inovações em uma organização. Os membros das equipes devem estar dispostos a explorar novas tecnologias e práticas de desenvolvimento, adquirir novas habilidades, e adaptar-se a novos métodos de trabalho.

- **Encorajamento à experimentação.** Uma cultura que valoriza a experimentação e o aprendizado com os erros é fundamental para a modernização de software. Os membros das equipes devem se sentir confortáveis para experimentar novas abordagens e soluções sem receio de fracassar, pois esse tipo de conduta permite a descoberta de soluções inovadoras e aprimoramento contínuo dos processos.

- **Colaboração e comunicação aberta.** Uma cultura que promove a colaboração e a comunicação aberta facilita a troca de conhecimento/experiências entre os membros das equipes. A comunicação eficaz ajuda a alinhar expectativas, resolver conflitos e garantir que todos estejam alinhados com os objetivos do projeto.

- **Resiliência e flexibilidade.** A modernização de sistemas legados pode ser um processo complexo e desafiador, sujeito a contratempos e obstáculos inesperados (imprevisibili-

dade). Uma cultura organizacional resiliente e flexível permite que as equipes se adaptem e respondam de maneira eficaz às mudanças e imprevistos. Em outras palavras, esse tipo de conduta envolve a capacidade de superar obstáculos e ajustar estratégias para que os objetivos de longo prazo sejam mantidos.

- **Liderança.** Os líderes devem demonstrar apoio ativo ao projeto, fornecendo recursos adequados, removendo obstáculos e promovendo uma mentalidade de inovação e melhoria contínua em toda a organização.

### A.4. Treinamento

Identificar habilidades e aprendizados necessários para lidar com a arquitetura de microsserviços, promovendo a devida capacitação técnica aos envolvidos, é um passo essencial em um cenário de modernização. Ao fornecer capacitação e treinamentos abrangentes para a equipe, as organizações podem criar perspectivas favoráveis quanto ao preparo dos evolvidos e, consequentemente, com a capacidade de enfrentamento de desafios inerentes à modernização de sistemas legados para arquiteturas de microsserviços. A capacitação e treinamentos não visam apenas assegurar uma transição suave e bem-sucedida, mas também capacitam as equipes, por exemplo, a aproveitarem ao máximo os benefícios desse estilo arquitetural (SOLDANI; TAMBURRI; Van Den Heuvel, 2018). A investigação da literatura cinza conduzida por esses autores revelou que um dos principais fatores de insucesso na transição de um sistema legado para a arquitetura de microsserviço foi a falta de conhecimento dos envolvidos nesse estilo arquitetural.

Como forma de nortear as organizações/empresas, a investigação conduzida da literatura sugere atenção especial com as seguintes habilidades: (i) estilo arquitetural de microsserviços, as equipes devem ter compreensão sólida sobre, por exemplo, os conceitos fundamentais, como separação de interesses, autonomia de microsserviços e escalabilidade horizontal; (ii) metodologias ágeis, as equipes devem entender os princípios e práticas-chave, como interação rápida, colaboração interdisciplinar e entrega incremental; (iii) abordagem de projeto de software, as equipes devem ser capacitadas a identificar e delimitar domínios de negócio, criar modelos de domínio ricos e definir fronteiras de contexto para microsserviços.

## B. Análise

A análise é uma atividade essencial no processo de modernização, pois permite que a equipe de desenvolvimento avalie a disponibilidade de artefatos de software nos sistemas legados a serem modernizados. Isso inclui repositórios de código-fonte, sistemas binários e documentação técnica. Como resultado, espera-se que as informações coletadas sejam suficientes para facilitar a formulação de um processo de tomada de decisão que determinará o ponto ideal de

iniciação para o processo de decomposição. Para isso, recomenda-se identificar os componentes e funcionalidades do sistema legado, compreender seus limites dentro do domínio de negócios, analisar as dependências entre os módulos e estabelecer uma base preliminar para o *design* e implementação da arquitetura de microsserviços (FREIRE *et al.*, 2021; DEHGHANI *et al.*, 2022; PARIKH *et al.*, 2022; KURYAZOV; JABBOROV; KHUJAMURATOV, 2020). A seguir, uma descrição detalhada dos passos sugeridos para a condução adequada desta atividade é apresentada.

### B.1. Modelo de negócio

Neste passo é essencial que o time de desenvolvimento compreenda as operações e processos de negócio do sistema legado e da organização. Em outras palavras, este passo envolve o entendimento pleno do "*core business*" e das nuances do domínio do sistema, que pode ser sintetizado na identificação das principais atividades, dos fluxos de trabalho, do pessoal envolvido e dos requisitos específicos de cada área ou departamento. Para isso, alguns documentos podem auxiliar nessa fase, a saber: (i) documentos de estratégia de negócio; (ii) plano e processos de negócios; (iii) documentação de produtos e serviços; (iv) análise de mercado e competitividade (PRASANDY *et al.*, 2020).

Durante esse passo, o time de desenvolvimento deve investigar como as diferentes partes do sistema legado suportam ou afetam os processos de negócio da organização. Além disso, o time também deve identificar os pontos de integração entre os sistemas e como essas interações podem ser redefinidas ou otimizadas na nova arquitetura de microsserviços. Em paralelo, o time deve considerar os requisitos regulatórios, de segurança e de conformidade que podem impactar a arquitetura de microsserviços. Esses requisitos visam assegurar que o sistema modernizado esteja em conformidade com as leis e regulamentações aplicáveis ao setor e à região em que a organização está atuando ou tem perspectiva para atuar (KYRYK *et al.*, 2022; MA *et al.*, 2022). Em resumo, pode-se dizer que a análise do modelo de negócio é um passo crítico para o sucesso da modernização para uma arquitetura de microsserviços, pois fornece *insights* valiosos sobre como os sistemas de TI podem suportar e alavancar os objetivos e operações de negócio da organização.

### B.2. Sistema legado

Durante este passo, o time de desenvolvimento deve realizar uma avaliação abrangente do sistema legado existente para entender sua arquitetura, componentes, funcionalidades e integrações. Em outras palavras, sugere-se começar pelas seguintes questões como: (i) O que eu tenho do sistema legado? (ii) O que eu devo produzir em termos de artefatos de documentação para auxiliar no entendimento do sistema?

Em resposta aos questionamentos apresentados, espera-se obter evidências sobre o código-fonte, o banco de dados, a infraestrutura de hospedagem, e quaisquer documentações disponíveis. Essas evidências podem ser sintetizadas em pontos problemáticos (ou que necessitam de melhorias), como código obsoleto, dependências complexas, gargalos de desempenho, e áreas de baixa escalabilidade. Essa análise ajuda a equipe a entender os desafios e riscos envolvidos na migração para microsserviços e a definir uma estratégia adequada para a modernização do sistema (PRASANDY *et al.*, 2020). Após conduzir a avaliação supracitada, os seguintes artefatos podem ser gerados:

| **Artefatos resultantes do Passo B.2** |
|---|
| **AT5.** Diagrama da arquitetura do sistema legado. |
| **AT6.** Mapeamento de funcionalidades críticas. |
| **AT7.** Documentação das restrições do sistema. |

### B.3. Banco de dados

Neste passo é sugerido que o time de desenvolvimento conduza uma análise do banco de dados ou abordagens de persistência do sistema legado, observando entidades e atributos que exerçam influência sobre o domínio da aplicação. Durante essa análise, o time de desenvolvimento deve examinar a estrutura do banco de dados existente, incluindo tabelas, índices, relacionamentos e procedimentos armazenados. O objetivo dessa análise é compreender como os dados são organizados, armazenados, acessados e manipulados pelo sistema legado. Em outras palavras, como as entidades e atributos exercem influência sobre o domínio da aplicação. Além disso, é essencial identificar possíveis problemas, como redundâncias, inconsistências e dependências complexas entre as tabelas (MAZZARA *et al.*, 2021).

Para conduzir uma análise detalhada em sistemas legados desenvolvidos em Delphi, algumas atividades são sugeridas:

- **Identificar o SGBD utilizado.** De acordo com o banco de dados no sistema legado, é possível identificar o SGBD utilizado e o potencial de funcionalidades. Por exemplo: *IBExpert*[1] para *Firebird*[2] ou *SQL Profiler*[3] para *SQL Server*[4].
- **Examinar a estrutura do banco de dados.** Examinar tabelas, relacionamentos, chaves primárias e estrangeiras para entender a estrutura do banco.

[1] <https://www.ibexpert.net/downloadcenter/>
[2] <https://firebirdsql.org/en/server-packages/>
[3] <https://learn.microsoft.com/pt-br/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-ver16>
[4] <https://www.microsoft.com/pt-br/sql-server/sql-server-downloads>

- **Mapear índices, procedimentos, *triggers* e *constraints*.** Identificar regras de negócios implementadas diretamente no banco de dados para entender como as consultas são otimizadas.

- **Verificar *DataModules*.** *DataModules* são unidades especializadas do Delphi, que normalmente centralizam os componentes de conexão e manipulação de dados.

- **Verificar componentes de conexões com o banco.** Normalmente, sistemas Delphi usam componentes como BDE (*Borland Database Engine*), IBX (*InterBase Express*), *dbExpress*, *ADOConnection*, *ZeosLib* ou *FireDAC*. Em geral, esses componentes são fortemente acoplados ao código do sistema e legado e difíceis de serem reutilizados durante um processo de transição para a arquitetura de microsserviços.

- **Examinar eventos de manipulação de dados.** Muitos componentes em Delphi possuem eventos de manipulação de dados como `BeforePost`, `AfterPost`, `BeforeDelete`, entre outros. Esses eventos são de extrema importância para entender o comportamento da aplicação.

Com base nessa análise, o time de desenvolvimento pode definir uma estratégia para reestruturar o banco de dados, visando suportar a arquitetura de microsserviços de forma eficiente e escalável. Nesse sentido, sugere-se reorganizar o banco de dados (ou seja, divisão de grandes tabelas) em microsserviços separados, sendo cada um com o seu banco de dados. Além disso, o time de desenvolvimento deve considerar fortemente a possibilidade de migrar o banco de dados legado para outro mais adequado para a arquitetura de microsserviços como, por exemplo, banco de dados NoSQL, e uso de APIs para acesso aos dados (LI; MA; LU, 2020; BANDARA; PERERA, 2020).

### B.4. Interface com o usuário

A condução de pesquisas UX/UI (do inglês, *User eXperience and User Interface*) tem papel fundamental no processo de modernização de sistemas legados para uma arquitetura de microsserviços. As pesquisas sugeridas têm como objetivo entender as necessidades e expectativas dos usuários finais em relação à interface do sistema modernizado. Para isso, este passo a coleta de *feedbacks* sobre a usabilidade, a experiência do usuário, o *design* visual e outros aspectos relacionados à interação com o software. Além disso, entrevistas com usuários, testes de usabilidade, análise de métricas de uso, *benchmarking* com sistemas similares e outras técnicas de pesquisa qualitativa e quantitativa podem ser utilizadas. Com base nos *insights* obtidos, o time de desenvolvimento pode projetar e implementar uma interface de usuário moderna, intuitiva e eficaz para os microsserviços, garantindo uma experiência positiva para os usuários finais (AUER *et al.*, 2021; BAMBERGER; KöRBER, 2022).

De acordo com Ayas, Leitner e Hebig (2021), a interface do sistema desempenha um papel estratégico em sistemas baseados em microsserviços, pois se comunica diretamente com os diversos serviços independentes que compõem a arquitetura. Para viabilizar essa comunicação, este passo requer um *design* bem planejado, capaz de garantir a fluidez na navegação e a coesão entre os diferentes módulos do sistema. Portanto, investir em pesquisas UX/UI não é apenas uma questão de estética, mas sim uma estratégia essencial para garantir que a transição para a arquitetura de microsserviços resulte em um sistema mais eficiente, intuitivo e alinhado às expectativas dos usuários.

### B.5. Stack tecnológica

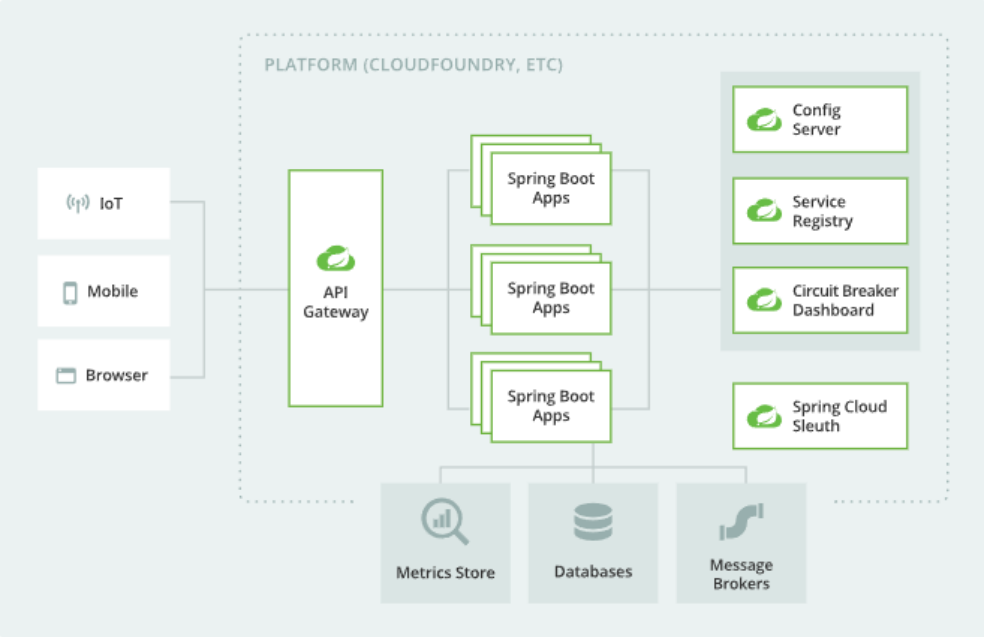
Este passo é essencial para o sucesso da modernização, pois envolve a definição da arquitetura de software e a seleção das tecnologias mais adequadas para a implementação dos microsserviços, que inclui uma escolha criteriosa das linguagens de programação, *frameworks*, bibliotecas, plataformas de hospedagem e serviços de infraestrutura necessários para suportar a nova arquitetura. A modelagem da estrutura tecnológica, conforme ilustrada na Figura 15, deve ser realizada com base em uma visão estratégica, garantindo que a nova arquitetura atenda às demandas da empresa e dos usuários finais. Como pode ser observado, nessa figura são ilustrados os elementos essenciais para funcionamento de sistemas baseados em microsserviços (elementos `Spring Boot Apps`), tais como: registro de serviços (`Service Registry`), configuração de serviços (`Config Server`), rastreamento de interações com o sistema (`Spring Cloud Sleuth`), entre outros. Como resultado, espera-se que a arquitetura seja capaz de lidar com variações de demanda, garantia de alta disponibilidade para os usuários e proteção de dados sensíveis contra ameaças cibernéticas. De acordo com Mazzara *et al.* (2021) e Kazanavičius e Mažeika (2023), as habilidades do time de desenvolvimento devem ser consideradas ao escolher as tecnologias, garantindo que os profissionais possam trabalhar com eficiência e que a curva de aprendizado não se torne um obstáculo para o projeto.

Diante do exposto neste passo, pode-se concluir que um planejamento detalhado e bem estruturado para as tecnologias que serão utilizadas em um sistema modernizado é indispensável para garantir uma transição "suave" para a arquitetura de microsserviços. Em outras palavras, a escolha de tecnologias alinhadas às necessidades do sistema e da equipe contribui significativamente para o sucesso da modernização, reduzindo riscos, aumentando a eficiência do desenvolvimento e garantindo um sistema robusto, flexível e preparado para o futuro.

## C. Decomposição

Decomposição é uma atividade que foca na arquitetura de software, com atenção especial na extração de candidatos a microsserviços do sistema legado. Segundo (LEWIS; FOWLER,

Figura 15 – Modelagem da estrutura tecnológica de microsserviços




Fonte: https://spring.io/microservices.

2019), o *design* de microsserviços deve considerar os aspectos organizacionais e de negócios que são oferecidos aos usuários. O desafio dessa atividade é determinar o tamanho ideal para os microsserviços, que devem ser divididos em unidades menores com baixo acoplamento e alta coesão. Uma vez projetado, cada microsserviço deve ser responsável por um contexto delimitado, fornecendo um conjunto coeso de funcionalidades com um escopo bem definido para atender a uma capacidade de negócios específica (KRAUSE *et al.*, 2020). Embora existam técnicas (semi)automatizadas que facilitam a decomposição de sistemas em microsserviços, é recomendado que, inicialmente, funcionalidades com baixo impacto ou risco sejam priorizadas durante um processo de modernização (KAZANAVIčIUS; MAŽEIKA, 2023). Para fazer isso, a equipe de desenvolvimento deve estabelecer uma estratégia para selecionar microsserviços candidatos que adicionarão maior valor e terão menos dependências externas. Em seguida, as etapas associadas à atividade de planejamento são descritas.

### C.1. Candidatos a microsserviços

Durante este passo, uma análise rigorosa das funcionalidades do sistema legado é essencial para determinar a viabilidade de potenciais candidatos a microsserviços (SELLAMI *et al.*, 2022). Dentre as abordagens existentes na literatura que podem dar suporte à decomposição, as mais frequentemente utilizadas são: (i) **decomposição por contexto delimitado**, que visa identificar as áreas do sistema legado que têm responsabilidades coesas e bem definidas, representando um contexto de negócios delimitado; (ii) *granularidade adequada*, que visa avaliar o tamanho e a complexidade das funcionalidades dos sistemas legados. Microsserviços muito

pequenos podem resultar em comunicação excessiva, enquanto aqueles muito grandes podem comprometer a escalabilidade e a manutenibilidade do sistema modernizado; e (iii) **acoplamento frouxo**, que visa identificar funcionalidades com poucas dependências externas que podem ser isoladas e mantidas de forma independente. Essa abordagem ajuda a minimizar o acoplamento entre microsserviços e facilita a evolução e a manutenção do sistema modernizado.

A identificação de candidatos a microsserviços em um sistema Delphi requer um equilíbrio entre análise técnica e visão estratégica do negócio. O foco deve estar em desacoplar funcionalidades de forma gradual, garantindo que cada novo serviço seja independente e escalável. Ferramentas como *ModelMaker*, *Pascal Analyzer* e *DUnit* podem ajudar a mapear dependências e modularidade. Os seguintes artefatos podem ser gerados com a execução deste passo:

| **Artefatos resultantes do Passo C.1** |
|---|
| **AT8.** Lista de funcionalidades candidatas a microsserviços. |
| **AT9.** Plano de migração e priorização de funcionalidades. |

## C.2. Classificação de microsserviços

Neste passo, os microsserviços candidatos são classificados em ordem de importância, visando promover a integridade contínua e valiosa do software (SELLAMI *et al.*, 2022). Nesse sentido, a investigação conduzida (veja Capítulo 3) sugere as seguintes diretrizes para classificação dos microsserviços: (i) os casos de uso mais críticos ou essenciais para o negócio; (ii) os microsserviços que oferecem o maior valor comercial ou que atendem às necessidades mais urgentes do cliente; (iii) os microsserviços que são pré-requisitos para outros microsserviços ou que têm poucas dependências externas; e (iv) os microsserviços que são mais simples de implementar ou que têm menor risco técnico. Vale ressaltar também que estratégias de pontuação e classificação, como análise de valor versus esforço, têm se mostrado uma alternativa viável em outros domínios de software e podem ser facilmente adaptadas ao contexto deste artigo.

Com base no conteúdo apresentado neste passo, sugere-se a elaboração de uma lista para ranqueamento dos microsserviços a serem implementados. Basicamente, essa lista deve conter um identificador para cada microsserviço, sua descrição de funcionalidade, suas relações de dependências (se houver), e suas relações com outros microsserviços (ou seja, quais microsserviços dependem do microsserviço que está sendo identificado).

## C.3. Design de decomposição

Neste passo, os desenvolvedores devem se concentrar em estabelecer a interface de comunicação para cada microsserviço, respeitando as capacidades de negócio do sistema. Nesse

sentido, sugere-se que os desenvolvedores também dediquem atenção especial ao gerenciamento de dependências entre microsserviços, visando minimizar o acoplamento e garantir a independência. Para isso, é recomendado que padrões arquitetônicos e boas práticas de engenharia de software sejam usados para projetar e implementar microsserviços, como contêineres, orquestração de contêineres, API *Gateway* e *Circuit Breaker* (RICHARDSON, 2018). Além disso, estratégias robustas de teste e validação também se mostraram uma alternativa viável para a decomposição e *design* de novos microsserviços, garantindo assim a qualidade e integridade desses serviços (TRABELSI *et al.*, 2022). Com resultado, os seguintes artefatos podem ser gerados neste passo:

| **Artefatos resultantes do Passo C.3** |
|---|
| **AT10.** Diagrama de arquitetura de microsserviços. |
| **AT11.** Plano de comunicação entre microsserviços. |
| **AT12.** Documentação das interfaces de APIs. |

### C.4. Reestruturação de dados

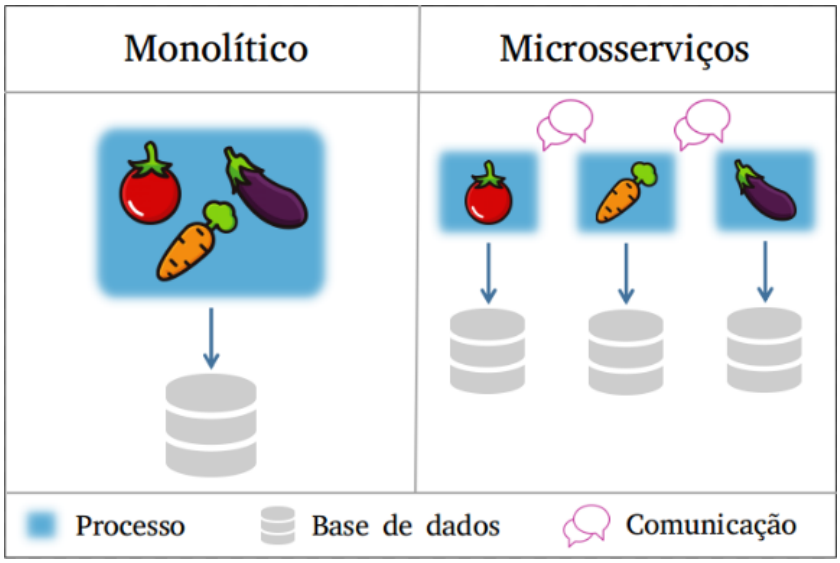
Similarmente ao passo anterior, os desenvolvedores devem reestruturar (ou seja, decompor) os bancos de dados do sistema legado para torná-los compatíveis com os microsserviços que serão desenvolvidos. Para fazer isso, os desenvolvedores devem se concentrar seus esforços em identificar os diferentes tipos de dados, relacionamentos entre entidades e acoplamento de entidades de banco de dados. Isso facilitará a decomposição do modelo de dados do sistema legado em esquemas menores e mais especializados, que podem então ser adaptados para atender aos requisitos específicos de cada microsserviço. Além disso, é recomendado que esta atividade seja realizada em paralelo com a anterior (PARIKH *et al.*, 2022).

Diante do exposto, para conduzir uma reestruturação dos dados em sistemas legados Delphi, algumas estratégias são sugeridas, a saber:

- **Identificação e decomposição de domínios de dados.** Cada microsserviço deve ter seu próprio domínio de dados delimitado, sendo assim, os modelos de dados da base legada devem ser decompostos em esquemas menores e mais especializados que atendam aos requisitos específicos de cada microsserviço, conforme ilustrado na Figura 16;
- **Gestão de consistência e integridade.** A consistência e a integridade dos dados, mesmo em um ambiente distribuído de microsserviços devem ser garantidas. Isso pode exigir o uso de padrões de transação distribuída, técnicas de replicação de dados e mecanismos de sincronização.

- **Segurança.** Medidas de segurança adequadas para proteger os dados em trânsito e em repouso devem ser implementadas. Isso pode incluir criptografia, controle de acesso baseado em função e auditoria de dados.
- **Testes e validação de dados.** Testes rigorosos para garantir a qualidade e a integridade dos dados após a migração devem ser realizados. Isso inclui testes de integração, testes de regressão e testes de desempenho.

Figura 16 – Decomposição de domínios de dados




Fonte: <https://pangarecode.wordpress.com/2018/04/17/monolitico-ou-microsservicos>

## C.5. Modelagem de API

Neste passo, os desenvolvedores devem concentrar seus esforços na modelagem das APIs, que representam os contratos de microsserviços (KYRYK *et al.*, 2022). Esta etapa pode ser resumida da seguinte maneira (ERL *et al.*, 2012): (i) identificação dos recursos ou funcionalidades (ou seja, entidades comerciais) que serão expostos pelos microsserviços; (ii) definição de *endpoints RESTful*, garantindo que cada recurso seja representado por um URI (do inglês, *Uniform Resource Identifier*) exclusivo e acessível por meio de métodos HTTP padrão, como `GET`, `POST`, `PUT` e `DELETE`; (iii) seleção de formatos de dados apropriados para representar os recursos da API, como exemplo, formato JSON (do inglês, *JavaScript Object Notation*); (iv) documentação dos contratos da API de forma clara e abrangente, especificando os parâmetros de entrada, respostas esperadas e possíveis códigos de status HTTP; (v) versionamento da API para garantir compatibilidade com versões futuras; (vi) *design* orientado à segurança para que mecanismos de autenticação e autorização sejam implementados em tais APIs para que os recursos e dados do sistema sejam protegidos; (vii) documentação da API para facilitar a descoberta e compreensão

de sua funcionalidade; e (viii) testes (por exemplo, unidade, integração e aceitação) para garantir o funcionamento correto das APIs e sua conformidade com os contratos especificados.

A modelagem de uma API em Delphi pode ser feita de forma organizada seguindo boas práticas e usando *frameworks* como *DMVCFramework*[5] ou *Horse*[6], garantindo eficiência, modularidade e escalabilidade. A implementação de segurança com JWT (do inglês, *JSON Web Token*), integração com bancos de dados e o uso de ferramentas como *Swagger*[7] para documentação tornam a API mais confiável e acessível. Para um ambiente produtivo, considerar *deploys* com *Docker*[8], balanceamento de carga e monitoramento são diferenciais que garantem alta disponibilidade e performance. Seguindo esses princípios, é possível desenvolver APIs eficientes e bem estruturadas, facilitando a integração com sistemas externos e melhorando a experiência dos usuários finais.

### C.6. Atributos de qualidade e métricas

Neste passo, é fundamental determinar quais atributos de qualidade e métricas devem ser associados ao sistema modernizado (TRABELSI *et al.*, 2022). De acordo com as evidências coletadas na Seção 3.2.2 (veja Figura 10), atributos de qualidade e métricas são elementos indispensáveis de sistemas baseados em microsserviços, dada a natureza intrínseca de tais sistemas. Esta investigação também revelou que revisões de código, testes de software e análise estática de código podem auxiliar na identificação e correção de defeitos. Paralelamente, métricas podem ser usadas para verificar a eficácia dos testes, a taxa de defeitos identificados, o tempo médio para retificar defeitos, cobertura de código e assim por diante. Da perspectiva do produto, métricas podem ser adotadas para avaliar a confiabilidade, desempenho, usabilidade e tempo de resposta do sistema modernizado (KRAUSE *et al.*, 2020). Vale ressaltar que a investigação conduzida (veja Capítulo 3) mostra que não existe uma combinação ideal para esse tópico (ou seja, combinação "bala de prata"). Os desenvolvedores devem analisar os interesses e cenários de criticidade do sistema que está sendo modernizado para escolha dos atributos e métricas. Nesse sentido, em trabalhos anteriores de nosso grupo de pesquisa (PASSINI; AFFONSO, 2018; AFFONSO; PASSINI; NAKAGAWA, 2019; PASSINI *et al.*, 2022), um módulo para configuração de atributos de qualidade em sistemas orientados a serviços foi desenvolvido. Esse módulo permite que o desenvolvedor defina atributos e pesos, conforme critério de importância, para configuração de níveis de qualidade em uma aplicação orientada a serviços.

[5] <https://github.com/danieleteti/delphimvcframework>
[6] <https://github.com/HashLoad/horse>
[7] <https://swagger.io>
[8] <https://www.docker.com>

### C.7. Documentação

Neste passo, os desenvolvedores devem concentrar seus esforços na preparação da documentação para o sistema que será modernizado nesta etapa. Em resumo, os artefatos essenciais de documentação devem ser concisos e compreensíveis, refletindo com precisão os modelos lógicos, o *design* e a arquitetura do sistema. Para isso, é recomendado que esses artefatos atendam aos seguintes critérios: cobertura da documentação, público-alvo e formato de acesso. A documentação deve ser revisada e atualizada consistentemente ao longo desta etapa e processo de modernização, pois inconsistências podem comprometer o processo e exigir o uso exclusivo do código-fonte para todo o desenvolvimento (PARIKH *et al.*, 2022).

Uma documentação para microsserviços pode ser dividida em várias categorias, conforme a Tabela 5. Sendo assim, entende-se que uma boa documentação garante que os microsserviços sejam fáceis de entender, integrar e manter. Portanto, utilizar padrões como *OpenAPI*, *AsyncAPI* e *C4 Model*, além de ferramentas como *Swagger*, *Confluence* e *GitHub Actions*, pode tornar esse processo mais eficiente.

Tabela 5 – Documentação para Microsserviços

| **Tipo** | **Objetivo** |
|---|---|
| Visão Geral | Apresentar a arquitetura geral e os principais conceitos do sistema |
| Arquitetura | Detalhar como os microsserviços se comunicam, padrões adotados e infraestrutura |
| APIs | Documentar *endpoints*, métodos, parâmetros e respostas |
| Banco de Dados | Especificar estrutura e estratégias de compartilhamento de dados |
| Mensageria | Explicar como eventos são trocados entre os microsserviços |
| Segurança | Descrever autenticação, autorização e padrões de segurança |
| DevOps e *Deploy* | Explicar pipelines de CI/CD, monitoramento e estratégias de *rollback* |

Fonte: Elaborada pelo autor.

## D. Desenvolvimento

Conforme evidenciado pelas descobertas da investigação conduzida (veja Capítulo 3), essa atividade pode ser realizada em duas abordagens: (i) desenvolver novo código do zero e (ii) extrair microsserviços do código-fonte legado. Na primeira, o código legado é identificado como de "baixo valor", e os desenvolvedores devem priorizar a preservação das funcionalidades do sistema legado implementando um novo código-fonte. Na segunda, o código legado é considerado valioso e pode ser reaproveitado na íntegra (ou com ajustes menores) para extrair e transformar as funcionalidades do sistema legado em microsserviços equivalentes. Independentemente da abordagem, também é fundamental considerar a estratégia de migração gradual

para esta atividade, onde a versão anterior do sistema legado deve ser descontinuada após a implementação do microsserviço para evitar problemas de manutenção decorrentes da coexistência de funcionalidades em dois locais distintos (PRASANDY *et al.*, 2020; AYAS; LEITNER; HEBIG, 2021; BANDARA; PERERA, 2020).

### D.1. Repositório de dados

Neste passo, a organização deve preparar a infraestrutura de desenvolvimento alinhada aos princípios e práticas do DevOps, permitindo o desenvolvimento independente e paralelo de microsserviços pelas equipes de desenvolvimento. A infraestrutura de DevOps facilita a entrega contínua de software, garantindo assim a concretização dos benefícios associados ao desenvolvimento baseado em microsserviços tanto da perspectiva organizacional quanto do cliente (ou seja, aqueles interessados em modernização) (KYRYK *et al.*, 2022).

A modernização de sistemas legados Delphi para arquitetura de microsserviços exige uma reestruturação do repositório de dados, garantindo desacoplamento, escalabilidade e performance. No ambiente de desenvolvimento Delphi, que tradicionalmente usa bancos relacionais como *Firebird*, *MySQL*[9] e *PostgreSQL*[10], a migração para microsserviços exige mudanças na organização e acesso aos dados. A seguir, uma lista de elementos e suas respectivas tecnologias são apresentadas como referência para apoiar o desenvolvimento de microsserviços em Delphi

- **ORM para Delphi**: *TMS Aurelius*[11], *Delphi MVC Framework*;
- **Banco Relacional**: *Firebird*, *PostgreSQL*, *MySQL*;
- **Banco NoSQL**: *MongoDB*[12], *Redis*[13];
- **Mensageria**: *RabbitMQ*[14], *Apache Kafka*[15];
- **API REST**: *Horse*[16], *RAD Server*[17];
- **Sincronização de Dados**: *JSON-RPC*[18], *REST*, *WebSockets*[19];

[9] <https://www.mysql.com/>
[10] <https://www.postgresql.org/>
[11] <https://www.tmssoftware.com/site/aurelius.asp>
[12] <https://www.mongodb.com/>
[13] <https://github.com/redis/redis/releases>
[14] <https://www.rabbitmq.com/>
[15] <https://kafka.apache.org/>
[16] <https://github.com/HashLoad/horse>
[17] <https://github.com/Embarcadero/RADServer-Docs>
[18] <https://www.jsonrpc.org/>
[19] <https://websocket.org/>

- **Versionamento**: *Git*[20], *GitHub*[21], *GitLab*[22].

## D.2. Casos de teste

Neste passo, os cenários de execução devem ser delineados para avaliar se o sistema ou suas funcionalidades atendem aos requisitos estabelecidos. O planejamento de testes tem sido uma alternativa viável na modernização de sistemas legados, pois envolve a identificação das condições de entrada, dos passos a serem seguidos e dos resultados esperados para cada funcionalidade. Portanto, pode-se dizer que os casos de teste servem de base para verificar se o software atende aos requisitos e se comporta conforme o esperado (KAZANAVIčIUS; MAžEIKA, 2023; AYAS; LEITNER; HEBIG, 2021).

A definição de casos de teste em um ambiente de microsserviços desenvolvidos em Delphi exige uma abordagem estruturada para garantir que cada serviço funcione corretamente e interaja adequadamente com os demais. Na Figura 17 é ilustrado um exemplo de caso de teste para um microsserviço de cadastro de clientes. Para definir casos de teste eficazes em microsserviços Delphi, algumas práticas são sugeridas, a saber (PRESSMAN, 2016; SOMMERVILLE, 2018): (i) Testes unitários para classes e métodos; (ii) Testes de integração para comunicação entre serviços; (iii) Testes de contrato para garantir compatibilidade; (iv) Testes de carga para avaliar escalabilidade; (v) Testes de segurança para evitar falhas críticas.

Figura 17 – Caso de Teste para um Microsserviço de Cadastro de Clientes

| ID | TC001 |
|---|---|
| **Descrição** | Testar o cadastro de um novo cliente com dados válidos. |
| **Pré-condições** | O microsserviço está rodando e conectado ao banco de dados. |
| **Passos** | 1. Enviar uma requisição `POST` para `/api/clientes` com os dados do cliente.<br>2. Validar a resposta do serviço. |
| **Dados de Entrada** | `{ "nome": "João Silva", "email": "joao@email.com" }` |
| **Resultado Esperado** | Código **201 Created** e o cliente salvo no banco de dados. |

Fonte: Elaborada pelo autor.

## D.3. Desenvolvimento de microsserviços

Neste passo, os microsserviços são desenvolvidos seguindo as especificações tecnológicas e de *design*, bem como os requisitos técnicos e de negócios estabelecidos nas etapas

[20] <https://git-scm.com/>
[21] <https://github.com/>
[22] <https://about.gitlab.com/pt-br/>

anteriores (ou seja, Passos B.1, C.1 a C.7). Como resultado, o objetivo é integrar os microsserviços de uma forma que garanta sua funcionalidade coletiva para o sistema que está sendo modernizado. Durante a implementação, é recomendado que os desenvolvedores sigam as melhores práticas em engenharia de software, como escrever código limpo e modular, para garantir a escalabilidade e resiliência dos microsserviços. Além disso, também é sugerido que ferramentas sejam usadas para desenvolver, gerenciar versões e implantar os microsserviços. Depois de desenvolvidos, os microsserviços são submetidos a uma série de testes, procedimentos de depuração e implantação em um ambiente de produção para uso por usuários finais (OSMAN *et al.*, 2022; COLANZI *et al.*, 2021b; FREITAS; FERREIRA; CUNHA, 2023; PRASANDY *et al.*, 2020; PRETI *et al.*, 2021).

O ambiente de desenvolvimento Delphi, tradicionalmente utilizado para o desenvolvimento de sistemas *desktop* e monolíticos, tem sido evoluído desde sua concepção, oferecendo atualmente diversas ferramentas para a criação de microsserviços escaláveis e eficientes. As tecnologias mais utilizadas pelas empresas atualmente são: *Horse*, *Swagger*, *Docker*, *RabbitMQ* e bancos de dados modernos. Essas tecnologias tornam possível a implementação de soluções baseadas em arquitetura de microsserviços. *Horse* é um *framework* leve e eficiente para criar APIs REST em Delphi, fornecendo um conjunto de *middlewares* para segurança, autenticação, *logs* e manipulação de requisições. Já o *Swagger* é uma ferramenta essencial para documentar microsserviços, que permite gerar uma interface interativa para testar *endpoints* e visualizar os contratos das APIs. O *Docker* fornece um conjunto de ferramentas permite o desenvolvedor empacotar e distribuir os microsserviços de maneira isolada e escalável. Além disso, o *FireDAC* é uma das ferramentas utilizadas no Delphi para conexão com banco de dados *PostgreSQL*, *MySQL*, *Firebird* ou *MongoDB*. No que diz respeito a entre microsserviços, o *RabbitMQ* é um software de mensageria que facilita a comunicação entre microsserviços de maneira assíncrona por meio de protocolos bem consolidados e estabelecidos. Portanto, como pode ser observado, o ambiente Delphi continua sendo uma excelente opção para construção de aplicações desenvolvimento de microsserviços quando combinado com tecnologias modernas.

### D.4. Migração de banco de dados

Neste passo, os bancos de dados de microsserviços devem ser criados com base no entendimento reunido nos Passos A.1, B.1, B.4 e C.1. A migração do banco de dados é uma tarefa complexa, abrangendo a transferência tanto do esquema de dados quanto dos dados em si. De acordo com as evidências reunidas no Capítulo 3, esta etapa deve ser realizada em paralelo com o desenvolvimento dos microsserviços, pois há uma relação intrínseca entre a funcionalidade do microsserviço e seus dados. Além disso, outro aspecto relevante a ser considerado para uma migração bem-sucedida do banco de dados é a migração de dados entre o antigo e o novo esquema de dados para garantir a integridade das informações entre os sistemas

legados e modernizados (MISHRA *et al.*, 2022; ZARAGOZA *et al.*, 2022; KAZANAVIčIUS; MAžEIKA, 2023).

Um dos maiores desafios na modernização de sistemas legados em Delphi para microsserviços é a migração do banco de dados, especialmente quando existe um forte acoplamento entre os componentes de banco de dados e o sistema legado. A transição de base de dados não precisa ser imediata, podendo a substituição ocorrer de forma gradual, começando pelos módulos menos críticos do sistema. Assim, à medida que as APIs forem sendo implementadas, os acessos diretos ao banco de dados podem ser substituídos por chamadas REST, adaptando os formulários Delphi para consumir os dados por meio das novas interfaces. Durante esse processo, os dados podem ser migrados de maneira controlada, utilizando ferramentas de extração e carregamento, replicações parciais ou mesmo mecanismos de sincronização baseados em eventos, quando necessário manter as bases em paralelo por algum tempo. Com todas as interações da aplicação Delphi passando a depender exclusivamente das APIs dos microsserviços, o sistema atinge um novo nível de desacoplamento. O antigo banco de dados legado deixa de ser um ponto de dependência central e pode ser mantido apenas para fins históricos ou de auditoria. Conforme reportado nesta seção, a migração de banco de dados é um processo exige planejamento e testes rigorosos em cada etapa, ao mesmo tempo que é possível preservar a robustez do cliente Delphi enquanto se adota uma arquitetura moderna e distribuída, mais alinhada com os padrões atuais de escalabilidade, manutenção e evolução contínua de software. Os seguintes artefatos podem ser gerados com a execução deste passo:

| **Artefatos resultantes do Passo D.4** |
|---|
| **AT13.** *Script* de migração. |
| **AT14.** Relatório de migração. |
| **AT15.** Estratégia de *rollback* (reversão). |

### D.5. Controle de versão

O controle de versão é uma prática essencial no desenvolvimento baseado em microsserviços, pois permite o rastreamento e o gerenciamento de alterações no código-fonte e na configuração do sistema ao longo do tempo. Além disso, o controle de versão permite o *rollback* para versões anteriores em caso de complicações e facilita a implantação de novas versões de software de forma regulada e automatizada. Nessa direção, também é recomendado que o versionamento para o contrato de microsserviço seja definido, bem como o controle de versão de seu código-fonte. Para isso, as equipes de desenvolvimento devem criar repositórios individuais para cada microsserviço neste passo (FREIRE *et al.*, 2021).

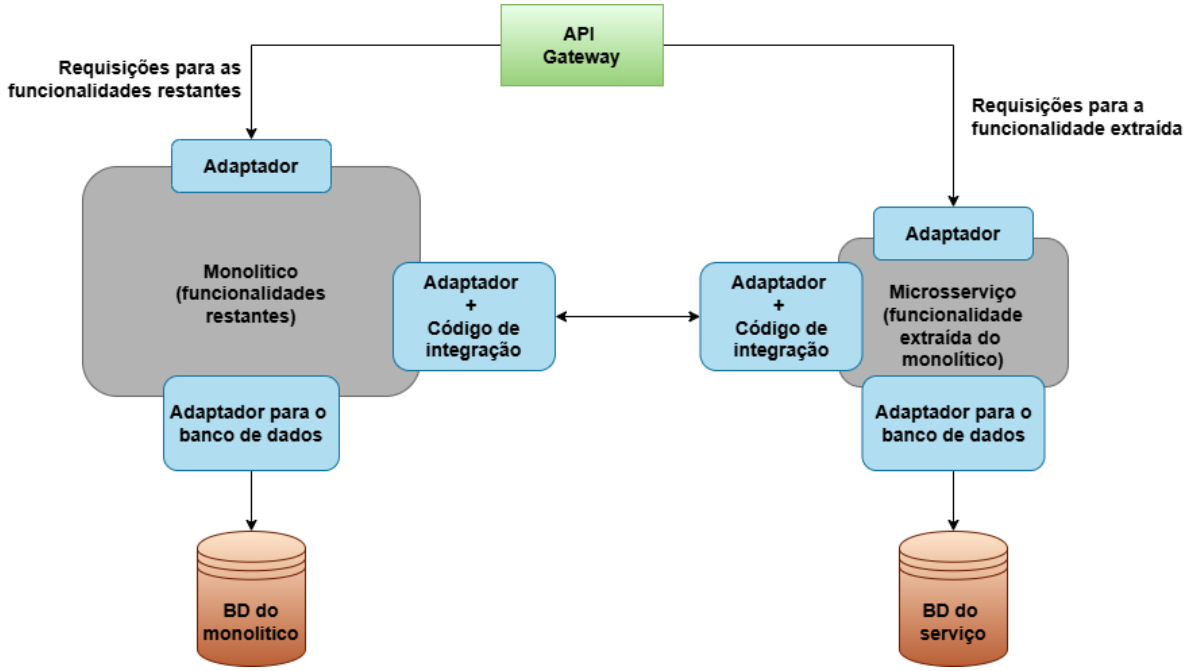
No contexto de aplicações Delphi, a adoção de controle de versão torna-se ainda mais crítica à medida que se evolui de uma arquitetura legada (ou monolítica) para uma baseada em microsserviços. Projetos desenvolvidos em Delphi frequentemente envolvem múltiplos módulos, componentes visuais e arquivos binários, o que exige organização rigorosa e estratégias adequadas para versionamento. A separação dos repositórios por microsserviço, por exemplo, ajuda a manter o escopo de cada módulo bem definido, garantindo que as mudanças em um serviço não afetem diretamente os demais. Além disso, o uso de ferramentas de versionamento como *Git*, aliado a práticas como o uso de arquivos `.dfm` em formato texto e o controle explícito de dependências e bibliotecas compartilhadas, permite que os desenvolvedores Delphi trabalhem com mais segurança, mesmo em equipes distribuídas. Portanto, o controle de versão em projetos Delphi vai além da simples rastreabilidade de código, esta prática deve propiciar sustentabilidade a base da transição arquitetural, dar suporte à automação de *builds* e testes, e promover uma cultura de entrega incremental e confiável, características indispensáveis em ambientes orientados a microsserviços.

### D.6. Monolítico para microsserviços

Este passo faz a transição do sistema de sua versão legada para uma modernizada. De acordo com (MA *et al.*, 2022; LI; MA; LU, 2020; BANDARA; PERERA, 2020), essa transição deve ocorrer gradualmente conforme os microsserviços se desenvolvem, permitindo que ambos os sistemas coexistam. Em projetos Delphi, essa coexistência pode ser viabilizada por meio da criação de camadas de integração que expõem partes do sistema monolítico como serviços, utilizando, por exemplo, *DataSnap*, *RAD Server* ou outras soluções RESTful compatíveis com Delphi. Essa organização em camadas permite que os microsserviços consumam funcionalidades do sistema legado enquanto novos módulos são implementados de forma independente. Durante esse período de transição, é essencial garantir a interoperabilidade entre o legado e os novos serviços, cuidando para que as estruturas de dados, regras de negócio e contratos de comunicação sejam bem definidos e compatíveis. Além disso, também é comum que um microsserviço inicialmente dependa do banco de dados legado, até que a separação completa dos dados possa ser feita com segurança. Para isso, os módulos antigos vão sendo desacoplados com o passar do tempo e substituídos por novos serviços, em uma abordagem evolutiva que reduz os riscos de reescrita total e interrupções no sistema. Nesse sentido, vale destacar que esse processo demanda atenção especial à orquestração e roteamento de requisições. Como ilustrado na Figura 18, *gateway* de API (ou um barramento intermediário) tem se mostrado uma boa estratégia para gerenciar o direcionamento entre o sistema legado e os novos serviços, assegurando que a experiência do usuário final não seja impactada negativamente durante a migração. Assim, pode-se assegurar que a modernização ocorre de forma segura, controlada e alinhada com a realidade tecnológica do ecossistema Delphi, pois as interfaces podem ser preservadas e o

sistema modernizado sendo a elas acopladas gradativamente.

Figura 18 – Monolítico para microsserviços




Fonte: Elaborada pelo autor.

## E. Integração

Esta atividade visa combinar e sincronizar os microsserviços desenvolvidos para compor o sistema modernizado. Durante essa atividade, os microsserviços são vinculados por meio de APIs ou outros canais de comunicação, facilitando a transferência de dados e informações essenciais para a funcionalidade geral do sistema. Além disso, essa atividade também pode envolver a integração do sistema modernizado com outros sistemas ou componentes externos para atender aos novos requisitos do sistema.

Do ponto de vista do desenvolvimento, a integração pode envolver a configuração de *pipelines* de implantação contínua e automação de processos, para facilitar a distribuição de microsserviços em diferentes ambientes, como desenvolvimento, teste e produção. Portanto, espera-se, após a conclusão desta atividade, que o sistema integrado esteja preparado para implantação e disponibilizado para utilização pelos usuários finais (PARIKH *et al.*, 2022).

### E.1. Malha de serviço

Neste passo, os microsserviços são empacotados em imagens e implantados em contêineres para inserção no ambiente operacional. Este passo também inclui a definição de mensagens, roteamento, balanceamento de carga, serviços de descoberta, entre outros elementos para viabilização da infraestrutura (ou ambiente operacional) necessária para funcionamento desse tipo

de aplicação. A malha de serviço se refere à comunicação e interação entre os vários microsserviços que compõem um sistema, bem como os mecanismos de monitoramento, diagnóstico e manutenção usados para garantir sua saúde e desempenho (KYRYK *et al.*, 2022; PARIKH *et al.*, 2022; AYAS; LEITNER; HEBIG, 2021).

No contexto de sistemas legados em Delphi, a implementação de uma malha de serviços pode representar um desafio adicional, uma vez que muitos desses sistemas não foram originalmente projetados para comunicação distribuída. Para contornar essa limitação, pode-se encapsular componentes Delphi em microsserviços RESTful utilizando bibliotecas como *RAD Server*, *DataSnap* ou até soluções personalizadas com *Indy*[23] ou *WebBroker*[24]. Esses microsserviços podem então ser executados em contêineres *Docker* e integrados a plataformas como *Kubernetes*[25], que oferecem suporte nativo à gestão de malhas de serviço. Dessa forma, a definição e implantação da malha de serviços é um passo estratégico na modernização de sistemas Delphi, pois viabiliza a comunicação escalável, resiliente e segura entre serviços distribuídos, permitindo o crescimento da arquitetura sem comprometer a confiabilidade.

### E.2. Integração Contínua e Entrega Contínua

De acordo com Pressman (2016), Sommerville (2018), os processos de software que permite entrega valiosa de software em funcionamento são baseados em três atividades: Integração Contínua (do inglês, *Continuos Integration* – CI), Entrega Contínua (do inglês, *Continuos Delivery* – CD) e implantação contínua. A CI exige o envio de alterações de código para serem mescladas na ramificação primária. Os processos automatizados de construção e teste garantem que o código na ramificação principal mantenha a qualidade de produção. Assim, as equipes podem identificar problemas relacionados à compatibilidade ou conflitos de código, mitigando a probabilidade de introduzir erros e problemas de qualidade no sistema. Na CD, um ambiente semelhante ao de produção recebe automaticamente as alterações de código que passam pelo processo de CI. Para garantir a entrega contínua em um ambiente de microsserviços, é importante ter um conjunto robusto de ferramentas de automação, testes automatizados abrangentes e uma cultura de colaboração e comunicação eficaz entre as equipes de desenvolvimento. A implantação contínua exige que as modificações de código tenham passado pelas duas etapas anteriores (ou seja, CI/CD) e estejam preparadas para implantação automatizada no ambiente de produção (ZARAGOZA *et al.*, 2022; KYRYK *et al.*, 2022; KAZANAVIčIUS; MAžEIKA, 2023; AYAS; LEITNER; HEBIG, 2021).

Um processo de CI/CD que deve ser considerado robusto deve incluir os seguintes recursos: (i) a criação e implantação de microsserviços de maneira independente; (ii) a implantação

[23] <https://docwiki.embarcadero.com/RADStudio/Athens/en/Indy>
[24] <https://docwiki.embarcadero.com/RADStudio/Sydney/en/About_Web_Broker>
[25] <https://kubernetes.io/releases/download/>

dos microsserviços em ambientes designados para desenvolvimento, teste e garantia de qualidade; (iii) a implantação simultânea de versões existentes e novas; e (iv) o empacotamento de imagens em contêineres dentro do ambiente de produção (DEHGHANI *et al.*, 2022; PARIKH *et al.*, 2022).

### E.3. Teste

Este passo representa a automação de testes unitários e de integração para microsserviços. Os testes unitários visam verificar o comportamento individual de pequenas partes do código, geralmente funções ou métodos, isolando-os de outras partes do sistema. Esses testes são automatizados para serem executados de forma rápida e repetida sempre que houver alterações no código, garantindo que novos recursos não quebrem os existentes e facilitando a detecção precoce de *bugs*. Os testes de integração visam verificar a interação entre diferentes componentes do sistema, como módulos, microsserviços ou sistemas externos. Esses testes automatizam a verificação da funcionalidade e compatibilidade do sistema como um todo, garantindo que os componentes individuais operem corretamente juntos. Ao automatizar os testes unitários e de integração, as equipes de desenvolvimento podem identificar e corrigir problemas de maneira proativa, reduzindo o tempo necessário para validar alterações no código e aumentando a confiabilidade e a estabilidade do software. Para automatizar os testes, é recomendável o uso de ferramentas de teste específicas, *frameworks* de desenvolvimento e práticas de integração contínua, que facilitam a execução automática de testes em ambientes controlados e a geração de relatórios detalhados sobre os resultados dos testes (KAZANAVIčIUS; MAžEIKA, 2023; AYAS; LEITNER; HEBIG, 2021).

A automação completa desses testes pode ser integrada a *pipelines* de CI/CD utilizando ferramentas como *Jenkins*[26], *GitLab CI*[27]*GitLab CI*, *Azure DevOps*[28], entre outras, com suporte para execução de *scripts* Delphi, coleta de resultados de testes e geração de relatórios. Além disso, também é recomendável configurar o versionamento dos testes juntamente com o código-fonte dos microsserviços, garantindo rastreabilidade das alterações e estabilidade dos testes ao longo do tempo. Automatizar testes em projetos Delphi modernos significa alinhar boas práticas tradicionais à infraestrutura atual de DevOps, adaptando-se às especificidades da linguagem, mas sem abrir mão da confiabilidade e da repetibilidade que a automação proporciona em ambientes de microsserviços. Como resultado, os seguintes artefatos podem ser gerados com a execução deste passo:

[26] <https://github.com/jenkinsci>
[27] <https://docs.gitlab.com/ci/>
[28] <https://azure.microsoft.com/pt-br/products/devops>

| **Artefatos resultantes do Passo E.3** |
| --- |
| **AT16.** Casos de testes automatizados. |
| **AT17.** Relatórios de execução e métricas. |

## F. Monitoramento

É uma atividade essencial em sistemas baseados em microsserviços devido à natureza distribuída e complexa desses sistemas. Portanto, o monitoramento em ambientes de microsserviços envolve a coleta e a análise de métricas relacionadas ao desempenho, como tempo de resposta, taxa de erros, uso de recursos (por exemplo, CPU[29], memória, rede) e disponibilidade. Essas métricas são coletadas em tempo de execução e podem ser visualizadas por meio de painéis e relatórios, fornecendo *insights* valiosos sobre a integridade do sistema e ajudando a identificar gargalos, falhas e oportunidades de otimização. Além disso, as atividades de monitoramento podem incluir a detecção e o alerta precoce de anomalias e problemas potenciais, permitindo que as equipes de operações e desenvolvimento tomem medidas corretivas rapidamente e minimizem o impacto sobre os usuários finais. A implementação de um sistema de monitoramento em ambientes de microsserviços exige a utilização de ferramentas e plataformas especializadas para a coleta, armazenamento e análise de dados de monitoramento, bem como práticas de observabilidade que visam aprimorar a compreensão e o diagnóstico do sistema em caso de problemas (FREIRE *et al.*, 2021; MA *et al.*, 2022).

### F.1. Dashboard

Este passo envolve a implementação de *dashboards* para fornecer *feedback* centralizado e contínuo sobre as métricas e a saúde dos microsserviços (FREIRE *et al.*, 2021). Para isso, um conjunto de ferramentas pode ser utilizado como *dashboards*, visando facilitar a observabilidade do sistema modernizado por meio da coleta de um conjunto de métricas, informações de rastreamento e *logs* (TOZZI, 2022). Métricas podem ser definidas como um medidor lógico usado para medir e registrar dados durante um período específico. Informações de rastreamento referem-se aos dados associados ao ciclo de vida de um único objeto transacional em um sistema, que podem ser específicos da transação ou relacionados a outros aspectos da operação do sistema. *Logs* lidam com eventos discretos que ocorrem durante a execução de um sistema, como mensagens de erro, eventos de auditoria ou metadados específicos de solicitações. Dentre as ferramentas disponíveis na literatura/mercado, o Grafana[30] tem se destacado como uma ferramenta importante para facilitar a observabilidade das aplicações baseadas em serviços/APIs pelos recursos oferecidos e pela disponibilidade de código aberto para customização.

[29] *Central Processing Unit.*
[30] <https://grafana.com>

A construção de *dashboards* para monitoramento e gerenciamento de microsserviços em aplicações Delphi tem se tornado cada vez mais viável e eficiente com o uso de *frameworks* modernos, bibliotecas de componentes e integrações com ferramentas de observabilidade. A seguir, são apresentadas algumas abordagens e recursos disponíveis para desenvolvedores Delphi interessados em criar soluções visuais robustas:

- **Delphi MVC Framework (DMVCFramework).** Este *framework* é uma das soluções mais utilizadas para construção de APIs RESTful no ecossistema Delphi. Essa solução permite que o desenvolvedor possa expor dados de microsserviços de forma estruturada, permitindo a integração com ferramentas externas de visualização, como o *Serenytics*, para criação de *dashboards* interativos e em tempo real. Essa abordagem é ideal para aplicações que desejam manter a lógica de *backend* em Delphi, aproveitando ao máximo seus recursos e performance.

- **Componentes TMS FNC Dashboard.** A TMS Software oferece o conjunto de componentes TMS FNC, que inclui elementos gráficos avançados como *gauges*, *charts*, *grids*, *cards* e *widgets*. Esses componentes são altamente customizáveis e multiplataforma, podendo ser usados tanto em aplicações VCL quanto *FireMonkey*. Com o *TMS FNC*, desenvolvedores Delphi podem criar *dashboards* sofisticados e responsivos diretamente na aplicação, integrando dados de diferentes microsserviços.

- **Dashboards Modernos com FireMonkey (FMX).** O *framework FireMonkey* permite a construção de interfaces modernas, com suporte a vetores, animações, e estilos personalizados, sendo ideal para aplicações que exigem um visual mais arrojado ou precisam ser executadas em múltiplas plataformas (Windows, macOS, Android e iOS). Com FMX é possível desenvolver *dashboards* dinâmicos, incorporando gráficos, indicadores e painéis informativos alimentados por dados dos microsserviços da aplicação.

- **Integração com Ferramentas de Observabilidade.** Para sistemas mais complexos ou ambientes em produção, é comum a necessidade de integração com ferramentas especializadas em monitoramento, como *Grafana* e *Prometheus*. Microsserviços desenvolvidos em Delphi podem expor métricas via *endpoints* HTTP, que são consumidos por essas plataformas para geração de *dashboards* analíticos, alertas e relatórios de desempenho. Essa abordagem combina a robustez do Delphi com ecossistemas modernos de observabilidade.

### F.2. Alertas de problemas

Este passo aborda o processamento de mensagens de aviso (ou alertas) quando o sistema transita para um estado crítico ou de atenção. Os alertas podem auxiliar na detecção precoce

e mitigação de problemas para evitar a interrupção dos microsserviços. O alerta representa uma prática fundamental para o monitoramento de microsserviços, pois permite a notificação imediata às equipes de operações e desenvolvimento sobre eventos ou problemas significativos que podem impactar a disponibilidade ou o desempenho do sistema (FREIRE *et al.*, 2021; LI; MA; LU, 2020).

Em arquiteturas baseadas em microsserviços, a capacidade de detectar e responder rapidamente a falhas ou comportamentos anômalos é fundamental para garantir a estabilidade e a continuidade dos sistemas. Embora os sistemas desenvolvidos em Delphi sejam associados como aplicações *desktop*, esse ambiente de desenvolvimento também oferece mecanismos eficazes para implementar estratégias de monitoramento e geração de alertas em tempo real. A seguir, são apresentadas algumas abordagens e ferramentas que podem ser utilizadas para implementar alertas eficientes em soluções baseadas em Delphi:

- **Monitoramento de Métricas via REST e DMVCFramework.** Com o uso do Delphi MVC *Framework*, os microsserviços podem disponibilizar *endpoints* específicos para métricas técnicas e de negócio (por exemplo, tempo de resposta, uso de memória, status de serviços externos). Esses dados podem ser coletados periodicamente por ferramentas de monitoramento como *Zabbix*, *Prometheus* ou *scripts* personalizados, permitindo o disparo de alertas em caso de anomalias, como lentidão ou indisponibilidade de serviços.

- **Notificações Baseadas em Eventos com Indy e WebSockets.** O Delphi, por meio da biblioteca *Indy Components*, oferece suporte à comunicação em rede e pode ser utilizado para implementar notificações assíncronas. Combinando *sockets* TCP[31] ou *WebSockets*, é possível que serviços Delphi emitam alertas para um painel central ou diretamente para os usuários, em tempo real. Essa estratégia é útil especialmente para aplicações que exigem resposta imediata a falhas, como sistemas financeiros ou industriais.

- **Integração com Serviços de Mensageria.** Para cenários distribuídos e com múltiplos microsserviços, a adoção de filas e *brokers* de mensagens, como *RabbitMQ*, *Kafka* ou até mesmo soluções mais simples como *Redis Pub/Sub*, pode facilitar o disparo e o gerenciamento de alertas. Microsserviços em Delphi podem publicar eventos de erro ou exceção em tópicos específicos, que são então consumidos por serviços especializados em tratamento de alertas como por exemplo, *bots* de notificação, SMS, e-mails ou dashboards.

- **Uso de Ferramentas de Observabilidade Externas.** Delphi também pode ser integrado a plataformas modernas de observabilidade, como *Grafana*, *Elastic Stack* (ELK) ou *New Relic*, por meio de *logs* estruturados e métricas exportadas. Essas ferramentas permitem

[31] do inglês, *Transmission Control Protocol*

a criação de regras visuais e automáticas para detecção de falhas, gerando alertas visuais, sonoros ou enviados por canais como *Slack*, *Teams* ou *Telegram*.

### F.3. Manutenção

Este passo requer a colaboração das equipes de desenvolvimento para garantir a coordenação e a manutenção de ações preventivas, corretivas e de melhoria com base nos dados coletados do monitoramento (ou seja, *dashboard*). Uma equipe de desenvolvimento precisa estar ciente das seguintes atividades de manutenção: manter as versões mais recentes do software para microsserviços; monitorar e corrigir *bugs* que afetam o comportamento ou a funcionalidade dos microsserviços; identificar e resolver problemas de desempenho em microsserviços; avaliar e ajustar a capacidade dos microsserviços para lidar com picos de tráfego e aumento de demanda; e, por fim, manter o monitoramento constante para identificar proativamente problemas e anomalias (COLANZI *et al.*, 2021b; LI; MA; LU, 2020).

### F.4. Reversão

Este passo sugere que as equipes de desenvolvimento possam reverter microsserviços (ou versões do sistema) em caso de falha. A reversão (ou *rollback*) permite que o sistema retorne ao ambiente legado e estável, facilitando assim a realização de uma manutenção eficaz. Portanto, pode-se dizer que habilitar o *rollback* é uma prática importante durante o ciclo de vida do microsserviço, pois fornece uma maneira de desfazer alterações que causaram problemas ou introduziram regressões no sistema. Ao incorporar estratégias de *rollback* como uma abordagem de manutenção, as equipes podem reduzir o risco de falhas e interrupções do sistema e garantir uma experiência do usuário mais confiável e consistente (FREIRE *et al.*, 2021). Os seguintes artefatos podem ser gerados com a execução deste passo:

| **Artefatos resultantes do Passo F.4** |
|---|
| **AT18.** Mecanismos de *rollback* implementados. |
| **AT19.** Documentação de reversão. |
| **AT20.** Planos de contingência atualizados. |

## 4.4 Considerações finais

O Processo Micro4Delphi surge como uma proposta estruturada para apoiar a modernização de sistemas legados desenvolvidos em Delphi, por meio de uma abordagem incremental e evolutiva. Fundamentado na revisão da literatura, o processo busca equilibrar teoria e prática, oferecendo uma solução viável e adaptável à realidade das organizações que ainda dependem

de aplicações legadas. Diante do exposto, pode-se dizer que a principal contribuição do Micro4Delphi está em sua capacidade de orientar equipes de desenvolvimento em um caminho gradativo de modernização, permitindo que os sistemas continuem operando enquanto são progressivamente transformados em uma arquitetura baseada em microsserviços. O processo considera aspectos importantes como a composição e o tamanho do time envolvido, bem como a necessidade de acesso ao sistema legado em execução, seu código-fonte e artefatos documentais, o que reforça a sua aplicabilidade prática. Além disso, ao enfatizar a participação ativa dos interessados, como clientes e especialistas de domínio, o Micro4Delphi valoriza a colaboração contínua ao longo do processo de modernização. Espera-se, assim, que os resultados obtidos contribuam não apenas para a renovação tecnológica, mas também para a sustentação de boas práticas de engenharia de software em projetos com alto grau de complexidade e legado histórico. Visando apresentar sua aplicabilidade e seus benefícios na prática, no Capítulo 5 é apresentado um estudo de caso, no qual o processo foi empregado em um cenário real de modernização.

# 5 ESTUDO DE CASO

## 5.1 Considerações Iniciais

Neste capítulo é apresentado o estudo de caso que foi desenvolvido para verificar a viabilidade, os pontos fortes e fracos do processo Micro4Delphi apresentado no Capítulo 4. Para isso, um sistema legado chamado Avance foi escolhido por ser desenvolvido em Delphi em uma abordagem estruturada e orientada a componentes. Na Seção 5.2 é apresentada uma visão geral desse sistema legado, onde são detalhados os principais requisitos e funcionalidades. Na Seção 5.3 é apresentada a aplicação do processo Micro4Delphi, seguindo as atividades e os passos executados para realizar a modernização do sistema legado para a arquitetura de microsserviços. Por fim, na Seção 5.4 são reportadas as considerações finais do capítulo.

## 5.2 O Sistema Avance

Para fins de entendimento do propósito do sistema escolhido como estudo de caso, nesta seção é apresentada uma descrição sobre a visão geral desse sistema, bem como a descrição apenas dos principais requisitos funcionais e não funcionais.

### Visão Geral

O sistema **Avance** é um software educacional, que tem como objetivo centralizar e facilitar o acesso a informações e funcionalidades pedagógicas, promovendo uma experiência digital segura, personalizada e eficiente. O sistema permite que professores cadastrem suas aulas de acordo com suas necessidades, escolas e turmas específicas e os alunos possam realizar as aulas de acordo com seu perfil. Além disso, esse sistema permite que relatórios de desempenho acadêmico e de acessos sejam emitidos, possibilitando um melhor gerenciamento das aulas.

### Requisitos Funcionais

Nesta seção são apresentados os requisitos funcionais do sistema **Avance**, destacando suas principais funcionalidades, regras de operação e os fluxos de interação esperados entre os usuários e a plataforma. A definição clara e precisa desses requisitos é fundamental não apenas para garantir que o sistema atenda adequadamente às necessidades dos usuários, mas também para fornecer uma base consistente que orientará as etapas subsequentes de implementação, testes e manutenção. A seguir, são descritos os principais requisitos funcionais do sistema:

- O sistema **Avance** permite que os usuários (ou seja, alunos e professores) realizem seu cadastro por meio do fornecimento de dados pessoais, possibilitando a autenticação no ambiente e o acesso às funcionalidades específicas de acordo com o perfil selecionado. No momento do cadastro, o usuário deve informar as seguintes informações: nome completo, data de nascimento, e-mail, senha de acesso, escola e turma vinculada.

- O sistema deve permitir o gerenciamento de diversas turmas, bem como os alunos matriculados em cada uma delas, assegurando a organização adequada do ambiente escolar. Quanto ao cadastro de turmas, o usuário deve informar as seguintes informações: identificação da turma (por exemplo: primeiro ano, segundo ano, entre outros), sala de aula, etc.

- No perfil do professor, o sistema deve permitir a criação de aulas específicas para cada turma, com base no conteúdo previamente registrado na plataforma. Além disso, os docentes podem elaborar exercícios relacionados a cada aula e turma, que devem ser concluídos pelos alunos durante o período das aulas, promovendo uma prática pedagógica mais dinâmica e direcionada.

- O sistema deve permitir o cadastro das escolas para as quais o sistema deve operar. Para isso, o usuário deve fornecer as seguintes informações: nome da instituição, cidade e estado.

- Em relação ao perfil do aluno, o sistema **Avance** oferece recursos para que cada estudante acompanhe o conteúdo estudado e os exercícios já realizados, contribuindo para o aprimoramento do processo de aprendizagem. O sistema permite visualizar o número de respostas corretas e incorretas em cada exercício, além de apresentar a solução recomendada para cada questão, incentivando a revisão e a consolidação do conhecimento.

Para fins de ilustração do sistema, na Figura 19 é apresentada a interface principal do ambiente destinado ao professor, onde são exibidas as opções de cadastro disponíveis no sistema, permitindo o gerenciamento de turmas, aulas, exercícios e demais recursos pedagógicos.

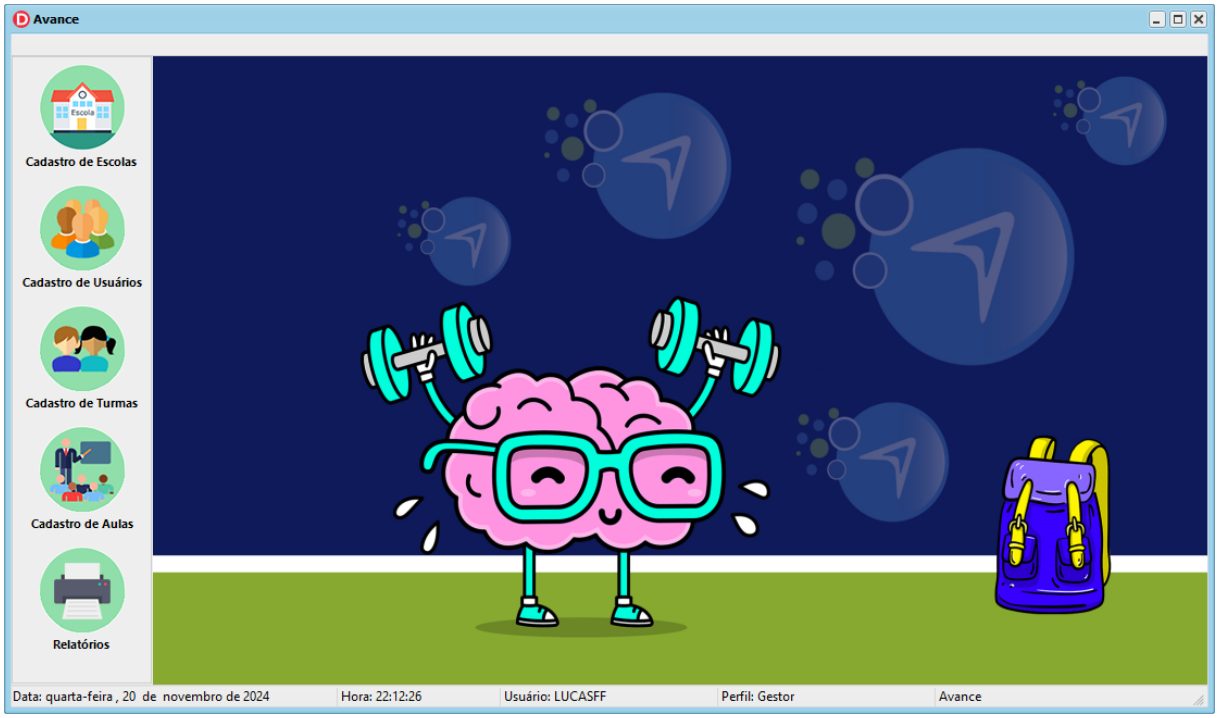
## Relatórios e consultas

Visando apoiar o acompanhamento e a gestão das atividades acadêmicas, o sistema **Avance** permite a impressão de diversos tipos de relatórios e consultas, a saber:

- Lista de usuários cadastrados, incluindo informações sobre escolas e turmas associadas;

- Relação de aulas disponíveis por turma, acompanhadas dos respectivos conteúdos, exercícios e questões associadas;

Figura 19 – Tela principal do sistema **Avance**



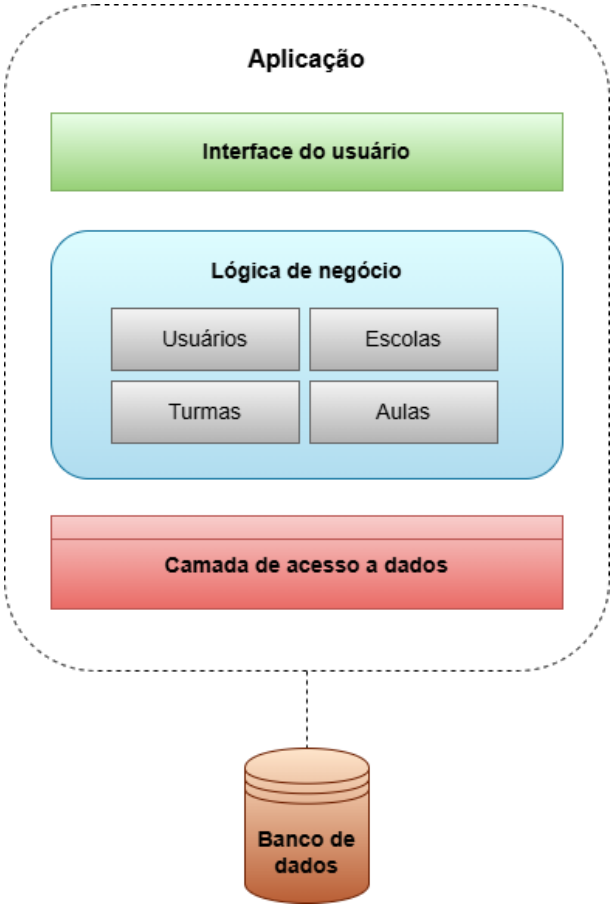

Fonte: Elaborada pelo autor.

- Monitoramento do desempenho acadêmico, permitindo o acompanhamento individualizado do progresso de cada aluno, incluindo o histórico de aulas assistidas, exercícios realizados e respostas fornecidas; e
- Controle de acessos, possibilitando a verificação do tempo de permanência dos alunos em cada aula, o que contribui para um gerenciamento mais preciso e completo das atividades pedagógicas.

## 5.3 Aplicando o Processo

Antes de iniciar a apresentação do Processo Micro4Delphi, vale destacar que o sistema **Avance** foi desenvolvido em Delphi 10.4 Tokio com um banco de dados Firebird versão 2.5. No que diz respeito a organização, esse sistema foi organizado em camadas e módulos, conforme ilustrado na Figura 20. Os componentes de conexão com o banco de dados ficam alocados em módulos de dados que servem como um contêiner, sem interface visual própria conhecidos como *datamodules*. Toda a lógica de negócios do sistema fica organizada em unidades separadas, arquivos com extensão ".pas" e os componentes da interface do usuário em formulários, arquivos com extensão ".dfm". Em termos de tamanho, este sistema possui 15 formulários, seis relatórios, 11 módulos de dados, 10 tabelas de banco de dados e 19 componentes de conectividade com o banco de dados, incluindo fontes de dados, tabelas e consultas.

Figura 20 – Arquitetura do sistema **Avance**




Fonte: Elaborada pelo autor.

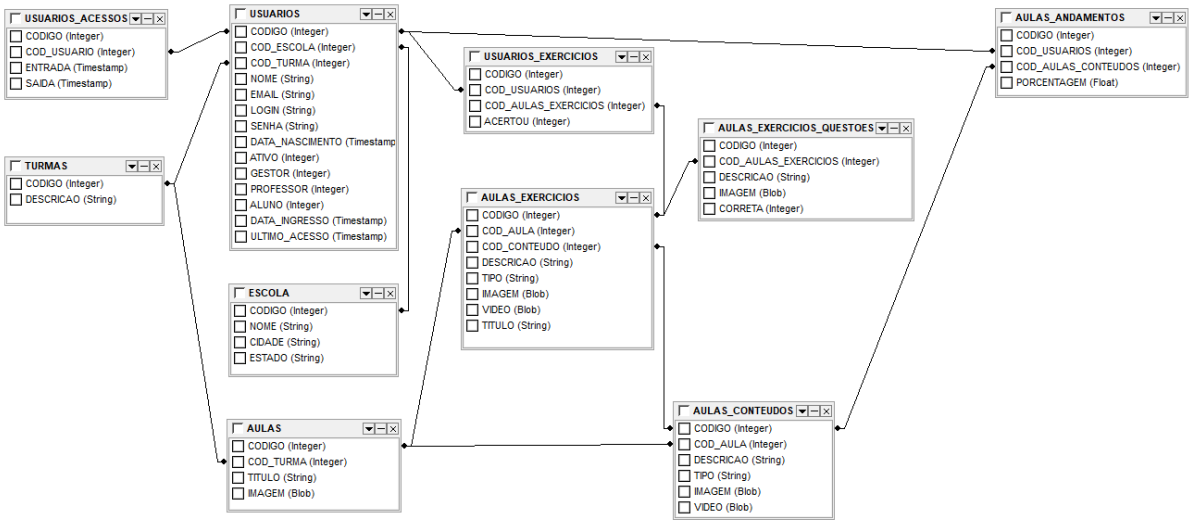
Como a modernização desse sistema está baseada em um processo de natureza cíclica e baseado em uma sequência de passos bem definida, optou-se por demonstrar o comportamento do processo Micro4Delphi (ou seja, a transição do sistema legado **Avance** para a arquitetura de microsserviços) para a criação do microsserviço `MSUser`, uma vez que a criação dos demais microsserviços requer o mesmo conjunto de atividades.

Inicialmente na **atividade de planejamento**, foi conduzida a definição do escopo do processo de modernização, juntamente com a definição da equipe, necessidade de trabalhar na questão da cultura organizacional e os treinamentos necessários para o sucesso da modernização (**Etapas A.1 a A4**). Nesse sentido, vale ressaltar que esse passo teve escopo de atuação limitado do ponto de vista de planejamento (ou seja, time de desenvolvimento e organização), uma vez que apenas um membro foi utilizado (ou seja, autor dessa dissertação de mestrado). Após essa atividade, alguns artefatos do sistema legado foram coletados, a saber: (i) código-fonte completo do sistema; (ii) banco de dados completo do sistema (ou seja, arquivo e modelo).

Na **atividade de análise**, cada uma das etapas prescritas no processo foi revisitada. Inicialmente, a análise envolveu a construção de um entendimento sobre o sistema legado

e seu banco de dados, além do modelo de negócios associado a tal sistema (**Etapas B.1 a B.3**). Nesse sentido, algumas funcionalidades relacionadas ao usuário foram identificadas, incluindo, por exemplo, a autenticação do usuário. Além disso, tabelas do banco de dados, como `USUARIOS_ACESSOS` e `USUARIOS`, foram identificadas como potenciais fontes de suporte para o microsserviço `MSUser`. Como o objetivo é transformar o sistema legado em microsserviços, decidiu-se que a interface do usuário deveria ser mantida e a linguagem nativa do sistema preservada (**Etapas B.3 e B.4**). Após essa atividade alguns artefatos foram coletados, a saber: (i) entendimento do modelo de negócio; (i) análise da arquitetura do sistema legado; (ii) análise do código fonte para mapeamento das funcionalidades do sistema; (iii) diagrama do banco de dados, conforme ilustrado na Figura 21; (iv) definição das tecnologias.

Figura 21 – Diagrama entidade relacionamento do sistema **`Avance`**



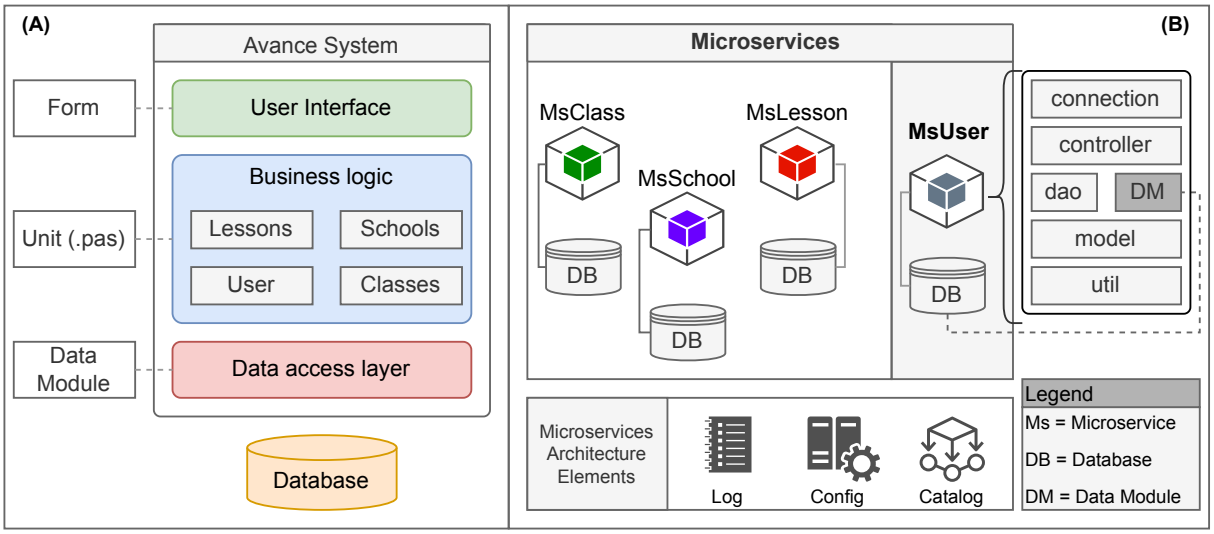

Fonte: Elaborada pelo autor.

Na **atividade de decomposição**, o código legado foi analisado para identificar os candidatos a microsserviços (**Etapa C.1**). Após a análise quatro candidatos a microsserviços foram identificados: (**MsClass**), (**MsSchool**), (**MsLesson**) e (**MsUser**), como ilustra a Figura 22. Vale ressaltar que esses microsserviços foram identificados/projetados conforme seus requisitos de domínio, de modo a minimizar o acoplamento existente entre as entidades, uma vez que esse sistema foi desenvolvido em uma abordagem baseada em componentes do Delphi, seguindo uma organização estruturada/modular de código.

Como o código-fonte do sistema **`Avance`** demonstrou ser de alto valor, a decisão foi que o sistema seja refatorado para que seu código-fonte e esquema de dados possam ser reutilizados na fase de desenvolvimento dos microsserviços (**Etapas C.2 a Etapas C.4**), de acordo com os novos interesses de cada microsserviço. Em seguida, as APIs equivalentes foram modeladas para os microsserviços identificados, com foco na definição de como esses microsserviços se comunicarão entre si e/ou com componentes externos ao sistema, juntamente com seus atributos

e métricas (**Etapas C.5 e C.6**). Por fim, a documentação desses microsserviços via *Swagger* deve ser concluída (**Etapa C.7**), pois visa fornecer às partes interessadas informações sobre como utilizá-los.

Figura 22 – Modernização de sistemas legados para MSA




Fonte: Elaborada pelo autor.

Durante a **atividade de desenvolvimento**, os microsserviços foram implementados, conforme ilustrado na Figura 22. Conforme ilustrado pelo microsserviço *MsUser*, cada microsserviço foi projetado com base em um estilo arquitetural em camadas, permitindo organização segmentada de código. Em outras palavras, cada microsserviço foi projetado para ter seu próprio banco de dados, sendo necessário definir uma unidade de conexão dedicada para cada um deles, conforme detalhado na Listagem 1. Com o desacoplamento do sistema legado em microsserviços, os componentes de conexão do banco de dados foram implementados via linha de comando. Na **linha 22**, o nome da conexão do banco de dados é definido, sendo então usado pelo componente de conexão `cnxDef` instanciado na **linha 45**. Entre as **linhas 47 e 55**, os parâmetros de conexão do banco de dados (ou seja, *driver* de conexão, usuário, senha, caminho do banco de dados e demais configurações do servidor) são definidos para cada microsserviço. Sobre o código fonte da referida listagem, deve ser destacada a similaridade desse código entre os demais microsserviços, pois apenas o valor da variável `Database` (Linha 50), que representa o caminho para a base de dados de cada microsserviço, deve ser diferenciado em cada um deles. Embora repetitiva, essa estratégia de codificação não cria dependência desnecessária de código entre os microsserviços, fazendo com que eles tenham repositórios de código independentes.

Listagem 1 – Código fonte da classe `udmFiredac`

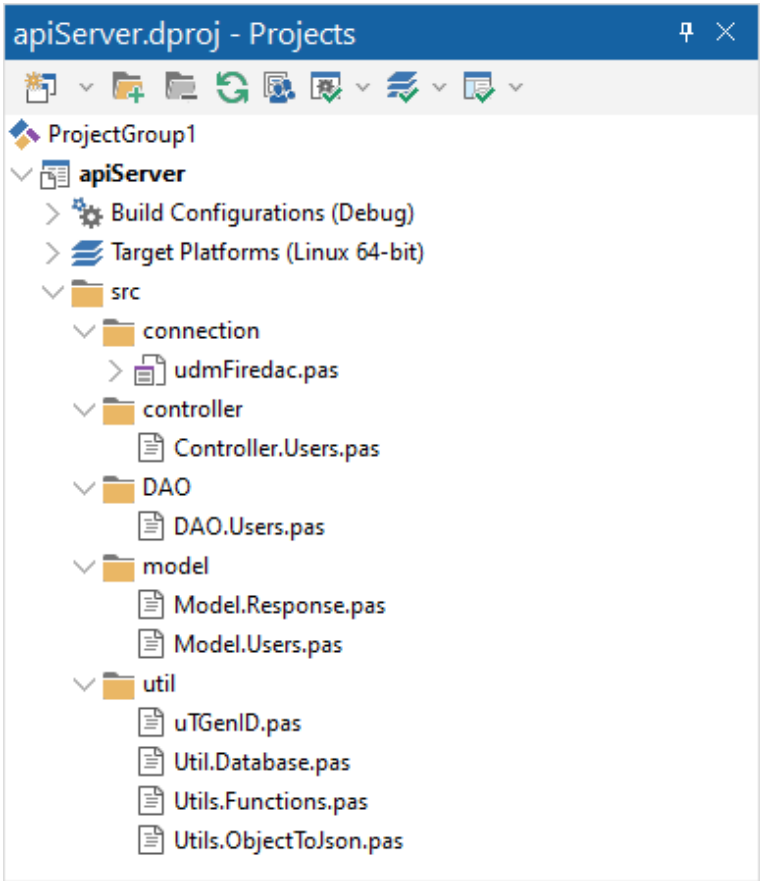
```
1  unit udmFiredac;
2
3  interface
4
```

```
5  uses
6   // Trecho de código omitido
7
8  type
9    TdmFiredac = class(TDataModule)
10     FDManager: TFDManager;
11     FDPhysFBDriverLink1: TFDPhysFBDriverLink;
12     FDGUIxWaitCursor1: TFDGUIxWaitCursor;
13     procedure FDManagerBeforeStartup(Sender: TObject);
14     procedure DataModuleCreate(Sender: TObject);
15   private
16     { Private declarations }
17   public
18     { Public declarations }
19     function getConnectionDefName: string;
20   end;
21
22 const FIREDAC_CONNECTION_DEF_NAME = 'Users_runtime';
23
24 var dmFiredac: TdmFiredac;
25
26 implementation
27
28 uses IdGlobalProtocols;
29
30 {$R *.dfm}
31
32 procedure TdmFiredac.DataModuleCreate(Sender: TObject);
33 begin
34   FDManager.Active := false;
35   FDPhysFBDriverLink1.vendorLib := '';
36 {$IFDEF MSWINDOWS}
37   FDPhysFBDriverLink1.vendorLib := ExtractFilePath(ParamStr(0)) + 'fbClient.dll';
38 {$ENDIF}
39 end;
40
41 procedure TdmFiredac.FDManagerBeforeStartup(Sender: TObject);
42 var cnxDef: IFDStanConnectionDef;
43 begin
44   FDManager.connectionDefs.Clear;
45   cnxDef := FDManager.connectionDefs.addConnectionDef;
46   cnxDef.Name := FIREDAC_CONNECTION_DEF_NAME;
47   cnxDef.Params.DriverID := 'FB';
48   cnxDef.Params.UserName := 'sysdba';
49   cnxDef.Params.Password := 'masterkey';
50   cnxDef.Params.Database :=
       ↪ 'C:\Development\Projects\Avance\Micro4Delphi\MsUsers\API\Database\Users.fdb';
51   cnxDef.Params.pooled := false;
52   cnxDef.Params.Add('Protocol=TCPIP');
53   cnxDef.Params.Add('Server=192.168.0.183');
```

```
54    cnxDef.Params.Add('Port=3050');
55    cnxDef.Params.Add('CharacterSet=WIN1252');
56    cnxDef.Apply;
57  end;
58
59  function TdmFiredac.getConnectionDefName: string;
60  begin
61    result := FIREDAC_CONNECTION_DEF_NAME;
62  end;
63
64  end.
```

A partir deste ponto, uma visão geral da organização do microsserviço **MsUser** é fornecida, conforme ilustrado na Figura 23. Essa figura reflete a estrutura organizacional ilustrada na Figura 22 (Lado B), onde cada camada foi mapeada como uma pasta dentro da pasta `src`. A seguir uma descrição de cada camada é apresentada:

Figura 23 – Organização do microsserviço MSUser




Fonte: Elaborada pelo autor.

- A camada `connection` contém a classe `udtmFiredac.pas` responsável pela conexão com o banco de dados, conforme detalhado na Listagem 1.

- A camada `controller` contém a classe `Controller.Users.pas` no fragmento de código apresentado na Listagem 2. Em resumo, o objetivo dessa classe é facilitar as interações entre a interface do usuário e a `camada DAO` para persistência de dados. Para ilustrar tais interações, esse controlador implementa dois procedimentos: os métodos `GET` (Linha 41) e `POST` (linha 74). Em ambos os métodos, é possível observar a criação do objeto `DAOUsers` a partir da camada `DAO` nas Linhas 46 e 81, respectivamente. Nas Linhas 51 e 86, são apresentadas as chamadas para os métodos `getUser` e `setUser` da classe `DAOUsers`, ambos descritos na classe `DAO.Users.pas` do (`pacote DAO`).

Listagem 2 – Código fonte da classe `Controller.Users`

```
1  unit Controller.Users;
2    // Trecho de código omitido
3
4  type
5    [SwagPath('Users', 'Users')]
6    TControllerUsers = class
7    private
8     // Trecho de código omitido
9    public
10     // Comandos Swagger para documentação
11     [SwagParamBody('body', TModelUsers)]
12
13     // Métodos Post
14     [SwagPOST('', 'Post', true)]
15     [SwagResponse(200, TModelUsers, 'Success')]
16     [SwagResponse(400, TModelResponse, 'Bad Request')]
17
18     // Procedimento post que chama o método setUsers da camada DAOUsers
19     procedure post;
20
21     // Métodos Get
22     [SwagGET('', 'Get', true)]
23     [SwagResponse(200, TModelUsers, 'Success')]
24     [SwagResponse(400, TModelResponse, 'Bad Request')]
25
26     // Procedimento Get que chama o método getUsers da camada DAOUsers
27     procedure get;
28     constructor Create(Req: THorseRequest; Res: THorseResponse);
29   end;
30
31 implementation
32
33 uses DAO.Users;
34
35 { TControllerUsers }
36 constructor TControllerUsers.Create(Req: THorseRequest; Res: THorseResponse);
37 begin
```

```
38    // Trecho de código omitido
39  end;
40
41  procedure TControllerUsers.get;
42  var
43    LRetorno: TModelResponse;
44    DAOUsers: TDAOUsers;
45  begin
46    DAOUsers := TDAOUsers.Create;
47
48    try
49      try
50        // Realiza a chamada do método getUsers da camada DAOUsers
51        FResponse.Status(200).Send<TJSONArray>(DAOUsers.getUsers);
52      except
53        on E: Exception do
54        begin
55          LRetorno := TModelResponse.Create;
56          LRetorno.Status := 400;
57          LRetorno.message := E.Message;
58
59          // Retorna o Json como objeto para a camada cliente
60          FResponse.Status(400).Send<TJSONObject>
61            (TJSON.ObjectToJsonObject(LRetorno, [joIgnoreEmptyArrays,
62            joIgnoreEmptyStrings]));
63        end;
64      end;
65    finally
66      // Trecho de código omitido
67    end;
68  end;
69
70  function TControllerUsers.getBody: TModelUsers;
71  // Trecho de código omitido
72  end;
73
74  procedure TControllerUsers.post;
75  var
76    Users: TModelUsers;
77    LRetorno: TModelResponse;
78    DAOUsers: TDAOUsers;
79  begin
80    Users := getBody;
81    DAOUsers := TDAOUsers.Create;
82
83    try
84      try
85        // Chama o método setUsers da camada DAOUsers
86        FResponse.Status(200).Send<TJSONObject>(DAOUsers.setUsers(Users));
87      except
```

```
88        on E: Exception do
89        begin
90          LRetorno := TModelResponse.Create;
91          LRetorno.Status := 400;
92          LRetorno.message := E.Message;
93
94          // Retorna Json como objeto para a camada do cliente
95          FResponse.Status(400).Send<TJSONObject>
96            (TJSON.ObjectToJsonObject(LRetorno, [joIgnoreEmptyArrays,
97            joIgnoreEmptyStrings]));
98        end;
99      end;
100   finally
101     // Trecho de código omitido
102   end;
103 end;
104 end.
```

- A camada DAO contém a classe (`DAO.Users`), cujo objetivo é lidar com a persistência de dados, conforme detalhado na Listagem 3. Para isso, esta classe implementa o método `getUsers` para consultar usuários e o método `postUsers` para incluir usuários no banco de dados.

Listagem 3 – Código fonte da classe `DAO.Users`

```
1  unit DAO.Users;
2    // Trecho de código omitido
3
4  type
5    TDAOUsers = class
6    private
7      // Trecho de código omitido
8    public
9      // Trecho de código omitido
10   end;
11
12 implementation
13
14 uses Util.Database;
15 { TDAOUsers }
16
17 procedure TDAOUsers.closeQuery;
18 begin
19   FDQuery.Close;
20 end;
21
22 destructor TDAOUsers.Destroy;
23 begin
24   // Trecho de código omitido
```

```
25  end;
26
27  function TDAOUsers.getUsers: TJSONArray;
28  var
29    Users: TModelUsers;
30    UsersList: TArray<TObject>;
31  begin
32    result := nil;
33    UsersList := TArray<TObject>.Create(nil);
34    FDQuery := TUtilDatabase.getFDQuery;
35
36    try
37      // Comando SQL para consulta
38      FDQuery.SQL.Clear;
39      FDQuery.SQL.Add('SELECT CODIGO, NOME');
40      FDQuery.SQL.Add('FROM USUARIOS');
41      FDQuery.SQL.Add('WHERE CODIGO > 0');
42      FDQuery.open;
43
44      SetLength(UsersList, FDQuery.RecordCount);
45
46      while not FDQuery.Eof do
47      begin
48        // Escreve o retorno para as propriedades do Model.Users
49        Users := TModelUsers.Create;
50        Users.id := FDQuery.FieldByName('CODIGO').asInteger;
51        Users.name := FDQuery.FieldByName('NOME').AsString;
52
53        // Adiciona os dados na lista de usuários
54        UsersList[FDQuery.recno - 1] := Users;
55        FDQuery.next;
56      end;
57
58      // Retorna a lista em formato Json para o controlador
59      if Length(UsersList) > 0 then
60        result := getJsonArray(UsersList);
61    finally
62      closeQuery;
63
64      if UsersList <> nil then
65        UsersList := nil;
66    end;
67  end;
68
69  function TDAOUsers.getJsonArray(const AValue: TArray<TObject>): TJSONArray;
70  var i: integer;
71  begin
72     // Trecho de código omitido
73  end;
74
```

```
75  function TDAOUsers.getJsonMsg(const tag, texto: string): TJsonObject;
76  begin
77    // Trecho de código omitido
78  end;
79
80  function TDAOUsers.postUsers(const Users: TModelUsers): TModelResponse;
81  begin
82    result := TModelResponse.Create;
83    result.status := 0;
84    result.message := '';
85
86    FDQuery := TUtilDatabase.getFDQuery;
87
88    try
89      // Comando SQL para consulta
90      FDQuery.SQL.Clear;
91      FDQuery.SQL.Add('SELECT CODIGO, NOME');
92      FDQuery.SQL.Add('FROM USUARIOS');
93      FDQuery.SQL.Add('WHERE CODIGO > 0');
94      FDQuery.open;
95
96      try
97        // Comando para incluir
98        FDQuery.Append;
99        if Users.ID = 0 then
100         FDQuery.FieldByName('CODIGO').asInteger :=
101           TGenID.getGenId('GEN_USUARIOS_ID')
102       else
103         FDQuery.FieldByName('CODIGO').asInteger := Users.ID;
104       FDQuery.FieldByName('NOME').AsString := Users.name;
105       FDQuery.Post;
106     finally
107       closeQuery;
108     end;
109   finally
110     result.status := 200;
111     result.message := 'Data entered successfully';
112   end;
113 end;
114
115 function TDAOUsers.setUsers(const Users: TModelUsers): TJsonObject;
116  // Trecho de código omitido
117 end;
118
119 end.
```

- A camada `model` contém uma classe designada (`Model.Users.pas`) que serve para instanciar as propriedades da tabela `USUARIOS` e a classe (`Model.Response.pas`) que é responsável pelo status e mensagens padronizadas para comunicação entre camadas da

aplicação.

- A camada `util` contém um conjunto de classes utilitárias para o funcionamento do microsserviço `MsUser`.

Vale destacar que a atividade de desenvolvimento do microsserviço **MsUser** é concluída com a documentação das *APIs* via *Swagger* entre as Linhas 10 e 27 (Listagem 2). Especificamente, na Linha 14 é apresentado o método `Post` com as mensagens de resposta associadas aos respectivos códigos detalhados nas Linhas 15 (`200` e "`Sucess`") e 16 (por exemplo, `400` e "`Bad Request`") . A documentação do método `Get` (Linhas 22 a 24) segue a mesma estrutura.

Portanto, pode-se afirmar que, após a conclusão da atividade de desenvolvimento, as **etapas D.1 a D.5** foram executadas integralmente neste estudo de caso, visto que o microsserviço agora está preparado para implantação em seu ambiente de execução. Em relação às demais atividades do processo Micro4Delphi, cabe destacar que as atividades de integração e monitoramento foram parcialmente conduzidas. Destacam-se o desenvolvimento de testes (**Etapa E.3**) por meio da *API Swagger* e o monitoramento de microsserviços no ambiente de execução (**Etapa F.1**). Embora essas atividades sejam relevantes para o processo de modernização, elas não abordam a transição de sistemas legados para microsserviços e, portanto, não serão detalhadas nesta seção.

Após finalizar e implantar o microsserviço **MSUser** utilizando o *Docker*, as alterações na camada visual do sistema legado são relativamente simples. Para isso, os métodos `getDados` (Linha 1) e `setDados` (Linha 13) devem ser utilizados, conforme apresentado na Listagem 4. O método `getDados` é responsável por buscar as informações da tabela `USUARIOS` e converter os dados do JSON de retorno do `MSUser` para o objeto *FDMemTable* do Delphi. O método `setDados` é responsável pela inclusão/alteração de registros na tabela `USUARIOS`.

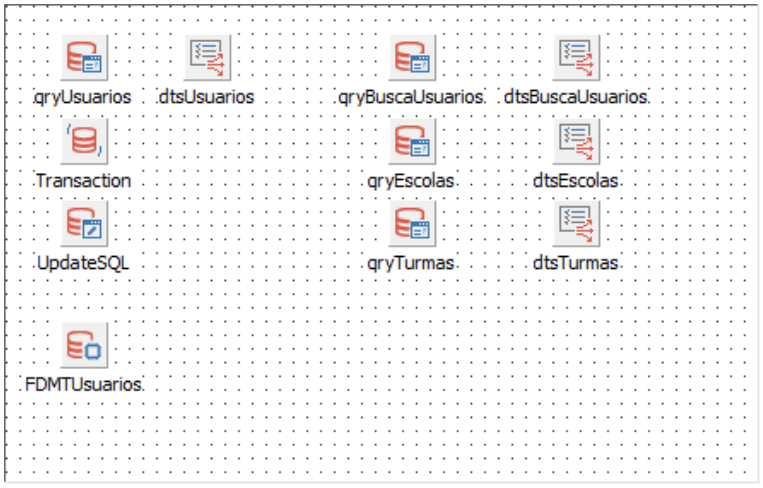
Listagem 4 – Código fonte da classe `MicroUsers`

```
1  procedure TfrmUsuarios.getDados;
2  begin
3     TRequest.New.BaseURL('http://localhost:8081/Usuarios')
4      .Adapters(TDataSetSerializeAdapter.New(dtmUsuarios.FDMTUsuarios))
5      .Accept('application/json').Get;
6
7    lblAPIUsuarios.Visible := true;
8    lblAPIUsuarios.Caption := 'Micro4DelphiUsuarios: ' +
9      TRequest.New.BaseURL('http://localhost:8081/ping').Get.Content;
10   Application.ProcessMessages;
11 end;
12
13 procedure TfrmUsuarios.setDados;
14 begin
15   TRequest.New.BaseURL('http://localhost:8081/Usuarios')
```

```
16      .ContentType('application/json')
17      .AddBody('{"codigo":"0","nome":"' + edtNome.Text + '"}').Post;
18
19    lblAPIUsuarios.Visible := true;
20    lblAPIUsuarios.Caption := 'Micro4DelphiUsuarios: ' +
21      TRequest.New.BaseURL('http://localhost:8081/ping').Get.Content;
22    Application.ProcessMessages;
23  end;
```

Com base no exposto, pode-se dizer que a utilização de componentes de acesso a dados no Delphi (veja Figura 24) não é mais necessária. Esses componentes são responsáveis pela criação de conexão com o banco de dados, além de permitir a conexão com os componentes visuais da interface. Essa abordagem de desenvolvimento gera um alto grau de acoplamento entre os componentes de desenvolvimento e o banco de dados do sistema. Portanto, pode-se dizer que as listagens de código apresentadas nas Listagens 1 a 4 substituem tais componentes, fazendo com que toda requisição da interface passe a ser controlada pelas APIs de cada microsserviço.

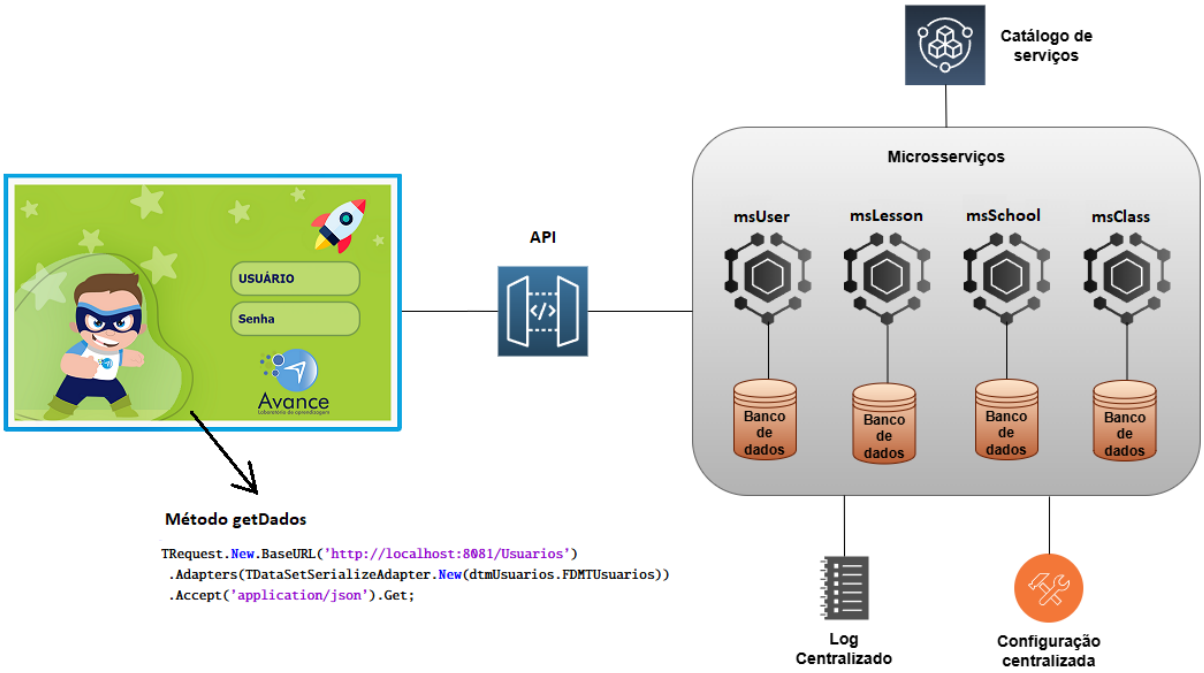
Figura 24 – Componentes de acesso a dados no Delphi




Fonte: Elaborada pelo autor.

Na Figura 25 é ilustrada a nova organização do sistema *Avance*, sendo possível observar a interface acesso ao sistema utilizando o microsserviço `MsUser` para realizar uma operação de autenticação. Para isso, essa interface solicita efetua uma chamada a esse microsserviço através do método `getDados`, que irá verificar se os dados fornecidos na interface (ou seja, usuário e senha) existem no banco de dados do sistema. Dessa forma, pode-se dizer que um ambiente modernizado foi elaborado, onde as requisições são realizadas via API, permitindo que o sistema baseado em microsserviço trabalhe totalmente desacoplado. Além de reduzir a dependência do código legado, essa abordagem permite maior flexibilidade para evoluções futuras, facilita a manutenção e a implementação de novas regras de negócio, e garante uma comunicação padronizada entre os módulos do sistema.

Figura 25 – Acesso do sistema Avance via `MsUser`



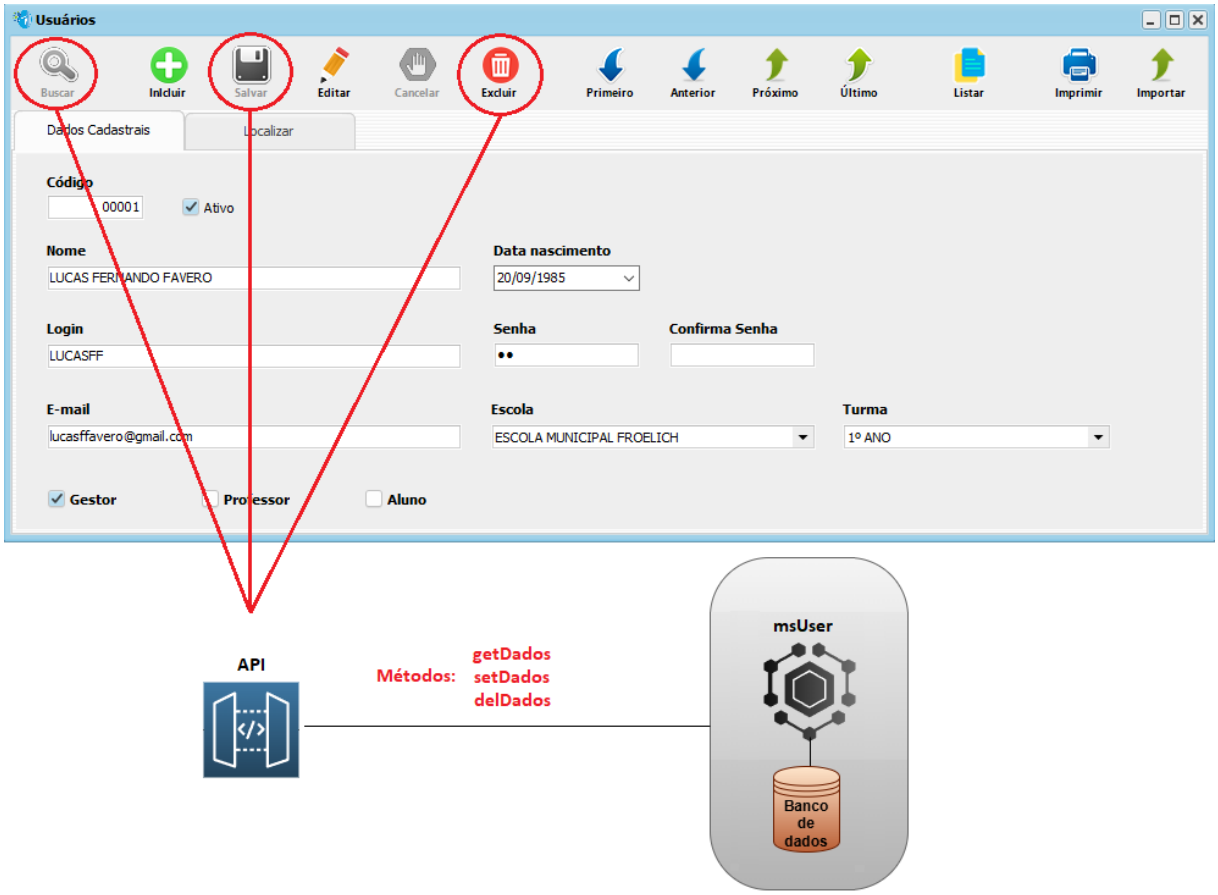

Fonte: Elaborada pelo autor.

Na Figura 26 é ilustrada e tela de gerenciamento de usuários, que permite realizar um conjunto de operações que são ilustradas no menu superior em botões. Para fins de apresentação, as funcionalidades de busca, salvamento e exclusão de usuários são utilizadas. Como pode ser observado, essas funcionalidades fazem referência ao conjunto de funcionalidades equivalentes via API (veja linhas em vermelho). Para isso a API do microsserviço `MsUser` implementa três métodos, a saber: `getDados`, `setDados` e `delDados`. Dessa forma, os componentes chamados não visuais, aqueles responsáveis pela conexão da interface com os componentes de acesso a dados (veja Figura 24), contidos nessa interface foram removidos e chamadas para as API foram inseridas em cada um dos botões destacados. Portanto, todas as funcionalidades legadas tanto de cadastros, pesquisas e relatórios não são mais necessárias, ou seja, tudo passa ser controlado e fornecido pelos microsserviços, onde cada um tem suas funcionalidades específicas e seus próprios banco de dados. As telas do sistema legado (front-end) permanecem como estão, porém, as funcionalidades legadas e componentes não são mais necessários. Por fim, vale ressaltar que essa mudança também abre caminho para integrar o sistema a outros serviços externos, melhorar a escalabilidade e atender de forma mais eficiente às demandas dos usuários.

Por fim, vale destacar que o código fonte desse estudo de caso[1] está disponível em repositório público, de modo que os interessados podem ter acesso a todo processo técnico de configuração das ferramentas necessárias para configuração do ambiente de execução. Nesse sentido, vale destacar que esse processo de configuração, embora seja um esforço adicional, não

[1] <https://github.com/LucasFFavero/Avance>

Figura 26 – Cadastro de usuários do sistema Avance via msUser




Fonte: Elaborada pelo autor.

deve ser considerado como uma barreira de impedimento para adoção do processo Micro4Delphi e/ou do ambiente de desenvolvimento Delphi, pois deve ser realizado obrigatória em outros ambientes de desenvolvimento e/ou linguagens de programação para viabilizar a adoção da arquitetura de microsserviços e seus recursos.

## 5.4 Considerações Finais

Neste capítulo foi apresentado um estudo de caso que teve como propósito avaliar a aplicabilidade do processo Micro4Delphi, o qual foi descrito no Capítulo 4. Na Seção 5.2 foi detalhada a visão geral do sistema legado **`Avance`**, incluindo seus principais requisitos e funcionalidades, de modo a contextualizar o cenário de aplicação. Em seguida, na Seção 5.3, foi descrita a execução do processo Micro4Delphi, contemplando as atividades e passos empregados para conduzir a modernização do sistema em direção a uma arquitetura baseada em microsserviços. A análise realizada evidenciou não apenas a viabilidade do processo, mas também sua capacidade de orientar, de maneira sistemática, a transformação de um sistema legado em Delphi, equilibrando continuidade operacional e evolução tecnológica. Assim, o estudo de caso

serviu como validação prática do processo proposto, oferecendo subsídios tanto para a comunidade acadêmica quanto para profissionais que enfrentam desafios semelhantes. No Capítulo 6 são apresentadas as conclusões desta dissertação, destacando suas principais contribuições, limitações e possíveis direções para trabalhos futuros.

# 6 CONCLUSÃO

Este trabalho apresentou o Micro4Delphi, um processo para apoiar a modernização de sistemas legados desenvolvidos em Delphi para a arquitetura baseada em microsserviços. Esse processo foi estruturado de forma a oferecer uma abordagem gradual e prática, contemplando não apenas aspectos técnicos, mas também organizacionais, reconhecendo que a modernização bem-sucedida depende da integração entre tecnologia, pessoas e processos. Para isso, cada atividade foi organizada em um conjunto de passos que orientam tanto a análise e adaptação do sistema legado, quanto a preparação e reestruturação das equipes de desenvolvimento e da própria organização. O Micro4Delphi busca, assim, fornecer um caminho sistemático para a transição, reduzindo os riscos inerentes à modernização, promovendo o desacoplamento gradual dos componentes legados e incentivando boas práticas arquiteturais e de gestão.

De acordo com a proposta apresentada nesta dissertação, o desenvolvimento desse tipo de processo envolve uma carga elevada de conceitos e conhecimentos de diferentes áreas. Inicialmente, no Capítulo 2 foram reportados os conceitos de arquitetura de software, as principais características da arquitetura monolítica, arquitetura orientada a serviços e uma descrição detalhada sobre a arquitetura de microsserviços, destacando suas características e benefícios, desafios que devem ser levados em consideração ao se adotar esse estilo arquitetural, as vantagens e desvantagens e as melhores práticas que se deve adotar para trabalhar com microsserviços. Além disso, as principais diferenças entre a arquitetura de microsserviços e arquiteturas monolíticas são também apresentadas. Em seguida, um mapeamento sobre a modernização de sistemas legados para a arquitetura de microsserviços, estabelecido com base em 43 estudos primários, foi apresentado no Capítulo 3. No Capítulo 4 foram reportados os detalhes do Micro4Delphi, processo desenvolvido neste projeto de mestrado acadêmico que tem por objetivo apoiar a modernização de sistemas legados desenvolvidos em Delphi para a arquitetura de microsserviços. No Capítulo 5 foi apresentado o estudo de caso que foi desenvolvido para avaliar a viabilidade, os pontos fortes e fracos do processo Micro4Delphi. Por fim, neste capítulo são apresentadas as conclusões desta dissertação de mestrado da seguinte maneira: na Seção 6.1 são descritas as contribuições dessa dissertação; na Seção 6.2 são reportadas as principais dificuldades enfrentadas durante a condução deste projeto de mestrado e suas limitações; e na Seção 6.3 são apresentadas as atividades que devem/podem ser conduzidas como trabalhos futuros.

## 6.1 Contribuições da dissertação

O processo proposto nesta dissertação apresenta contribuições relevantes para as áreas de desenvolvimento e engenharia de software, especialmente no contexto da modernização de sistemas legados. Até onde se tem conhecimento, esta é a primeira iniciativa sistematizada que efetivamente apoia a transição de sistemas desenvolvidos em Delphi para uma arquitetura baseada em microsserviços. Além de oferecer uma solução prática para um problema recorrente na indústria, o processo também se configura como um arcabouço teórico, capaz de inspirar a formulação de novas iniciativas ou o aprimoramento de abordagens já existentes.

O estudo de caso[1] apresentado no Capítulo 5 exemplifica a aplicação do processo e fornece uma visão detalhada sobre a modernização de um sistema legado em Delphi, destacando a reorganização do sistema a partir de uma perspectiva orientada a microsserviços. Esse estudo de caso, portanto, pode servir como um guia ou referência prática para profissionais e organizações interessadas em conduzir uma transformação semelhante para atender às demandas do cenário computacional contemporâneo. Os resultados reforçam a utilidade e o potencial do processo Micro4Delphi em facilitar a jornada de transformação digital, oferecendo um caminho estruturado para profissionais e pesquisadores que enfrentam os desafios da modernização de sistemas legados em Delphi, ao mesmo tempo em que contribui para o avanço do conhecimento na área.

Outra contribuição deste trabalho é o mapeamento sistemático da literatura apresentado no Capítulo 3. Para essa finalidade, o mapeamento foi baseado no protocolo formal de pesquisa (veja Apêndice A) para responder quatro questões de pesquisa. A partir das informações coletadas, foi possível estabelecer um panorama geral da área, identificando as lacunas, tendências, ambientes de aplicação, tecnologias utilizadas, entre outros tópicos. Dessa forma, espera-se fornecer à comunidade científica uma fonte de conhecimento acerca da modernização de sistemas legados para microsserviços.

## 6.2 Dificuldades e limitações

Durante a condução deste trabalho, algumas dificuldades e limitações foram identificadas e merecem ser registradas, tanto para contextualizar os resultados alcançados quanto para orientar futuras pesquisas na área. A seguir, uma descrição dessas dificuldades e/ou limitações é apresentada:

- Uma das principais dificuldades enfrentadas esteve relacionada à escassez de literatura específica sobre a modernização de sistemas legados desenvolvidos em Delphi para arquiteturas de microsserviços. Embora existam diversos estudos sobre modernização de

[1] <https://github.com/LucasFFavero/Avance>

sistemas legados em geral e sobre microsserviços, nenhum aborda de maneira explícita do contexto do Delphi, o que exigiu um esforço adicional para adaptar conceitos, práticas e técnicas a essa realidade. Essa limitação também reforça a relevância do processo proposto, por preencher uma lacuna importante na área.

- Outra dificuldade significativa foi a necessidade de lidar com sistemas legados pouco documentados e, em alguns casos, com baixo nível de manutenção ao longo dos anos. Essa situação aumentou a complexidade para compreender a estrutura do sistema legado, identificar os módulos candidatos à extração como microsserviços e avaliar os impactos da modernização. Esse cenário é, contudo, comum em projetos reais de modernização e, portanto, reflete a realidade enfrentada por muitas organizações.

- Em relação à aplicação prática do processo, a realização do estudo de caso também apresentou limitações em relação ao ambiente organizacional onde foi conduzido, como restrições de tempo e de recursos humanos para execução de algumas atividades. Além disso, a validação do processo ocorreu em um único estudo de caso por restrições de tempo, o que limita a generalização imediata dos resultados para outros contextos organizacionais ou técnicos.

- Por fim, é importante destacar que a transição de sistemas legados para uma arquitetura de microsserviços é, por natureza, um processo de longo prazo e altamente dependente do contexto específico da organização. Assim, o processo proposto neste trabalho não pretende ser uma solução definitiva ou única para todos os cenários, mas sim uma orientação prática que pode e deve ser adaptada conforme as características e necessidades de cada cenário de utilização.

As dificuldades enfrentadas ao longo deste projeto reforçam a complexidade do processo de modernização de sistemas legados em Delphi para arquiteturas de microsserviços. A escassez de literatura específica exigiu a adaptação de conceitos e práticas, evidenciando a originalidade e relevância da proposta. Além disso, soma-se a baixa documentação e o histórico de manutenção limitado dos sistemas legados analisados representaram desafios adicionais, mas também refletiram a realidade comum de muitas organizações.

Do ponto de vista prático, a aplicação do processo demonstrou que fatores organizacionais, como tempo e recursos disponíveis, impactam diretamente sua execução, confirmando a necessidade de flexibilidade e adaptação em cada contexto. Além disso, o caráter gradual e dependente do cenário organizacional mostra que a modernização não deve ser vista como uma solução imediata, mas como um esforço contínuo e estruturado por parte dos interessados.

Por fim, cabe reconhecer que a transição de sistemas legados para uma arquitetura de microsserviços é um processo contínuo, de longo prazo e sujeito a múltiplos fatores técnicos e

organizacionais. Nesse sentido, o Micro4Delphi não se apresenta como solução universal (ou "bala de prata"), mas como um guia estruturado que pode apoiar organizações nesse caminho de transformação digital.

Apesar do cenário exposto nesta seção, acredita-se que os resultados deste projeto de mestrado acadêmico contribuem significativamente para o avanço do conhecimento e para a prática de modernização de sistemas legados em Delphi, podendo o processo Micro4Delphi servir como um arcabouço teórico para pesquisas e iniciativas futuras.

## 6.3 Trabalhos futuros

Em relação aos trabalhos futuros envolvendo o processo Micro4Delphi, estão previstas as seguintes direções de continuidade e aprofundamento: (i) condução de novos estudos de caso, visando ampliar a avaliação do processo em diferentes domínios de software, como aplicações Web, móveis e sistemas corporativos. Essa diversidade permitirá verificar a aplicabilidade do Micro4Delphi em contextos tecnológicos distintos e com diferentes exigências organizacionais. (ii) instanciação do arcabouço teórico do Micro4Delphi em outras linguagens de programação, com o objetivo de adaptar o processo para ser utilizado com outras tecnologias além do Delphi, possibilitando avaliar sua flexibilidade e comportamento em diferentes ambientes de desenvolvimento; e (iii) aplicação do processo ambientes industriais reais, o que permitirá observar a eficácia e os desafios do Micro4Delphi em contextos práticos e complexos, caracterizados por demandas reais do mercado, equipes multidisciplinares, restrições operacionais e infraestrutura robusta, contribuindo para o refinamento e validação do processo.

# REFERÊNCIAS

AFFONSO, F. J.; PASSINI, W. F.; NAKAGAWA, E. Y. A reference architecture to support the development of mobile applications based on self-adaptive services. *Pervasive and Mobile Computing*, v. 53, p. 33 – 48, 2019. ISSN 1574-1192. Nenhuma citação no texto.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A systematic mapping study in microservice architecture. *Journal of Systems and Software*, v. 174, p. 110890, 2021. Nenhuma citação no texto.

AMUNDSEN, M. *et al.* Microservice architecture: Aligning principles, practices, and culture. In: . [S.l.: s.n.], 2016. Nenhuma citação no texto.

AUER, F. *et al.* From monolithic systems to microservices: An assessment framework. *Information and Software Technology*, Elsevier B.V., v. 137, 2021. ISSN 09505849. Cited by: 38; All Open Access, Green Open Access, Hybrid Gold Open Access. Disponível em: <https://doi.org/10.1016/j.infsof.2021.106600>. Nenhuma citação no texto.

AWS. Microsserviços. 2023. Disponível em: <https://aws.amazon.com/pt/microservices/>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

AYAS, H. M.; LEITNER, P.; HEBIG, R. The migration journey towards microservices. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science and Business Media Deutschland GmbH, v. 13126 LNCS, p. 20 – 35, 2021. ISSN 03029743. Disponível em: <https://doi.org/10.1007/978-3-030-91452-3_2>. Nenhuma citação no texto.

BAMBERGER, B.; KöRBER, B. Migrating monoliths to microservices integrating robotic process automation into the migration approach. *Journal of Automation, Mobile Robotics and Intelligent Systems*, Industrial Research Institute for Automation and Measurements, v. 2022, n. 1, p. 72 – 82, 2022. ISSN 18978649. Disponível em: <https://doi.org/10.14313/JAMRIS/1-2022/8>. Nenhuma citação no texto.

BANDARA, C.; PERERA, I. Transforming monolithic systems to microservices - an analysis toolkit for legacy code evaluation. In: . Institute of Electrical and Electronics Engineers Inc., 2020. p. 95 – 100. ISBN 978-172818653-5. Disponível em: <https://doi.org/10.1109/ICTer51097.2020.9325443>. Nenhuma citação no texto.

Bass; Clements, P.; Kazman, R. Software architecture in practice. *Addison-Wesley Professional*, 3rd. ed., 2012. Nenhuma citação no texto.

BOGNER, J. *et al.* Microservices in practice: A survey of microservice architecture practitioners. *Empirical Software Engineering*, v. 26, n. 4, p. 1–35, 2021. Nenhuma citação no texto.

BOOCH, G. Object-oriented analysis and design with applications (2nd ed.). benjamin/cummings publishing company, inc. 1994. Nenhuma citação no texto.

CARVALHO. Extraction of configurable and reusable microservices from legacy systems: an exploratory study. *Em Proceedings of the 23rd International Systems and Software Product Line Conference*, 2017. Disponível em: <http://dl.acm.org/citation.cfm?doid=3336294.3336319>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

CLEMENTS, P. *et al. Documenting Software Architectures: Views and Beyond.* Pearson Education, 2010. ISBN 9780132488594. Disponível em: <https://books.google.com.br/books?id=UTZbsrA4qAsC>. Nenhuma citação no texto.

COLANZI, T. *et al.* Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices. In: *The 15th Brazilian Symposium on Software Components, Architectures, and Reuse*. New York, NY, USA: Association for Computing Machinery, 2021. p. 61–70. ISBN 9781450384193. Nenhuma citação no texto.

COLANZI, T. *et al.* Are we speaking the industry language? the practice and literature of modernizing legacy systems with microservices. In: . Association for Computing Machinery, 2021. p. 61 – 70. ISBN 978-145038419-3. Disponível em: <https://doi.org/10.1145/3483899.3483904>. Nenhuma citação no texto.

DEHGHANI, M. *et al.* Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. *Software and Systems Modeling*, Springer Science and Business Media Deutschland GmbH, v. 21, n. 3, p. 1115 – 1133, 2022. ISSN 16191366. Cited by: 2; All Open Access, Green Open Access. Disponível em: <https://doi.org/10.1007/s10270-022-00977-3>. Nenhuma citação no texto.

DIESTE, O.; GRIMáN, A.; JURISTO, N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 14, n. 5, p. 513–539, 2009. ISSN 1382-3256. Nenhuma citação no texto.

DRAGONI *et al.* Present and ulterior software engineering. In: *Microservices: Yesterday, Today, and Tomorrow.* [S.l.]: Springer, 2017. p. 195–216. Nenhuma citação no texto.

EMBARCADERO. *Native Apps For Any Device From One Codebase With Delphi!* 2025. On-line. Disponível em: <https://www.embarcadero.com/products/delphi>, Acessado em 6 de outubro de 2025. Nenhuma citação no texto.

ERL, T. *et al. SOA with REST*. Philadelphia, PA: Prentice Hall, 2012. Nenhuma citação no texto.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.l.]: Addison-Wesley, 2003. Nenhuma citação no texto.

FOWLER. Patterns of enterprise application architecture. pearson education. 2002. Nenhuma citação no texto.

FOWLER. Richardson maturity model. 2010. Disponível em: <https://martinfowler.com/articles/richardsonMaturityModel.html>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

FREIRE, A. F. A. A. *et al.* Migrating production monolithic systems to microservices using aspect oriented programming. *Software - Practice and Experience*, John Wiley and Sons Ltd, v. 51, n. 6, p. 1280 – 1307, 2021. ISSN 00380644. Cited by: 8. Disponível em: <https://doi.org/10.1002/spe.2956>. Nenhuma citação no texto.

FREITAS, F.; FERREIRA, A.; CUNHA, J. A methodology for refactoring orm-based monolithic web applications into microservices. *Journal of Computer Languages*, Elsevier Ltd, v. 75, 2023. ISSN 25901184. Disponível em: <https://doi.org/10.1016/j.cola.2023.101205>. Nenhuma citação no texto.

GARLAN; SHAW. An introduction to software architecture. 1993. Nenhuma citação no texto.

IBM. Soa vs. microservices. 2023. Disponível em: <https://www.ibm.com/topics/soa>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

JAMBUNATHAN, B.; KALPANA, Y. Microservice design for container based multi-cloud deployment. *International Journal of Engineering and Technology (IJET)*, v. 8, 02 2016. Nenhuma citação no texto.

KAZANAVIčIUS, J.; MAžEIKA, D. An approach to migrate from legacy monolithic application into microservice architecture. In: D., N. *et al.* (Ed.). Institute of Electrical and Electronics Engineers Inc., 2023. ISBN 979-835030383-4. Disponível em: <https://doi.org/10.1109/eStream59056.2023.10135021>. Nenhuma citação no texto.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Nenhuma citação no texto.

KITCHENHAM, B. A.; DYBA; JORGENSEN. M. evidence-based software engineering. In: . [S.l.]: Proceedings of the 26th International Conference on Software Engineering., 2004. p. 273–281. ISBN 0769521630. Nenhuma citação no texto.

KRAUSE, A. *et al.* Microservice decomposition via static and dynamic analysis of the monolith. In: . Institute of Electrical and Electronics Engineers Inc., 2020. p. 9 – 16. ISBN 978-172817415-0. Cited by: 18; All Open Access, Green Open Access. Disponível em: <https://doi.org/10.1109/ICSA-C50368.2020.00011>. Nenhuma citação no texto.

KURYAZOV, D.; JABBOROV, D.; KHUJAMURATOV, B. Towards decomposing monolithic applications into microservices. In: . Institute of Electrical and Electronics Engineers Inc., 2020. ISBN 978-172817385-6. Disponível em: <https://doi.org/10.1109/AICT50176.2020.9368571>. Nenhuma citação no texto.

KYRYK, M. *et al.* Methods and process of service migration from monolithic architecture to microservices. In: . Institute of Electrical and Electronics Engineers Inc., 2022. p. 553 – 558. ISBN 978-166546861-9. Cited by: 2. Disponível em: <https://doi.org/10.1109/TCSET55632.2022.9767055>. Nenhuma citação no texto.

LENARDUZZI, V.; TAIBI, D.; JANES, A. Migrating from monolithic systems to microservices: An industrial survey. *IEEE Software*, v. 39, n. 2, p. 28–35, 2022. Nenhuma citação no texto.

LEWIS; FOWLER. Microservices - a definition of this new architectural term. In: . [S.l.: s.n.], 2014. p. 1, 2, 13, 26, 43, 47. Disponível em: <https://martinfowler.com/articles/microservices.html>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

LEWIS, J.; FOWLER, M. *Microservices Guide*. 2019. On-line. Avaliable in: <https://martinfowler.com/microservices>, accessed on 6 de outubro de 2025. Nenhuma citação no texto.

LI, C.-Y.; MA, S.-P.; LU, T.-W. Microservice migration using strangler fig pattern: A case study on the green button system. In: . Institute of Electrical and Electronics Engineers Inc., 2020. p. 519 – 524. ISBN 978-172819255-0. Disponível em: <https://doi.org/10.1109/ICS51289.2020.00107>. Nenhuma citação no texto.

MA, S.-P. *et al.* Microservice migration using strangler fig pattern and domain-driven design. *Journal of Information Science and Engineering*, Institute of Information Science, v. 38, n. 6, p. 1285 – 1303, 2022. ISSN 10162364. Disponível em: <https://doi.org/10.6688/JISE.202211_38(6).0010>. Nenhuma citação no texto.

MACKENZIE *et al. Reference model for service oriented architecture 1.0 OASIS standard*. [S.l.]: Pearson Education, 2016. v. 12. 18 p. Nenhuma citação no texto.

MAZZARA, M. *et al.* Microservices: Migration of a mission critical system. *IEEE Transactions on Services Computing*, Institute of Electrical and Electronics Engineers Inc., v. 14, n. 5, p. 1464 – 1477, 2021. ISSN 19391374. Cited by: 16; All Open Access, Green Open Access. Disponível em: <https://doi.org/10.1109/TSC.2018.2889087>. Nenhuma citação no texto.

MISHRA, R. *et al.* Transition from monolithic to microservices architecture: Need and proposed pipeline. In: . Institute of Electrical and Electronics Engineers Inc., 2022. ISBN 978-166545046-1. Cited by: 0. Disponível em: <https://doi.org/10.1109/INCOFT55651.2022.10094556>. Nenhuma citação no texto.

NADAREISHVILI, I. *et al. Microservice Architecture: Aligning Principles, Practices, and Culture*. 1. ed. [S.l.]: O'Reilly Media, 2021. ISBN 978-1491956250. Nenhuma citação no texto.

NEWMAN, S. *Building microservices: Designing Fine-Grained Systems.* [S.l.]: O'Reilly Media, Beijing Sebastopol, CA, first edition, 2015. 2, 16, 17, 18, 19, 20, 21, 29, 63, 64, 117 p. ISBN 978-1-4919-5035-7. Nenhuma citação no texto.

NEWMAN, S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, Incorporated, 2019. ISBN 978-1-4920-4784-1. Disponível em: <https://books.google.com.br/books?id=iul3wQEACAAJ>. Nenhuma citação no texto.

OSMAN, M. H. *et al.* From monolith to microservices: A semi-automated approach for legacy to modern architecture transition using static analysis. *International Journal of Advanced Computer Science and Applications*, Science and Information Organization, v. 13, n. 10, p. 907 – 916, 2022. ISSN 2158107X. Disponível em: <https://doi.org/10.14569/IJACSA.2022.01310107>. Nenhuma citação no texto.

PARIKH, A. *et al.* Monolithic to microservices architecture - a framework for design and implementation. In: . Institute of Electrical and Electronics Engineers Inc., 2022. p. 90 – 96. ISBN 979-835039784-0. Disponível em: <https://doi.org/10.1109/ICCPC55978.2022.10072238>. Nenhuma citação no texto.

PASSINI, W. F.; AFFONSO, F. J. Developing self-adaptive service-oriented mobile applications: A framework based on dynamic deployment. *International Journal of Software Engineering and Knowledge Engineering*, v. 28, n. 11n12, p. 1537–1558, 2018. Nenhuma citação no texto.

PASSINI, W. F. *et al.* Design of frameworks for self-adaptive service-oriented applications: A systematic analysis. *Software: Practice and Experience*, v. 52, n. 1, p. 5–38, 2022. Nenhuma citação no texto.

PETERSEN, K. *et al.* Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, UK: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77. Nenhuma citação no texto.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, v. 64, p. 1–18, 2015. ISSN 0950-5849. Disponível em: <https://doi.org/10.1016/j.infsof.2015.03.007>. Nenhuma citação no texto.

PRASANDY, T. *et al.* Migrating application from monolith to microservices. In: . Institute of Electrical and Electronics Engineers Inc., 2020. p. 726 – 731. ISBN 978-172817071-8. Disponível em: <https://doi.org/10.1109/ICIMTech50083.2020.9211252>. Nenhuma citação no texto.

PRESSMAN, R. Engenharia de software: Uma abordagem profissional. In: . [S.l.]: AMGH Editora Ltda., 2016. (7ª edição), p. Capítulo 13. ISBN 978-85-8055-534-9. Nenhuma citação no texto.

PRETI, J. P. D. *et al.* Monolithic to microservices migration strategy in public safety secretariat of mato grosso. In: . Institute of Electrical and Electronics Engineers Inc., 2021. ISBN 978-166543897-1. Disponível em: <https://doi.org/10.1109/ICECCE52056.2021.9514268>. Nenhuma citação no texto.

RICHARDSON, C. *Microservices patterns*. [S.l.]: Manning Publications Company, 2018. ISBN 9781617294549. Nenhuma citação no texto.

RICHARDSON, C. *Microservices Patterns. With examples in Java.* [S.l.: s.n.], 2019. Nenhuma citação no texto.

ROCHA, D. P. da. *Monólise: Uma Técnica para Decomposição de Aplicações Monolíticas em Microsserviços*. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, São Leopoldo, 2018. Nenhuma citação no texto.

SELLAMI, K. *et al.* Improving microservices extraction using evolutionary search. *Information and Software Technology*, Elsevier B.V., v. 151, 2022. ISSN 09505849. Cited by: 3. Disponível em: <https://doi.org/10.1016/j.infsof.2022.106996>. Nenhuma citação no texto.

SEVOCAB. *IEEE Computer Society. Software and Systems Engineering Vocabulary*. 2023. [On-line]. Disponível em: <https://pascal.computer.org/sev_display/index.action>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

SHARMA. Build domain-driven microservice-based applications with spring, spring cloud, and angular. In: *S. Mastering Microservices with Java 9*. [S.l.]: Packt Publishing Ltd, 2017. Nenhuma citação no texto.

SILVA, C. E. da; JUSTINO, Y. de L.; ADACHI, E. Spread: service-oriented process for reengineering and devops. 2022. Nenhuma citação no texto.

SOLDANI, J.; TAMBURRI, D. A.; HEUVEL, W.-J. V. D. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, v. 170, p. 110708, 2021. Nenhuma citação no texto.

SOLDANI, J.; TAMBURRI, D. A.; Van Den Heuvel, W.-J. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, v. 146, p. 215–232, 2018. ISSN 0164-1212. Nenhuma citação no texto.

SOMMERVILLE, I. Engenharia de software. 10ª edição. Pearson Education do Brasil, 2018. Nenhuma citação no texto.

TAIBI, D.; LENARDUZZI, V. On the definition of microservice bad smells. In: *Proceedings of the International Conference on Software Architecture (ICSA)*. [S.l.]: IEEE, 2018. p. 105–110. Nenhuma citação no texto.

TIOBE. *TIOBE Index for August 2025*. 2025. On-line. Disponível em: <https://www.tiobe.com/tiobe-index>, Acessado em 6 de outubro de 2025. Nenhuma citação no texto.

TOZZI, C. *The 3 pillars of observability: Logs, metrics and traces*. 2022. Available: <https://www.techtarget.com/searchitoperations/tip/The-3-pillars-of-observability-Logs-metrics-and-traces>, Accessed on 6 de outubro de 2025. Nenhuma citação no texto.

TRABELSI, I. *et al.* From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis. *Journal of Software: Evolution and Process*, John Wiley and Sons Ltd, 2022. ISSN 20477481. Cited by: 2. Disponível em: <https://doi.org/10.1002/smr.2503>. Nenhuma citação no texto.

TSERPES. A microservices methodology for the creation of short, fast-paced and stream processing pipelines. In: . [S.l.: s.n.], 2019. p. S2405959519301092. ISBN 24059595. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S2405959519301092>, Accessado em 12 de dezembro de 2023. Nenhuma citação no texto.

WOHLIN, C. *et al. Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN 3642290434. Disponível em: <https://www.springer.com/gp/book/9783642290435>. Acesso em: 05/06/2020. Nenhuma citação no texto.

XIAO; WIJEGUNARATNE; QIANG. Reflections on soa and microservices. In: . [S.l.]: International conference on enterprise systems, 2016. p. 60–67. Nenhuma citação no texto.

ZARAGOZA, P. *et al.* Materializing microservice-oriented architecture from monolithic object-oriented source code. *Communications in Computer and Information Science*, Springer Science and Business Media Deutschland GmbH, v. 1622 CCIS, p. 143 – 168, 2022. ISSN 18650929. Cited by: 1. Disponível em: <https://doi.org/10.1007/978-3-031-11513-4_7>. Nenhuma citação no texto.

ZIMMERMANN, O. *Microservices Tenets – Agile Approaches to Service-Oriented Architecture*. [S.l.]: Springer, 2022. Nenhuma citação no texto.

# APÊNDICE A – PROTOCOLO DE PESQUISA

## A.1 Introdução

Este estudo visa estabelecer uma visão abrangente sobre o projeto de modernização de sistemas legados em Delphi para a arquitetura de microsserviços. Como essa visão pode resultar em modelos e/ou arquiteturas voltados para diferentes domínios de sistemas, uma classificação para obtenção de estudos voltados para modernização ou migração de sistemas legados deve ser realizada. Para isso, foi adotada a técnica de mapeamento sistemático (do inglês, *Systematic Mapping Study – SMS*) baseada no processo proposto por Petersen *et al.* (2008), Petersen, Vakkalanka e Kuzniarz (2015), o qual é composto por cinco passos principais:

- **[Passo 1]** ***Definição das Questões de Pesquisa.*** Neste passo são definidos os objetivos da pesquisa e o protocolo de mapeamento sistemático, que representa um plano pré-determinado que descreve questões de pesquisa, assim como o mapeamento sistemático deve ser conduzido;
- **[Passo 2]** ***Condução da Pesquisa.*** Neste passo os estudos primários são recuperados e identificados;
- **[Passo 3]** ***Triagem dos Artigos***. Neste passo os estudos identificados são avaliados de acordo com os critérios de inclusão e exclusão definidos no protocolo de pesquisa;
- **[Passo 4]** ***Elaboração dos resumos.*** Neste passo os estudos incluídos são analisados e classificados; e
- **[Passo 5]** ***Extração de dados e mapeamento dos estudos.*** Neste passo ocorre a construção da representação final do mapeamento, que deve ser realizada com base no esquema final criado no Passo 4.

Como forma de apresentar os passos supracitados, este documento foi organizado em outras cinco seções. Na Seção A.2 são apresentadas as questões de pesquisa propostas para este SMS. Já na Seção A.3 é reportada a estratégia de pesquisa utilizada neste SMS, sendo destacada a *string* de busca, as bases de pesquisa utilizadas e os critérios de inclusão e exclusão utilizados para selecionar os estudos. Em seguida, na Seção A.4 é mostrado como os dados foram coletados e sintetizados. Na Seção A.5 são reportadas as limitações deste tipo de trabalho. Por

fim, na Seção A.6 é apresentado um breve relato sobre como os resultados devem ser reportados.

## A.2 Questões de pesquisa

O objetivo principal deste SMS é identificar estudos primários voltados para a modernização de sistemas legados para a arquitetura de microsserviços. As Questões de Pesquisa (QP) estabelecidas são:

- **QP 1:** Quais são as características demográficas dos estudos sobre modernização de sistemas legados para o estilo MSA?
  - Quando os estudos são publicados?
  - Onde são publicados os estudos?
  - Como as publicações são distribuídas entre a academia e a indústria?
- **QP 2:** Existem soluções que apoiam a modernização de sistemas legados para o estilo MSA?
  *O objetivo dessa questão é identificar as soluções (ou seja, metodologia, método, processo, abordagem, entre outros) existentes na literatura. Como resultado, essa questão pode apresentar um panorama de soluções voltadas para a modernização de sistemas legados para o estilo MSA.*
  - **QP 2.1:** Quais domínios de aplicação que se beneficiaram com soluções para modernizar sistemas legados para o estilo MSA?
    *O objetivo desta questão é mapear os domínios de aplicação que têm utilizado soluções no projeto de modernização de sistemas legados. Como exemplo, pode-se citar o domínio de sistemas legados voltados para o comércio em geral.*
  - **QP 2.2:** Como essas soluções foram avaliadas?
    *O objetivo dessa questão é entender como essas soluções têm sido avaliadas. Por exemplo: prova de conceito, estudo de caso, experimentos, casos reais em ambientes industriais.*
  - **QP 2.3:** Quais são as evidências que motivam a adoção de soluções de modernização de sistemas legados ao estilo MSA?
    *O objetivo dessa questão é reunir evidências que possam ter norteado a adoção dessas soluções. Por exemplo: nível de maturidade, aplicabilidade na indústria, entre outros.*

  - **QP 2.4:** Quais são os atributos de qualidade abordados no processo de modernização de sistemas legados para o estilo MSA?
    *O objetivo dessa questão é elencar os atributos de qualidade utilizados no projeto dessas soluções. Exemplos: confiabilidade, disponibilidade, tempo de resposta, etc.*

- **QP 3:** Quais são as atividades executadas/conduzidas durante o processo de modernização?
  *O objetivo dessa questão é entender quais são os passos que devem ser executados. Além disso, também é interesse entender se esses passos são executados manualmente ou apoiados por processos automatizados.*

- **QP 4:** Como os microsserviços têm sido organizados em uma aplicação baseada em microsserviços durante a modernização?
  *O objetivo dessa questão é entender se a nova aplicação contempla novas features, além de outras características inerentes ao estilo arquitetural de microsserviços.*

## A.3 Estratégia de pesquisa

A estratégia de pesquisa foi definida com base nas QPs apresentadas na Seção A.2. Essa estratégia é composta por critério de seleção das bases de pesquisa, lista de bases pesquisadas, idioma do estudo e, palavras-chave e seus termos relacionados. A seguir, detalhes dessa estratégia são apresentados:

1. **Critério de seleção.** Os seguintes critérios foram utilizados para selecionar as bases de pesquisa deste mapeamento (DIESTE; GRIMáN; JURISTO, 2009): (i) frequência de atualização de conteúdo; (ii) disponibilidade de textos na íntegra; (iii) qualidade dos resultados apresentados pela base; e (iv) versatilidade para exportar os dados obtidos;

2. **Lista de bases pesquisadas.** As bases de pesquisa selecionadas para o mapeamento são apresentadas na Tabela 6. Segundo Kitchenham e Charters (2007) e Petersen, Vakkalanka e Kuzniarz (2015), tais bases atendem aos critérios de seleção de bases de pesquisa supracitados e são consideradas suficientes para a condução desse tipo de estudo para a área de Engenharia de Software.

3. **Idioma dos estudos.** Apenas estudos primários escritos em inglês devem ser considerados neste SMS. Essa restrição deve-se ao fato desse idioma ser o mais comum em publicações científicas, além de auxiliar em futuras replicações deste trabalho.

Tabela 6 – Lista de bases de pesquisa

| **Base de pesquisa** | **Endereço** |
|---|---|
| ACM Digital Library | <dl.acm.org> |
| IEEE Xplore | <www.ieeexplore.ieee.org> |
| ScienceDirect | <www.sciencedirect.com> |
| Scopus | <www.scopus.com> |

Fonte: Elaborada pelo autor.

4. **Palavras-chave:** *microservice*, *modernization*, com os seguintes sinônimos:

   a) ***microservice:*** *microservice, microservices, micro service, micro services, micro-service, micro-services*;

   b) ***modernização:*** *decomposing, decoupling, migrating, migration, modernization, modernizing, re engineering, re-engineering, reengineering, reverse engineering, transition.*

5. ***String* de pesquisa:** O operador booleano `OR` foi utilizado para relacionar os termos principais e seus respectivos sinônimos. Todos esses termos foram combinados utilizando o operador booleano `AND`. Portanto, a *string* de busca final é apresentada na Tabela 7.

Tabela 7 – *String* de pesquisa

```
("decomposing" OR "decoupling" OR "migrating" OR "migration" OR
    "modernization" OR "modernizing" OR "re engineering" OR
"re-engineering" OR "reengineering" OR "reverse engineering" OR
                        "transition")
                          AND
   ("micro service" OR "micro services" OR "micro-service" OR
    "micro-services" OR "microservice" OR "microservices")
```

Fonte: Elaborada pelo autor.

Definir os critérios de inclusão (do inglês, *Inclusion Criteria* – IC) e critérios de exclusão (do inglês, *Exclusion Criteria* – EC) para um mapeamento é um importante elemento para condução e execução do mesmo. A partir da definição desses critérios é possível incluir estudos primários relevantes para responder às QPs e excluir estudos que não permitem extrair respostas para as mesmas. Para este SMS, o seguinte IC foi definido:

- **IC 1:** O estudo apresenta uma solução para apoiar a modernização/migração de sistemas legados para a arquitetura de microsserviços.

Em contrapartida, os ECs definidos para este mapeamento foram:

- **EC 1:** O estudo não aborda a modernização de sistemas legados para a arquitetura de microsserviços;
- **EC 2:** O estudo é um editorial, artigo de opinião, palestra, opinião, tutorial, pôster ou painel;
- **EC 3:** O estudo foi escrito em um idioma diferente do definido pela pesquisa;
- **EC 4:** O estudo possui apenas um resumo ou não está disponível para ser lido na íntegra;
- **EC 5:** O estudo é um trabalho desenvolvido anteriormente pelo mesmo autor. Nesse caso, apenas a versão mais recente do estudo ou a mais completa será considerada.
- **EC 6:** O estudo é um trabalho secundário.

Após a definição dos critérios de seleção, a *string* de busca (veja Tabela 7) foi customizada e executada em cada uma das bases de publicação (veja Tabela 6), sendo explorados os seguintes campos de busca: título, resumo e palavras-chave. Conforme apresentado na Tabela 8, a "Etapa 1" representa o número de estudos resultantes para cada meio de publicação, sendo totalizado em 1.037 estudos. Em seguida, "Etapa 2", os estudos obtidos em cada base foram unificados em um arquivo e as seguintes atividades foram conduzidas: remoção dos estudos duplicados; e remoção de trabalhos que não foram considerados estudos primários. Essa etapa resultou em 639 estudos, que, a partir do conjunto de estudos primários selecionados, o título e o resumo de cada estudo foram lidos e avaliados de acordo com critérios de seleção. Quando necessário, a introdução e a conclusão também foram lidas. Dúvidas na aplicação dos critérios foram resolvidas por meio de mediações com o orientador deste projeto de mestrado acadêmico. Após a aplicação dos critérios de seleção, 98 estudos resultaram na "Etapa 3". Por fim, "Etapa 4", optou-se por aplicar critérios de qualidade para seleção final de estudos por dois motivos: número elevado de estudos e melhoria da qualidade do trabalho. Para isso, oito critérios de qualidade foram definidos, como apresentado na Tabela 9. Após classificar os 98 estudos com base em tais critérios, a lista final de estudos para esta última etapa (Etapa 4) teve como limiar de seleção todos os estudos com percentual acima de 80% da pontuação total, resultando em 43 estudos.

Tabela 8 – Lista de bases de pesquisa

| Base de pesquisa | Etapa 1 | Etapa 2 | Etapa 3 | Etapa 4 |
|---|---|---|---|---|
| ACM Digital Library | 83 | 639 | 98 | 43 |
| IEEE Xplore | 268 | | | |
| ScienceDirect | 58 | | | |
| Scopus | 629 | | | |
| TOTAL | 1.037 | | | |

Fonte: Elaborada pelo autor.

Tabela 9 – Critérios de Qualidade

| ID | Descrição | Pontuação |
|---|---|---|
| CQ1 | Os autores declaram claramente os objetivos da pesquisa? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ2 | Os autores discutem as limitações de seu estudo? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ3 | Os autores apresentam as descobertas claramente? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ4 | Existem evidências de que os resultados do estudo podem ser usados por outros pesquisadores/praticantes? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ5 | O corpo do estudo atende integralmente às questões previstas no resumo? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ6 | As propostas descritas foram avaliadas empiricamente? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |
| CQ7 | Como o estudo foi avaliado ? | 0: Não avaliado;<br>0.5: Exemplo ilustrativo;<br>1: Estudo de caso;<br>1.5: Avaliação por pesquisadores e especialistas;<br>2: Aplicação na indústria. |
| CQ8 | O estudo apresenta ideias para investigação futura? | 0: Não;<br>1: Parcialmente;<br>2: Sim. |

Fonte: Elaborada pelo autor.

## A.4 Extração e síntese de dados

Esta fase resultará em um conjunto de dados que descreve cada estudo primário identificado na Seção A.3. Conforme sugerido por Petersen, Vakkalanka e Kuzniarz (2015), um formulário para cada estudo primário deve ser preenchido, visando facilitar as tarefas de extração e síntese de dados. Com base nos campos propostos pelos autores supracitados, um formulário específico para este mapeamento foi elaborado, como pode ser visualizado na Tabela 10. Esses tópicos de pesquisa desse formulário serão utilizados posteriormente para responder às questões de pesquisa estabelecidas na Seção A.2.

Tabela 10 – Formulário de extração de dados

| **Tópico de Pesquisa** | **Valor** | **QP** |
|---|---|---|
| ID do estudo | Valor inteiro | — |
| Ano de publicação | Ano | QP1 |
| Local de publicação | Nome do local da publicação | QP1 |
| Tipo da publicação | Academia ou Indústria | QP1 |
| Tipo da solução | Metodologia, método, processo ou abordagem | QP2 |
| Domínio de aplicação | Domínio em que modelo/arquitetura foi projetado(a) | QP2.1 |
| Avaliação da solução | Prova de conceito, estudo de caso, experimentos e casos reais na industria | QP2.2 |
| Evidências de adoção | Nível de maturidade, aplicabilidade na indústria e ganhos/benefícios | QP2.3 |
| Atributos de qualidade | Confiabilidade, disponibilidade e tempo de resposta | QP2.4 |
| Atividades executadas | Planejamento, análise, decomposição, desenvolvimento, integração e monitoramento | QP3 |
| Organização microsserviços | APIs, containers, mensageria e novas features | QP4 |

Fonte: Elaborada pelo autor.

## A.5 Limitações

Nesta seção são apresentadas as possíveis limitações que um trabalho dessa natureza (SMS) pode apresentar, a saber:

1. **Omissão de artigos.** O mapeamento utilizará um processo de pesquisa automatizado, sendo a *string* de busca adaptada e processada pelo mecanismo de busca/pesquisa de cada base. Conforme reportado na Seção A.3, quatro bases de pesquisa para a obtenção das

publicações serão utilizadas e, apesar dos esforços em tornar este trabalho mais confiável, não é possível afirmar que todos os estudos primários relacionados à temática deste SMS poderão ser recuperados. Entretanto, com o intuito de calibrar a *string* final, pesquisas com diferentes variações deverão ser realizadas para validar se estudos que deveriam ter sido recuperados foram retornados. O caso oposto também deve ser validado, ou seja, estudos que não devem ser recuperados. Em seguida, a *string* final será posteriormente adaptada para os padrões de cada uma das bases de pesquisa. Portanto, acredita-se que dessa maneira nenhum estudo relevante seja excluído do mapeamento.

2. **Confiabilidade dos pesquisadores.** Todos os envolvidos no mapeamento são pesquisadores da área de Engenharia de Software. Nenhum dos estudos incluídos neste mapeamento são de autoria dos membros do grupo de pesquisa ou de pesquisadores relacionados com os autores. Sendo assim, acredita-se que os resultados apresentados neste SMS devam apresentar o atual estado da arte da área pesquisada sem a introdução involuntária de viés.

3. **Parcialidade.** Embora um conjunto de diretrizes seja definido com a finalidade de garantir que o mapeamento seja imparcial, não é possível garantir que toda a informação coletada seja completamente imparcial, pois está sujeita a interpretação da pessoa que está conduzindo o mapeamento.

4. **Extração de dados.** Outra eventual limitação deste mapeamento está relacionada ao processo de extração de dados. As informações utilizadas para responder as QPs podem nem sempre estar presentes de maneira explícita no texto e algumas informações devem ser interpretadas. Entretanto, quando houver dúvidas/divergências a respeito de um estudo, o mesmo deve ser discutido entre os pesquisadores envolvidos a fim de estabelecer um consenso quanto ao critério a ser aplicado e/ou dado a ser extraído. Adicionalmente, vale destacar que um formulário de pesquisa foi adotado com o objetivo de auxiliar na tarefa de coleta e síntese dos dados, produzindo um arquivo padronizado de dados.

5. **Avaliação da qualidade.** Como o objetivo deste mapeamento é identificar soluções para o processo de modernização de sistemas legados para a arquitetura de microsserviços, a qualidade dos estudos não será avaliada neste SMS. Entretanto, deve-se reconhecer a importância da avaliação de qualidade e esse passo poderá ser considerado em uma futura versão deste mapeamento.

6. **Replicabilidade.** Em função das bases de pesquisa utilizadas estarem disponíveis em ambiente virtual e serem constantemente atualizadas, é possível que em uma futura pesquisa os resultados não sejam exatamente iguais, o que poderia dificultar a reprodução dos dados coletados a partir deste SMS.

## A.6 Síntese dos resultados

Ao reportar os resultados deste mapeamento, as seguintes atividades devem ser realizadas: (i) discussão sobre os resultados (ou seja, descrição de cada estudo primário e detalhes de qualquer meta-análise que tenha sido realizada); (ii) discussão das principais descobertas, forças e fraquezas do mapeamento e os significados dessas descobertas como, por exemplo, aplicabilidade, benefícios, efeitos adversos e riscos; (iii) conclusão, incluindo recomendações como, por exemplo, possíveis implicações práticas para o desenvolvimento de software e para a área de pesquisa e questões de pesquisa não respondidas; e, por fim, (iv) elaboração de uma discussão sobre as limitações do mapeamento.

# APÊNDICE B – Lista de Estudos Primários

Na Tabela 11 estão listados os estudos primários incluídos no SMS descrito no Apêndice A. Na coluna **ID** são apresentados os identificadores dos estudos. Na coluna **Referência** são exibidos o(s) autor(es) e o título com um link externo para a referência *on-line* de cada estudo. Por fim, na coluna **Ano** são apresentados os anos de publicação dos estudos.

Tabela 11 – Lista final de estudos selecionados para a extração e síntese de dados

| ID | Referência | Ano |
|---|---|---|
| S1 | Buchgeher, Georg and Ramlerf, Rudolf and Stummer, Heinz and Kaufmann, Hannes, Adopting Microservices for Industrial Control Systems: A Five Step Migration Path. | 2021 |
| S2 | Lenarduzzi, Valentina and Lomio, Francesco and Saarimäki, Nyyti and Taibi, Davide, Does migrating a monolithic system to microservices decrease the technical debt?. | 2020 |
| S3 | Trabelsi, Imen and Abdellatif, Manel and Abubaker, Abdalgader and Moha, Naouel and Mosser, Sébastien and Ebrahimi-Kahou, Samira and Guéhéneuc, Yann-Gaël, From legacy to microservices: A type-based approach for microservices identification using machine learning and semantic analysis. | 2022 |
| S4 | Sellami, Khaled and Ouni, Ali and Saied, Mohamed Aymen and Bouktif, Salah and Mkaouer, Mohamed Wiem, Improving microservices extraction using evolutionary search. | 2022 |
| S5 | Krause, Alexander and Zirkelbach, Christian and Hasselbring, Wilhelm and Lenga, Stephan and Kroger, Dan, Microservice Decomposition via Static and Dynamic Analysis of the Monolith. | 2020 |
| S6 | Mazzara, Manuel and Dragoni, Nicola and Bucchiarone, Antonio and Giaretta, Alberto and Larsen, Stephan T. and Dustdar, Schahram, Microservices: Migration of a Mission Critical System. | 2021 |

*Continua na próxima página*

*Continuando da página anterior*

| ID | Referência | Ano |
|---|---|---|
| S7 | Freire, Augusto Flávio A. A. and Sampaio, Américo Falcone and Carvalho, Luis Heustakio L. and Medeiros, Otávio and Mendonça, Nabor C., Migrating production monolithic systems to microservices using aspect oriented programming. | 2021 |
| S8 | Kalia, Anup K. and Xiao, Jin and Krishna, Rahul and Sinha, Saurabh and Vukovic, Maja and Banerjee, Debasish, Mono2Micro: A practical and effective tool for decomposing monolithic Java applications to microservices. | 2021 |
| S9 | Wang, Yu-Te and Ma, Shang-Pin and Lai, Yue-Jun and Liang, Yan-Cih, Qualitative and quantitative comparison of Spring Cloud and Kubernetes in migrating from a monolithic to a microservice architecture. | 2023 |
| S10 | Zaragoza, Pascal and Seriai, Abdelhak-Djamel and Seriai, Abderrahmane and Bouziane, Hinde-Lilia and Shatnawi, Anas and Derras, Mustapha, Refactoring monolithic object-oriented source code to materialize microservice-oriented architecture. | 2021 |
| S11 | Löhnertz, Jakob and Oprescu, Ana Maria, Steinmetz: Toward automatic decomposition of monolithic software into microservices. | 2020 |
| S12 | Mishra, Ridhima and Jaiswal, Nishtha and Prakash, Rishi and Barwal, Paras Nath, Transition from Monolithic to Microservices Architecture: Need and proposed pipeline. | 2022 |
| S13 | Dehghani, MohammadHadi and Kolahdouz-Rahimi, Shekoufeh and Tisi, Massimo and Tamzalit, Dalila, Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. | 2022 |
| S14 | Osman, Mohd Hafeez and Saadbouh, Cheikh and Sharif, Khaironi Yatim and Admodisastro, Novia and Basri, Muhammad Hadri, From Monolith to Microservices: A Semi-Automated Approach for Legacy to Modern Architecture Transition using Static Analysis. | 2022 |
| S15 | Colanzi, Thelma and Amaral, Aline and Assunção, Wesley and Zavadski, Arthur and Tanno, Douglas and Garcia, Alessandro and Lucena, Carlos, Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices. | 2021 |
| S16 | Auer, Florian and Lenarduzzi, Valentina and Felderer, Michael and Taibi, Davide, From monolithic systems to Microservices: An assessment framework. | 2021 |

*Continua na próxima página*

*Continuando da página anterior*

| ID | Referência | Ano |
|---|---|---|
| S17 | Zaragoza, Pascal and Seriai, Abdelhak-Djamel and Seriai, Abderrahmane and Shatnawi, Anas and Bouziane, Hinde-Lilia and Derras, Mustapha, Materializing Microservice-oriented Architecture from Monolithic Object-oriented Source Code. | 2022 |
| S18 | Kyryk, Marian and Tymchenko, Oleksandr and Pleskanka, Nazar and Pleskanka, Mariana, Methods and process of service migration from monolithic architecture to microservices. | 2022 |
| S19 | Stranner, Heimo and Strobl, Stefan and Bernhart, Mario and Grechenig, Thomas, Microservice decompositon: A case study of a large industrial software migration in the automotive industry. | 2020 |
| S20 | Ma, Shang-Pin and Li, Chia-Yu and Lee, Wen-Tin and Lee, Shin-Jie, Microservice Migration Using Strangler Fig Pattern and Domain-Driven Design. | 2022 |
| S21 | Li, Chia-Yu and Ma, Shang-Pin and Lu, Tsung-Wen, Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System. | 2020 |
| S22 | Ma, Shang-Pin and Lu, Tsung-Wen and Li, Chung-Chieh, Migrating Monoliths to Microservices based on the Analysis of Database Access Requests. | 2022 |
| S23 | Salii, Sh. and Ajdari, J. and Zenuni, Xh., Migrating to a microservice architecture: benefits and challenges. | 2023 |
| S24 | Carvalho, Luiz and Garcia, Alessandro and Colanzi, Thelma Elita and Assuncao, Wesley K. G. and Pereira, Juliana Alves and Fonseca, Baldoino and Ribeiro, Marcio and De Lima, Maria Julia and Lucena, Carlos, On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices. | 2020 |
| S25 | Gomes Barbosa, Marx Haron and Maia, Paulo Henrique M., Towards Identifying Microservice Candidates from Business Rules Implemented in Stored Procedures. | 2020 |
| S26 | Bamberger, Burkhard and Körber, Bastian, Migrating Monoliths to Microservices Integrating Robotic Process Automation into the Migration Approach. | 2022 |

*Continua na próxima página*

*Continuando da página anterior*

| ID | Referência | Ano |
|---|---|---|
| S27 | Parikh, Aman and Kumar, Pranav and Gandhi, Parshav and Sisodia, Jignesh, Monolithic to Microservices Architecture - A Framework for Design and Implementation. | 2022 |
| S28 | Kuryazov, Dilshodbek and Jabborov, Dilshod and Khujamuratov, Bekmurod, Towards Decomposing Monolithic Applications into Microservices. | 2020 |
| S29 | Freitas, Francisco and Ferreira, André and Cunha, Jácome, A methodology for refactoring ORM-based monolithic web applications into microservices. | 2023 |
| S30 | Yang, Zhumei and Wu, Sijie and Zhang, Cheng, A Microservices Identification Approach based on Problem Frames. | 2022 |
| S31 | Kazanavičius, Justas and Mažeika, Dalius, An Approach to Migrate from Legacy Monolithic Application into Microservice Architecture. | 2023 |
| S32 | Volynsky, Evgeny and Mehmed, Merlin and Krusche, Stephan, Architect: A Framework for the Migration to Microservices. | 2022 |
| S33 | Laigner, Rodrigo and Kalinowski, Marcos and Diniz, Pedro and Barros, Leonardo and Cassino, Carlos and Lemos, Melissa and Arruda, Darlan and Lifschitz, Sergio and Zhou, Yongluan, From a Monolithic Big Data System to a Microservices Event-Driven Architecture. | 2020 |
| S34 | Santos, Ana and Paula, Hugo, Microservice decomposition and evaluation using dependency graph and silhouette coefficient. | 2021 |
| S35 | Prasandy, Teguh and Titan and Murad, Dina Fitria and Darwis, Taufik, Migrating application from monolith to microservices. | 2020 |
| S36 | Haugeland, Sindre Gronstol and Nguyen, Phu H. and Song, Hui and Chauvel, Franck, Migrating Monoliths to Microservices-based Customizable Multi-tenant Cloud-native Apps. | 2021 |
| S37 | Goncalves, Nuno and Faustino, Diogo and Silva, Antonio Rito and Portela, Manuel, Monolith Modularization towards Microservices: Refactoring and Performance Trade-offs. | 2021 |
| S38 | Preti, Joao Paulo Delgado and Souza, Adriano Neres Araujo and Freiberger, Evandro Cesar and De Almeida Lacerda, Tiago, Monolithic to Microservices Migration Strategy in Public Safety Secretariat of Mato Grosso. | 2021 |

*Continua na próxima página*

*Continuando da página anterior*

| ID | Referência | Ano |
|---|---|---|
| S39 | Vera-Baquero, Alejandro and Phelan, Owen and Slowinski, Pawel and Hannon, John, Open Source Software as the Main Driver for Evolving Software Systems Toward a Distributed and Performant E-Commerce Platform: A Zalando Fashion Store Case Study. | 2021 |
| S40 | Bajaj, Deepali and Bharti, Urmil and Goel, Anita and Gupta, S.C., Partial Migration for Re-architecting a Cloud Native Monolithic Application into Microservices and FaaS. | 2020 |
| S41 | Michael Ayas, Hamdy and Leitner, Philipp and Hebig, Regina, The Migration Journey Towards Microservices. | 2021 |
| S42 | Batista, Cesar and Proenca, Bruno and Cavalcante, Everton and Batista, Thais and Morais, Felipe and Medeiros, Henrique, Towards a Multi-Tenant Microservice Architecture: An Industrial Experience. | 2022 |
| S43 | Bandara, Chamika and Perera, Indika, Transforming monolithic systems to microservices - An analysis toolkit for legacy code evaluation. | 2020 |

Fonte: Elaborada pelo autor.

# APÊNDICE C – Questionário

O questionário está organizado em três seções, contendo 60 questões. A primeira seção é voltada para a detecção de parâmetros relacionados ao perfil do profissional, a segunda voltada para os interesses de modernização e, por fim, a terceira voltada para cada atividade do processo.

## C.1 Informação básica

1. Qual é o seu papel na empresa?
    a) Analista de negócios
    b) Arquiteto
    c) Designer
    d) Desenvolvedor
    e) Outro
2. Quantos anos de experiência profissional?
    a) Menos de 2 anos
    b) De 2 a 3 anos
    c) De 3 a 5 anos
    d) De 5 a 10 anos
    e) Mais de 10 anos
3. Qual é sua experiência com a arquitetura de microsserviços?
    a) Não utilizo
    b) Cerca de 1 ano
    c) De 2 a 3 anos
    d) De 3 a 5 anos
    e) Mais de 5 anos
4. Qual é o tipo de informação existente do sistema legado
    a) Sistema em binário
    b) Código fonte sem documentação

c) Código fonte documentado

d) Código fonte + documentação

e) Código fonte + documentação + modelos

## C.2 Motivação para modernização

5. O sistema legado envolve um processo de negócio complexo?

   a) Concordo totalmente

   b) Concordo

   c) Neutro

   d) Discordo

   e) Discordo totalmente

6. Existem questões técnicas relativamente complexas no sistema legado?

   a) Concordo totalmente

   b) Concordo

   c) Neutro

   d) Discordo

   e) Discordo totalmente

7. O estágio atual do sistema legado tem dificultado a implementação de novas features no sistema?

   a) Concordo totalmente

   b) Concordo

   c) Neutro

   d) Discordo

   e) Discordo totalmente

8. O estágio atual do sistema legado tem dificultado a expansão dos negócios da empresa?

   a) Concordo totalmente

   b) Concordo

   c) Neutro

   d) Discordo

   e) Discordo totalmente

## C.3 Atividade 1 - Planejamento

9. Os passos da **atividade de Planejamento** estão claramente definidos?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

10. Os **objetivos do projeto** estão claramente definidos e documentados?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

11. No passo **Definir Escopo** fica claro o que precisa ser feito para iniciar o Projeto?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

12. O passo **Estruturar a Equipe** abrange totalmente as práticas necessárias, os papéis e as responsabilidades de cada membro?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

13. O passo **Cultura Organizacional** é essencial para o Projeto?

    a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

14. O passo **Capacitação e Treinamentos** reporta todas as orientações que uma equipe precisa ter e desenvolver para o sucesso do Projeto?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

15. No passo **Definir Tecnologias** estão todas as considerações essenciais que uma equipe precisa seguir para trabalhar com um sistema baseado na arquitetura de microsserviços?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

16. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Planejamento**?

## C.4 Atividade 2 - Análise

17. Os passos da **atividade de Análise** estão claramente definidos?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

18. No passo **Analisar o Modelo de Negócio** fica claro o que precisa ser investigado sobre o sistema e a organização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

19. O passo **Analisar o Sistema Legado** define corretamente o objetivo de avaliar o sistema legado para o processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

20. O passo de **Analisar o Banco de Dados** é essencial para definir uma estratégia para reestruturar o banco de dados?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

21. O passo **Realizar Pesquisas UX/UI** é essencial para identificação de candidatos a microsserviços?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

22. O passo **Identificar Principais Funcionalidades** é essencial para o sucesso do Projeto?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

23. No passo **Modelar Estrutura Tecnológica** estão todas as orientações que uma equipe precisa desenvolver para um bom planejamento da nova arquitetura?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

24. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Análise**?

## C.5 Atividade 3 - Decomposição

25. Os passos da **atividade de Decomposição** estão claramente definidos?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

26. No passo **Identificar Funcionalidades Candidatas** a microsserviços fica claro o objetivo e quais caminhos devem ser seguidos?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

27. O passo **Definir Design da Decomposição** é essencial para o processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

28. O passo de **Definir Reestruturação dos Dados** é essencial para o processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

29. No passo **Modelar APIs dos Microsserviços**, as práticas recomendadas são suficientes para entender e executar as operações?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

30. O passo **Priorizar Microsserviços** a serem desenvolvidos é essencial para o sucesso do Projeto?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

31. O passo **Definir Padrão Arquitetural** consegue demonstrar como funciona uma arquitetura de microsserviços?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

32. O passo **Definir Especificação Técnica** é essencial para definir como as equipes irão construir os microsserviços?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

33. O passo **QA de Software e Métricas** é importante para o sucesso do projeto?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

34. O passo **Elaborar Documentação** é essencial para o desenvolvimento e acompanhamento do projeto?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

35. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Decomposição**?

## C.6 Atividade 4 - Desenvolvimento

36. Os passos da **atividade de Desenvolvimento** estão claramente definidos?

    a) Concordo totalmente

    b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

37. No passo **Reestruturar o Repositório de Dados** fica claro o objetivo e a importância para o desenvolvimento de microsserviços?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

38. O passo **Definir Casos de Teste** é essencial para o processo de modernização e o correto funcionamento do sistema?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

39. O passo de **Definir Reestruturação dos Dados** é essencial para o processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

40. No passo **Implementar Microsserviços**, as práticas recomendadas são suficientes para entender e executar as operações de acordo com padrões de projetos?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

41. No passo **Implementar Comunicação entre Microsserviços** as informações são suficientes para entender a importância para o correto funcionamento e sucesso do Projeto?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

42. O passo **Definir Controle de Versão** é essencial para o desenvolvimento de microsserviços?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

43. O passo **Migrar Banco de Dados Legado** contém todas as informações necessárias para que as equipes consigam construir os microsserviços com bases independentes e alimentá-las corretamente?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

44. O passo **Migrar Monolítico para Microsserviços** consegue abordar a real importância de uma migração bem sucedida para o sucesso do projeto?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

45. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Desenvolvimento**?

## C.7 Atividade 5 - Integração

46. Os passos da **atividade de Integração** estão claramente definidos?

    a) Concordo totalmente
    b) Concordo
    c) Neutro
    d) Discordo
    e) Discordo totalmente

47. No passo **Definir malha de Microsserviços e Suporte** fica claro o objetivo e a importância para o desenvolvimento de microsserviços?

    a) Concordo totalmente
    b) Concordo
    c) Neutro
    d) Discordo
    e) Discordo totalmente

48. O passo **Garantir Integração Contínua** é essencial para o processo de modernização e o correto funcionamento do sistema?

    a) Concordo totalmente
    b) Concordo
    c) Neutro
    d) Discordo
    e) Discordo totalmente

49. O passo **Garantir Entrega Contínua** é importante para o processo de modernização e desenvolvimento de software?

    a) Concordo totalmente
    b) Concordo
    c) Neutro

d) Discordo

e) Discordo totalmente

50. No passo **Garantir Deployment Contínuo**, as práticas recomendadas são importantes e fazem sentido para o processo de modernização?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

51. O passo **Automatizar Testes de Unidade e Integração** é essencial e importante para garantir a qualidade do software?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

52. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Integração**?

## C.8 Atividade 6 - Monitoramento

53. Os passos da **atividade de Monitoramento** estão claramente definidos?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

54. O passo **Elaborar Dashboard** é importante para o sucesso do processo de modernização?

    a) Concordo totalmente

    b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

55. O passo **Emitir Alertas** é essencial para o acompanhamento e detecção de problemas para garantir o correto funcionamento dos microsserviços?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

56. No passo **Efetuar Manutenção** as práticas recomendadas são importantes e fazem sentido para o sucesso do processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

57. O passo **Permitir Reversão** é essencial para garantir confiabilidade e segurança para lidar com falhas no processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

58. No passo **Diagnóstico** as práticas recomendadas são importantes e fazem sentido para o sucesso do processo de modernização?

a) Concordo totalmente

b) Concordo

c) Neutro

d) Discordo

e) Discordo totalmente

59. No passo **Lidar com Inconsistências** fica claro a importância do gerenciamento de microsserviços e as práticas que devem ser seguidas para o sucesso do processo de modernização?

    a) Concordo totalmente

    b) Concordo

    c) Neutro

    d) Discordo

    e) Discordo totalmente

60. Na sua opinião quais são as sugestões de mudanças ou melhorias para a atividade de **Monitoramento**?

# DADOS CURRICULARES

| **IDENTIFICAÇÃO** | |
|---|---|
| **Nome** | Lucas Fernando Fávero |
| **Data de nascimento** | 20/09/1985 |
| **Nacionalidade** | Brasileira |
| **Nome em citações bibliográficas** | Fávero, L. F. |
| **Currículo Lattes** | http://lattes.cnpq.br/0548860613594021 |
| **ORCID** | https://orcid.org/0009-0005-8797-8152 |
| **FORMAÇÃO ACADÊMICA** | |
| **2003-2007** | Graduação em Processamento de Dados - Graduado. Faculdade de Tecnologia (Fatec), Taquaritinga. |
| **2008-2009** | Pós Graduação em Engenharia de Sistemas - Pós Graduado. Escola Superior Aberta do Brasil (Esab), Ribeirão Preto. |
| **2013-2014** | Pós Graduação em Gerenciamento de Projetos Práticas do PMI - Pós Graduado. Serviço Nacional de Aprendizagem Comercial (Senac), São José do Rio Preto. |
| **2022-2025** | Mestrado em Ciência da Computação - Mestre. Universidade Estadual Paulista "Júlio de Mesquita Filho" (Unesp), Rio Claro. |
| **PRODUÇÃO BIBLIOGRÁFICA** | |
| Fávero, L. F., Mário, G. S. and Affonso, F. (2025). Micro4Delphi: A Process for the Modernization of Legacy Systems in Delphi to Microservice Architecture. *In:* Proceedings of the 27th International Conference on Enterprise Information Systems - Volume 2: ICEIS; ISBN 978-989-758-749-8; ISSN 2184-4992, SciTePress, pages 328-339. DOI: 10.5220/0013434400003929. | |
| Fávero, L. F., Almeida, N. R. d., and Affonso, F. J. (2025). A Systematic Mapping Study on the Modernization of Legacy Systems to Microservice Architecture. Applied System Innovation, 8(4), 86. https://doi.org/10.3390/asi8040086. | |
| Almeida, N. R. d., Fávero, L. F., and Affonso, F. J. (2025). From Domain-Driven Design To Microservice Architecture: A Systematic Mapping Focused On Modernization of Legacy Systems. *In:* 20th Iberian Conference on Information Systems and Technologies, pages 1-12. Artigo aceito e apresentado na conferência. | |