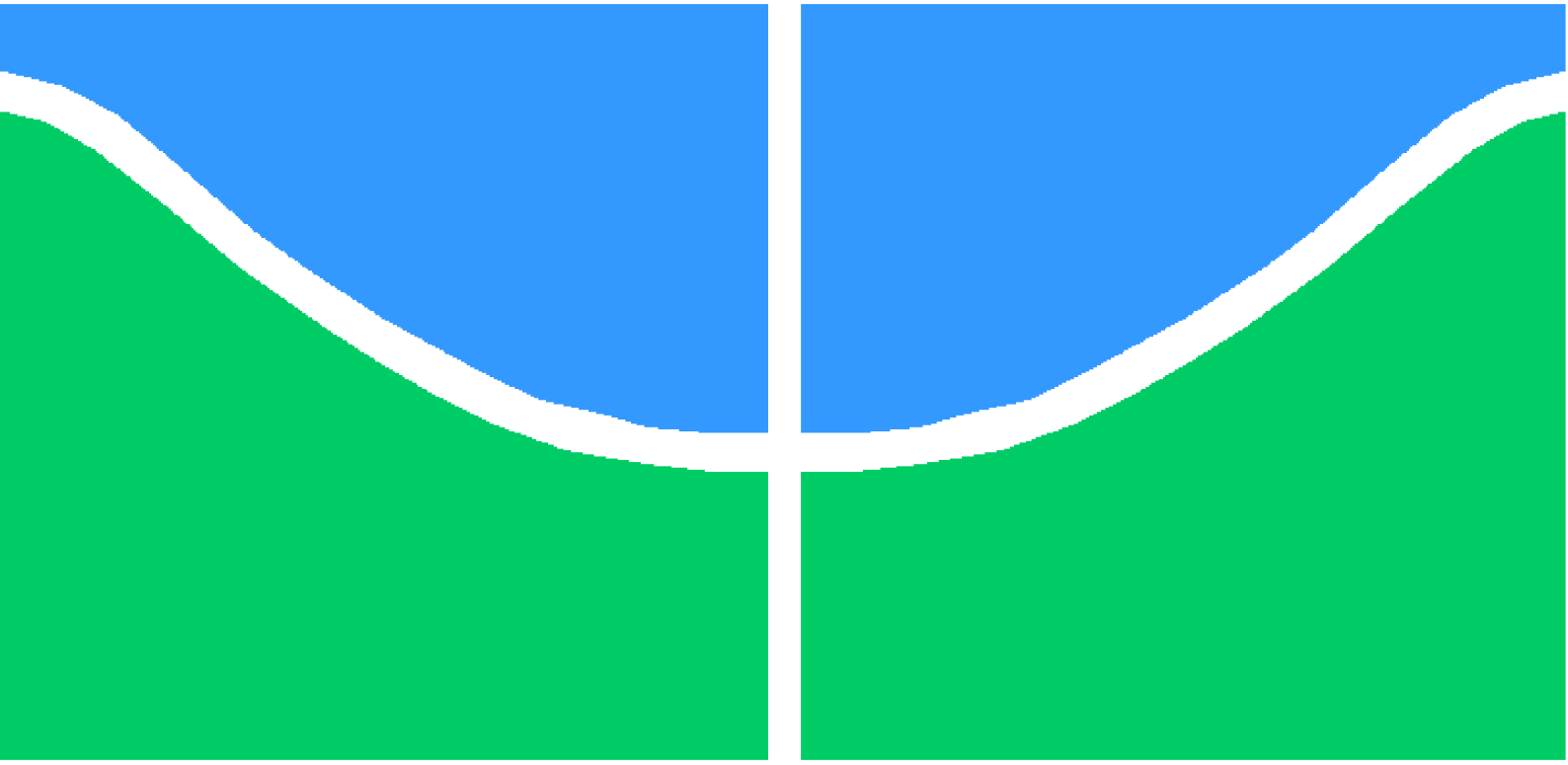


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços

Autor: Mateus Brandão Teixeira
Felipe Correa Andrade

Orientador: Prof. Dr. Maurício Serrano

Coorientador: Profa. Dra. Milene Serrano

Brasília, DF

2024

Mateus Brandão Teixeira
Felipe Correa Andrade

Um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Maurício Serrano

Coorientador: Profa. Dra. Milene Serrano

Brasília, DF

2024

Mateus Brandão Teixeira
Felipe Correa Andrade
Um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços/ Mateus Brandão Teixeira
Felipe Correa Andrade. – Brasília, DF, 2024-
107 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Maurício Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2024.

1. Microsserviços. 2. Catálogo. I. Prof. Dr. Maurício Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços

CDU 02:141:005.6

Mateus Brandão Teixeira
Felipe Correa Andrade

Um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 12/07/2024:

Prof. Dr. Maurício Serrano
Orientador

Profa. Dra. Milene Serrano
Coorientador

Prof. Dr. Fernando Willian Cruz
Examinador 1

Gabriel Davi Silva Pereira
Examinador 2

Brasília, DF
2024

Agradecimentos

Com profunda gratidão, elevamos nossos primeiros agradecimentos a Deus, fonte de nossa perseverança e inspiração. É em Sua presença que encontramos a força necessária para superar os desafios e as adversidades enfrentadas ao longo deste percurso. Estendemos nossa sincera gratidão às nossas famílias, cujo apoio incondicional, incentivo e motivação foram a base de nossa jornada. Cada membro da família foi um alicerce, oferecendo-nos o suporte emocional e a compreensão essenciais para a realização deste trabalho e para a concretização de nossas aspirações.

Expressamos nossa estima e reconhecimento aos professores da instituição, cuja dedicação ao ensino e à transmissão do conhecimento foram pilares de nossa formação acadêmica e profissional. Em particular, nosso profundo agradecimento aos professores Maurício Serrano e Milene Serrano, nossos orientadores, cuja orientação, paciência e comprometimento foram decisivos para a realização deste trabalho. A todos que, direta ou indiretamente, contribuíram para nossa formação e para a conclusão deste trabalho, nosso sincero muito obrigado. Vocês foram parte essencial desta conquista e compartilham conosco o sucesso deste momento.

"Ainda que eu ande pelo vale da sombra da morte,
não temerei mal algum,
pois tu estás comigo."
(Bíblia Sagrada, Salmos 23, 4)

Resumo

Este trabalho de conclusão de curso (TCC) aborda o desafio de implementar microsserviços eficientes e resilientes. O foco deste TCC é o desenvolvimento de um catálogo abrangente de boas práticas, meticulosamente compilado para guiar desenvolvedores através das complexidades inerentes à arquitetura de microsserviços. Com o aumento da complexidade dos sistemas de software e a demanda por entrega rápida e contínua, o projeto se mostra de grande ajuda para equipes que buscam otimizar seus processos de desenvolvimento. O catálogo proposto surge como um recurso interessante, oferecendo orientações baseadas em evidências e exemplos práticos. Para validar algumas práticas sugeridas, o TCC inclui a realização de Provas de Conceito (PoCs), cada uma focada em demonstrar a aplicabilidade e os benefícios de práticas específicas. Essas PoCs vão desde a inicialização de um microsserviço baseado em domínio de negócio até a implementação de padrões como API Gateway e Circuit Breaker. Este resumo encapsula a essência do TCC, que não apenas fornece um guia teórico mas também coloca à prova as práticas recomendadas, oferecendo contribuições tangíveis para a comunidade de desenvolvimento e reforçando o papel vital dos microsserviços na engenharia de software contemporânea.

Palavras-chaves: Microsserviços. Boas práticas de desenvolvimento. Arquitetura de software. Provas de Conceito (PoCs). Escalabilidade de aplicações. Catálogo.

Abstract

This thesis addresses the challenge of implementing efficient and resilient microservices. At its core, this thesis develops a comprehensive catalog of best practices, meticulously compiled to guide developers through the complexities inherent in microservices architecture. With the increasing complexity of software systems and the demand for rapid and continuous delivery, this project proves itself for teams seeking to optimize their development processes. The proposed catalog emerges as a valuable resource, offering evidence-based guidance and practical examples. To validate the suggested practices, this study includes the execution of Proof of Concepts (PoCs), each one focused on demonstrating the applicability and benefits of specific practices. These PoCs range from the initiation of a business domain-based microservice to the implementation of standards such as API Gateway and Circuit Breaker. This summary encapsulates the essence of this Monograph, which not only provides a theoretical guide but also tests the recommended practices, offering tangible contributions to the development community and reinforcing the vital role of microservices in contemporary software engineering.

Key-words: Microservices. Best development practices. Software architecture. Proof of Concepts (PoCs). Application scalability. Catalog.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SOA	*Arquitetura Orientada a Serviços*
PoC	*Proof of Concept / Prova de Conceito*
API	*Application Programming Interface*
SAP	*Systems, Applications, and Products in Data Processing*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language:*
HTTP	*Hypertext Transfer Protocol*
DDD	*Domain-Driven Design*

Sumário

1 Introdução

Neste capítulo, tem-se a exposição do cenário que compreende este trabalho, atuando no contexto de Arquitetura de Software. O contexto introduzirá brevemente os desafios existentes na área, ressaltando a necessidade de uma arquitetura robusta visando a sustentabilidade e a evolução de um software. Na sequência, tem-se a Questão de Pesquisa que orienta este estudo. Posteriormente, constam a Justificativa, seguida dos Objetivos desta pesquisa. Por fim, há a organização da monografia em capítulos.

1.1 Contexto

Na Engenharia de Software, a natureza e a definição de 'arquitetura' têm sido amplamente debatidas e ponderadas. Desde sua concepção, a indústria tem se esforçado para entender e definir claramente o que exatamente constitui a arquitetura de um software.(FOWLER, 2023b) Enquanto alguns consideram a arquitetura como a organização fundamental de um sistema ou como a forma que os componentes de mais alto nível estão interconectados, outros argumentam que essa definição pode ser muito simplista ou mesmo ambígua(FOWLER, 2023b).

Ralph Johnson, em uma conversa durante a QCon[1](JOHNSON, 2010), refletiu sobre essa questão e sua definição de arquitetura. Ele desafiou a ideia comum de que arquitetura refere-se às 'decisões de *design* que precisam ser tomadas no início de um projeto', propondo que, na verdade, poderia ser mais apropriadamente descrita como 'as decisões que você gostaria de poder tomar logo no início de um projeto'. Sua definição culminou na ideia de que 'Arquitetura trata do que é importante. Seja lá o que for". Nessa afirmação, está a essência de que a arquitetura de software é realmente sobre discernimento. É sobre reconhecer e priorizar os elementos que, se não tratados corretamente, podem resultar em consequências graves para o projeto (FOWLER, 2023b).

Essa perspectiva de Johnson é especialmente relevante na era contemporânea, à medida que os sistemas de software tornam-se mais complexos e multifacetados. Esse aumento de complexidade é evidente quando se observam as tendências e evoluções no âmbito da arquitetura de software. Ao longo do tempo, várias abordagens arquiteturais foram propostas e adotadas, tais como: arquitetura em camadas (ou n-Tier) (IBM, 2023b); arquitetura orientada a serviços(SOA) (IBM, 2023c), arquitetura monolítica(TechTarget,

[1] A QCon é uma série de conferências internacionais voltadas para desenvolvedores de software, arquitetos e líderes técnicos. Organizada pela InfoQ, seu objetivo é compartilhar práticas, tendências e inovações recentes em engenharia de software, arquitetura de sistemas e liderança em tecnologia.

2023), arquitetura de microsserviços (WHAT..., 2023a), dentre outras.

Atualmente, à medida que os sistemas evoluem em complexidade e tamanho, diversas dessas arquiteturas revelam-se desafiadoras em termos de manutenção e escalabilidade. Tomando como exemplo a arquitetura monolítica, percebe-se que, com a contínua evolução dos requisitos, implementar mudanças sem comprometer a integridade do código e a operacionalidade da aplicação torna-se um desafio crescente(TechTarget, 2023). Para contornar as desvantagens inerentes à arquitetura monolítica, a arquitetura de microsserviços ganhou destaque. Nessa abordagem, uma aplicação é estruturada em componentes independentes, onde cada componente representa um processo do aplicativo, funcionando como um serviço distinto. Estes serviços interagem entre si através de interfaces bem definidas, utilizando APIs ágeis e leves. Devido à sua independência, cada serviço pode ser atualizado, implantado e escalado individualmente, permitindo atender mais adequadamente as demandas de funções específicas do aplicativo(AWS, 2023b).

No entanto, apesar de os microsserviços trazerem inúmeras vantagens, eles também trazem consigo desafios distintos. Questões como escalabilidade adequada, interconexão entre serviços, registro eficaz de *logs*, monitoramento e depuração surgem no cenário. É diante dessa realidade que o debate sobre as melhores práticas para o desenvolvimento e a implantação de microsserviços se torna imperativo(Red Hat, 2023a).

Ao reconhecer a importância e o potencial dos microsserviços no cenário atual de desenvolvimento de software, este trabalho visa delinear e catalogar as melhores práticas associadas a essa arquitetura, auxiliando profissionais e organizações na jornada de adoção de microsserviços.

1.2 Questão de Pesquisa

A questão de pesquisa primordial deste estudo é a seguinte: Quais são as práticas mais recomendadas na área de Arquitetura de Software para o desenvolvimento e a implantação de uma arquitetura de microsserviços?

De acordo com a Questão de Pesquisa, provemos um Catálogo de boas práticas de desenvolvimento e implantação de arquitetura de microsserviços, no intuito de facilitar a consulta por parte dos interessados.

A arquitetura de microsserviços tem ganhado destaque nos últimos anos como uma abordagem de desenvolvimento de software que permite que as aplicações sejam divididas em componentes menores e independentes, conhecidos como microsserviços. Isso traz flexibilidade, escalabilidade e facilidade de manutenção para sistemas de software complexos. (AWS, 2023b) No entanto, a implementação bem-sucedida de microsserviços requer a adoção de boas práticas que ajudem a mitigar desafios como a complexidade da

comunicação entre microsserviços, segurança e gerenciamento de dependências.

A questão de pesquisa busca identificar as boas práticas no desenvolvimento e na implantação de microsserviços. Para isso, foi necessário analisar fontes confiáveis, como livros, artigos científicos e documentos técnicos. Algumas boas práticas, já identificadas em leituras preliminares sobre o tema (FOWLER, 2023a), incluem:

- **Descomposição de Serviços**: Dividir as funcionalidades de maneira adequada em microsserviços para manter a coesão e o baixo acoplamento[2];
- **Gerenciamento de Dependências**: Gerenciar cuidadosamente as dependências entre microsserviços para evitar conflitos e garantir que as atualizações ocorram de forma segura;
- **Segurança**: Implementar práticas robustas de segurança, como autenticação e autorização, para proteger os microsserviços contra ameaças;
- **Monitoramento e Rastreamento**: Estabelecer ferramentas e processos para monitorar o desempenho e rastrear problemas em microsserviços em execução;
- **Documentação Abundante**: Manter documentação detalhada para cada microsserviço, facilitando a compreensão e a manutenção.

Em resposta à questão de pesquisa, foi obtido um conjunto de práticas recomendadas para o desenvolvimento e a implantação de uma arquitetura de microsserviços, principalmente orientando-se pela literatura especializada. Com o intuito de apresentar essas práticas de forma mais adequada, foi providenciado um catálogo de boas práticas, contendo um descritivo conceitual e embasado de cada uma, além de alguns exemplos concretos - implementados e disponíveis em repositório - que conferem aos interessados maiores detalhes do desenvolvimento e implantação.

Diante do exposto, contribuiu-se tanto para a divulgação de conhecimentos sobre a arquitetura de microsserviços quanto para a aplicação desses conhecimentos por parte dos interessados.

1.3 Justificativa

O conceito de 'Arquitetura de Microsserviços' emergiu recentemente para caracterizar uma abordagem específica no *design* de aplicativos de software, organizando-os como

[2] Baixo acoplamento significa que as mudanças em um serviço têm impacto mínimo ou nenhum impacto em outros serviços.

conjuntos de serviços que podem ser implementados de maneira autônoma. Os microsserviços trazem consigo uma série de benefícios que ressoam fortemente com as necessidades contemporâneas das organizações. Um deles é a agilidade: ao serem implantados de forma independente, os microsserviços facilitam a gestão de correções e lançamentos de novas funcionalidades. Outra vantagem importante é a composição de equipes menores e focadas. Dada a natureza autônoma de um microsserviço, ele pode ser gerenciado por uma equipe pequena, fomentando agilidade e produtividade. A flexibilidade no uso da tecnologia é outra característica valiosa dos microsserviços. As equipes têm a liberdade de selecionar tecnologias que mais bem atendem às necessidades de seu serviço. Por fim, a escalabilidade é uma das vantagens mais destacadas dos microsserviços. Eles permitem a ampliação horizontal de serviços específicos, otimizando o uso de recursos e evitando a necessidade de expandir todo o aplicativo.

A arquitetura microsserviços, apesar de suas vantagens, também apresenta desafios inerentes. A complexidade do sistema é ampliada pela multiplicidade de serviços autônomos, tornando o desenvolvimento e o teste mais desafiadores. A falta de governança pode conduzir a uma diversidade de linguagens e estruturas, complicando a manutenção. Adicionalmente, a latência e o congestionamento de rede podem ser aumentados devido à quantidade de comunicações inter-serviços. A ausência de padronização e de um conjunto bem definido de melhores práticas pode exacerbar os desafios enfrentados ao adotar a arquitetura de microsserviços. Estas lacunas, ao invés de favorecer, podem reduzir e até anular as vantagens que os microsserviços prometem oferecer.

Dessa forma, compreendendo a relevância dos microsserviços na engenharia de software atual e sua proeminente posição no mercado, tornou-se relevante a elaboração de um catálogo que congregue as melhores práticas. Tal instrumento serviria como um referencial pertinente para desenvolvedores e arquitetos de software, proporcionando diretrizes embasadas na comunidade especializada para a implementação de microsserviços. Dado o contexto apresentado e a lacuna identificada na literatura e prática atual, a elaboração deste trabalho justificou-se.

1.4 Objetivos

Dentro do contexto da pesquisa, abordado na seção 1.2, foram definidos objetivos gerais e específicos com o propósito de oferecer respostas ao questionamento proposto.

1.4.1 Objetivo Geral

O objetivo geral deste trabalho de conclusão de curso é contribuir para a comunidade de desenvolvimento de software interessada em aplicações baseadas em micros-

serviços, elaborando um catálogo abrangente das principais práticas recomendadas. Para auxiliar nessa contribuição, selecionaremos e validaremos as três práticas mais críticas por meio da execução de Provas de Conceito (PoCs), demonstrando sua eficácia e aplicabilidade prática no desenvolvimento de sistemas de software.

1.4.2 Objetivos Específicos

Para que fosse possível atingir o objetivo geral, é importante cumprir os objetivo específicos a seguir:

- Realizar uma revisão sistemática da literatura e consultar fontes especializadas para identificar e documentar um conjunto de práticas recomendadas para o desenvolvimento de microserviços.
- Analisar o conjunto de práticas compiladas para selecionar as três práticas consideradas mais críticas ou impactantes para a eficácia da arquitetura de microserviços, com base em critérios como frequência de recomendação, potencial de impacto e aplicabilidade.
- Desenvolver e implementar Provas de Conceito para cada uma das três práticas selecionadas, com o objetivo de demonstrar sua aplicabilidade e eficácia no contexto de um projeto de desenvolvimento de software.
- Analisar os resultados obtidos das PoCs, documentando as observações, os desafios enfrentados, as soluções adotadas e as implicações práticas das práticas validadas.
- Consolidar as práticas recomendadas, os *insights* obtidos da análise das PoCs e as diretrizes para aplicação prática em um catálogo estruturado, destinado a orientar desenvolvedores e equipes de projeto na adoção eficiente da arquitetura de microserviços.

1.5 Organização da Monografia

Esta monografia segue a seguinte organização:

- Capítulo 2 - Referencial Teórico: Este capítulo tem como objetivo fornecer os principais conceitos que compõem a base de fundamentos do presente trabalho, concentrando-se na compreensão dos princípios fundamentais relacionados ao desenvolvimento e à implantação de microsserviços, de acordo com o tema abordado;

- Capítulo 3 - Refencial Tecnológico: Este capítulo tem como objetivo apresentar o contexto tecnológico, evidenciando os principais instrumentos/recursos (ex. ferramentas, linguagens, platataformas e outros) que possibilitam a realização do trabalho como um todo;

- Capítulo 4 - Metodologia: Este capítulo tem como objetivo esclarecer sobre a metodologia adotada como norteadora do presente trabalho, ao expor os detalhes que definem a classificação da pesquisa e os métodos de desenvolvimento e análise de resultados;

- Capítulo 5 - Estudo Exploratório: Este capítulo tem como objetivo descrever em detalhes a proposta do trabalho, descreve o catálogo de boas práticas e demonstra a aplicação prática de algumas dessas práticas através de três Provas de Conceito (PoCs).

- Capítulo 6 - Análise de Resultados: Este capítulo tem como objetivo apresentar a análise dos resultados obtidos através das PoCs, utilizando ferramentas como SonarQube, JQuery e Grafana para validar a eficácia das boas práticas propostas.

- Capítulo 7 - Conclusão: Este capítulo tem como objetivo apresentar a conclusão do trabalho, retomando os objetivos, questões de pesquisa, principais resultados do Estudo Exploratório e perspectivas para trabalhos futuros.

2 Referencial Teórico

Neste capítulo, será apresentada uma fundamentação teórica, embasada na literatura, visando mair compreensão sobre os conceitos e princípios inerentes ao trabalho, em especial sobre arquiteturas de software, culminando na abordagem contemporânea dos microserviços. Em um primeiro momento, há uma visão geral sobre o termo Arquitetura de Software e, na sequência, busca-se maior detalhamento quanto às especificidades de três abordagens arquiteturais: arquitetura monolítica, arquitetura orientada a serviços (SOA) e, finalmente, arquitetura de microserviços. Após uma compreensão clara dessas três arquiteturas e suas evoluções, há colocações sobre as vantagens inerentes à escolha dos microserviços, procurando apresentar um comparativo entre essa arquitetura e a monolítica. Também será dada atenção especial aos desafios e às complicações que podem surgir ao se adotar essa abordagem. Em seguida, consta um olhar sobre os padrões arquiteturais frequentemente associados a essa abordagem. Para solidificar a compreensão prática e as implicações reais dessas abordagens, são cobertos cenários de uso emblemáticos. O capítulo será concluído com um resumo final, fornecendo uma consolidação dos *insights* e saberes compartilhados ao longo deste capítulo.

2.1 Arquitetura de Software

A arquitetura de software é frequentemente entendida de maneira intuitiva por engenheiros e desenvolvedores, principalmente aqueles com certa experiência no campo. Porém, definir arquitetura de software de forma clara e concisa é uma tarefa desafiadora. Uma questão recorrente é a tentativa de distinguir a arquitetura do mero *design* de software, sendo que a primeira pode ser considerada uma faceta mais especializada da última (CONCEITO. . . , 2023).

Garlan e Shaw, em sua obra 'An Introduction to Software Architecture', destacam a profundidade e a complexidade da arquitetura de software, posicionando-a além da mera elaboração de algoritmos e estruturas de dados. Eles enfatizam que, quando se aborda a arquitetura, entra-se em um domínio do *design* que lida com desafios holísticos e estruturais do sistema. Para esses autores, a arquitetura concentra-se não apenas na funcionalidade, mas em como o sistema como um todo é organizado, controlado e interconectado. Eles apontam questões como a composição dos elementos de *design*, a escalabilidade, os protocolos de comunicação, e a decisão sobre alternativas de *design* como essenciais no âmbito da arquitetura de software (GARLAN; SHAW, 1993).

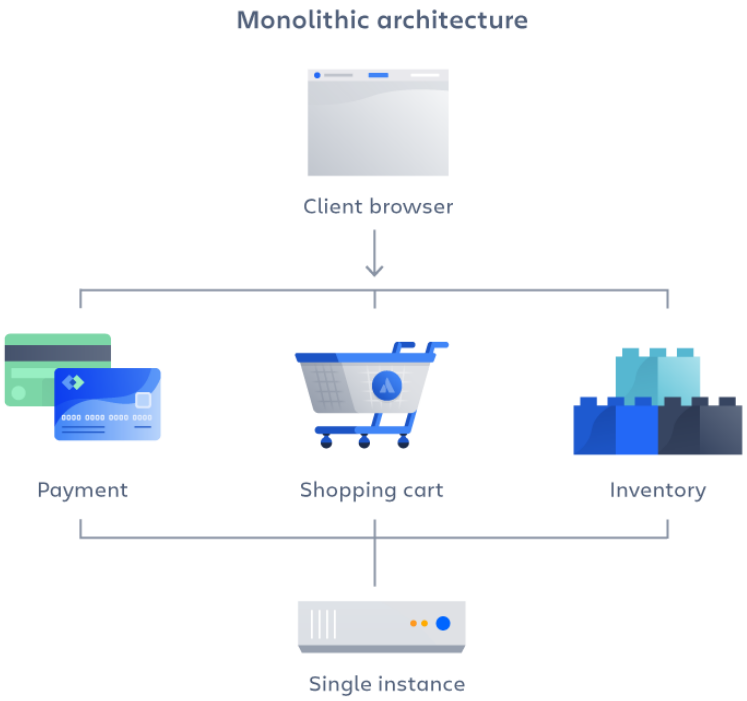
À medida que os sistemas de software se expandiam, tanto em escopo quanto em

funcionalidade, tornou-se evidente que o *design* e a estrutura desses sistemas precisavam de um planejamento meticuloso. Esse planejamento permitiu não apenas atender às demandas imediatas, mas também acomodar mudanças futuras e crescimento. No início da programação, os sistemas eram frequentemente construídos como monolitos, onde todos os componentes do software estavam interligados em uma única unidade. Embora essa abordagem fosse adequada para sistemas menores, logo se tornou evidente que ela não era escalável para sistemas maiores e mais complexos. Com a popularização da Internet e o surgimento de sistemas distribuídos, a necessidade de arquiteturas mais flexíveis tornou-se primordial. Surgiram abordagens como a Arquitetura Orientada a Serviços (SOA), que visava descompor sistemas em serviços independentes que podiam ser reutilizados em diferentes contextos (AppMaster, 2023). No entanto, à medida que a complexidade e a escala dos sistemas cresceram, algumas limitações do SOA começaram a aparecer, especialmente em termos de granularidade e gerenciamento. Foi nesse contexto que a arquitetura de microsserviços emergiu como uma evolução natural do SOA. Enquanto a SOA se concentra em serviços de negócios maiores, os microsserviços descompõem o sistema em partes ainda menores e mais focadas, geralmente organizadas em torno de capacidades de negócios específicas. Os microsserviços ampliaram os princípios da SOA, enfatizando a independência completa, não apenas em termos de funcionalidade, mas também em termos de desenvolvimento, implantação e escalabilidade. Além disso, enquanto a SOA tendia a centralizar a governança e a reutilização de serviços, a arquitetura de microsserviços favorece a descentralização (BERRIO-CHARRY; VERGARA-VARGAS; UMAñA-ACOSTA, 2020).

2.2 Arquitetura monolítica

A arquitetura monolítica refere-se ao design de software no qual um programa é construído como uma única e coesa unidade, independente de outros sistemas. Este termo, frequentemente associado a estruturas grandes e imutáveis, reflete eficazmente a natureza da arquitetura monolítica em software. Toda a lógica e funcionalidades são compactadas em uma base de código singular, implicando que qualquer modificação exige acesso direto a essa base, seguido de um relançamento completo da aplicação. Enquanto esta abordagem pode ser benéfica nas fases iniciais de um projeto devido à sua simplicidade, ela pode tornar-se limitante e ineficiente conforme o sistema cresce, tornando as atualizações mais desafiadoras e demoradas (ATLASSIAN, 2023a). Uma ilustração desse conceito pode ser visualizada na imagem de exemplo apresentada a seguir.

Figura 1 – Arquitetura monolítica



Fonte: (ATLASSIAN, 2023a)

2.2.1 *Vantagens da arquitetura monolitica*

Com a evolução da engenharia de software, diversos estilos arquiteturais emergiram, cada um adequado às necessidades e desafios do seu tempo. A arquitetura monolítica é uma dessas abordagens que, embora considerada tradicional, tem se mantido relevante em certos contextos de aplicação. O *design* monolítico foca na unidade, onde todas as funções são desenvolvidas em conjunto, em um único código-base. Este estilo de arquitetura, apesar de antiga, possui suas peculiaridades que podem se adequar bem a certos contextos de negócios. Segundo (ATLASSIAN, 2023a), as principais vantagens de optar por ela incluem:

- Simplicidade na Implementação: Há apenas um arquivo executável ou diretório a ser tratado.
- Desenvolvimento Coeso: Com tudo em uma única base de código, o desenvolvimento torna-se mais linear.
- Desempenho Aprimorado: Em alguns cenários, uma única API centralizada pode ser mais eficiente do que múltiplas APIs em microsserviços.

- Testes Diretos: Por ser uma entidade única, os testes de ponta a ponta são mais simples e diretos.
- Depuração Facilitada: A centralização do código facilita o rastreamento e correção de falhas.

2.2.2 *Desvantagens da arquitetura monolitica*

Enquanto a abordagem monolítica tem suas forças que a tornam atrativa, ela também vem com suas limitações que podem se tornar obstáculos em ambientes de desenvolvimento dinâmicos e em rápida evolução. Segundo (ATLASSIAN, 2023a), as principais desvantagens associadas a essa arquitetura incluem:

- Desenvolvimento Lento: À medida que o aplicativo cresce, torna-se mais desafiador e demorado fazer alterações.
- Questões de Escalabilidade: Diferentemente dos microsserviços, é difícil dimensionar partes individuais do monólito.
- Confiabilidade Reduzida: Uma falha em um módulo pode comprometer todo o sistema.
- Resistência à Mudança Tecnológica: Atualizar ou modificar tecnologias no monólito é um processo trabalhoso.
- Rigidez: A estrutura monolítica é, por natureza, menos flexível do que as abordagens modernas.
- Implementações Custosas: Qualquer pequena alteração requer a recompilação e reimplementação do sistema como um todo.

2.3 Arquitetura orientada a serviços

A arquitetura monolítica, dominante antes da ascensão da SOA, tinha desafios inerentes em termos de escalabilidade, manutenção e integração com outros sistemas. Cada atualização ou correção exigia que o sistema inteiro fosse temporariamente desativado e então reimplantado, tornando o processo trabalhoso e muitas vezes disruptivo. Frente a esses desafios, surgiu a SOA (*services oriented architecture* ou arquitetura orientada a serviços). SOA representa uma abordagem de *design* de software focada em tornar os componentes reutilizáveis através de interfaces de serviços que se comunicam por meio de uma linguagem comum em uma rede. Em vez de ter um código monolítico único e

massivo, a SOA divide o software em serviços independentes, cada um responsável por uma tarefa ou função específica. Estes serviços, completos com suas integrações de dados e código, podem ser acessados e atualizados remotamente, conferindo uma flexibilidade e modularidade até então não vistas(Red Hat, 2023b).

A grande revolução trazida pela SOA foi a capacidade de integrar esses componentes de software, mesmo que implantados e mantidos separadamente, permitindo que se comunicassem e trabalhassem conjuntamente. Assim, as aplicações poderiam operar de forma coesa, mesmo estando em sistemas diferentes, oferecendo uma solução elegante e eficiente para os problemas anteriormente enfrentados pela arquitetura monolítica (Red Hat, 2023b).

2.3.1 *Funcionamento da arquitetura orientada a serviços*

Na Arquitetura Orientada a Serviços (SOA), cada serviço é autônomo, contendo todo o código e dados necessários para realizar uma função de negócios específica, como processar uma proposta ou verificar o crédito de um cliente. Estes serviços apresentam acoplamento fraco por meio de interfaces, o que simplifica a interação entre eles, já que demandam mínimo conhecimento sobre a implementação uns dos outros. A comunicação entre os serviços é formalizada através de um contrato de serviço, a interface, que é independente da linguagem de programação ou da plataforma usada no desenvolvimento do serviço. Assim, serviços podem ser originários de várias fontes, sejam elas em Java, Microsoft .Net, ou até mesmo aplicativos comerciais como SAP. Essas interfaces são frequentemente descritas usando WSDL, um padrão baseado em XML(IBM, 2023c).

Para a comunicação, os serviços utilizam protocolos padrão, como SOAP/HTTP ou Restful HTTP, facilitando a interação e troca de informações. Importante ressaltar que a governança desempenha um papel fundamental na SOA, gerenciando o ciclo de vida dos serviços. Uma vez prontos, os serviços são listados em um registro, facilitando sua descoberta e reutilização por outros desenvolvedores. Por fim, embora possam ser desenvolvidos do zero, muitos desses serviços são, na verdade, funções de sistemas legados expostas como interfaces, aproveitando recursos e funcionalidades preexistentes(IBM, 2023c).

2.3.2 *Benefícios da arquitetura orientada a serviços*

Conforme destacado pela (Red Hat, 2023b), a Arquitetura Orientada a Serviços (SOA) apresenta diversos benefícios quando comparada à abordagem tradicional monolítica:

- Agilidade no Lançamento: A capacidade de reutilizar serviços permite uma entrega mais rápida e flexível das aplicações, eliminando a necessidade de começar sempre do zero.

- Maximização de Recursos Legados: SOA facilita a integração de sistemas legados em novas plataformas ou ambientes, expandindo sua utilidade.

- Eficiência de Custos: A agilidade e eficiência no processo de desenvolvimento proporcionam uma redução significativa nos custos.

- Manutenção Simplificada: A natureza independente dos serviços facilita atualizações e modificações, sem perturbar o ecossistema inteiro.

- Escalabilidade Robusta: SOA suporta a integração de diversos serviços, plataformas e linguagens, juntamente com protocolos de comunicação padronizados, promovendo escalabilidade sem grandes complicações.

- Confiabilidade Aumentada: Serviços menores são mais fáceis de depurar e, portanto, resultam em aplicações mais estáveis.

- Disponibilidade Universal: Os recursos e serviços na SOA estão acessíveis a todos, promovendo uma distribuição uniforme de funcionalidades.

2.4 Arquitetura de microsserviços

Microsserviços são estruturados como pequenas aplicações que podem ser escaladas, testadas e implantadas de maneira autônoma. Sua singularidade é evidenciada pela posse de uma única responsabilidade, seja esta relacionada à função principal do serviço ou à forma como ele se comporta. A singularidade do microsserviço remete à sua essência de ter apenas um motivo para sofrer alterações ou ser substituído. Mais do que isso, ele é delineado pela capacidade de realizar exclusivamente uma tarefa, proporcionando compreensão facilitada de seu propósito e funcionamento. Essa única tarefa ou responsabilidade do microsserviço pode manifestar-se de diferentes maneiras. Pode estar atrelada a requisitos funcionais específicos, como realizar uma operação de negócios específica. Por outro lado, pode estar associada a requisitos não funcionais ou transversais, lidando, por exemplo, com aspectos de performance ou segurança (THÖNES, 2015).

A adoção de microsserviços tem crescido graças à simplicidade no desenvolvimento decorrente da sua granularidade, à escalabilidade proporcionada pelos contêineres e à conveniência de implantação em plataformas de nuvem. Porém, é fundamental considerar os desafios associados à gestão e monitoramento desses serviços (FLORES, 2023).

Microsserviços permitem desenvolvimento em várias linguagens e podem ser criados de maneira autônoma por equipes distintas. Eles se contrapõem às abordagens monolíticas, onde os aplicativos tendem a ser implantados como uma única entidade em contêineres web. (TAIBI; LENARDUZZI; PAHL, 2017). Muitos arquitetos de software têm defendido esse estilo arquitetônico. Contudo, frente às implicações financeiras e logísticas de uma migração, alguns profissionais permanecem cautelosos em relação à adoção dos microsserviços, em grande parte devido à incerteza sobre suas vantagens e possíveis obstáculos. Nesse cenário, é comum que as decisões arquiteturais dos desenvolvedores sejam influenciadas por experiências anteriores ou pelo apelo de inovações recentes. Assim, é crucial entender os motivos que conduzem à preferência pelos microsserviços, explorar as principais razões para sua adoção atual e identificar áreas que possam precisar de maior refinamento (TAIBI; LENARDUZZI; PAHL, 2017).

2.4.1 *Vantagens da Arquitetura de microsserviços*

Os microsserviços ganharam popularidade não apenas entre desenvolvedores, mas também entre executivos e líderes de projetos. Diferentemente de muitas tendências arquitetônicas, que normalmente atraem apenas equipes técnicas, os microsserviços também ressoam com os líderes empresariais. Isso ocorre porque essa arquitetura espelha a maneira como muitos líderes desejam organizar e gerir suas equipes e processos. Além disso, os microsserviços são totalmente compatíveis com a ideia de produzir software para a nuvem, no movimento conhecido como Cloud Native (Cloud Native Computing Foundation,). Esse alinhamento com práticas Cloud Native não só representa um modelo técnico avançado, mas também facilita a implementação de um modelo operacional desejado para os negócios, promovendo escalabilidade, resiliência e agilidade (IBM, 2023a).

De acordo com a (ATLASSIAN, 2023b), os microsserviços oferecem uma série de vantagens que melhoram o processo de desenvolvimento e a entrega de software:

- **Agilidade**: Equipes menores e autônomas podem adotar práticas ágeis, acelerando o ciclo de desenvolvimento graças à estrutura independente dos microsserviços.
- **Escalabilidade**: Devido à sua natureza distribuída, os microsserviços podem ser facilmente escalados horizontalmente, permitindo a rápida adição de novas instâncias conforme a demanda.
- **Lançamentos Rápidos**: Os microsserviços favorecem ciclos de lançamento mais curtos e frequentes, facilitando a experimentação e a implementação de novos recursos.
- **Flexibilidade Tecnológica**: As equipes têm liberdade para escolher as ferramentas e tecnologias que preferirem, sem ficarem restritas a um único padrão.

- **Qualidade Aprimorada**: A modularidade dos microsserviços permite que as equipes se concentrem em garantir a qualidade de serviços individuais.
- **Confiabilidade**: Ao segmentar funcionalidades, os riscos associados a atualizações e mudanças são reduzidos. Isolar e corrigir problemas em um serviço não impacta todo o sistema, aumentando a estabilidade geral.

2.4.2 *Monolito x microsserviços*

Neste capítulo, buscaremos elucidar as principais diferenças e características de cada uma destas abordagens arquiteturais. Conforme dito anteriormente, enquanto o monolito é tradicionalmente conhecido por sua estrutura coesa e unitária, os microsserviços trazem uma divisão em serviços menores e independentes, cada um com sua própria responsabilidade. Para ilustrar mais claramente as distinções e particularidades de cada abordagem, apresentamos a seguir uma tabela comparativa que destaca os principais pontos de cada estrutura. Esta tabela fornece *insights* sobre quando e porque optar por uma arquitetura em detrimento da outra.

Tabela 1 – Resumo das diferenças: monolítico x microsserviços

Categoria	**Arquitetura monolítica**	**Arquitetura de microsserviços**
Design	Base de código única com várias funções interdependentes.	Componentes de software independentes com funcionalidade autônoma que se comunicam entre si usando APIs.
Desenvolvimento	Requer menos planejamento no início, mas fica cada vez mais complexo de entender e manter.	Requer mais planejamento e infraestrutura no início, mas fica mais fácil de gerenciar e manter com o tempo.
Implantação	Aplicação inteira implantada como uma única entidade.	Cada microsserviço é uma entidade de software independente que requer implantação individual em contêineres.
Depuração	Rastreie o caminho do código no mesmo ambiente.	Requer ferramentas avançadas de depuração para rastrear a troca de dados entre vários microsserviços.
Modificação	Pequenas mudanças introduzem riscos maiores, pois afetam toda a base de código.	Você pode modificar microsserviços individuais sem afetar toda a aplicação.
Escala	Você precisa escalar toda a aplicação, mesmo que apenas determinadas áreas funcionais tenham um aumento na demanda.	Você pode escalar microsserviços individuais conforme necessário, o que economiza custos gerais de escalabilidade.
Investimento	Baixo investimento inicial à custa de maiores esforços contínuos e de manutenção.	Investimento adicional de tempo e custo para configurar a infraestrutura necessária e desenvolver a competência da equipe. No entanto, economia de custos, manutenção e adaptabilidade a longo prazo.

Fonte: (AWS, 2023a)

2.4.3 Desafios e Complicações na Implementação de Microsserviços

Embora o emprego da arquitetura de microsserviços traga muitos benefícios, existem alguns desafios a serem enfrentados para que essa arquitetura não se torne um problema a longo prazo. Um sistema baseado em qualquer arquitetura não deve ser construído pensando apenas no tempo de desenvolvimento, mas também levando em conta características como a manutenibilidade a longo prazo no tempo em que o sistema já se encontra

em produção. Para que isso seja garantido, (FELIPE, 2020) cita alguns dos desafios ao utilizar a arquitetura de microsserviços no desenvolvimento de aplicações:

- **Consistência dos dados:** quando temos dados sendo acessados e registrados por vários serviços diferentes, podemos ter inconsistências de dados entre os serviços, se o modelo de replicação dos dados entre os serviços não for adequado ou se houver atraso na publicação das alterações, os usuários poderão ter inconsistências nos dados da aplicação.
- **Comunicação entre os sistemas e entre as equipes:** para que não haja inconsistência nas regras de negócio tratadas pelos microsserviços, é necessária uma comunicação bem estruturada tanto em termos síncronos quanto em termos assíncronos entre os microsserviços. O mesmo vale para a comunicação entre as equipes que tratam e implementam as regras de negócio.
- **Monitoramento:** monitoramento é importante em sistemas de qualquer arquitetura, porém, quando se trata de microsserviços, é algo crucial para garantir o bom funcionamento do sistema. Quando se tem vários microsserviços funcionando em sinergia para gerar valor, é imprescindível saber quais microsserviços estão funcionando e como estão funcionando. A forma como é feito o monitoramento dos serviços é crucial para a tomada de decisões e para realizar correções em mau funcionamento.
- **Testes:** por envolver diversos serviços e camadas, a arquitetura de microsserviços possui um fluxo de dados muito mais complexo e de difícil entendimento às vezes. Portanto, realizar ou escrever testes para sistemas baseados na arquitetura de microsserviços é uma tarefa muito mais difícil do que nas aplicações monolíticas.
- **Construção:** construir microsserviços demanda uma grande capacidade analítica para identificar a dependência entre os serviços, uma vez que uma alteração em um serviço pode acarretar alterações em vários outros serviços.
- **Versionamento:** decidir como controlar a versão de uma aplicação em microsserviços não é uma tarefa fácil. É necessário analisar se isso será realizado em um repositório único ou se cada serviço terá seu próprio repositório. Cada microsserviço tem sua própria versão, mas a agregação de serviços também pode gerar uma versão própria. Fica muito mais difícil pensar na compatibilidade entre versões anteriores.
- ***Deploying*:** A etapa de *deploy* de uma aplicação de microsserviços deve ser obrigatoriamente automatizada. A complexidade dessa etapa é tão grande que se torna inviável realizá-la manualmente.

2.4.4 Padrões Comuns de Microsserviços

No contexto da arquitetura de microsserviços, a utilização de padrões de *design* é fundamental para garantir a eficiência, escalabilidade e manutenibilidade dos sistemas. Esses padrões representam soluções comprovadas para desafios recorrentes enfrentados durante o desenvolvimento e implantação de microsserviços. Para chegar a esses padrões, são empregados métodos como análise de casos de uso, identificação de problemas comuns em diferentes implementações e colaboração entre especialistas na área. Esses métodos permitem a sistematização de práticas que atendem às necessidades específicas dos microsserviços. (VELEPUCHA; FLORES, 2023)

Nesta seção, apresentamos uma tabela com a relação dos padrões mais comuns aplicados na construção de sistemas baseados em microsserviços, proporcionando uma visão quantitativa dos padrões existentes e destacando suas características e benefícios.

Tabela 2 – Principais padrões de *design* em microsserviços

Padrão	**Descrição**
API Gateway	Atua como um ponto de entrada único para o conjunto de microsserviços, gerenciando roteamento, composição e políticas de segurança.
Service Discovery	Permite que os microsserviços encontrem e se comuniquem uns com os outros dinamicamente através de um registro centralizado ou distribuído.
Circuit Breaker	Protege serviços contra falhas em cascata ao monitorar chamadas e interromper solicitações para serviços não responsivos.
Event Sourcing	Armazena o estado dos dados como uma sequência de eventos, permitindo reconstruir o estado atual a qualquer momento.
CQRS (Command Query Responsibility Segregation)	Separa as operações de leitura e escrita em modelos distintos para otimizar o desempenho e a escalabilidade.
Saga Pattern	Gerencia transações distribuídas garantindo consistência eventual através de uma série de passos compensatórios.
Strangler Pattern	Facilita a migração de sistemas legados para microsserviços, permitindo a substituição gradual de funcionalidades.
Bulkhead	Isola recursos críticos para limitar o impacto de falhas e melhorar a resiliência do sistema.
Backend for Frontend (BFF)	Cria backends específicos para diferentes interfaces de usuário, otimizando a comunicação entre o frontend e os microsserviços.
Distributed Tracing	Implementa rastreamento de solicitações através de múltiplos serviços para facilitar o monitoramento e a depuração.

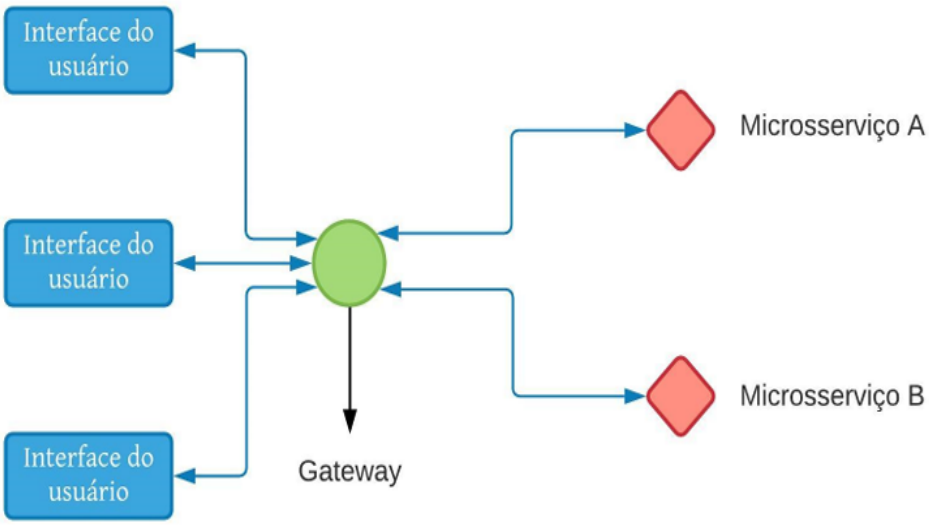
- **Comunicação entre Serviços - Padrão API Gateway:** Em ambientes de microsserviços, a gestão e roteamento de requisições podem se tornar complexos devido à quantidade de serviços independentes. O padrão API Gateway envolve a utiliza-

ção de um *gateway* (ou *proxy*) para centralizar e simplificar o acesso aos serviços, oferecendo um ponto único de entrada, roteamento de requisições, composição de respostas e até mesmo a implementação de políticas de segurança e limitação de taxa (NEGRI, 2021).

- Ponto de Entrada Único: O API Gateway oferece um único ponto de entrada para os clientes externos se comunicarem com o sistema. Em vez de conhecerem os *endpoints* individuais de cada serviço, os clientes fazem suas requisições para o *gateway*.
- Roteamento de Requisições: O *gateway* é responsável pelo roteamento de requisições para os microsserviços apropriados com base em informações como a URL, o método HTTP ou outros cabeçalhos. Ele encaminha a requisição para o serviço relevante de acordo com as regras de roteamento configuradas.
- Composição de Respostas: Em alguns casos, o API Gateway pode até mesmo compor múltiplas respostas de diferentes microsserviços para fornecer uma resposta única e coesa ao cliente externo. Isso é útil para reduzir o número de chamadas de API que o cliente precisa fazer.
- Políticas de Segurança: O *gateway* pode implementar políticas de segurança, autenticação e autorização para proteger os microsserviços subjacentes. Ele age como um ponto de controle centralizado para garantir que apenas solicitações autorizadas sejam encaminhadas.
- Limitação de Taxa: Em ambientes onde é importante controlar a taxa de acesso aos serviços para evitar sobrecargas, o API Gateway pode implementar limitações de taxa para equilibrar a carga de forma adequada.
- Monitoramento e Registro: O *gateway* também pode ser configurado para coletar métricas de desempenho, registro de solicitações e respostas, facilitando o monitoramento e a solução de problemas.

Em resumo, o padrão API Gateway simplifica a comunicação entre serviços em ambientes de microsserviços, proporcionando uma camada de controle centralizada que melhora a segurança, o desempenho e a capacidade de gerenciamento do sistema. Ele se torna uma parte essencial da infraestrutura de microsserviços, permitindo que os clientes externos interajam com o sistema de forma transparente, mesmo quando há muitos microsserviços funcionando nos bastidores.

Figura 2 – Api Gateway fazendo o direcionamento das requisições para os microsserviços



Fonte: (NETO; AUGUSTO, 2021)

- **Decomposição de Aplicações baseada em Domínios de Negócio:** Este padrão é uma estratégia de arquitetura que envolve a decomposição de uma aplicação monolítica em microsserviços, e essa decomposição é orientada pelos domínios de negócio ou áreas funcionais da aplicação. Essa abordagem é frequentemente associada ao conceito de Domain-Driven Design (DDD), que é uma metodologia de *design* de software que coloca um forte foco na modelagem do domínio de negócio.(VELEPUCHA; FLORES, 2023)

  - A ideia principal por trás desse padrão é que uma aplicação monolítica pode se tornar complexa e difícil de gerenciar à medida que cresce. Diversas funcionalidades, regras de negócio e componentes diferentes podem estar interconectados, tornando o código difícil de entender e manter. A decomposição com base em domínios de negócio visa resolver esse problema e esses são alguns dos principais pontos deste padrão:
  - *Domain-Driven Design* (DDD): O DDD é uma abordagem que enfatiza a modelagem do domínio de negócio como o ponto central do *design* de software. Ele sugere a criação de modelos de domínio ricos em termos de negócio, como entidades, agregados e objetos de valor. Ao aplicar o DDD, você ganha uma compreensão profunda dos domínios de negócio em sua aplicação.
  - Decomposição Orientada por Domínio: Com base na compreensão adquirida por meio do DDD, você pode identificar os diferentes domínios ou áreas funcionais de sua aplicação. Cada um desses domínios pode ser mapeado para um microsserviço separado.

  - Responsabilidade Clara: Cada microsserviço é projetado para ter uma responsabilidade clara dentro do domínio que representa. Isso significa que um microsserviço deve conter apenas a lógica de negócio relacionada àquele domínio específico, evitando interdependências complexas.
  - Flexibilidade e Escalabilidade: A decomposição com base em domínios permite que você dimensione e mantenha cada microsserviço de forma independente. Isso facilita a escalabilidade horizontal e torna o sistema mais flexível para acomodar mudanças em áreas específicas da aplicação.
  - Facilidade de Entendimento: Ao dividir a aplicação em microsserviços alinhados com os domínios de negócio, o sistema geral se torna mais fácil de entender e manter. Cada equipe pode se concentrar em seu próprio microsserviço, reduzindo a complexidade global do código.

  Em resumo, a decomposição de aplicações baseada em domínios de negócio é uma estratégia poderosa para tornar sistemas de software mais escaláveis, flexíveis e fáceis de manter. Ela aproveita o conhecimento profundo dos domínios de negócio para criar microsserviços que refletem as áreas funcionais da aplicação, permitindo uma melhor organização e gerenciamento do sistema como um todo. Essa abordagem é especialmente valiosa em ambientes de microsserviços, onde a modularidade e a clareza nas responsabilidades são essenciais.

- **Tratamento de Falhas - Padrão *Circuit Breaker*:** Em ambientes distribuídos, onde diferentes serviços se comunicam entre si, é possível que a falha em um serviço específico provoque uma cascata de falhas em outros serviços interconectados. Essa situação pode levar a uma degradação generalizada do sistema e afetar a disponibilidade e o desempenho de todo o ecossistema. É aí que entra o padrão 'Circuit Breaker' (Disjuntor, em tradução literal).(FALAHAH; SURENDRO; SUNINDYO, 2021)

  O padrão Circuit Breaker é uma estratégia de tratamento de falhas que funciona de maneira análoga ao disjuntor elétrico de uma instalação elétrica. A ideia principal é a seguinte:

  - Monitoramento de Falhas: O sistema monitora continuamente o serviço ou recurso que está sendo chamado. Ele acompanha o número de falhas ou erros que ocorrem ao interagir com esse serviço.
  - Estado Normal: Inicialmente, o Circuit Breaker está em um 'estado normal' e permite que as chamadas ao serviço ocorram normalmente. Todas as solicitações são enviadas ao serviço alvo.
  - Transição para o Estado Aberto: Se o sistema detectar um aumento anormal no número de falhas ou erros ao chamar o serviço (por exemplo, devido a

um serviço não responsivo ou sobrecarregado), o Circuit Breaker pode fazer a transição para o 'estado aberto'. Nesse estado, o Circuit Breaker bloqueia temporariamente todas as chamadas ao serviço problemático, evitando que novas solicitações sejam encaminhadas.

- Tempo de Espera e Recuperação: Enquanto o Circuit Breaker está no estado aberto, ele pode aguardar um período determinado antes de permitir que as chamadas ao serviço sejam reativadas. Esse tempo de espera dá ao serviço problemático a chance de se recuperar ou resolver seus problemas.
- Transição de Volta ao Estado Normal: Após o término do período de espera, o Circuit Breaker pode fazer a transição de volta ao 'estado normal' e permitir que as chamadas ao serviço sejam encaminhadas novamente. Se o serviço continuar a funcionar sem problemas, o sistema retorna à operação normal. Se novas falhas ocorrerem, o Circuit Breaker poderá repetir o processo de transição para o 'estado aberto'.

O padrão Circuit Breaker oferece vários benefícios, incluindo a prevenção de falhas em cascata, a redução do tráfego desnecessário durante falhas e a melhoria da resiliência do sistema como um todo. Isso é particularmente valioso em arquiteturas de microsserviços, onde várias partes do sistema interagem entre si e as falhas podem se propagar rapidamente. O Circuit Breaker atua como um mecanismo de proteção, garantindo que as falhas em um serviço não afetem negativamente o funcionamento de todo o sistema.

- **Outros Padrões Relevantes:** Além desses padrões, existem muitos outros mostrados em (VELEPUCHA; FLORES, 2023) que desempenham papéis importantes na arquitetura de microsserviços, alguns exemplos são:
  - Padrão Service Registry and Discovery: Como os microsserviços podem descobrir e se comunicar entre si em um ambiente dinâmico;
  - Padrão de Token JWT (JSON Web Token): Como lidar com autenticação e autorização entre serviços;
  - Padrão de Log Correlacionado: Como correlacionar logs entre serviços para rastrear o fluxo de requisições;
  - Padrão de Rastreamento Distribuído: Ferramentas e práticas para rastrear uma chamada através de vários serviços;
  - Padrão de Implantação Azul-Verde: Reduzir o tempo de inatividade ao implantar novas versões de um microsserviço;
  - Padrão de Canary Release: Liberar uma nova versão para um subconjunto de usuários para testar e validar;

2.4.5 Cenários de Uso

A adoção da arquitetura de microsserviços tem se mostrado uma abordagem inovadora e eficaz para o desenvolvimento de sistemas de software. Diversas empresas líderes em tecnologia, como Netflix[1], Uber[2], LinkedIn[3], SoundCloud[4] entre outras, têm adotado essa abordagem e colhido resultados notáveis. Neste subtópico, foram examinados esses cenários de uso e foi realizada uma análise crítica de suas experiências, destacando os benefícios, desafios e lições aprendidas.

- Netflix: Lições para o *Design* Arquitetural

  A Netflix é frequentemente citada como um exemplo pioneiro na adoção de microsserviços. O artigo (MAURO, 2015) descreve as melhores práticas arquiteturais que a Netflix adotou ao longo de sua jornada com microsserviços. O artigo mostra como a Netflix conseguiu escalar sua plataforma de *streaming* e oferecer uma experiência de visualização excepcional para milhões de usuários em todo o mundo.

- Uber: Refatorando com microsserviços

  A Uber, uma das empresas líderes em transporte compartilhado, optou por refatorar sua infraestrutura monolítica em direção aos microsserviços. O artigo (ENGINEERING, 2021) aborda a implementação de microsserviços pela Uber. É possivel notar como essa transição permitiu à Uber ganhar flexibilidade, escalabilidade e melhorar a experiência de seus motoristas e passageiros.

- LinkedIn: Evolução da Arquitetura de Serviço

  O LinkedIn, uma rede social profissional líder, passou por uma significativa evolução em sua arquitetura de serviços. O artigo (IHDE, 2015) detalha essa transição. O LinkedIn adotou microsserviços para melhorar a escalabilidade, a resiliência e a capacidade de inovação.

- SoundCloud: Lidando com o Monolito

  O SoundCloud, uma plataforma de compartilhamento de música, enfrentou desafios semelhantes ao lidar com uma arquitetura monolítica. O artigo (CALCADO, 2014)

[1] Netflix: uma plataforma de streaming de filmes e séries mundialmente conhecida, que utiliza microsserviços para escalar e distribuir seu conteúdo globalmente de maneira eficiente.

[2] Uber: uma empresa de tecnologia focada em mobilidade urbana e entrega de alimentos, que adota microsserviços para manter sua infraestrutura global, conectando milhões de motoristas e usuários em tempo real.

[3] LinkedIn: uma rede social voltada para conexões profissionais e oportunidades de emprego, que utiliza a arquitetura de microsserviços para lidar com seu grande volume de usuários e dados.

[4] SoundCloud: uma plataforma de streaming de música que permite aos usuários compartilhar e descobrir novas músicas, também aproveitando a flexibilidade dos microsserviços para suportar suas operações de forma escalável.

descreve como o SoundCloud abordou esse desafio por meio de microsserviços. Essa transição melhorou a agilidade e a capacidade de entrega de novos recursos.

Estes casos de uso serviram como exemplos simples de como as empresas estão abraçando a arquitetura de microsserviços e como essa abordagem pode impactar positivamente a escalabilidade, a agilidade e a inovação no desenvolvimento de software.

Além de destacar os aspectos positivos desses cenários de uso, também foi conduzida uma análise crítica das possíveis falhas e desafios encontrados durante a adoção de microsserviços. Afinal, é importante compreender que nem todas as empresas obtêm sucesso sem obstáculos na implementação de microsserviços.

2.5 Resumo do capítulo

Neste capítulo, exploramos diversas arquiteturas de software, incluindo a Arquitetura Monolítica, Arquitetura Orientada a Serviços e Arquitetura de Microsserviços. Comparamos as características e benefícios de cada uma delas.

- Arquitetura de Software:

  Introdução às diferentes abordagens arquiteturais em sistemas de software. A importância de escolher a arquitetura correta para atender aos requisitos do projeto.

- Arquitetura Monolítica:

  Descrição da arquitetura monolítica, em que todos os componentes do sistema estão integrados em um único aplicativo. Vantagens e desvantagens dessa abordagem, incluindo a simplicidade, mas também a falta de escalabilidade e flexibilidade.

- Arquitetura Orientada a Serviços:

  Discussão sobre a arquitetura orientada a serviços (SOA) e sua ênfase na modularidade e reutilização de componentes. Destaque para a necessidade de um barramento de serviços e protocolos de comunicação padronizados.

- Arquitetura de Microsserviços:

  Exploração da arquitetura de microsserviços, em que o sistema é dividido em componentes independentes que se comunicam por meio de APIs. Benefícios, como escalabilidade e manutenibilidade, e desafios, como complexidade na gestão e monitoramento.

- Arquitetura Monolítica x Arquitetura de Microsserviços:

Comparação das duas arquiteturas em termos de escalabilidade, flexibilidade, manutenção e outros aspectos. Destaque para a tendência de migração de sistemas monolíticos para arquiteturas de microsserviços.

- Desafios e Complicações:

  Análise dos desafios enfrentados ao implementar e gerenciar sistemas baseados em microsserviços. Questões como consistência de dados, comunicação entre serviços e monitoramento.

- Padrões Comuns de Microsserviços:

  Exploração dos padrões de design, infraestrutura, segurança e monitoramento comuns em arquiteturas de microsserviços. Destaque para o Padrão API Gateway, Circuit Breaker, Service Registry e Discovery, CQRS e Event Sourcing.

- Cenários de Uso:

  Estudo de casos de sucesso de empresas que adotaram arquiteturas de microsserviços. Análise crítica de sucessos e falhas em implementações reais, incluindo exemplos da Netflix, Uber, LinkedIn e SoundCloud.

- Resumo do Capítulo:

  Recapitulação das principais ideias e conclusões do capítulo. Destaque para a importância de escolher a arquitetura adequada com base nos requisitos e desafios específicos do projeto. Este capítulo fornece uma base sólida para entender as diferentes arquiteturas de software e os princípios dos microsserviços, preparando o terreno para a implementação de um Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços, que será abordado nos próximos capítulos.

3 Suporte Tecnológico

Neste capítulo, foram abordadas as principais ferramentas e tecnologias selecionadas para a condução deste trabalho. A estrutura deste capítulo foi dividida em três seções principais. A primeira seção 3.1, foca nas tecnologias planejadas para serem empregadas na fase subsequente do TCC na construção do catálogo de boas práticas. A segunda seção 3.2, detalha outras ferramentas significativas que conferem suporte a elaboração do presente trabalho. Por fim, a terceira seção 3.3, reflete sobre a importância do planejamento tecnológico e traz um resumo das tecnologias e ferramentas abordadas.

3.1 Ferramentas para desenvolvimento do catálogo

Para a elaboração do catálogo de boas práticas em microsserviços, foi fundamental a seleção de ferramentas adequadas que proporcionem eficiência, integridade e uma experiência de usuário agradável. Ao considerar a natureza deste projeto, bem como os requisitos e desafios associados ao desenvolvimento de um catálogo deste tipo, várias tecnologias e plataformas foram consideradas. A escolha dessas ferramentas não apenas define a abordagem de desenvolvimento, mas também tem implicações significativas na manutenibilidade, escalabilidade e adaptabilidade do catálogo no futuro.

3.1.1 Git e GitHub

O Git é um sistema de controle de versão distribuído que permite rastrear mudanças no código ao longo do tempo(Git, 2023). Ele se tornou uma ferramenta indispensável para colaboração e versionamento de código em projetos modernos. Em conjunção com o Git, o GitHub oferece uma plataforma para hospedagem de repositórios e facilita a colaboração entre desenvolvedores, fornecendo recursos como 'pull requests', revisão de código e integração contínua(GITHUB, 2023). Para este projeto, o GitHub foi utilizado para gerenciar o código-fonte do catálogo, permitindo uma colaboração eficaz entre os envolvidos e assegurando que todas as alterações e atualizações estejam adequadamente documentadas e versionadas.

3.1.2 GitBook

O GitBook é uma ferramenta moderna de documentação que integra diretamente com o GitHub. Ele permite que os desenvolvedores e colaboradores escrevam documen-

tações ricas, manuais ou até mesmo livros usando a linguagem Markdown(WHAT..., 2023b). A sua integração direta com o GitHub significa que as atualizações no código podem ser refletidas diretamente na documentação. Para este projeto, o GitBook será utilizado para criar e manter a documentação associada ao catálogo, garantindo que os usuários e desenvolvedores tenham acesso a informações atualizadas e relevantes sobre as boas práticas listadas.

3.1.3 Spring e Spring Boot

O *Spring Framework*, comumente referido apenas como Spring, é uma das plataformas Java mais populares, destinada à criação de aplicações corporativas de alto desempenho(SPRING FRAMEWORK, 2023). Ao longo dos anos, ele revolucionou o desenvolvimento Java ao oferecer uma abordagem mais simples, flexível e abrangente, comparado aos padrões tradicionais de desenvolvimento J2EE. Surgindo como uma extensão natural do Spring Framework, o Spring Boot visa simplificar o processo de configuração e desenvolvimento de aplicações Spring. Ele herda todas as características poderosas do Spring e adiciona camadas de abstração para facilitar ainda mais a vida dos desenvolvedores(SPRING..., 2023). Ele oferece uma convenção sobre a configuração, eliminando a necessidade de especificar beans no arquivo XML de configuração do Spring. Esta abordagem 'convenção sobre configuração' permite que os desenvolvedores se concentrem na lógica do negócio, em vez de detalhes de configuração. Ao adotar o Spring Boot como nossa plataforma de implementação, buscamos não apenas validar algumas das boas práticas sugeridas em nosso catálogo, mas também demonstrar sua aplicabilidade em uma das frameworks mais populares e respeitadas da indústria.

3.2 Ferramentas de apoio

Neste contexto de apoio, várias ferramentas desempenharam um papel essencial durante a condução deste trabalho. Abaixo, detalhamos algumas destas ferramentas e suas respectivas finalidades:

- Monday: Utilizado como uma ferramenta de apoio para a implementação do Quadro Kanban, facilitando a organização e o rastreamento das atividades ao longo do projeto.
- Telegram: Essencial para comunicações instantâneas e troca de mensagens rápidas entre o autor e os orientadores.

- Slack: Funcionou como um repositório para as orientações e feedbacks ao longo do trabalho, sendo uma ferramenta crucial para manter o alinhamento entre o autor e os orientadores.
- Teams: Adotado para reuniões periódicas, sejam elas semanais ou conforme necessário, e para chats mais extensos.
- Overleaf: Adotado como a principal ferramenta para redação e formatação deste trabalho, garantindo uma apresentação consistente e profissional do conteúdo.
- Zotero: Esta ferramenta de gerenciamento de referências foi essencial para organizar e citar as diversas fontes consultadas durante a pesquisa.

Estas ferramentas, combinadas, garantiram uma execução eficiente e organizada do projeto, desde sua concepção até sua finalização.

3.3 Considerações finais

Neste capítulo, delineamos as principais ferramentas e tecnologias selecionadas para a condução e apoio do presente trabalho. Discutimos inicialmente sobre a relevância do Spring e sua extensão, o Spring Boot, destacando sua importância no contexto de aplicar algumas das boas práticas mencionadas no catálogo. Em seguida, abordamos uma série de ferramentas de apoio que foram cruciais em diversas etapas do projeto, incluindo comunicação, escrita e organização de referências. Ferramentas como Monday, Telegram, Slack, Teams e Zotero desempenharam papéis vitais, cada uma contribuindo de forma única para o sucesso da pesquisa. O Latex, em particular, serviu como alicerce fundamental na elaboração do texto, assegurando a qualidade e consistência desejadas. A combinação dessas ferramentas proporcionou um ambiente de trabalho robusto, facilitando o desenvolvimento contínuo e a comunicação eficaz entre todas as partes envolvidas. No intuito de conferir um resumo sobre as tecnologias utilizadas, segue a Tabela 3.

Tabela 3 – Resumo das Tecnologias

Nome	**Descrição**	**Link**	**Versão**
Git	Sistema de controle de versão distribuído.	<https://git-scm.com>	2.42.0
Github	Plataforma para armazenamento e colaboração de código-fonte.	<https://github.com>	-
GitBook	Plataforma para publicar livros e documentos, com integração ao Git.	<https://www.gitbook.com>	3.0
Spring Boot	Ferramenta que simplifica a construção de microsserviços com Java.	<https://spring.io/>	3.1.4
Monday	Plataforma de gestão de projetos.	<monday.com>	-
Telegram	Aplicativo de mensagens instantâneas com foco em segurança e privacidade.	<https://web.telegram.org>	-
Overleaf	Editor *online* de documentos LaTeX.	<https://overleaf.com>	-
Slack	Ferramenta de comunicação para equipes.	<https://slack.com/>	-
Teams	Plataforma de colaboração e comunicação da Microsoft.	<https://teams.microsoft.com/>	-
Zotero	Gerenciador de referências.	<https://www.zotero.org/>	-

Fonte: Autor

4 Metodologia

Este capítulo apresentará a abordagem metodológica adotada para a execução da pesquisa, tanto em sua vertente teórica quanto prática. Inicialmente, serão detalhados aspectos fundamentais da Classificação da Pesquisa, contemplando a Abordagem metodológica escolhida, a Natureza do estudo em questão, os Objetivos almejados e os Procedimentos metodológicos implementados. Também será introduzido o tópico da Metodologia Investigativa, discutindo as estratégias empregadas para inquirir e construir um entendimento sobre o fenômeno estudado. Na sequência, é introduzida a Metodologia Orientada a Provas de Conceito, essencial para o embasamento metodológico do projeto, destacando sua pertinência para a pesquisa e os passos concretos para sua execução. Posteriormente, a discussão evolui para a Metodologia de Desenvolvimento adotada, com ênfase nas práticas ágeis. Além disso, é descrita a Metodologia de Análise de Resultados, com exposição dos seus procedimentos e características específicas. Segue-se a apresentação dos Fluxos de Atividades/Subprocessos e dos Cronogramas de Atividades/Subprocessos, abrangendo tanto a primeira quanto a segunda etapa do Trabalho de Conclusão de Curso. Concluindo o capítulo, as Considerações Finais sintetizam os principais tópicos abordados, fornecendo um panorama conciso do método utilizado na pesquisa.

4.1 Classificação da Pesquisa

Gerhardt e Silveira (2009) enfatizam que toda pesquisa começa com uma questão ou dúvida que necessita de esclarecimento. O ato de pesquisar é, portanto, uma jornada de questionamento e investigação, direcionada para resolver determinadas incógnitas. Adicionalmente, os autores propõem uma classificação da pesquisa em quatro dimensões críticas: abordagem, natureza, objetivos e procedimentos, cujos detalhes serão discutidos nos próximos segmentos."

4.1.1 Abordagem

A pesquisa adotou principalmente uma metodologia qualitativa. Essa abordagem foca em aprofundar e entender um assunto específico, buscando explicar o 'porquê' dos fenômenos, sem a necessidade de submetê-los a verificações empíricas, pois se baseia em dados não quantificáveis, (GERHARDT; SILVEIRA, 2009). Embora a ênfase seja qualitativa, também serão integrados dados quantitativos coletados através de métricas extraídas das Pocs.

4.1.2 Natureza

Este estudo se enquadra na categoria de pesquisa aplicada, conforme definido por Gerhardt e Silveira (2009), com o propósito de gerar conhecimento destinado à aplicação direta e à resolução de questões específicas. Desta forma, o trabalho possui uma natureza prática, visando a encontrar soluções para os desafios apresentados na INTRODUÇÃO estabelecida.

4.1.3 Objetivos

Em termos de classificação de objetivos, o estudo pode ser categorizado como exploratório. Essa abordagem de pesquisa é projetada para aumentar o entendimento e a proximidade com o assunto em questão, com o intuito de obter insights mais profundos que possam contribuir para a formulação de hipóteses ou clarificar o tema com maior precisão (GERHARDT; SILVEIRA, 2009)

4.1.4 Procedimentos

Este trabalho se caracteriza por uma pesquisa que se baseia em provas de conceito. O principal objetivo de uma prova de conceito é validar a viabilidade de uma ideia ou conceito em um contexto prático. Além disso, como um procedimento complementar, foi realizada uma pesquisa bibliográfica, cujos detalhes podem ser encontrados na seção de METODOLOGIA

4.2 Método Investigativo

Segundo o livro (KAHLMEYER-MERTENS et al., 2007), uma vez definido o tema, a pesquisa bibliográfica inicial busca proporcionar uma maior familiarização do pesquisador com a área de estudo em foco. Esta fase é crucial, já que auxilia na delimitação do escopo do trabalho a ser realizado. Por conseguinte, logo após a definição do tema deste estudo, foi conduzida uma pesquisa bibliográfica em bases científicas. Abaixo, encontra-se a tabela contendo as strings de busca utilizadas durante essa fase.

4.2.1 Critérios de Seleção

Dado o vasto número de artigos disponíveis, estabelecemos critérios rigorosos de seleção para garantir a relevância e a qualidade das fontes escolhidas. Esses critérios foram

Tabela 4 – Strings de busca utilizadas na pesquisa bibliográfica

Base	***String* de Busca**	**Nº de artigos**
Google Acadêmico	microservices	50.800
Google Acadêmico	desenvolvimento de microsserviços	1.010
Google Acadêmico	Implantação eficaz de arquiteturas de microsserviços	470
IEEE	microservices AND monolithic architecture	260
IEEE	microservices AND services oriented	300
Google Acadêmico	software architecture guide, 2020-2023	2780

Fonte: Autor

desenvolvidos com base nas necessidades específicas deste estudo e nas melhores práticas de pesquisa bibliográfica:

- Relevância Temática: Priorizamos artigos que estivessem diretamente relacionados aos microserviços, especialmente aqueles que discutiam práticas de desenvolvimento, implantação e desafios arquitetônicos. Isso incluiu trabalhos que abordavam a modularidade, escalabilidade, autonomia e padrões de comunicação entre os serviços.
- Autoridade e Credibilidade: Demos preferência a fontes escritas por especialistas reconhecidos na área de arquitetura de software e microserviços, incluindo pesquisadores acadêmicos e profissionais da indústria com um histórico comprovado de contribuições significativas para o campo.
- Atualidade: Concentramos em artigos publicados nos últimos cinco anos para garantir que as informações estivessem atualizadas com as tendências e práticas mais recentes na arquitetura de microserviços.
- Aplicabilidade Prática: Artigos que incluíam estudos de caso, exemplos práticos ou discussões detalhadas sobre a implementação de microserviços receberam atenção especial, pois oferecem insights valiosos sobre como as práticas recomendadas são aplicadas no mundo real.
- Fundamentação Teórica: Artigos que contribuíam para a fundamentação teórica dos microserviços, oferecendo uma compreensão mais profunda dos princípios subjacentes a essa abordagem arquitetônica, também foram considerados.

Ao construir um catálogo de boas práticas para o desenvolvimento e implementação de microserviços, é crucial estabelecer critérios de seleção que garantam a qualidade, relevância e aplicabilidade das diretrizes incluídas.

As boas práticas devem estar intimamente relacionadas aos princípios fundamentais dos microserviços, como a modularidade, escalabilidade, desacoplamento, autonomia

e distribuição. Devem abordar questões específicas sobre a implementação, comunicação, testes e gerenciamento de microserviços.

As diretrizes devem ser comprovadas e reconhecidas como práticas eficazes na implementação de microserviços. Elas devem incluir estratégias para lidar com desafios comuns, como segurança, tolerância a falhas, monitoramento e versionamento.

As boas práticas devem ser descritas de maneira clara, oferecendo orientações práticas e diretas para desenvolvedores e equipes de desenvolvimento. Devem incluir exemplos reais, padrões de codificação, ferramentas recomendadas e abordagens testadas. Baseada em Evidências e Experiências Práticas e elas devem ser fundamentadas em experiências reais de implementação de microserviços, bem como em pesquisas e estudos que respaldem sua eficácia. É importante considerar fontes confiáveis, relatos de casos, estudos de benchmarking e experiências de mercado.

A atualidade das boas práticas é crucial. Devem refletir os desafios e tendências atuais no desenvolvimento e implementação de microserviços. Informações desatualizadas ou práticas obsoletas devem ser evitadas.

4.2.1.1 Resultados

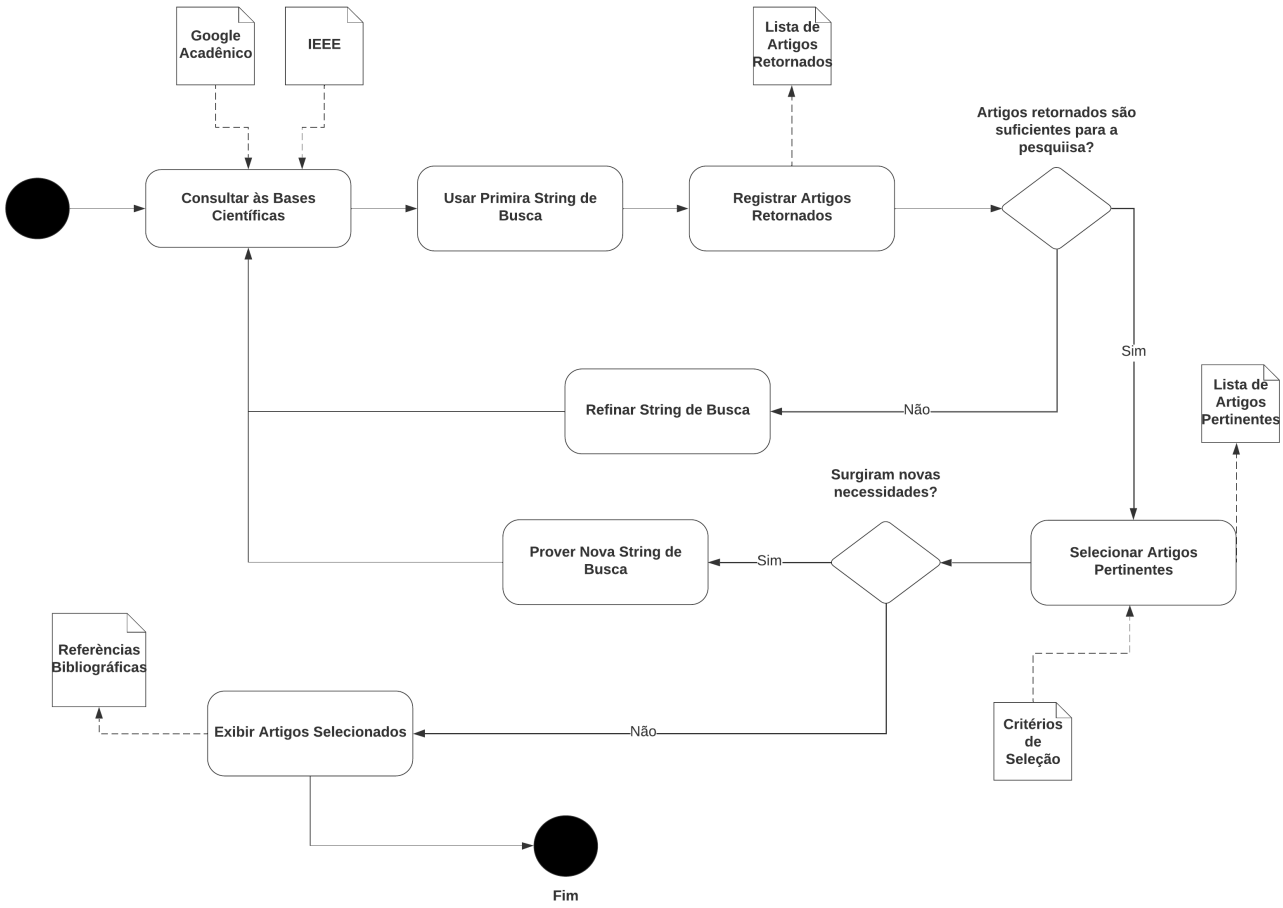
Após uma análise prévia, alguns artigos selecionados orientaram a pesquisa do trabalho, bem como suas referências. Sendo algumas delas:

- Microservices a definition of this new architectural term: (FOWLER, 2023a)
- What are microservices? (WHAT..., 2023a)

O resultado do processo de Pesquisa Bibliográfica é apresentado no Referencial Teórico, concentrando-se na de implentação de microsserviços e suas melhores práticas. Cada tópico é minuciosamente detalhado, buscando uma fundamentação adequada para a realização do trabalho.

Adicionalmente, são realizados estudos com base na literatura especializada para atender aos desafios identificados nas provas de conceito. O processo investigativo é conduzido conforme descrito anteriormente: consultas em bases científicas e correlatas utilizando strings de busca, ajustes nas strings de busca para abranger as palavras-chave principais, recuperação e registro dos artigos obtidos, seleção dos artigos com base em critérios e clareza na exposição dos materiais utilizados para fundamentar a solução que mitiga o desafio. Um resumo desse sub-processo, que representa as atividades da metodologia investigativa, é ilustrado por meio de modelagem em BPMN.

Figura 3 – Modelagem da metodologia investigativa



Fonte: Autor

4.3 Método Orientado a Provas de Conceito

para Guimarães (2008) uma Prova de Conceito é uma realização prática de uma ideia ou método, com o objetivo de demonstrar sua viabilidade. Portanto a Metodologia Orientada a Provas de Conceito (PoC) adotada neste trabalho é essencial para validar as boas práticas de desenvolvimento de microsserviços em um contexto real. Esta abordagem permite verificar a aplicabilidade, eficácia e impacto das práticas recomendadas em um ambiente de software controlado.

Conforme Missao e Jr (2003), a Prova de Conceito segue uma organização clara e direta. Inicia-se com a planejamento e preparação da infraestrutura, onde selecionamos produtos e operações específicos que funcionarão como um piloto. Em seguida, passamos para a fase de implementação, onde o software ou os procedimentos são implantados. A etapa de operação assistida é onde acompanhamos em tempo real o funcionamento, fazendo ajustes e corrigindo falhas conforme necessário. Por fim, na fase de avaliação, medimos o desempenho da solução implementada para verificar se atende aos objetivos

propostos.

Com base na organização proposta por Missao e Jr (2003) no presente trabalho iniciamos cada PoC com uma fase de planejamento criterioso. Aqui, estabelecemos os objetivos específicos, delimitamos o escopo e identificamos os indicadores-chave que nos permitiram medir o sucesso da prática em questão. Isso envolve uma análise aprofundada das exigências técnicas e operacionais e uma previsão dos desafios que podemos encontrar. Em seguida a fase de implementação é onde a teoria se transforma em prática. Configuramos um ambiente de desenvolvimento que reflete um contexto de produção e começamos a codificar. Durante esta etapa, aplicamos a boa prática em foco. Com a PoC em funcionamento, passamos para a execução e monitoramento. Essa fase é vital, pois coletamos dados em tempo real e observamos como a prática se comporta sob condições variadas. Usamos ferramentas de monitoramento e *logging* para coletar uma ampla gama de métricas, das técnicas às de negócio, as quais nos forneceram insights valiosos sobre a eficácia da prática.

4.4 Metodologia de Desenvolvimento

No desenvolvimento do catálogo de boas práticas para o desenvolvimento e implantação de microsserviços, a escolha da metodologia foi essencial. Escolhemos uma abordagem ágil, reconhecendo a complexidade do processo e a necessidade de adaptação contínua. A metodologia ágil, em particular o Scrum (SACHDEVA, 2016) e o Kanban (ANDERSON, 2021), permitiu entregas incrementais e colaboração efetiva entre nós.

A decisão de adotar o Kanban foi tomada com o intuito de proporcionar uma visualização clara do progresso do projeto, além de viabilizar a priorização de tarefas e a identificação de gargalos. Conforme destacado (ANDERSON, 2021), o Kanban é uma metodologia ágil de gestão de projetos que se fundamenta em um quadro de tarefas, no qual as atividades são movidas de uma coluna para outra conforme avançam em seu desenvolvimento.

Em (ANDERSON, 2021) ainda e dito que, em sua forma mais simples, o Kanban é composto por três colunas: 'To Do' (A fazer), 'Doing' (Em andamento) e 'Done' (Concluído). A coluna 'To Do' representa as tarefas que ainda não foram iniciadas, enquanto a coluna 'Doing' abrange aquelas que estão em processo de desenvolvimento. Por fim, a coluna 'Done' indica as tarefas que foram concluídas. Além disso, o Kanban oferece uma considerável flexibilidade e adaptabilidade, permitindo ajustes de prioridade e adequações às mudanças de requisitos e necessidades do projeto.

No contexto do Scrum, a intenção é fazer uso principalmente do *Product Backlog*, que consiste em uma lista de requisitos sujeita a modificações à medida que o desenvol-

vimento do projeto progride (SACHDEVA, 2016). No entanto, busca-se uma abordagem mais leve em termos de processos em comparação com o Scrum tradicional, eliminando o uso de papéis ou rituais, como reuniões diárias e retrospectivas. Na versão combinada com o Kanban, o Quadro Kanban, especificamente através do Trello, está previsto para representar o Sprint Backlog em cada momento do projeto. Destaca-se ainda que a metodologia de desenvolvimento aqui descrita será empregada em conjunto com a Metodologia Orientada a Provas de Conceito, já abordada anteriormente.

4.5 Método de Análise de resultados

Para a análise dos resultados obtidos no desenvolvimento do catálogo de boas práticas para microsserviços, adotamos uma abordagem que combina a avaliação de métricas quantitativas provenientes das provas de conceito (PoCs) com a análise de *feedback* qualitativo coletado por meio de questionários de experiência do usuário. A parte quantitativa da análise de resultado focou em métricas de desempenho obtidas durante a realização das Provas de Conceito, que atuam como testes práticos das boas práticas em estudo. Avaliamos elementos como a latência, que é o tempo que cada microsserviço leva para responder a uma solicitação; o throughput, ou seja, quantas solicitações o sistema consegue processar em um determinado intervalo de tempo; a utilização de recursos, monitorando quanto de CPU e memória está sendo usado pelos microsserviços; e o número de falhas, contabilizando quão frequentemente e em quais circunstâncias os microsserviços falham. Esses dados nos deram uma visão clara do impacto operacional das práticas de microsserviços que analisamos.

Paralelamente, para capturar a experiência dos usuários, que foram escolhidos aleatoriamente em grupos compostos por pessoas que já trabalham ou trabalharam na área de desenvolvimento e implementação de microsserviços, elaboramos um questionário (Formulário de Pesquisa). Esse questionário foi aplicado aos usuários finais que interagiram com o catálogo, com o objetivo de avaliar:

- Usabilidade: Facilidade de uso e aprendizado do catálogo.
- Satisfação: Nível de contentamento com a funcionalidade e desempenho do catálogo.
- Eficiência: esforços necessários para realizar práticas propostas no catálogo.

Os dados qualitativos obtidos dos questionários foram analisados para identificar tendências, pontos de satisfação e insatisfação, e qualquer correlação entre a experiência do usuário e a implementação das boas práticas. Esta análise multifacetada foi fundamental para validar a eficácia das práticas recomendadas e fornecer uma boa base para conclusões e recomendações no trabalho final.

4.6 Fluxo De atividades/Subprocessos

Durante a elaboração deste trabalho, executamos uma variedade de atividades com o propósito de conduzir as demandas de maneira coesa e dentro dos prazos estabelecidos. Na fase inicial do TCC, as atividades iniciam-se com a definição do tema, estendendo-se até a apresentação dos resultados perante a banca examinadora. Desta forma, elaborou-se um plano de atividades específico, conforme detalhado a seguir:

1. **Definir Tema**

   Descrição: Atividade que tem como objetivo selecionar material relevante sobre o tema escolhido, disponível nas bases científicas, além de fornecer apoio para o pesquisador sobre trabalhos anteriores sobre o tema.

   Status: Concluída;

2. **Realização da Pesquisa Bibliográfica**

   Descrição: Atividade que visa realizar uma pesquisa extensiva nas bases científicas para reunir material relevante relacionado ao tema escolhido. Essa pesquisa bibliográfica fornecerá uma base sólida para o desenvolvimento do trabalho.

   Status: Concluída;

3. **Formular Proposta Inicial**

   Descrição: Atividade que consiste na formulação da proposta inicial do trabalho, definindo os objetivos, escopo e principais hipóteses ou questões a serem abordadas, presente no Capítulo 1 - Introdução.

   Status: Concluída;

4. **Elaborar Referencial Teórico**

   Descrição: Atividade que envolve a elaboração do referencial teórico, baseando-se em teorias e estudos existentes relevantes ao tema. Este processo fornece a base conceitual para o trabalho. Presente no Capítulo 2 - Referêncial Teórico.

   Status: Concluída;

5. **Estabelecer Suporte Tecnológico**

   Descrição: Atividade que visa identificar e estabelecer as ferramentas e tecnologias necessárias para a implementação do trabalho. Presente no Capítulo 3 - Suporte Tecnológico.

   Status: Concluída;

6. **Descrever a Metodologia**

   Descrição: Atividade que consiste em detalhar a metodologia que será adotada no desenvolvimento do trabalho, incluindo abordagens, técnicas e ferramentas específicas a serem utilizadas. Presente neste Capítulo.

   Status: Concluída;

7. **Refinar Proposta**

   Descrição: Atividade que previu a revisão e o refinamento da proposta inicial com base nas orientações recebidas durante a realização da pesquisa bibliográfica e do referencial teórico. Antigo Capítulo 5 - Proposta. Atual capítulo de Estudo Exploratório.

   Status: Concluída;

8. **Desenvolver POC Inicial**

   Descrição: Atividade que consistiu no desenvolvimento das Prova de Conceito (POCs) inicial para validar a viabilidade técnica da proposta. Antigo Capítulo 5 - Proposta. Atual capítulo de Estudo Exploratório.

   Status: Concluída;

9. **Descrever Resultados Parciais**

   Descrição: Atividade que envolveu a descrição dos resultados parciais obtidos durante o desenvolvimento do trabalho. Presente no antigo Capítulo 6 - *Status*, atual capítulo de Conclusão.

   Status: Concluída;

10. **Apresentar TCC1**

    Descrição: Atividade que representou a apresentação do Trabalho de Conclusão de Curso 1 (TCC1) perante a banca examinadora.

    Status: Concluída;

4.6.1 Segunda Etapa do TCC

Nesta fase do TCC, as atividades iniciaram-se com a implementação das correções sugeridas pela banca, seguidas pelo desenvolvimento, análise dos resultados, conclusão da monografia e apresentação do trabalho perante a banca. O plano de atividades específico para esta segunda etapa da monografia é detalhado a seguir:

1. **Aplicar Correções**

Descrição: Esta etapa envolveu a implementação das correções sugeridas pela banca examinadora. Foi essencial abordar e incorporar os *feedbacks* recebidos para aprimorar a qualidade geral do trabalho.

Status: Concluída;

2. **Realização de Atividades de Desenvolvimento**

   Descrição: Após a aplicação das correções, a fase de desenvolvimento foi iniciada, concentrando-se na execução das atividades necessárias para atender aos objetivos propostos no TCC. Presente no Capítulo 5 - Estudo Exploratório.

   Status: Concluída;

3. **Realização de Análise de Resultados**

   Descrição: A análise de resultados foi uma etapa crucial que envolveu a avaliação e interpretação dos dados coletados durante a realização das atividades de desenvolvimento. Os resultados são examinados à luz dos objetivos do trabalho. Presente no Capítulo 6 - Análise de Resultados.

   Status: Concluída;

4. **Finalizar Monografia**

   Descrição: Nesta fase, a monografia foi finalizada, incorporando os resultados da análise e garantindo que todos os elementos do trabalho estejam coerentes. Revisões finais, formatação e ajustes são realizados.

   Status: Concluída;

5. **Apresentar Trabalho (Segunda Etapa)**

   Descrição: A última etapa envolve a preparação e apresentação do trabalho para a banca examinadora. Esta fase é crucial para comunicar os resultados alcançados e defender as contribuições do TCC.

   Status: Não iniciado;

4.7 Cronogramas de Atividades/Subprocessos

As Figuras 4 e 5 representam, respectivamente, os cronogramas para primeira e segunda etapa do TCC.

Figura 4 – Cronograma TCC 1

Atividades/Subprocessos	Agosto	Setembro	Outubro	Novembro	Dezembro
Definir tema					
Realização de Pesquisa Bibliográfica					
Formular Proposta Inicial					
Elaborar Referencial Teórico					
Estabelecer Suporte Tecnológico					
Descrever a Metodologia					
Desenvolvedor POC Inicial					
Descrever Resultados Parciais					
Apresentador TCC1					

Fonte: Autor

Figura 5 – Cronograma TCC 2

Atividades/Subprocessos	Março	Abril	Maio	Junho	Julho
Aplicar Correções					
Realização das Atividades de Desenvolvimento					
Realização da Análise de Resultados					
Finalizar Monografia					
Apresentador TCC2					

Fonte: Autor

4.8 Resumo

O presente capítulo teve como intuito apresentar, detalhadamente, os aspectos envolvendo a metodologia que foi utilizada no trabalho.

Em relação à classificação da pesquisa, foi utilizada uma abordagem híbrida (quantitativa e qualitativa); sua natureza é aplicada; seus objetivos são definidos como exploratório; e seus procedimentos são de pesquisa bibliográfica e POC.

Sobre a metodologia de desenvolvimento, foi utilizada uma combinação entre o *Scrum, Kanban*. Para metodologia de análise de resultados, foram apresentadas as etapas do desenvolvimento das POCs.

Além disso, foi apresentado o fluxo de atividades, mostrando as atividades e subprocessos que serão utilizados na primeira e segunda etapa do TCC, e, ao final, foi apresentado um cronograma considerando as etapas do trabalho.

5 Estudo Exploratório

Neste capítulo, apresentamos a proposta do trabalho, conferindo motivação e contextualização para o uso de boas práticas no desenvolvimento de microsserviços. Em seguida, descrevemos o catálogo proposto, que tem como objetivo melhorar a qualidade e eficiência no processo de desenvolvimento. A aplicação dessas boas práticas é demonstrada através de três Provas de Conceito (vide sessão 5.4), detalhando os critérios necessários para a sua realização. Cada prova de conceito define o desafio a ser superado e os principais requisitos inerentes a esse desafio. Adicionalmente, são apresentadas as soluções propostas e a análise dos resultados, destacando cada etapa do desenvolvimento. Por fim, encerramos com as considerações finais do capítulo.

5.1 Contexto

Nos últimos anos, a arquitetura de microsserviços emergiu como uma solução vital para organizações que buscam agilidade e escalabilidade em seus sistemas de software. A abordagem de decompor uma aplicação em um conjunto de serviços menores, cada um executando um processo de negócios único e operando de maneira independente, tem sido amplamente adotada devido à sua flexibilidade e capacidade de acelerar o desenvolvimento(VIEIRA, 2023). O desenvolvimento eficaz requer uma compreensão sólida de princípios como desacoplamento de serviços, comunicação inter-serviços, gerenciamento de dados distribuídos e estratégias de monitoramento e resiliência. O *Spring Boot*, um ecossistema robusto para a construção de microsserviços, oferece inúmeras funcionalidades que simplificam esses processos, mas a falta de aplicação de boas práticas de desenvolvimento pode levar a resultados inconsistentes e problemas operacionais.

Este trabalho procura preencher essa lacuna, fornecendo um catálogo de boas práticas que aborda os desafios arquiteturais mais amplos dos microsserviços.

5.2 Motivação

A motivação para o desenvolvimento deste trabalho surge da observação de que, no cenário atual de engenharia de software, a abordagem de microsserviços vem se estabelecendo como um paradigma fundamental para a criação de sistemas distribuídos escaláveis e resilientes. Com o aumento da complexidade dos sistemas e a necessidade de agilidade no desenvolvimento e na entrega, surge a questão de como melhorar e padronizar os processos envolvidos.

Este trabalho é impulsionado pela necessidade de consolidar um conjunto de práticas eficazes que possam ser adotadas por desenvolvedores e equipes ao trabalhar com a arquitetura de microsserviços. Embora o *Spring Boot* seja uma das plataformas mais populares para esse propósito, há outras alternativas no mercado, como *Quarkus*, *Micronaut* e *Node.js*. A escolha do *Spring Boot* neste estudo se deve à sua ampla adoção na comunidade, que representa uma parcela significativa dos desenvolvedores que trabalham com microsserviços, embora o percentual exato possa variar de acordo com o contexto regional e o setor de atuação. Essa popularidade, combinada com a vasta documentação e o suporte robusto, torna o *Spring Boot* uma escolha estratégica. Contudo, o catálogo de boas práticas aqui proposto é flexível e pode ser adaptado para outras plataformas, sugerindo a inclusão de seções futuras, eventualmente contribuídas por outros desenvolvedores, com Provas de Conceito (PoCs) em ambientes alternativos, proporcionando um guia mais abrangente para diferentes contextos de desenvolvimento.

O impulso para este estudo é intensificado por conta da lacuna percebida entre a teoria e a prática. Muitas das práticas recomendadas em literatura científica e fóruns de discussão não são amplamente testadas ou podem não ser diretamente aplicáveis a todos os contextos. Portanto, o objetivo é validar algumas práticas de maior impacto por meio de Provas de Conceito, fornecendo *insights* práticos e direcionamento para a comunidade de desenvolvimento de software, e especialmente para aqueles trabalhando com *Spring Boot*.

Em última análise, este trabalho procura contribuir para o campo da Engenharia de Software oferecendo um estudo prático para o desenvolvimento de microsserviços, um recurso essencial para organizações que buscam inovação e eficiência operacional em um mercado tecnológico que está em constante evolução.

5.3 Catálogo

O núcleo desta proposta é a criação de um catálogo de boas práticas, destinado a auxiliar desenvolvedores na implementação de microsserviços de forma mais eficiente. Este catálogo busca ir além de uma simples lista de recomendações, apresentando orientações fundamentadas e exemplos práticos que refletem estratégias e técnicas amplamente aceitas no campo do desenvolvimento de software. As práticas incluídas foram selecionadas através de um processo que envolveu pesquisa bibliográfica Metodologia, e cada prática foi analisada com base em sua relevância, eficácia e aplicabilidade, considerando o contexto de uso.

O catálogo está dividido em seções temáticas, cada uma abordando um aspecto crítico do desenvolvimento de microsserviços. As seções cobrem tópicos como configuração

de serviços, segurança, comunicação entre serviços, gestão de dados, testes automatizados e estratégias de *deploy* e monitoramento. Dentro de cada seção, as práticas são apresentadas e exemplificadas.

Além de servir como um guia para desenvolvedores e equipes de projeto, o catálogo também funciona como base para a execução das Provas de Conceito (PoCs). Estas PoCs foram elaboradas para demonstrar a eficácia de três práticas e fornecer uma compreensão mais profunda de como elas podem ser incorporadas em projetos reais.

Ao criar este catálogo, nosso objetivo é estabelecer um recurso valioso que auxilie no desenvolvimento de microsserviços e apoie os profissionais na superação dos desafios técnicos comuns desta abordagem arquitetural. Este catálogo é um documento vivo, sujeito a atualizações e melhorias contínuas à medida que novas práticas e técnicas sejam identificadas e validadas.

A comunidade poderá contribuir, por enquanto, enviando solicitações para os mantenedores do projeto. No futuro, abriremos para alterações com políticas de *commit* e outras diretrizes que garantirão a qualidade e a consistência das contribuições.

Para acessar o catálogo completo, visite Catálogo de Boas Práticas para o Desenvolvimento e Implantação de Microsserviços

5.4 Provas de Conceito para Desenvolvimento de Microsserviços

Para verificar a aplicação das boas práticas identificadas na pesquisa bibliográfica e incorporadas no desenvolvimento do catálogo, foram conduzidas três Provas de Conceito (PoCs) selecionadas. Cada PoC foi escolhida com base na relevância da prática a ser demonstrada, buscando ilustrar de maneira concreta sua funcionalidade e eficiência no processo de desenvolvimento de sistemas. Essas PoCs têm o objetivo de fornecer *insights* sobre a aplicabilidade das práticas recomendadas em cenários reais de desenvolvimento de software.

5.4.1 PoC 1 - Iniciando um Microsserviço Baseado em Domínios de Negócio

A seção a seguir apresenta a prova de conceito (PoC) elaborada para o desenvolvimento de um microsserviço baseado em um domínio de negócios. Inicialmente, apresentamos a Definição da PoC. Em seguida, detalhamos os Requisitos da PoC necessários para a implementação. Finalizamos com a Análise de Resultados da PoC.

A PoC 1 é uma demonstração prática de como criar um microsserviço do zero, orientado a um domínio de negócio específico. Este processo envolve a identificação e

definição do domínio de negócio, a coleta e especificação dos requisitos funcionais e não funcionais, e a construção do microsserviço utilizando Spring Boot.

5.4.1.1 Definição da PoC 1

O objetivo dessa prova de conceito é demonstrar a criação eficaz de um microsserviço do zero, focando em um domínio de negócio específico para ilustrar como a modularização pode ser realizada de acordo com as necessidades de negócios.

5.4.1.2 Requisitos da PoC 1

Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- Identificação do Domínio de Negócio: Escolher um domínio de negócio específico para o microsserviço.
- Definição dos Requisitos: Levantar e definir os requisitos funcionais e não funcionais.
- Desenvolvimento do Microsserviço: Construir o microsserviço utilizando Spring Boot, focando em atender os requisitos identificados.

5.4.1.3 Análise dos Resultados da PoC 1

A análise de resultados envolve a avaliação desta PoC em diversos termos de desempenho técnico:

- Desempenho Operacional: Medimos a latência, que é o tempo de resposta do microsserviço, e o *throughput*, que é o volume de operações que o serviço consegue processar em um determinado período.
- Eficiência de Recursos: Monitoramos o uso de CPU e memória para garantir que o microsserviço está otimizado para operar com eficiência.
- Estabilidade: Registramos o número e a natureza das falhas para entender a resiliência do serviço e identificamos áreas que podem necessitar de melhorias.

Toda a documentação detalhada e a análise completa dos resultados desta PoC estão disponíveis no link: Documentação e Análise de Resultados da PoC 1, permitindo um estudo mais aprofundado sobre o desenvolvimento e a performance do microsserviço.

5.4.2 PoC 2 - Implementando um API Gateway

A seção a seguir apresenta a prova de conceito (PoC) elaborada para a implementação de um API Gateway. Inicialmente, apresentamos a Definição da PoC. Em seguida, detalhamos os Requisitos da PoC necessários para a implementação. Finalizamos com a Análise de Resultados da PoC.

A PoC 2 é uma demonstração prática de como implementar um API Gateway para gerenciar e rotear solicitações entre diferentes microsserviços. Este processo envolve a definição dos requisitos do API Gateway, a configuração e implementação utilizando ferramentas apropriadas, e a validação da eficácia do *gateway* em termos de gerenciamento de tráfego e segurança.

5.4.2.1 Definição da PoC 2

O objetivo dessa prova de conceito é demonstrar a implementação eficaz de um API Gateway, focando na integração e gerenciamento de microsserviços, para ilustrar como um *gateway* pode melhorar a segurança, a escalabilidade e a eficiência do sistema.

5.4.2.2 Requisitos da PoC 2

Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- Definição dos Requisitos do Gateway: Levantar e definir os requisitos funcionais e não funcionais do API Gateway.
- Configuração do API Gateway: Configurar o API Gateway utilizando uma ferramenta como o *Spring Cloud Gateway* ou *Kong*, conforme as necessidades do sistema.[1]
- Integração com Microsserviços: Integrar o API Gateway com os microsserviços existentes, garantindo roteamento e gerenciamento eficientes.

5.4.2.3 Análise dos Resultados da PoC 2

A análise de resultados envolve a avaliação desta PoC em diversos termos de desempenho técnico:

[1] Kong é uma plataforma de gerenciamento de APIs que atua como um gateway para microsserviços, fornecendo funcionalidades como roteamento, autenticação, segurança e monitoramento. É altamente escalável e permite a integração com diversas ferramentas de observabilidade e controle.

- Facilidade de gerenciamento de diferentes APIs: Para facilitar o gerenciamento de diferentes microsserviços, configuramos o API Gateway para rotear todas as solicitações através de uma única aplicação. Isso centraliza o ponto de entrada para todos os microsserviços, simplificando o acesso e o gerenciamento das APIs para o *front-end*.
- Segurança: Analisamos os mecanismos de segurança implementados pelo API Gateway, incluindo autenticação e autorização. A implementação desses mecanismos mostrou-se eficaz, com autenticação rápida e autorização precisa, garantindo que apenas usuários autorizados pudessem acessar os serviços protegidos. Não foram identificadas vulnerabilidades significativas durante os testes, confirmando a robustez das políticas de segurança aplicadas.

Toda a documentação detalhada e a análise completa dos resultados desta PoC estão disponíveis no link: Documentação e Análise de Resultados da PoC 2, permitindo um estudo mais aprofundado sobre a implementação e a performance do API Gateway.

5.4.3 PoC 3 - Circuit Breaker

Essa seção descreve a prova de conceito (PoC) para a implementação de um Circuit Breaker. A seção é introduzida com a Definição da PoC, seguida dos Requisitos da PoC. Finaliza-se com a Análise de Resultados da PoC.

A PoC 3 é uma demonstração prática de como implementar o padrão Circuit Breaker para aumentar a resiliência do sistema, prevenindo falhas em cascata entre microsserviços. Este processo envolve a escolha de um microsserviço crítico, a configuração do Circuit Breaker para detectar falhas e a implementação de uma lógica de *fallback*.

5.4.3.1 Definição da PoC 2

Essa prova de conceito tem o intuito de validar a implementação do padrão Circuit Breaker para aumentar a resiliência do sistema, prevenindo falhas em cascata entre microsserviços.

5.4.3.2 Requisitos da PoC 2

Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- Escolher um microsserviço crítico na aplicação para implementar o Circuit Breaker.

- Configurar o Circuit Breaker para detectar falhas e interromper automaticamente as solicitações, evitando sobrecarga no serviço.
- Implementar uma lógica de

5.5 Resumo

Neste capítulo, apresentamos a proposta do trabalho, destacando a importância de adotar boas práticas no desenvolvimento de microsserviços. Propusemos um catálogo detalhado dessas práticas e delineamos sobre três Provas de Conceito (PoCs) para testar sua eficácia.

A primeira PoC foca na criação de um microsserviço desde o início, concentrando-se em um domínio de negócio. A segunda PoC explora a implementação de um API Gateway, e a terceira valida o padrão de Circuit Breaker para resiliência de um sistema.

Os resultados das PoCs reforçarão o valor do catálogo como um recurso prático para melhorar a qualidade e eficiência no desenvolvimento de sistemas escaláveis e resilientes. Este capítulo sublinha o compromisso do trabalho com a aplicação prática das boas práticas, com ideia de contribuir para avanços na Engenharia de Software moderna.

6
Análise de Resultados

Neste capítulo, apresentamos a análise detalhada dos resultados obtidos através das Provas de Conceito (PoCs) desenvolvidas. As PoCs foram elaboradas para validar a aplicação das boas práticas identificadas na pesquisa bibliográfica, presente no capítulo 2 e incorporadas no desenvolvimento do catálogo. A análise dos resultados visa fornecer uma compreensão clara sobre a eficácia, eficiência e impacto dessas práticas no desenvolvimento de microsserviços.

Para coletar métricas de usabilidade do catálogo, utilizamos um formulário, que nos proporcionou uma visão clara da experiência do usuário. O Grafana foi utilizado para monitorar e visualizar o desempenho dos microsserviços em tempo real, possibilitando a identificação de gargalos e a otimização dos recursos utilizados.

Os resultados obtidos a partir dessas análises são fundamentais para validar a eficácia das boas práticas propostas no catálogo e para identificar áreas de melhoria contínua. Este capítulo detalha cada uma dessas análises, que corroboram com as conclusões obtidas ao longo do estudo. Ao final, espera-se que as evidências apresentadas contribuam significativamente para a compreensão dos benefícios e desafios na adoção de microsserviços, bem como para a consolidação das melhores práticas recomendadas.

O sistema usado como *case* para o presente trabalho é o Agenday, uma aplicação de agendamento de consultas desenvolvida para permitir que profissionais de diferentes áreas ofereçam seus serviços e permitam que clientes agendem horários de forma prática e eficiente. O sistema inclui funcionalidades como cadastro de profissionais, gestão de horários disponíveis, catálogo de serviços e agendamento de consultas. A escolha de um sistema de agendamento de consultas para as PoCs se deu pela complexidade e relevância que esse tipo de aplicação possui em termos de requisitos de desempenho, escalabilidade e manutenibilidade. Sistemas de agendamento precisam lidar com um alto volume de operações simultâneas, garantindo disponibilidade e performance, além de necessitarem de uma arquitetura flexível que permita fácil adição de novos serviços e funcionalidades. O Agenday foi desenvolvido desde o início como um sistema baseado em microsserviços. Cada funcionalidade, como gerenciamento de usuários, cadastro de serviços, gestão de horários e o próprio agendamento, foi implementada como um serviço independente. Essa abordagem modular visava desde o princípio a escalabilidade, manutenibilidade e flexibilidade.

6.1 Análise

A aplicação das boas práticas validadas nas PoCs resultou em bons resultados no desempenho do sistema. A latência média das requisições foi mantida abaixo de 50ms. Embora nosso ambiente de testes tenha sido limitado a uma execução local, a arquitetura de microsserviços do Agenday permite uma escalabilidade muito superior. A capacidade de escalar individualmente cada serviço conforme a demanda, ao contrário de uma abordagem monolítica, é uma vantagem clara. Em um ambiente de produção, isso permitiria uma utilização mais eficiente dos recursos, reduzindo os custos e melhorando a capacidade de resposta do sistema. A manutenção do sistema tornou-se mais fácil e eficiente com a separação das funcionalidades em serviços independentes. Por exemplo, durante a implementação de uma nova funcionalidade para agendamentos, foi possível atualizar apenas o serviço de registro sem impactar o restante da aplicação, o que reduziu o tempo de deploy. Cada serviço pode ser atualizado, depurado e implantado independentemente, reduzindo o risco de impacto em outras partes do sistema. Isso aumentou a agilidade no desenvolvimento e na implementação de novas funcionalidades. Embora a arquitetura de microsserviços tenha introduzido uma maior complexidade na gestão da infraestrutura e na comunicação entre serviços, as vantagens em termos de desempenho, escalabilidade e manutenibilidade compensaram esses desafios. A utilização de ferramentas como o API Gateway e o Circuit Breaker ajudou a mitigar os riscos e a manter a estabilidade do sistema.

6.1.1 Análise PoC 1

6.1.1.1 Definição da PoC 1

Demonstrar a criação eficaz de um microsserviço do zero, focando em um domínio de negócio específico para ilustrar como a modularização pode ser realizada de acordo com as necessidades de negócios.

6.1.1.2 Definição dos Requisitos da Poc 1

- **1. Identificação do Domínio de Negócio:** Agendamento de Consultas.
- **2. Definição dos Requisitos do Sistema:** Para a definição dos requisitos, foi utilizado um *product backlog*, que é uma lista dinâmica e ordenada das funcionalidades desejadas para o produto. Essa lista foi organizada e priorizada utilizando a técnica MoSCoW (*Must have*, *Should have*, *Could have*,

Won't have this time). Essa técnica permite a priorização eficaz dos requisitos, classificando-os em quatro categorias principais:

* **Must have:** Requisitos essenciais que devem ser implementados para que o produto seja considerado funcional.
* **Should have:** Requisitos importantes que são desejáveis, mas não essenciais para o funcionamento básico do produto.
* **Could have:** Requisitos que são desejáveis, mas não são críticos e podem ser implementados se houver tempo e recursos disponíveis.
* **Won't have this time:** Requisitos que são reconhecidos, mas que não serão implementados na presente iteração.

A seguir, apresentamos as tabelas detalhadas que agrupam os requisitos por épico, especificando a descrição de cada requisito, seus critérios de aceitação e sua priorização.

Tabela 5 – Requisitos do Épico 01 - Gerenciamento de Instituições

ID	**Descrição**	**Critérios de Aceitação**	**Priorização**
REQ001	Como administrador, quero cadastrar uma nova instituição com detalhes como nome, tipo, endereço e contato.	Formulário de cadastro com campos obrigatórios. Validação dos dados inseridos.	Must Have
REQ002	Como administrador, quero atualizar ou remover instituições para manter as informações corretas e atualizadas.	Opções de editar e excluir para cada instituição cadastrada. Confirmação antes da exclusão de uma instituição.	Must Have
REQ003	Como administrador, quero visualizar detalhes completos das instituições cadastradas para gerenciar eficientemente.	Lista detalhada com todas as instituições. Opções para visualizar mais detalhes de cada instituição.	Should Have

Fonte: Autor

Tabela 6 – Requisitos do Épico 02 - Gerenciamento de Especialidades

ID	**Descrição**	**Critérios de Aceitação**	**Priorização**
REQ004	Como administrador, quero adicionar novas especialidades ao sistema para diversificar os serviços oferecidos.	Formulário para inserção de nova especialidade. Validação dos dados inseridos.	Must Have
REQ005	Como administrador, quero editar ou remover especialidades para atualizar ou simplificar a oferta de serviços.	Opções de editar e excluir para cada especialidade. Confirmação antes da remoção de uma especialidade.	Should Have

Fonte: Autor

Tabela 7 – Requisitos do Épico 03 - Gerenciamento de Colaboradores

ID	**Descrição**	**Critérios de Aceitação**	**Priorização**
REQ006	Como administrador, quero cadastrar colaboradores no sistema, incluindo informações como nome, CPF, e-mail, e seus horários disponíveis.	Formulário de cadastro com campos para nome, CPF, e-mail e horários disponíveis. Validação dos dados inseridos para garantir a precisão das informações. Possibilidade de definir múltiplos intervalos de disponibilidade por dia.	Must Have
REQ007	Como administrador, quero atualizar ou remover colaboradores para manter a equipe atualizada com as necessidades do serviço.	Opções de editar e excluir para cada colaborador cadastrado. Capacidade de alterar informações pessoais e ajustar horários disponíveis. Confirmação antes da exclusão de um colaborador.	Must Have
REQ008	Como administrador, quero gerenciar os horários de disponibilidade de cada colaborador para atualizar ou bloquear horários específicos conforme necessário.	Interface para visualizar e modificar os horários disponíveis de cada colaborador. Funcionalidade para adicionar, editar ou remover blocos de disponibilidade. Visualização clara de horários já agendados para evitar conflitos.	Should Have

Fonte: Autor

Tabela 8 – Requisitos do Épico 04 - Gerenciamento de Agendamentos

ID	Descrição	Critérios de Aceitação	Priorização
REQ009	Como cliente, quero agendar serviços diretamente pelo sistema para facilitar a gestão do meu tempo.	Interface simplificada para seleção de serviço, data e horário. Confirmação imediata do agendamento.	Must Have
REQ010	Como cliente, quero cancelar ou reagendar serviços para ajustar meus compromissos conforme necessário.	Opções de cancelamento e reagendamento acessíveis na área do cliente. Notificações enviadas ao prestador de serviço e ao cliente sobre alterações.	Should Have

Fonte: Autor

Tabela 9 – Requisitos do Épico 05 - Catálogo de Serviços

ID	Descrição	Critérios de Aceitação	Priorização
REQ011	Como cliente, quero visualizar um catálogo de serviços para escolher o serviço que melhor atenda às minhas necessidades.	Listagem de todos os serviços disponíveis com detalhes como descrição, preço e duração. Funcionalidades de filtragem por categoria, localização e avaliação.	Must Have
REQ012	Como administrador, quero adicionar serviços ao catálogo para disponibilizar mais opções aos clientes.	Formulário para inserção de novos serviços com campos para nome, descrição, preço e duração. Validação dos dados inseridos para garantir a precisão das informações.	Should Have
REQ013	Como administrador, quero atualizar ou remover serviços do catálogo para manter as informações corretas e atuais.	Opções de editar e excluir para cada serviço listado. Confirmação antes da exclusão de um serviço.	Could Have

Fonte: Autor

6.1.1.3 Análise dos Resultados da Poc 1

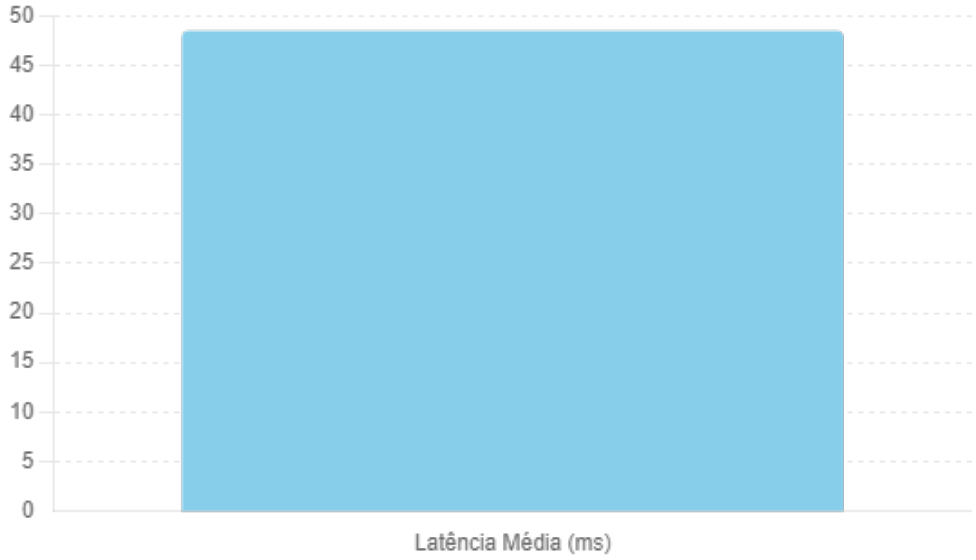
A análise de resultados envolveu a avaliação desta PoC em diversos termos de desempenho técnico. Os dados foram coletados em uma aplicação local rodando em um container. Os testes foram realizados no JMeter com 1000 threads, simulando 1000 usuários quase simultâneos fazendo requisições.

– **Desempenho Operacional**: A latência, que é o tempo de resposta do microsserviço, foi medida em milissegundos. Observamos que a latência média

permaneceu abaixo de 50 ms, mesmo sob alta carga, o que demonstra a eficiência do microsserviço em responder rapidamente às solicitações. A Figura 6 ilustra a latência média medida durante os testes.

O throughput, que é o número de requisições bem-sucedidas, foi calculado como 20.64 operações por segundo, conforme mostrado na Figura 7. Esse resultado indica que o microsserviço é capaz de processar um número significativo de requisições por segundo, mantendo um bom desempenho mesmo sob condições de alta demanda.

Figura 6 – Latência



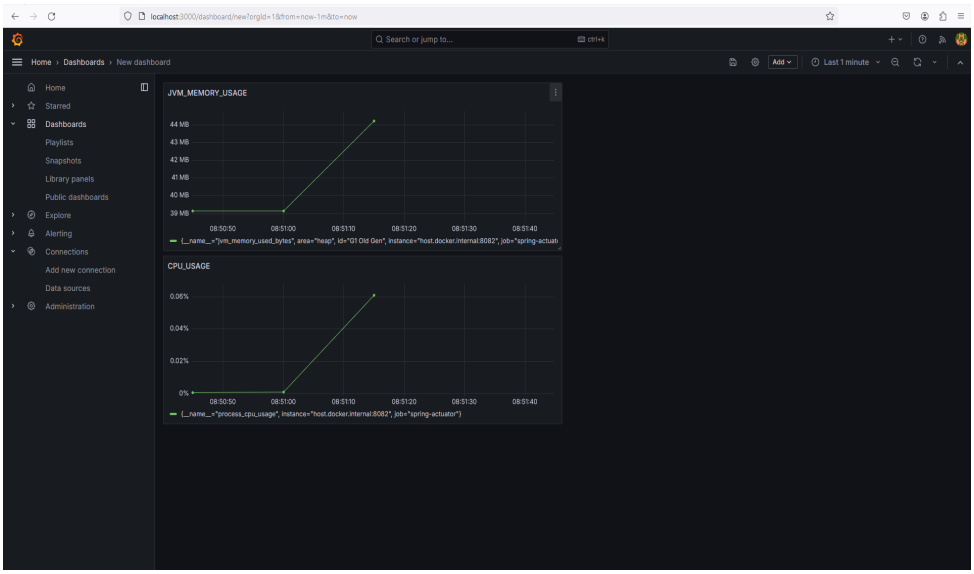
Fonte: Autor

Figura 7 – *Throughput*

Fonte: Autor

– **Eficiência de Recursos**: O uso de CPU e memória foi otimizado, com o microsserviço utilizando em média 0.06% da capacidade da CPU e 44 MB de memória alocada. Esses resultados garantem uma operação eficiente e estável. A Figura 8 ilustra a eficiência de recursos, mostrando o uso de CPU e memória durante os testes.

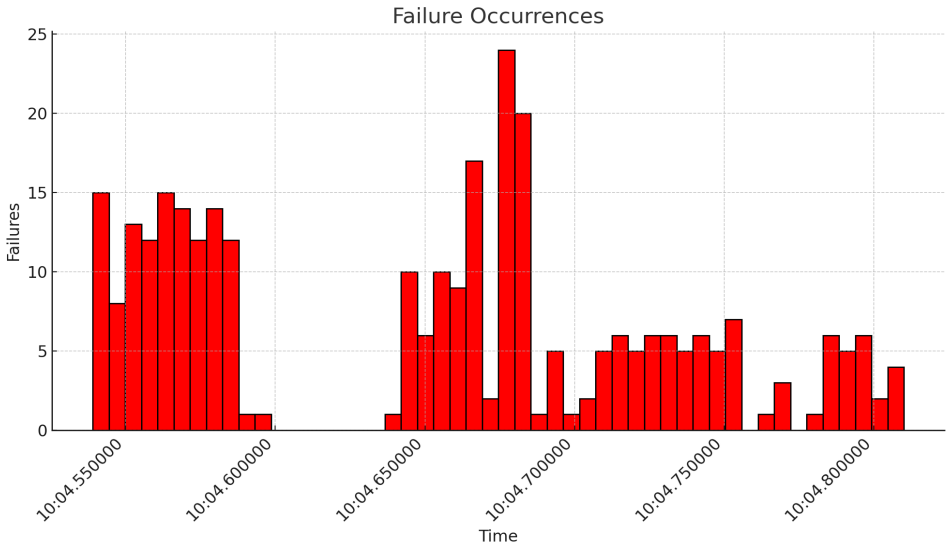
Figura 8 – Eficiência de Recursos



Fonte: Autor

– **Estabilidade**: Registramos o número e a natureza das falhas ocorridas, identificando um total de 304 falhas durante os testes de estresse. A grande maioria dessas falhas ocorreu devido a limitações ao gerenciar múltiplas conexões ao banco de dados MySQL em um ambiente Docker local. Essas limitações na rede interna do Docker resultaram em falhas nas conexões, onde em alguns momentos a aplicação não conseguiu estabelecer novas conexões com o banco de dados MySQL. Grande parte do impacto ocasionado por essas falhas foi sanada na PoC 3 com a implementação do Circuit Breaker. A Figura 9 ilustra a ocorrência das falhas durante o teste.

Figura 9 – Occorência de Falhas



Fonte: Autor

6.1.1.4 Conclusão da Poc 1

A partir dos dados analisados, observa-se que a latência média do microsserviço está razoavelmente baixa, indicando um bom tempo de resposta. O *throughput* é bom, mas pode ser melhorado. A presença de 304 falhas pode ser atribuída a fatores como as limitações do ambiente local com contêineres, o que sugere que esses números podem ser mais favoráveis em um ambiente de produção otimizado.

Grande parte dessas falhas ocorreu devido a limitações ao gerenciar múltiplas conexões ao banco de dados MySQL em um ambiente Docker local. Essas limitações na rede interna do Docker resultaram em falhas nas conexões, onde a aplicação não conseguiu estabelecer novas conexões com o banco de dados MySQL devido à exaustão dos recursos de rede.

Embora os dados tenham sido coletados em um ambiente local rodando em contêineres, o que pode gerar diferenças no desempenho em comparação com um ambiente de produção real, os resultados obtidos são relevantes para simular cenários de uso em produção. Fatores como configuração da infraestrutura, balanceamento de carga e tráfego de rede podem impactar os resultados. Contudo, espera-se que um ambiente de produção otimizado, com melhor configuração e recursos disponíveis, apresente resultados ainda mais favoráveis, com menor impacto sobre o *throughput* e menos falhas observadas."

Além disso, grande parte do impacto ocasionado por essas conexões foi sanada na PoC 3 com a implementação do Circuit Breaker. Esta solução ajudou a melhorar

a resiliência do sistema e a reduzir o número de falhas de conexão, proporcionando uma maior estabilidade e confiabilidade ao microsserviço.

Toda a documentação detalhada e a análise completa dos resultados desta PoC estão disponíveis no *link*: Documentação e Análise de Resultados da PoC 1.

6.1.2 Análise PoC 2

6.1.2.1 Definição da Poc 2

Essa prova de conceito tem como objetivo implementar um API Gateway para gerenciar as solicitações entre a interface do usuário e os microsserviços, demonstrando as vantagens em termos de segurança, eficiência e gerenciamento de tráfego.

6.1.2.2 Definição dos Requisitos da PoC 2

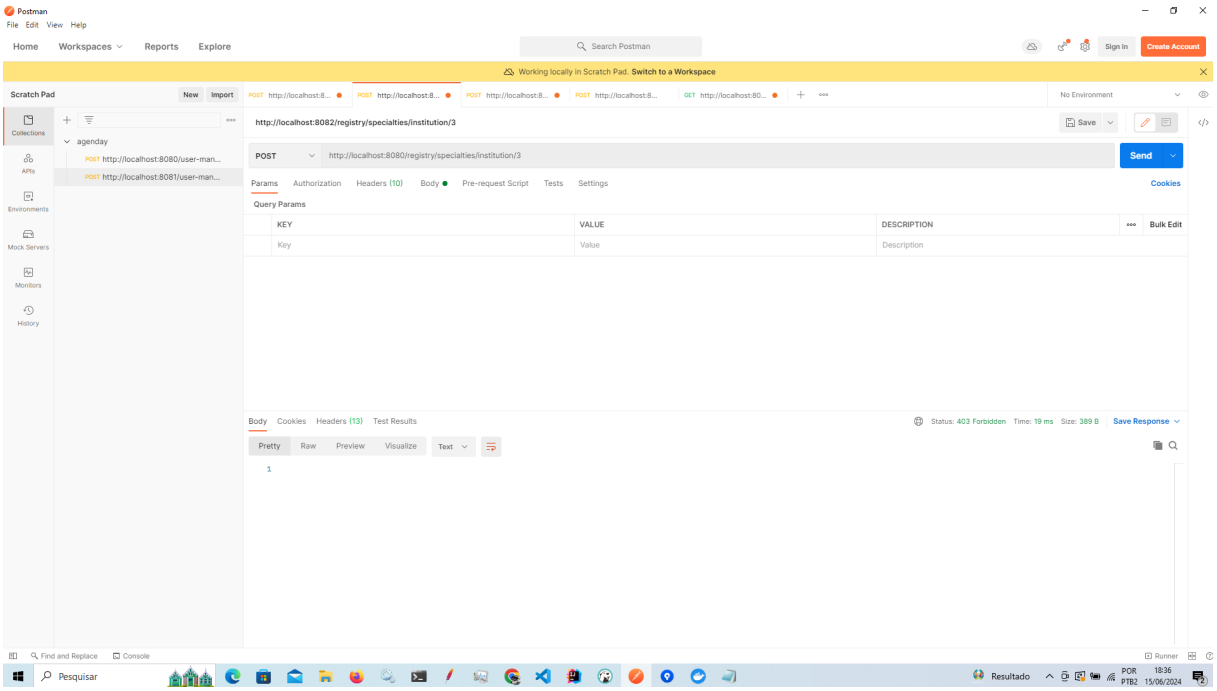
Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- Desenvolver um API Gateway como ponto de entrada único para todas as solicitações da aplicação.
- Configurar o Gateway para rotear solicitações para os microsserviços apropriados.
- Implementar funcionalidades de segurança, como autenticação e autorização, no Gateway.

6.1.2.3 Análise dos Resultados da PoC 2

1. **Eficácia na proteção contra solicitações mal-intencionadas**: Para testar a eficácia do API Gateway na proteção contra solicitações mal-intencionadas, realizamos os seguintes testes:
    1. Tentamos acessar o endpoint /registry/specialties/institution/3 na porta 8080 sem fornecer um token JWT.
    2. Conforme a Figura 10 o Gateway retornou um status 403 (Forbidden), indicando que a solicitação foi barrada devido à ausência de autenticação.

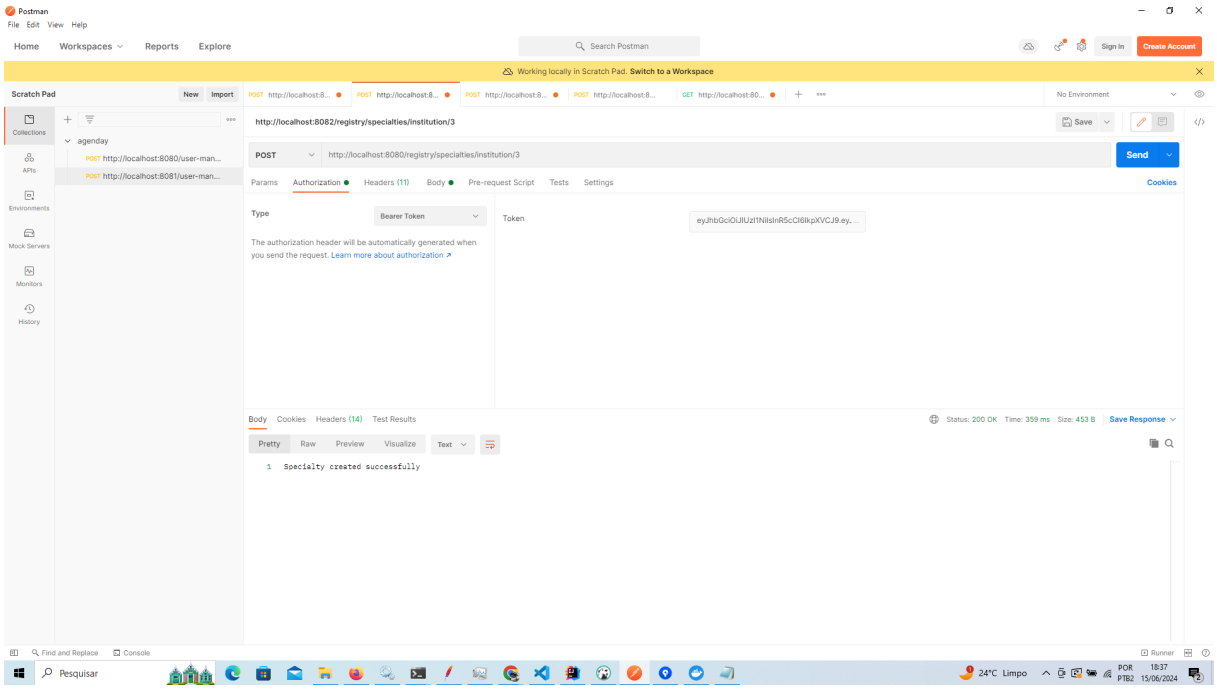
Figura 10 – Teste de Segurança Reprovado Gateway



Fonte: Autor

3. Realizamos uma requisição de login para obter o token JWT necessário para autenticação.
4. Utilizando o token JWT obtido na etapa anterior, tentamos acessar novamente o endpoint "/registry/specialties/institution/3".
5. Conforme a Figura 11 a requisição foi bem-sucedida e retornou um status 200 (OK), indicando que o acesso foi permitido com a autenticação correta.

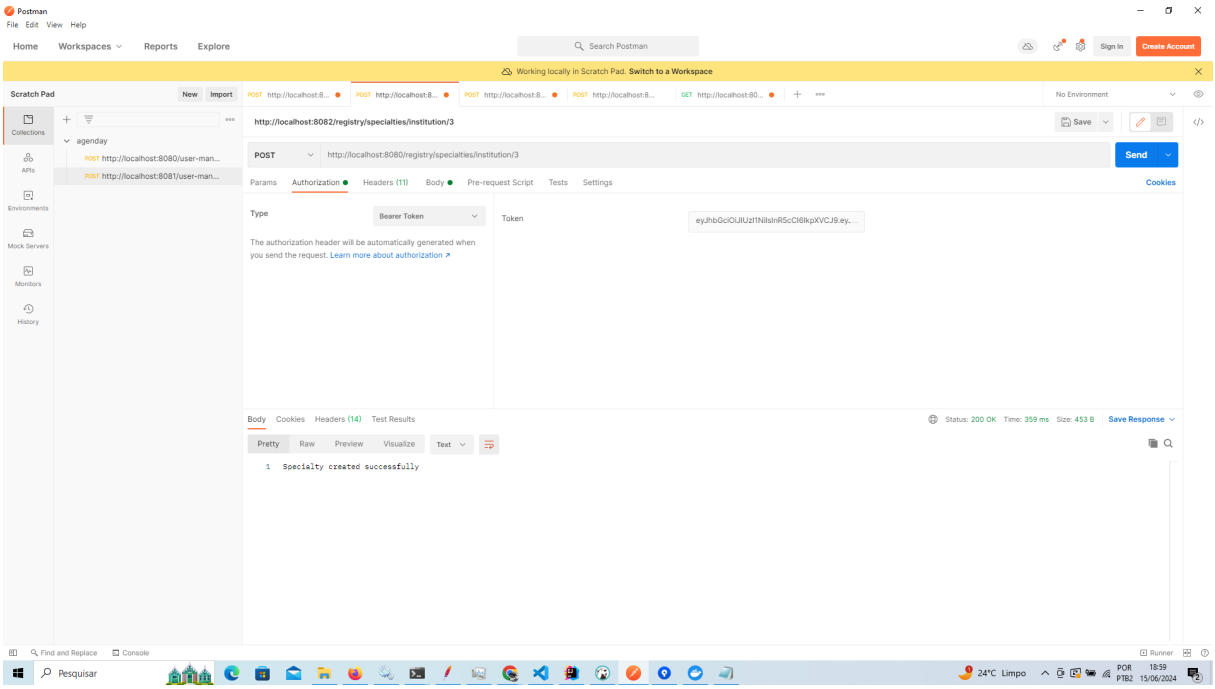
Figura 11 – Teste de Segurança Aprovado Gateway



Fonte: Autor

2. **Facilidade de gerenciamento de diferentes APIs**: Para facilitar o gerenciamento de diferentes microsserviços, configuramos o API Gateway para rotear todas as solicitações através de uma única aplicação. Isso centraliza o ponto de entrada para todos os microsserviços, simplificando o acesso e o gerenciamento das APIs para o front-end. A imagem abaixo mostra a requisição ao serviço de login, com o gateway redirecionando para o serviço http://localhost:8081 e retornando a resposta, conforme ilustrado na Figura 12.

Figura 12 – Roteamento Gateway



Fonte: Autor

6.1.2.4 Conclusão da Poc 2

A implementação do API Gateway demonstrou-se eficaz em centralizar e gerenciar as solicitações entre a interface do usuário e os microsserviços. A configuração adequada do gateway permitiu a roteamento eficiente das requisições, garantindo segurança através da autenticação e autorização baseada em tokens JWT.

Os testes realizados mostraram que o API Gateway é capaz de proteger contra solicitações mal-intencionadas, barrando acessos não autenticados e permitindo apenas solicitações devidamente autenticadas. A configuração de CORS global facilitou a comunicação entre o *front-end* e o *back-end*, eliminando problemas de bloqueio de origem cruzada.

Além disso, o uso do API Gateway simplificou significativamente o gerenciamento das APIs, permitindo que o *front-end* interaja com diversos microsserviços através de um único ponto de entrada. Isso não apenas facilita a manutenção e a escalabilidade da aplicação, mas também melhora a eficiência operacional.

Em resumo, a implementação do API Gateway atendeu aos requisitos propostos e demonstrou os benefícios em termos de segurança, eficiência e gerenciamento de tráfego, validando a viabilidade da solução em um ambiente de microsserviços.

Toda a documentação detalhada e a análise completa dos resultados desta PoC estão disponíveis no *link*: Documentação e Análise de Resultados da PoC 2.

6.1.3 Análise PoC 3

6.1.3.1 Definição da PoC 3

Essa prova de conceito tem o intuito de validar a implementação do padrão Circuit Breaker para aumentar a resiliência do sistema, prevenindo falhas em cascata entre microsserviços.

6.1.3.2 Definição dos Requisitos da PoC 3

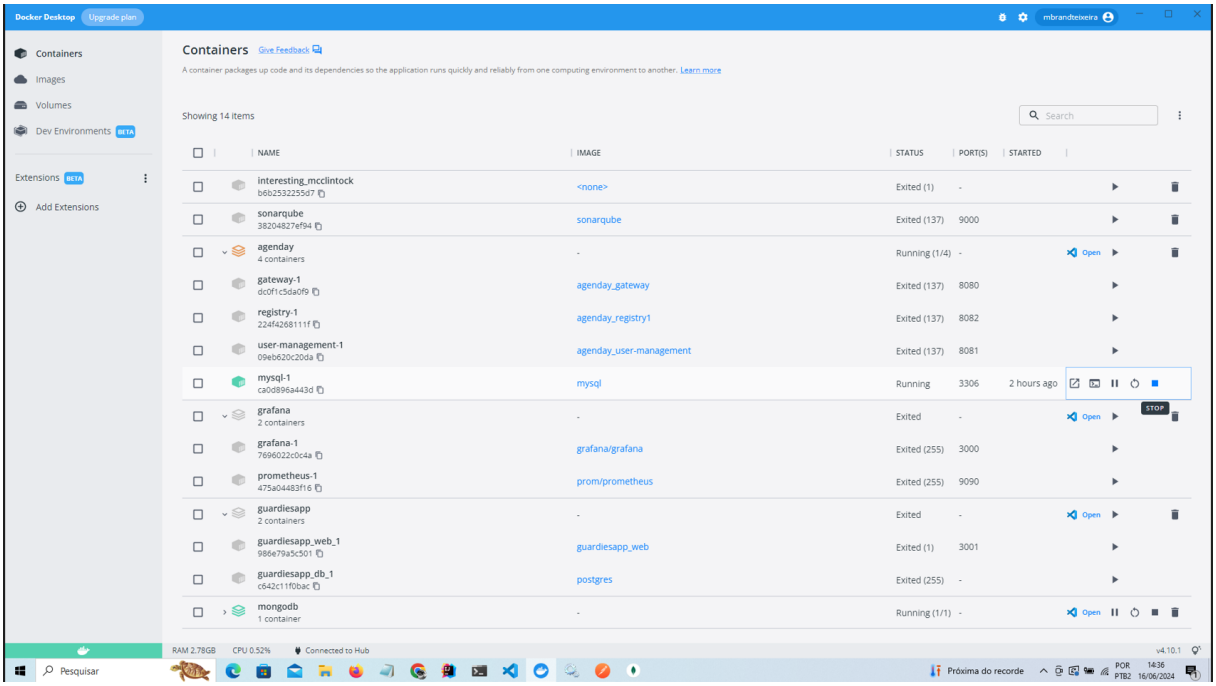
Os seguintes requisitos devem ser atendidos para que essa prova de conceito seja considerada concluída:

- Escolher um microsserviço crítico na aplicação para implementar o Circuit Breaker.
- Configurar o Circuit Breaker para detectar falhas e interromper automaticamente as solicitações, evitando sobrecarga no serviço.
- Implementar uma lógica de *fallback* para oferecer uma resposta alternativa quando um serviço falhar.

6.1.3.3 Análise dos Resultados da PoC 3

- **Simulação de Falhas**: Simulamos falhas derrubando a conexão com o banco de dados MySQL para testar a resiliência do sistema. Observamos que o Circuit Breaker entrou em ação conforme esperado. Durante o período em que o Circuit Breaker estava ativo, as requisições foram redirecionadas para um banco de dados NoSQL, onde foram armazenadas para reprocessamento ao final do dia. Essa abordagem permitiu a resiliência do sistema.
    1. A Figura 13 mostra a interface do Docker Desktop, onde o container do MySQL foi parado manualmente para simular a falha.

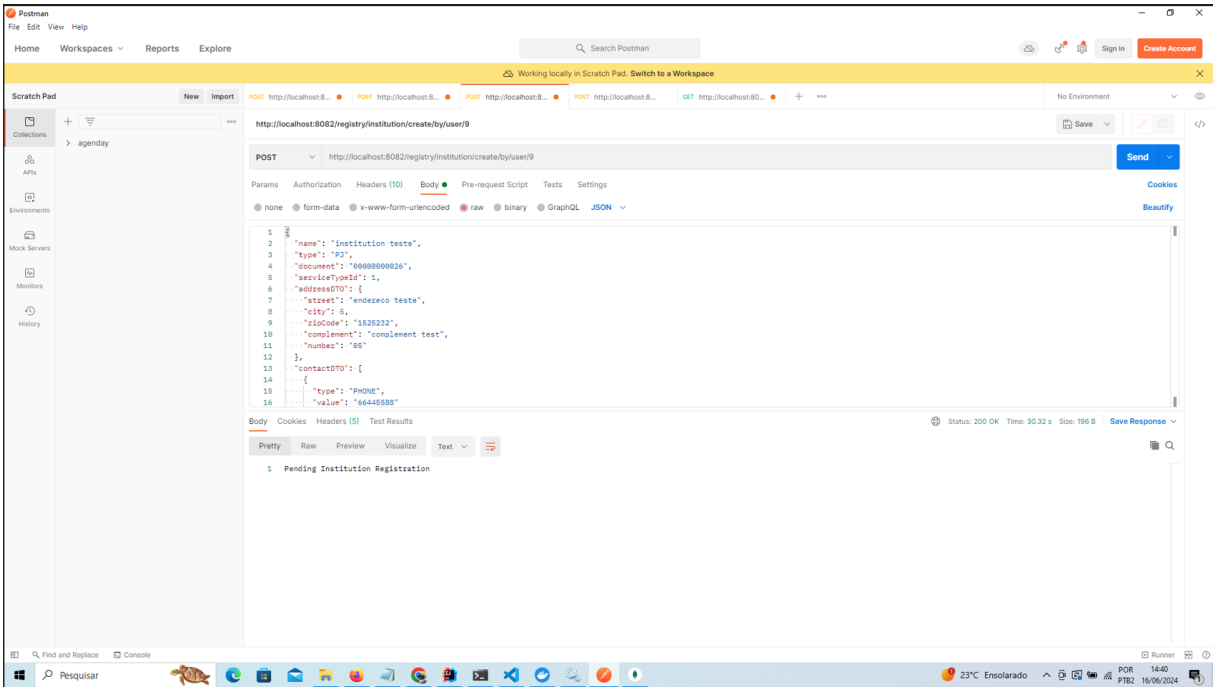
Figura 13 – Docker Desktop



Fonte: Autor

2. A Figura 14 apresenta a interface do Postman exibindo a requisição ao serviço de criação de instituição, demonstrando que a requisição foi redirecionada e tratada corretamente, mesmo durante a falha do banco de dados MySQL.

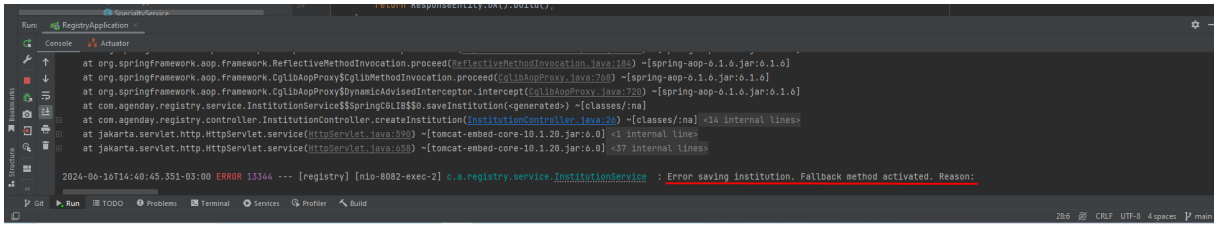
Figura 14 – Interface do Postman



Fonte: Autor

3. A Figura 15 mostra o *log* de erros onde é possível ver a mensagem de ativação do método fallback, indicando que a aplicação redirecionou corretamente as requisições para o banco de dados NoSQL.

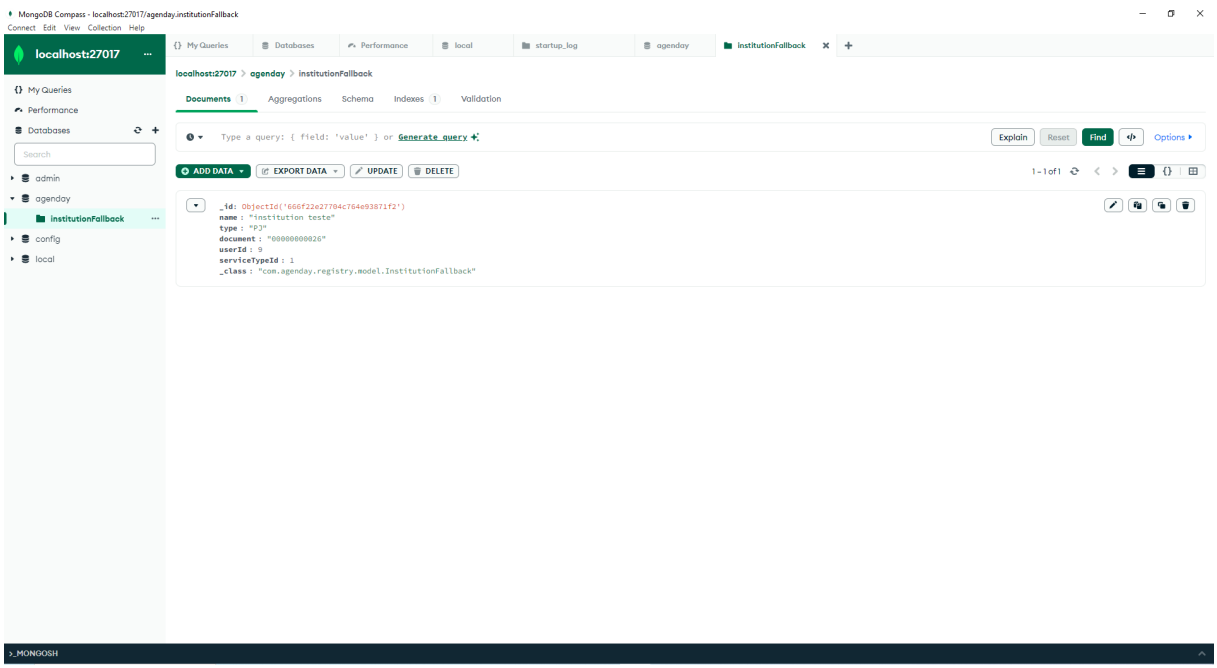
Figura 15 – Logs



Fonte: Autor

4. A Figura 16 mostra a interface do MongoDB Compass exibindo os dados redirecionados para o banco de dados NoSQL após a ativação do Circuit Breaker.

Figura 16 – Interface do MongoDB



Fonte: Autor

– **Eficiência do Mecanismo de *Fallback***: O mecanismo de *fallback* garantiu que as funcionalidades essenciais continuassem operando, mesmo durante as falhas, mantendo a disponibilidade do sistema. As requisições armazenadas no banco de dados NoSQL foram reprocessadas com sucesso, assegurando que nenhuma informação fosse perdida durante o período de falha.

– **Estabilidade Geral do Sistema**: A implementação do Circuit Breaker resultou em uma melhoria significativa na estabilidade geral do sistema. Reduzimos

as falhas em cascata. Essa abordagem assegurou uma experiência mais estável e confiável para os usuários.

6.1.3.4 Conclusão da Poc 3

O mecanismo de fallback garantiu que as funcionalidades essenciais continuassem operando, mesmo durante as falhas, mantendo a disponibilidade do sistema. As requisições armazenadas no banco de dados NoSQL foram reprocessadas com sucesso, assegurando que nenhuma informação fosse perdida. A implementação do Circuit Breaker resultou em uma melhoria significativa na estabilidade geral do sistema. Reduzimos as falhas em cascata. Essa abordagem assegurou uma experiência mais estável e confiável para os usuários.

Toda a documentação detalhada e a análise completa dos resultados desta PoC estão disponíveis no *link*: Documentação e Análise de Resultados da PoC 3.

6.2 Análise dos Resultados do Formulário

Com o intuito de avaliar a eficácia e a utilidade do nosso catálogo de boas práticas para o desenvolvimento e implantação de microsserviços, conduzimos uma pesquisa por meio de um formulário online. O formulário pode ser acessado através do seguinte *link*: Formulário de Pesquisa.

6.2.1 Metodologia

O questionário foi elaborado para obter *feedback* detalhado dos usuários sobre o catálogo de boas práticas, sendo fundamental para compreender a percepção dos profissionais de TI e identificar áreas de melhoria. Os participantes foram profissionais com variados níveis de experiência em arquitetura de microsserviços, escolhidos pela relevância de suas experiências e capacidade de fornecer *insights*. Eles utilizaram o catálogo de boas práticas de forma independente e responderam ao questionário *online* sem assistência direta dos autores.

6.2.2 Formulário

As Tabelas 10, 11, 12, 13, 14 e 15 apresentam respectivamente as perguntas feitas no formulário e as respostas dos participantes da pesquisa, conforme apresentado na Seção 6.2. A pesquisa foi conduzida para avaliar o catálogo de boas práticas para o desenvolvimento e implantação de microsserviços.

Tabela 10 – Perguntas do questionário

Número	**Pergunta**	**Relevância**
1	Como você classificaria sua experiência geral com o catálogo?	Obter uma visão geral da satisfação do usuário, fornecendo um indicador inicial da eficácia do catálogo.
2	O catálogo foi fácil de navegar e utilizar?	Avaliar a usabilidade, garantindo que os usuários possam acessar e compreender facilmente as informações apresentadas.
3	O conteúdo do catálogo foi útil e relevante?	Determinar se o conteúdo atende às necessidades dos usuários e se as boas práticas são aplicáveis ao seu contexto de trabalho.
4	Quais seções ou tópicos foram mais úteis para você?	Identificar as partes do catálogo que agregam mais valor, ajudando a focar em áreas de destaque ou necessidade de melhoria.
5	Há algum tópico que você gostaria de ver adicionado ao catálogo?	Coletar sugestões para expansão do conteúdo, garantindo que o catálogo evolua para cobrir mais aspectos relevantes.
6	Você encontrou algum problema ou erro ao usar o catálogo? Se sim, por favor, descreva.	Identificar problemas técnicos ou de conteúdo que possam afetar a experiência do usuário, direcionando esforços para correção.
7	Você tem alguma sugestão de melhoria?	Permitir feedback aberto, permitindo que os usuários sugiram melhorias baseadas em suas experiências específicas.
8	Qual é o seu cargo atual?	Entender o perfil profissional dos respondentes, ajudando a contextualizar as respostas e verificar se o catálogo atende a diferentes níveis de experiência.
9	Quantos anos de experiência você tem nesse ou em outro cargo semelhante?	Avaliar a experiência dos usuários com microsserviços, proporcionando insights sobre como diferentes níveis de experiência influenciam a percepção do catálogo.

Fonte: Autor

Tabela 11 – Resumo dos questionários respondidos pelos participantes (Parte 1)

Participante	**Experiência Geral**	**Facilidade de Navegação**	**Utilidade do Conteúdo**
Participante 1	Excelente	Muito Fácil	Útil
Participante 2	Excelente	Fácil	Muito Útil
Participante 3	Boa	Fácil	Muito Útil
Participante 4	Excelente	Muito Fácil	Útil
Participante 5	Boa	Muito Fácil	Útil
Participante 6	Boa	Muito Fácil	Útil
Participante 7	Boa	Fácil	Útil

Fonte: Autor

Tabela 12 – Resumo dos questionários respondidos pelos participantes (Parte 2)

Participante	**Seções Úteis**	**Tópicos Adicionais**	**Problemas ou Erros**
Participante 1	-	-	-
Participante 2	Performance e Otimização	Não	Não
Participante 3	-	-	-
Participante 4	Comunicação e Integração	-	-
Participante 5	Resiliência e Estabilidade	Não	Não
Participante 6	Boas práticas de arquitetura e design	-	Não
Participante 7	-	Qualidade e Manutenção	Nenhum problema

Fonte: Autor

Tabela 13 – Resumo dos questionários respondidos pelos participantes (Parte 3)

Participante	**Sugestões de Melhoria**
Participante 1	-
Participante 2	Não
Participante 3	-
Participante 4	-
Participante 5	Eu gosto bastante de utilizar imagens em documentos
Participante 6	Não
Participante 7	Está excelente

Fonte: Autor

Tabela 14 – Resumo dos questionários respondidos pelos participantes (Parte 4)

Participante	Cargo Atual
Participante 1	Estudante de engenharia automotiva
Participante 2	Engenheiro de Software
Participante 3	Desenvolvedor
Participante 4	Desenvolvedor
Participante 5	Desenvolvedor
Participante 6	Desenvolvedor
Participante 7	Desenvolvedor

Fonte: Autor

Tabela 15 – Resumo dos questionários respondidos pelos participantes (Parte 5)

Participante	Experiência
Participante 1	Menos de 1 ano
Participante 2	1-3 anos
Participante 3	Mais de 10 anos
Participante 4	7-10 anos
Participante 5	1-3 anos
Participante 6	4-6 anos
Participante 7	1-3 anos

Fonte: Autor

6.2.3 Resultados da Pesquisa

Os resultados da pesquisa indicam que a maioria dos participantes considerou o catálogo como uma ferramenta útil e bem estruturada. A análise detalhada dos resultados das respostas pode ser encontrada no link a seguir: Pesquisa de Usabilidade.

Os principais pontos destacados pelos respondentes incluem:

- **Clareza das Informações:** A maioria dos participantes relatou que as informações presentes no catálogo são claras e de fácil compreensão.
- **Aplicabilidade das Práticas:** Muitos dos respondentes afirmaram que as práticas recomendadas são aplicáveis ao seu ambiente de trabalho.
- **Utilidade Geral:** A maioria dos participantes consideraram o catálogo como uma ferramenta útil para a implementação de microsserviços.

Além disso, foram coletados diversos feedbacks qualitativos que destacaram a necessidade de exemplos práticos adicionais e casos de uso para facilitar a aplicação das boas práticas recomendadas.

6.2.4 Conclusões da Análise

A pesquisa confirmou a relevância e a eficácia do catálogo de boas práticas para o desenvolvimento e implantação de microsserviços. Os feedbacks recebidos serão utilizados para melhorar ainda mais o conteúdo e torná-lo uma referência ainda mais valiosa para profissionais da área.

7
Conclusão

Este capítulo apresenta a conclusão do trabalho. Inicialmente, retomamos o contexto que motivou a realização deste estudo. Em seguida, apresentamos a status, fornecendo uma visão geral dos objetivos do trabalho, da questão de pesquisa e dos principais comportamentos revelados ao longo do Estudo Exploratório. Finalmente, oferecemos uma perspectiva sobre os futuros trabalhos, enfatizando as oportunidades de aprimoramento e evolução deste Estudo Exploratório.

7.1 Contexto

Nos últimos anos, a arquitetura de microsserviços emergiu como uma solução vital para organizações que buscam agilidade e escalabilidade em seus sistemas de software. A abordagem de decompor uma aplicação em um conjunto de serviços menores, cada um executando um processo de negócios único e operando de maneira independente, tem sido amplamente adotada devido à sua flexibilidade e capacidade de acelerar o desenvolvimento (VIEIRA, 2023). O desenvolvimento eficaz requer uma compreensão sólida de princípios como desacoplamento de serviços, comunicação inter-serviços, gerenciamento de dados distribuídos e estratégias de monitoramento e resiliência. O Spring Boot, um ecossistema robusto para a construção de microsserviços, oferece inúmeras funcionalidades que simplificam esses processos, mas a falta de aplicação de boas práticas pode levar a resultados inconsistentes e problemas operacionais.

Este trabalho procurou preencher essa lacuna, fornecendo um catálogo de boas práticas que não apenas apoia o uso otimizado do Spring Boot, mas também aborda os desafios arquiteturais mais amplos dos microsserviços. Analisamos tendências atuais, estudos de caso e literatura acadêmica para garantir que o catálogo seja relevante, atualizado e aplicável em diversos contextos.

7.2 Status do Trabalho

Na fase inicial deste projeto, buscou-se documentar o embasamento teórico que fundamentou os princípios desta pesquisa. Com o objetivo de apresentar uma solução para um problema concreto, este trabalho evidencia, por meio de provas de conceito,

sua capacidade potencial de resolução. A tabela 16 oferece uma visão sobre o status das atividades realizadas na primeira etapa do TCC.

Tabela 16 – Atividades da Etapa 1 do TCC

Atividades	**Status**
Definir tema	Concluído
Realização da Pesquisa Bibliográfica	Concluído
Formular Proposta Inicial	Concluído
Estabelecer referencial teórico	Concluído
Definir suporte tecnológico	Concluído
Especificar metodologia	Concluído
Produzir prova de conceito	Concluído
Descrever Resultados Parciais	Concluído
Determinar proposta do trabalho	Concluído
Revisar monografia	Concluído
Apresentar TCC 1	Concluído

Fonte: Autor

Na fase subsequente do Trabalho de Conclusão de Curso (TCC), as tarefas compreenderam aprimoramentos na redação da monografia, destacando-se, sobretudo, o desenvolvimento da aplicação destinada à integração no âmbito acadêmico. A Tabela 17 apresenta o status das atividades executadas na segunda etapa do TCC.

Tabela 17 – Atividades da Etapa 2 do TCC

Atividades	**Status**
Realizar melhorias	Concluído
Realizar desenvolvimento da aplicação	Concluído
Análise de resultados	Concluído
Finalizar monografia	Concluído
Apresentar TCC 2	A fazer

Fonte: Autor

7.3 Futuros Trabalhos

Após atingir os objetivos estabelecidos neste trabalho, reconhecemos que o campo da engenharia de software está em constante evolução, com novas práticas e tec-

nologias emergindo regularmente. Portanto, pretendemos expandir este estudo de várias maneiras.

Primeiramente, planejamos desenvolver mais Provas de Conceito (PoCs) para validar novas práticas e abordagens que possam surgir. Essas PoCs futuras irão focar em diferentes aspectos da arquitetura de microsserviços, como segurança aprimorada, gestão avançada de dados distribuídos, técnicas de monitoramento mais robustas e estratégias de resiliência. Através dessas novas PoCs, esperamos continuar a fornecer insights práticos e detalhados que possam ser aplicados em cenários reais de desenvolvimento de software.

Além disso, o catálogo de boas práticas será continuamente atualizado para refletir as melhores e mais recentes práticas no desenvolvimento de microsserviços. À medida que novas práticas forem identificadas e validadas através de estudos de caso, literatura acadêmica e PoCs adicionais, elas serão incorporadas ao catálogo inicialmente por meio dos mantenedores e posteriormente pretendemos criar politicas de commit para que a comunidade possa contribuir por si só. Este processo de atualização contínua garantirá que o catálogo permaneça um recurso valioso e relevante para desenvolvedores e equipes de projeto que trabalham com Spring Boot.

Ao manter o catálogo atualizado e expandir o número de PoCs, nosso objetivo é criar uma base de conhecimento dinâmica e em constante crescimento que apoie a excelência no desenvolvimento de microsserviços. Esperamos que este trabalho contínuo contribua significativamente para a comunidade de desenvolvimento de software, oferecendo soluções práticas e testadas para os desafios complexos que surgem ao implementar arquiteturas de microsserviços.

Referências

ANDERSON, D. J. *Kanban: Enhancing Agility in Modern Development Practices*. 3rd. ed. [S.l.]: Agile Publications, 2021. Citado na página 56.

AppMaster. *Evolução do design da arquitetura de software*. 2023. Acessado em: 5 out. 2023. Disponível em: <https://appmaster.io/pt/blog/evolucao-do-design-da-arquitetura-de-software>. Citado na página 30.

ATLASSIAN. *Microsserviços versus arquitetura monolítica*. 2023. Acessado em: 6 out. 2023. Disponível em: <https://www.atlassian.com/br/microservices/microservices-architecture/microservices-vs-monolith>. Citado 3 vezes nas páginas 30, 31 e 32.

ATLASSIAN. *Princípios dos microsserviços*. 2023. Acessado em: 6 out. 2023. Disponível em: <https://www.atlassian.com/br/microservices>. Citado na página 35.

AWS. *Monolítico x microsserviços — Diferença entre arquiteturas de desenvolvimento de software*. 2023. Acessado em: 7 out. 2023. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-monolithic-and-microservices-architecture/>. Citado na página 37.

AWS. *O que são microsserviços?* 2023. Acessado em: 16 set. 2023. Disponível em: <https://aws.amazon.com/pt/microservices/>. Citado na página 24.

BERRIO-CHARRY, E.; VERGARA-VARGAS, J.; UMAñA-ACOSTA, H. A component-based evolution model for service-based software architectures. In: *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*. [S.l.: s.n.], 2020. p. 111–115. Citado na página 30.

CALCADO, P. *Building Products at SoundCloud – Part I: Dealing with the Monolith*. 2014. Disponível em: <https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-1-dealing-with-the-monolith>. Citado na página 44.

Cloud Native Computing Foundation. *Cloud Native Computing Foundation*. <https://www.cncf.io/>. Acesso em: 16 de setembro de 2024. Citado na página 35.

CONCEITO: Arquitetura de Software. 2023. Acessado em: 5 out. 2023. Disponível em: <https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/concepts/software_architecture_4269A354.html>. Citado na página 29.

ENGINEERING, U. *Rewriting Uber Engineering: The Opportunities Microservices Provide*. 2021. Disponível em: <https://eng.uber.com/building-tincup-microservice-implementation/>. Citado na página 44.

FALAHAH; SURENDRO, K.; SUNINDYO, W. D. Circuit breaker in microservices: State of the art and future prospects. *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, v. 1077, n. 1, p. 012065, feb 2021. Disponível em: <https://dx.doi.org/10.1088/1757-899X/1077/1/012065>. Citado na página 42.

FELIPE, L. *Introdução a Microsserviços*. 2020. Disponível em: <https://www.luisdev.com.br/2020/12/26/introducao-a-microsservicos/?gclid=EAIaIQobChMIi-vBsfG0-gIVCj-RCh2_mg9dEAAYBCAAEgLEffD_BwE>. Citado na página 38.

FLORES, P. V. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*, v. 11, p. 88339–88358, 2023. Citado na página 34.

FOWLER, M. *Microservices*. 2023. Acessado em: 19 set. 2023. Disponível em: <https://martinfowler.com/articles/microservices.html>. Citado 2 vezes nas páginas 25 e 54.

FOWLER, M. *Software Architecture Guide*. 2023. Acessado em: 16 set. 2023. Disponível em: <https://martinfowler.com/architecture/>. Citado na página 23.

GARLAN, D.; SHAW, M. An introduction to software architecture. In: *Advances in software engineering and knowledge engineering*. [S.l.]: World Scientific, 1993. p. 1–39. Citado na página 29.

GERHARDT, T. E.; SILVEIRA, D. T. *Métodos de pesquisa*. [S.l.]: Plageder, 2009. Citado 2 vezes nas páginas 51 e 52.

Git. *Git - git Documentation*. 2023. Acessado em: 9 out. 2023. Disponível em: <https://git-scm.com/docs/git>. Citado na página 47.

GITHUB. *GitHub Documentation*. 2023. Disponível em: <https://docs.github.com/pt/get-started/quickstart/hello-world>. Citado na página 47.

GUIMARãES, J. H. N. O. *Método para manutenção de sistema de software utilizando técnicas arquiteturais*. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, 9 2008. Mestrado em Sistemas Digitais. Citado na página 55.

IBM. *O que são microsserviços?* 2023. Acessado em: 6 out. 2023. Disponível em: <https://www.ibm.com/br-pt/topics/microservices>. Citado na página 35.

IBM. *O que é arquitetura de três camadas (tiers)*. 2023. Acessado em: 16 set. 2023. Disponível em: <https://www.ibm.com/br-pt/topics/three-tier-architecture>. Citado na página 23.

IBM. *O que é SOA (Arquitetura Orientada a Serviços)?* 2023. Acessado em: 16 set. 2023. Disponível em: <https://www.ibm.com/br-pt/topics/soa>. Citado 2 vezes nas páginas 23 e 33.

IHDE, S. *From a Monolith to Microservices + REST: The Evolution of LinkedIn's Service Architecture*. 2015. Disponível em: <https://www.infoq.com/presentations/linkedin-microservices-urn/>. Citado na página 44.

JOHNSON, R. In: *QCon Londres*. Londres: [s.n.], 2010. Citado na página 23.

KAHLMEYER-MERTENS, R. S. et al. *Como elaborar projetos de pesquisa: linguagem e método*. [S.l.]: FGV Editora, 2007. Citado na página 52.

MAURO, T. *Adopting Microservices at Netflix: Lessons for Architectural Design*. 2015. Disponível em: <https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/>. Citado na página 44.

MISSAO, C. K.; JR, E. D. B. *Desenvolvimento de uma metodologia de negócios para sistemas de informação relacionados com o gerenciamento da cadeia de su...* [S.l.], 2003. Relatório técnico. Citado 2 vezes nas páginas 55 e 56.

NEGRI, P. *API Gateway: O que é e qual a sua função?* 2021. Disponível em: <https://www.iugu.com/blog/api-gateway>. Citado na página 40.

NETO, M. P. M.; AUGUSTO, V. d. S. e. S. *Padrões para produção de aplicações utilizando Microsserviços*. Dissertação (Trabalho de Conclusão de Curso), 2021. Disponível em: <http://repositorio.ifg.edu.br/handle/prefix/700>. Citado na página 41.

Red Hat. *O que são microsserviços?* 2023. Acessado em: 16 set. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/microservices/what-are-microservices>. Citado na página 24.

Red Hat. *O que é SOA (Arquitetura Orientada a Serviços)?* 2023. Acessado em: 6 out. 2023. Disponível em: <https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-service-oriented-architecture>. Citado na página 33.

SACHDEVA, S. Scrum methodology. *Int. J. Eng. Comput. Sci*, v. 5, n. 16792, p. 16792–16800, 2016. Citado 2 vezes nas páginas 56 e 57.

SPRING Boot Reference Documentation. 2023. Acessado em: Out. 9, 2023. Disponível em: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#legal>. Citado na página 48.

SPRING FRAMEWORK. *Spring Framework Documentation*. 2023. Acessado em: Out 9, 2023. Disponível em: <https://docs.spring.io/spring-framework/reference/index.html>. Citado na página 48.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, v. 4, n. 5, p. 22–32, 2017. Citado na página 35.

TechTarget. *What is monolithic architecture in software?* 2023. Accessed on: September 16, 2023. Disponível em: <https://www.techtarget.com/whatis/definition/monolithic-architecture>. Citado na página 24.

THÖNES, J. Microservices. *IEEE Software*, v. 32, n. 1, p. 116–116, jan 2015. Citado na página 34.

VELEPUCHA, V.; FLORES, P. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*, v. 11, p. 88339–88358, 2023. Citado 3 vezes nas páginas 39, 41 e 43.

VIEIRA, H. *Microsserviços: melhores práticas, vantagens e exemplos*. 2023. Objective. Available online: <https://www.objective.com.br/insights/microsservicos/>. Citado 2 vezes nas páginas 63 e 93.

WHAT are microservices? 2023. Accessed on: September 16, 2023. Disponível em: <http://microservices.io/index.html>. Citado 2 vezes nas páginas 24 e 54.

WHAT is GitBook? 2023. Acessado em: Out 9, 2023. Disponível em: <https://docs.gitbook.com/>. Citado na página 48.

APÊNDICE A – Apêndices

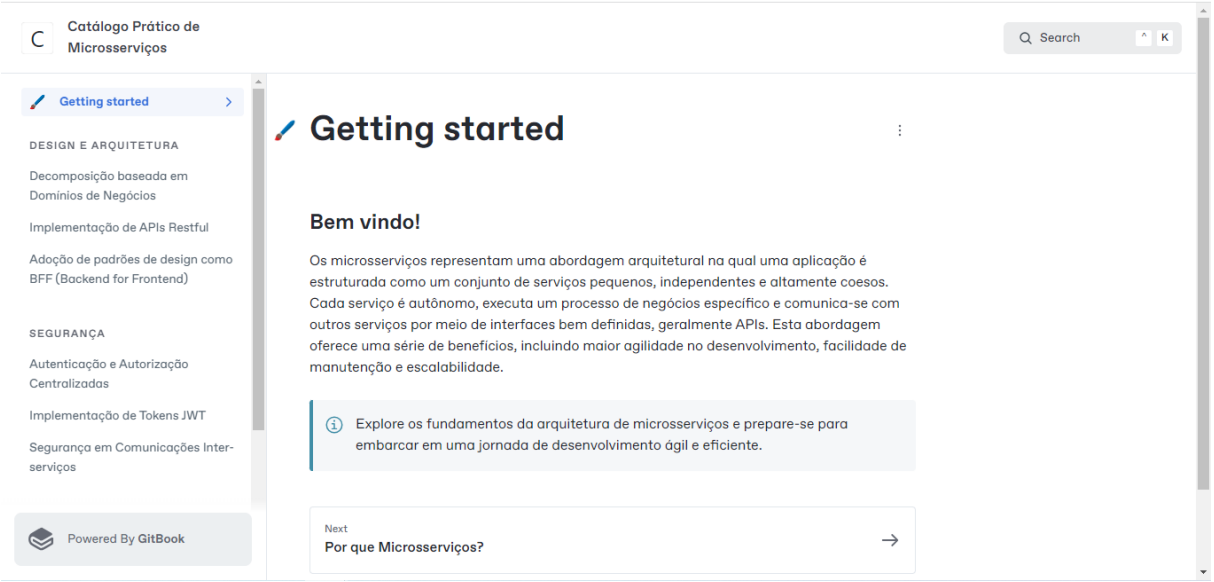
Este apêndice contém prints do início da prototipação do catálogo e uma série de perguntas e seus respectivos resultados obtidos através de um questionário aplicado. O objetivo é apresentar as opiniões e sugestões dos participantes sobre o catálogo desenvolvido, avaliando aspectos como facilidade de uso, relevância do conteúdo e sugestões de melhoria.

As perguntas do questionário foram cuidadosamente elaboradas para abranger diferentes aspectos da experiência dos usuários com o catálogo, desde a navegação e usabilidade até a relevância do conteúdo e sugestões para melhorias. O questionário buscou capturar tanto feedback quantitativo quanto qualitativo, permitindo uma avaliação abrangente das percepções dos usuários.

A.1 Protótipo

Esta é uma imagem meramente ilustrativa, mostrando o local e a ferramenta utilizados para a criação do catálogo no TCC.

Figura 17 – Protótipo do sistema



Fonte: Autor

A.2 Perguntas Formulário

A.2.1 Perguntas 1 e 2

A primeira pergunta visava entender o nível de consentimento dos participantes em participar do estudo e se estavam confortáveis em compartilhar suas opiniões de forma voluntária. A segunda pergunta buscou avaliar a experiência geral dos usuários com o catálogo, medindo a satisfação geral com a ferramenta.

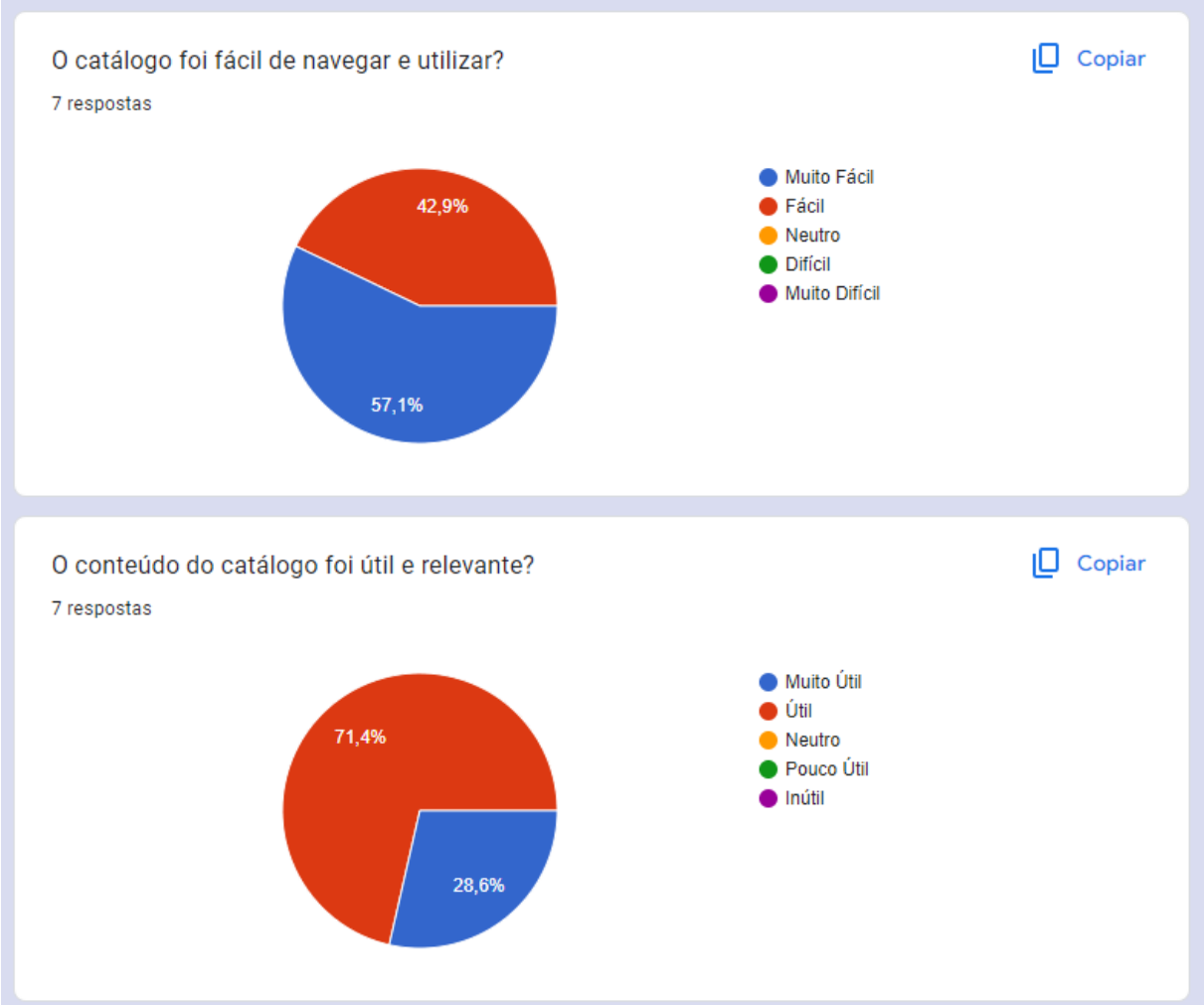
Figura 18 – Perguntas 1 e 2

Fonte: Autor

A.2.2 Perguntas 3 e 4

A terceira pergunta foi elaborada para entender a facilidade de navegação e uso do catálogo, permitindo identificar possíveis dificuldades enfrentadas pelos usuários. A quarta pergunta buscou avaliar a utilidade e relevância do conteúdo do catálogo, verificando se ele atende às necessidades dos usuários.

Figura 19 – Perguntas 3 e 4

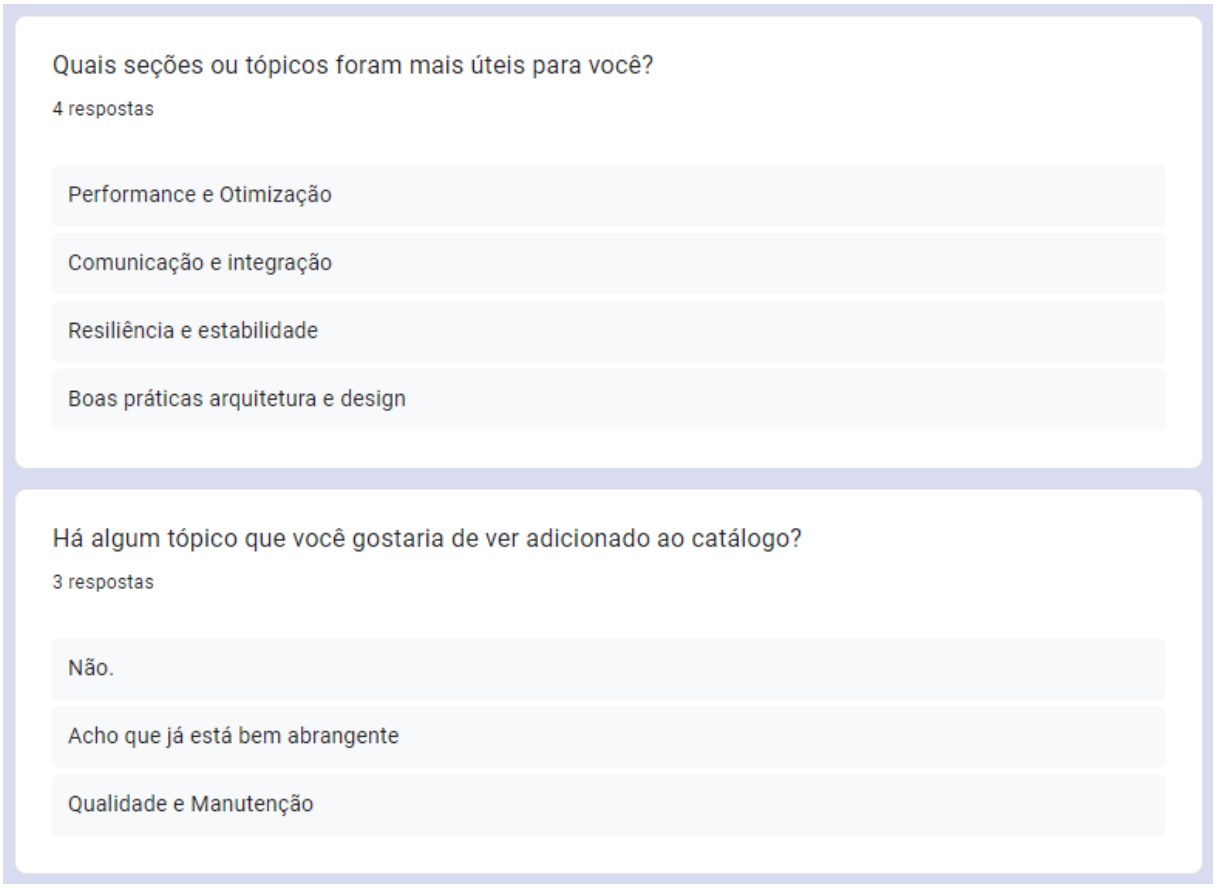


Fonte: Autor

A.2.3 Perguntas 5 e 6

A quinta pergunta foi incluída para identificar quais seções ou tópicos do catálogo foram considerados mais úteis pelos usuários, ajudando a destacar os pontos fortes do conteúdo. A sexta pergunta buscou coletar sugestões de melhorias dos usuários, possibilitando identificar áreas de melhoria e futuras atualizações do catálogo.

Figura 20 – Perguntas 5 e 6

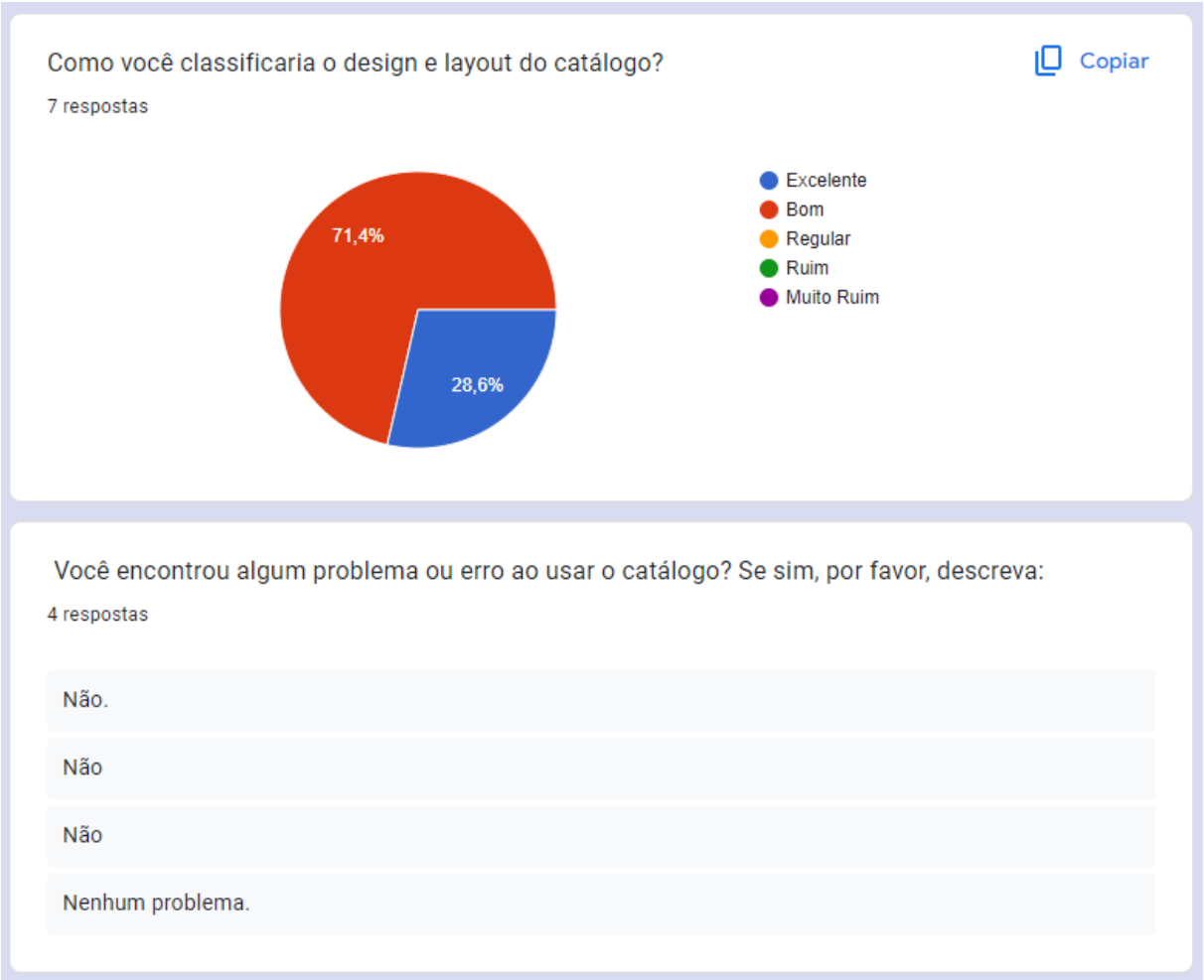


Fonte: Autor

A.2.4 Perguntas 7 e 8

A sétima pergunta foi formulada para classificar o design e layout do catálogo, permitindo avaliar a estética e a apresentação visual da ferramenta. A oitava pergunta teve como objetivo identificar problemas ou erros encontrados ao usar o catálogo, fornecendo uma visão das dificuldades técnicas ou funcionais enfrentadas pelos usuários.

Figura 21 – Perguntas 7 e 8

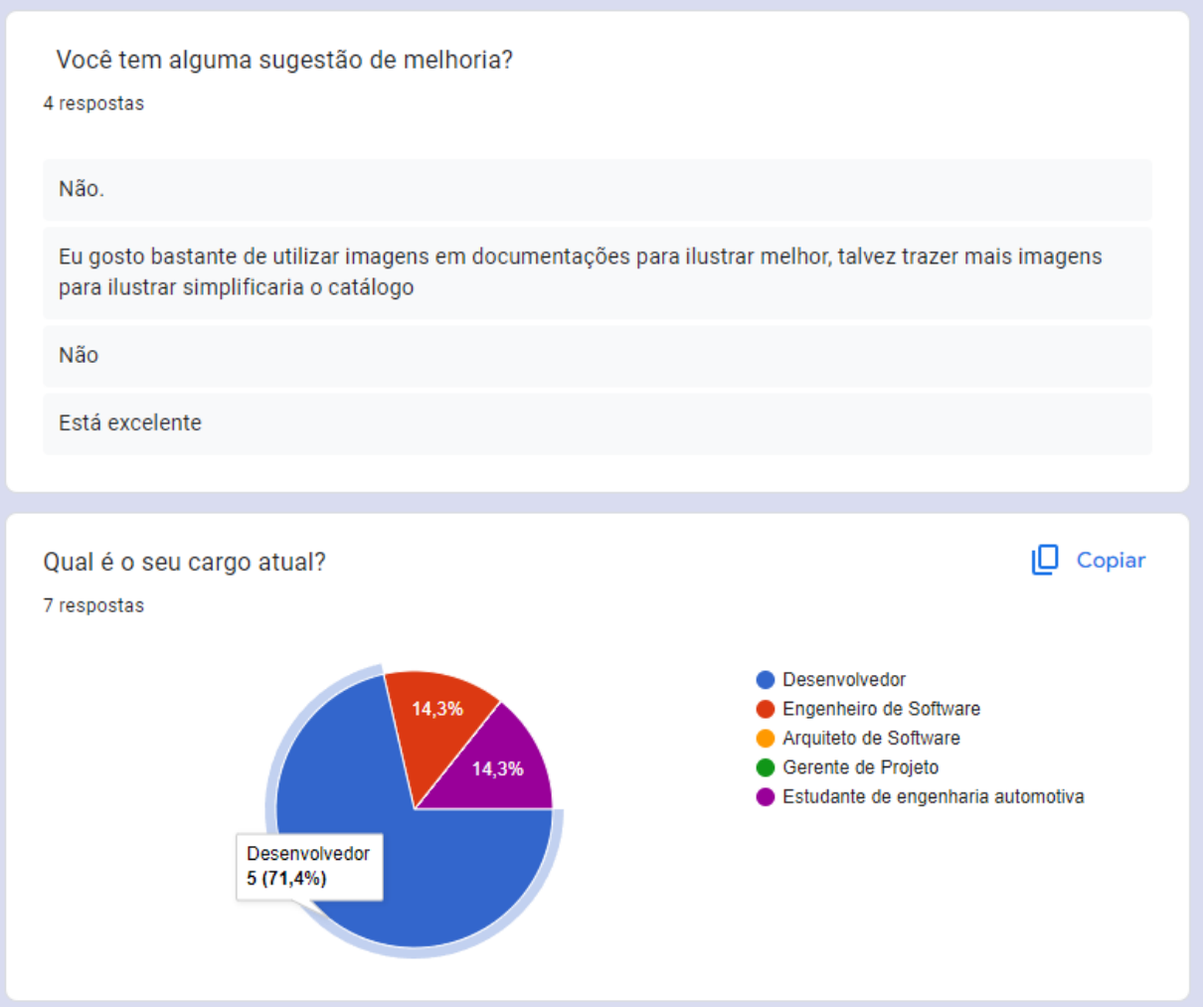


Fonte: Autor

A.2.5 Perguntas 9 e 10

A nona pergunta buscou obter sugestões específicas de melhoria, possibilitando identificar ajustes e aprimoramentos necessários. A décima pergunta coletou informações sobre o cargo atual dos participantes, ajudando a entender o perfil dos respondentes e suas áreas de atuação.

Figura 22 – Perguntas 9 e 10

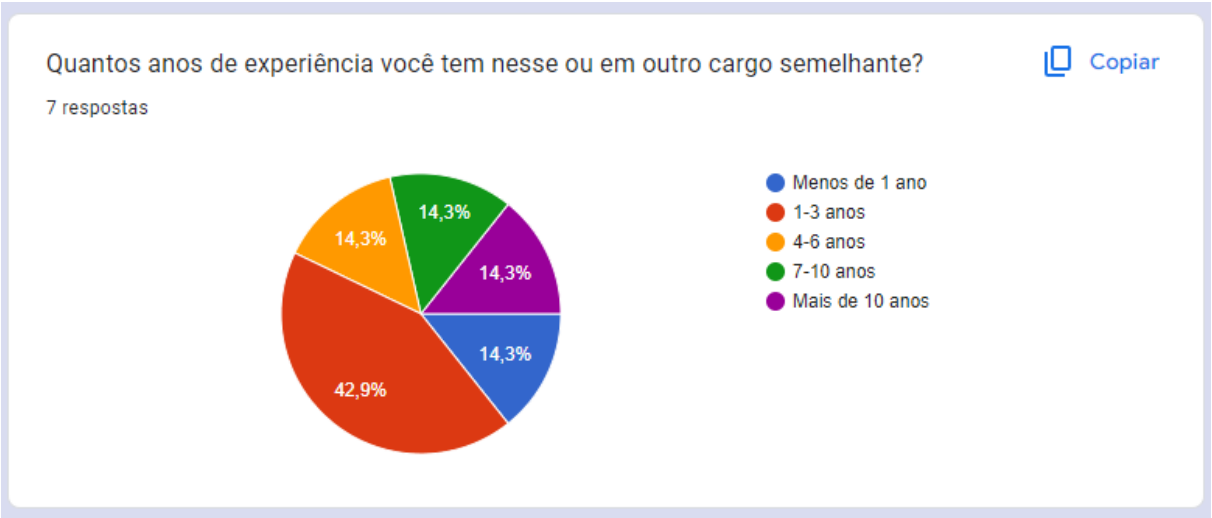


Fonte: Autor

A.2.6 Pergunta 11

A décima primeira pergunta buscou coletar informações sobre a experiência dos participantes em seus cargos atuais ou similares, ajudando a entender o nível de experiência e conhecimento dos usuários.

Figura 23 – Pergunta 11



Fonte: Autor