Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados

Autor: Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho

Orientadora: Profa. Dra. Milene Serrano

Orientador: Prof. Dr. Maurício Serrano

Brasília, DF

2024

Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho

Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados

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

Orientadora: Profa. Dra. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2024

Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho

Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados/ Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho. – Brasília, DF, 2024-

138 p. : il. (algumas color.) ; 30 cm.

Orientadora: Profa. Dra. Milene Serrano

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

1. Arquitetura Portas e Adaptadores. 2. Microsserviços. I. Profa. Dra. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados

CDU 02:141:005.6

Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho

Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados

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, 5 de Setembro de 2024:

Profa. Dra. Milene Serrano
Orientadora

Prof. Dr. Maurício Serrano
Coorientador

Profa. Dra. Elaine Venson
Examinador 1

Gabriel Davi Silva Pereira
Engenheiro de Sotfware na Praxent
Pós-Graduando em Arquitetura de Software (PUC-Minas)
Examinador 2

Brasília, DF
2024

Agradecimentos

Agradecemos primeiramente aos Professores Milene Serrano e Maurício Serrano, por aceitar nos orientar neste trabalho tão significativo, pelo enorme esforço e dedicação, além de paciência, em guiar-nos em todos os aspectos possíveis da execução deste trabalho, sendo grandes exemplos de profissionais e pessoas a seguir através de suas ações. E às nossas amizades que cultivamos durante nossa jornada acadêmica, deixando esse desafio mais leve e divertido.

Eu, Matheus Afonso, gostaria de expressar minha gratidão primeiramente à minha esposa, que desde o início do meu trabalho tem me apoiado, cuidado e amado em todos os detalhes, sendo foco de atenção e afeto. Agradeço à minha família, que sempre me apoiou e incentivou a dar o melhor de mim em todas as minhas ações. Agradeço, por fim, ao meu querido cão Raimundo, que é sempre motivo de alegria nos meus dias.

Eu, Thiago Carvalho, expresso meu profundo agradecimento e dedico este trabalho aos meus pais, José Orlando de Carvalho e Elisabete Mesquita Peres de Carvalho, por todo investimento em minha educação, pelos conselhos, cobranças e, acima de tudo, pelo amor incondicional que sempre me proporcionaram. Aos meus irmãos, agradeço pelas pequenas conversas e trocas de experiências que enriqueceram minha jornada. Aos amigos que sempre me fortaleceram durante essa caminhada, especialmente Rossicler, Saulo e Arnaud, agradeço por estarem presentes nos momentos fáceis e difíceis, crescendo junto comigo. Aos meus animaizinhos, Marx, Koda e Perseu, pela companhia e diversão em todos os momentos. E, por fim, dedico este trabalho à minha querida e amada namorada, Beatriz Liarte, por ser meu porto seguro ao longo de toda a minha jornada. Agradeço por dividir todos os sonhos comigo, por toda parceria, amor, companheirismo e pelo constante incentivo em ser um ser humano melhor.

Sem cada um de vocês, nada disso seria possível. Nossa eterna gratidão.

"São as perguntas que não sabemos responder que mais nos ensinam. Elas nos ensinam a pensar. Se você dá uma resposta a um homem, tudo o que ele ganha é um fato qualquer. Mas, se você lhe der uma pergunta, ele procurará suas próprias respostas."
(ROTHFUSS, 2011, p. 545)

Resumo

Nos últimos anos, a Arquitetura de *software* passou a ter uma relevância ainda maior, principalmente, visando auxiliar no gerenciamento da complexidade dos sistemas. Isso ocorre, pois a mesma define e organiza os elementos arquiteturais de um sistema. Através dela, define-se uma estrutura que satisfaça as necessidades técnicas, bem como operacionalize os serviços oferecidos de acordo com essas necessidades. Uma das mais conhecidas abordagens é a Arquitetura de Microsserviços, na qual é proporcionado o desenvolvimento da aplicação como um todo, considerando partes relevantes do sistema sendo implementadas "isoladamente", e encapsuladas em serviços de menor granularidade e com responsabilidades únicas. Em conjunto, essas partes, ou seja, esses microsserviços, compõem a aplicação. Na atualidade, dada a abrangência dos produtos de *software*, é pertinente considerar a combinação de vários modelos arquiteturais. Por exemplo, implementar um *software* combinando a Arquitetura de Microsserviços e a Arquitetura de Portas e Adaptadores. No caso dessa última, há necessidade de colocar o domínio de negócio no centro da aplicação, considerando um modelo em camadas, desenhado como círculos concêntricos. Adicionalmente, há necessidade de atender prazos, dada a dinâmica de mercado. Portanto, orientar-se por Reutilização de *Software*, na qual se faz uso de soluções prontas, em diferentes níveis de abstração (ex. modelagens e código), sendo essas soluções bem estabelecidas pela comunidade especializada, torna-se algo bem vantajoso. O presente trabalho é um estudo exploratório que revela comportamentos relevantes observados ao longo do desenvolvimento de uma aplicação *Back-end* orientada à combinação das Arquiteturas de Microsserviços e Portas e Adaptadores, com viés de Reutilização de *Software*. O estudo conferiu insumos, guiando-se por um método centrado em provas de conceito, que permitiram explorar várias nuances do desenvolvimento de uma típica aplicação *Back-end*. Os resultados obtidos a partir do desenvolvimento são apresentados, destacando-se aspectos positivos e negativos identificados durante a conclusão de cada prova de conceito, estando disponível então para consulta por terceiros interessados em conhecer as arquiteturas abordadas, que estão ganhando popularidade na comunidade de *software*.

Palavras-chave: Arquitetura de *Software*. Arquitetura de Microsserviços. Arquitetura de Portas e Adaptadores. Reutilização de *Software*. Princípios *SOLID*. Aplicações *Back-end*.

Abstract

In recent years, software architecture has become even more important, mainly to help manage the complexity of systems. This is because it defines and organizes the architectural elements of a system. It seeks to define a structure that that satisfies technical needs, as well as operationalizing the services offered in accordance with these needs. One of the best-known approaches is Microservices Architecture, in which the application is developed as a whole, considering relevant parts of the system being implemented "in isolation", and encapsulated in services of lesser granularity and with unique responsibilities. Together, these parts, i.e. these microservices, make up the application. Nowadays, given the scope of software products, it is pertinent to consider combining various architectural models. For example, implementing software combining Microservices Architecture and Ports and Adapters Architecture. In the case of the latter, there is a need to place the business domain at the center of the application, considering a layered model, designed as concentric circles. In addition, there is a need to meet deadlines, given market dynamics. Therefore, orienting oneself towards Software Reuse, in which ready-made solutions are used at different levels of abstraction (e.g. modeling and code), and these solutions are well established by the specialized community, becomes something very advantageous. This paper is an exploratory study that reveals relevant behaviors observed during the development of a back-end application oriented towards the combination of Microservices and Ports and Adapters Architectures, with a bias towards Software Reuse. The study provided valuable inputs, guided by a method centered on proof of concepts, which allowed exploring various nuances of developing a typical back-end application. The results obtained from the development are presented, highlighting the positive and negative aspects identified during the completion of each proof of concept, and are available for consultation by third parties interested in learning about the architectures discussed, which are gaining popularity in the software community.

Key-words: Software Architecture. Microservices Architecture. Ports and Adapters Architecture. Software reuse. SOLID principles. Back-end applications.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
BPMN	*Business Process Modeling Notation*
CEP	Código de Endereçamento Postal
HTTP	*Hypertext Transfer Protocol*
NoSQL	*Not Only Structured Query Language*
POC	*Proof of Concept*
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
YAML	*Yet Another Markup Language*

Sumário

Parte I

Introdução

1 Introdução

Este capítulo apresenta o contexto do trabalho realizado, demonstrando os conceitos relacionados aos tópicos Justificativa, Questão de Pesquisa e Objetivos, Geral e Específicos. Por fim, tem-se a Organização da Monografia. No contexto deste trabalho, tem-se como objeto de estudo a área de Arquitetura de *Software*, com a utilização da Arquitetura Portas e Adaptadores em Microsserviços, em princípio, visando conferir maior capacidade de Reutilização de *Software*. Reutilização de *Software* é algo desejado na Engenharia de *Software*, uma vez que permite o reaproveitamente de soluções comumente recomendadas pela comunidade para problemas recorrentes no desenvolvimento de *software*. Além disso, tais soluções podem ser componentizadas, conferindo maior facilidade para uso das mesmas em novos contextos similares.

1.1 Contexto

A Arquitetura de *software* é definida por Jaiswal (JAISWAL, 2019) como uma "planta" para um sistema. Ela fornece uma abstração para gerenciar a complexidade do sistema e estabelecer uma comunicação e um mecanismo de coordenação entre elementos. Ela define uma estrutura que procura satisfazer as necessidades técnicas e operacionais, enquanto otimiza atributos comuns de qualidade, como desempenho e segurança.

Além disso, segundo Bergman et al. (2008), padrões arquiteturais ajudam a definir o comportamento e as características básicas de uma aplicação. Ele afirma que ter conhecimento das características de cada padrão de arquitetura, seus pontos fortes e fracos, é necessário para escolher uma que seja aderente ao negócio. Em uma arquitetura monolítica, todas as funcionalidades são encapsuladas em uma só aplicação; logo, os módulos não podem ser executados de forma independente. Utilizar uma arquitetura monolítica no início de um projeto pode ser uma boa estratégia, pois permite explorar tanto a complexidade do sistema quanto às limitações de seus elementos. Porém, uma vez que a aplicação começa a escalar, essa arquitetura pode apresentar pontos significativamente negativos, conforme apontado em Bergman et al. (2008):

- Dificuldade de compreensão e manutenibilidade, resultando em uma redução de produtividade no desenvolvimento;
- Dificuldade de manter um *deploy* contínuo, já que uma mudança em uma pequena parte da aplicação requer, por vezes, a reconstrução de toda a aplicação;
- Custoso esforço para proporcionar escalabilidade, e

- Dependência longa com as tecnologias escolhidas em um primeiro momento, e que podem não mais satisfazer as necessidades evolutivas da aplicação (PONCE; MÁRQUEZ; ASTUDILLO, 2019).

Segundo Ponce, Márquez e Astudillo (2019), a Arquitetura de Microsserviços é uma abordagem que proporciona o desenvolvimento de uma aplicação dividida em pequenos serviços, cada um rodando seu próprio processo e comunicando-se entre si. Por serem serviços menores, em teoria, eles são mais fáceis de manter e também são mais tolerantes a falhas, já que uma falha em um serviço não compromete a aplicação inteira, como acontece na arquitetura monolítica (PONCE; MÁRQUEZ; ASTUDILLO, 2019).

Com base em Griffin (2021), a Arquitetura de Portas e Adaptadores apresenta uma abordagem distinta na organização e na modelagem de aplicações. Essa arquitetura coloca o modelo de domínio no centro da aplicação, cercado por camadas dedicadas ao gerenciamento dos objetos dentro desse modelo. Além disso, é adicionada uma camada de interface que abrange todas as outras camadas, fornecendo meios para que as solicitações possam acessar e interagir com as camadas internas da aplicação via portas, estabelecendo interfaces que atendam às necessidades do cliente. As implementações que atendem aos requisitos das portas são denominadas adaptadores, desempenhando um papel fundamental na conexão entre as camadas e garantindo a funcionalidade esperada do sistema. Quando aplicada da forma correta, esta arquitetura pode trazer benefícios, dentre eles: manutenibilidade facilitada; maior escalabilidade; menor acoplamento entre componentes; e até mesmo menor quantidade de linhas de código para novas funcionalidades (GRIFFIN, 2021).

1.2 Justificativa

De acordo com Martin (MARTIN, 2017), se a arquitetura de uma aplicação for algo tratado ao final, e não planejado previamente, então o sistema ficará cada vez mais caro para desenvolver. Martin (2017) chega a mencionar que a mudança em termos arquiteturais, caso seja necessária e tardia, será praticamente impossível de ocorrer para parte ou todo o sistema.

Além disso, de acordo com Ane (VAROTO, 2002), a arquitetura pode ser definida como um conjunto de componentes e os relacionamentos existentes entre eles, considerando a Reutilização de *Software* como um princípio de relevância nas tomadas de decisão. Dentre os diversos benefícios que essa reutilização tende a oferecer, Varoto (2002) destaca:

- Reutilização de idéias arquitetônicas, discernindo sobre o que adere bem ou não como solução técnica e de negócios para o problema em questão;

- Reutilização de estilos e padrões de arquitetura, adequando o negócio à representação técnica disponível, apresentando como uma primeira referência para a escolha da arquitetura, e
- Reutilização de componentes de *software*, aproveitando partes codificadas do sistema para evitar replicação, e definindo interfaces e funcionalidades encapsuladas.

Dentre os benefícios apresentados, a reutilização de componentes de *software* vem se tornando uma prática comum no desenvolvimento de *software* da atualidade, evitando replicação de código e facilitando a manutenção do sistema (VAROTO, 2002). Porém, para que a reutilização seja possível, é necessário que os componentes sejam bem definidos e, portanto, diversos padrões arquiteturais para o desenvolvimento de *software* foram criados com o objetivo de facilitar a criação de componentes reutilizáveis.

Esse trabalho apresenta a utilização da Arquitetura de Portas e Adaptadores em Microsserviços como algo que pode mitigar problemas de cunho arquitetural, mais especificamente considerando a necessidade de *deploy* contínuo, e facilitar a manutenção evolutiva de um sistema orientando-se por Reutilização de *Software*.

Como colocado anteriormente, a Arquitetura de Portas e Adaptadores faz uso de uma abordagem de menor acoplamento (LARMAN, 2007) entre os componentes arquiteturais, sendo assim, os componentes podem ser mais facilmente substituídos. Entende-se por menor acoplamento, segundo Larman (2007), uma menor dependência entre os componentes arquiteturais, permitindo remover ou incorporar funcionalidades de maneira mais simples. Adicionalmente, a Reutilização de *Software* tende a ser facilitada, quando há maior modularização (LUCRéDIO, 2009). Nesse sentido, faz-se uso combinado da Arquitetura Portas e Adaptadores e Microsserviços. Microsserviços permitem modularizar várias partes lógicas do sistema, estruturando-as como pequenos serviços (PONCE; MÁRQUEZ; ASTUDILLO, 2019). Esses pequenos serviços podem ser entendidos como componentes arquiteturais que proporcionam reutilização. Reutilizando vários Microsserviços pode-se obter uma solução - em tese - mais rapidamente. Essa é a principal premissa do trabalho.

1.3 Questão de Pesquisa

Ao longo deste trabalho, a seguinte questão de pesquisa foi respondida: Quais são os comportamentos de relevância observados ao longo do desenvolvimento de uma aplicação que combina **Arquitetura de Microsserviços e Arquitetura Portas e Adaptadores, considerando como foco Reutilização de *Software***?

Comportamentos considerados de relevância são aqueles que revelam evidências de cunho arquitetural e que permitam Reutilização de *Software* em soluções similares. Por

exemplo: os componentes arquiteturais, modelados de acordo com essa visão combinada, Portas e Adaptadores e Microsserviços, são facilmente reutilizados por terceiros? Há de fato baixo acoplamento entre esses componentes arquiteturais? Há facilidade de inserção de novos componentes? Há facilidade de remoção de componentes que não atendem mais as necessidades?

Apresenta-se, nesse contexto, provas de conceito que abordem preocupações pontuais, dentre elas inserção e remoção de novos componentes arquiteturais, revelando sobre os comportamentos observados.

1.4 Objetivos

Para responder a questão de pesquisa, este trabalho cumpre com alguns objetivos, sendo apresentados como Objetivo Geral e Objetivos Específicos, os quais são expostos a seguir.

1.4.1 Objetivo Geral

Estudo exploratório sobre os comportamentos de relevância, associados à Reutilização de *Software*, observados ao longo do desenvolvimento de uma aplicação orientada às Arquiteturas Microsserviços e Portas e Adaptadores.

Por ser um Objetivo Geral abrangente, foram especificados alguns Objetivos Específicos. Ao serem cumpridos esses Objetivos Específicos, almeja-se cumprir o Objetivo Geral.

1.4.2 Objetivos Específicos

- Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada;
- Levantamento dos principais pontos relacionados à Arquitetura Portas e Adaptadores, usando como base a literatura especializada;
- Estudo sobre Reutilização de *Software*, usando como base a literatura especializada;
- Especificação de componentes reutilizáveis, considerando um ou mais problema(s) recorrente(s) em termos arquiteturais, e
- Registro dos comportamentos de relevância - no contexto de Reutilização de *Software* - em uma aplicação orientada às Arquiteturas Microsserviços e Portas e Adaptadores.

1.5 Organização da Monografia

Esta monografia está dividida em capítulos, sendo eles:

- Capítulo 2 - Referencial Teórico: apresenta as fundamentações conceituais relacionadas aos tópicos de Arquitetura de *software*, Arquitetura de Microsserviços, Arquitetura Portas e Adaptadores e Reutilização de *Software*;
- Capítulo 3 - Suporte Tecnológico: apresenta as tecnologias e ferramentas utilizadas para a realização do trabalho, incluindo desenvolvimento das provas de conceito, hospedagem, versionamento, modelagens, comunicação entre os autores, dentre outros;
- Capítulo 4 - Metodologia: acorda sobre a classificação da pesquisa, bem como sobre os métodos para pesquisa, desenvolvimento e análise de resultados;
- Capítulo 5 - Estudo Exploratório: aborda o estudo exploratório do trabalho, bem como sua motivação, o detalhamento e objetivos de cada POC, bem como os principais aspectos relacionados ao seu desenvolvimento;
- Capítulo 6 - Análise de Resultados: Apresenta os principais resultados obtidos com a elaboração das Provas de Conceito, conferindo um resumo das análises de resultado abordadas no Capítulo 5 - Estudo Exploratório, e
- Capítulo 7 - Conclusão: Descreve a conclusão do trabalho, retomando o contexto, objetivos concluídos, questão de pesquisa, e conferindo ideias para trabalhos futuros.

Parte II

Referêncial Teórico

2 Referêncial Teórico

Este capítulo apresenta a fundamentação teórica para o desenvolvimento do trabalho, facilitando a compreensão de tópicos fundamentais ao tema. No intuito de conferir um roteiro de leitura coerente, são apresentadas definições de termos importantes relacionados à Arquitetura de *Software*, perpassando pela definição da Arquitetura Monolítica e pela Arquitetura de Microsserviços, evidenciando vantagens e desvantagens de acordo com a literatura. Em seguida, têm-se modelos arquiteturais conhecidos como a Arquitetura Limpa e a Arquitetura de Portas e Adaptadores, focando em suas aplicabilidades, vantagens e desvantagens também de acordo com a literatura apresentada. A ideia é ambientar o leitor no contexto para auxiliá-lo na compreensão das escolhas das arquiteturas de interesse desse trabalho, no caso: Arquitetura de Microsserviços e Arquitetura de Portas e Adaptadores.

Além disso, é abordada uma visão geral acerca das aplicações web desenvolvidas em *Server Side*, Aplicações *Back-end*, sendo apresentados, os principais aspectos referentes ao seu desenvolvimento, com destaque às boas práticas, convenções e demais tópicos relevantes. Há ainda um olhar mais específico aos aspectos de Reutilização de *Software*.

Por fim, constam as Considerações Finais do Capítulo, com um resumo sobre os principais aspectos que foram apresentados ao longo do capítulo.

2.1 Arquitetura de *Software*

A Arquitetura de *Software* é uma das principais *baselines* no que compreende proporcionar ganhos efetivos em agilidade e eficiência em manutenção e evolução dos sistemas de informação corporativos, fator preponderante para ambientes competitivos (SORDI; MARINHO; NAGY, 2006). À medida que estes sistemas crescem, cresce também a necessidade de entender práticas, princípios, estratégias e padrões que atendam estes sistemas (RODRIGUES; WERNER, 2009). Assim, a Arquitetura de *Software* firma-se como um elemento chave no processo de desenvolvimento das aplicações de maneira escalável e sustentável, sendo um dos principais fatores de sucesso para o desenvolvimento do *software*.

Quando se fala sobre arquitetura, remete-se à ideia de objetos e estruturas robustas, tais como: prédios, casas e pontes. Entretanto, em uma visão mais aprofundada, e no contexto da Engenharia de *Software*, tomando como base os autores mencionados anteriormente, a Arquitetura de *Software* incorre em não somente atentar-se à definição em si, com foco em estruturas que organizam elementos arquiteturais, mas também em

aspectos de desenvolvimento, ou seja, de cunho mais técnico e aplicado.

Diante do exposto, a Arquitetura de *Software* é definida como estruturas que incluem componentes, suas propriedades externas e os relacionamentos existentes entre elas, definindo a abstração do sistema que está sendo construído (BASS; CLEMENTS; KAZMAN, 2003). Além disso, é definido que a Arquitetura de *Software* comporta-se como uma ferramenta facilitadora para comunicação, análise e crescimento dos sistemas (BASS; CLEMENTS; KAZMAN, 2003).

Nas últimas décadas, o desenho/planejamento de uma arquitetura tornou-se uma importante demanda na Engenharia de *Software*. Uma boa arquitetura pode ajudar um sistema a garantir requisitos chave em diversas nuances, com destaque para: desempenho, confiabilidade, escalabiliade e interoperabilidade. Já uma arquitetura mal projetada pode incorrer em um verdadeiro desafio a ser transposto por desenvolvedores e demais especialistas técnicos (GARLAN, 2000).

Ainda de acordo com Garlan (2000), a Arquitetura de *Software* pode ter um papel importante em pelo menos seis aspectos no desenvolvimento de *software*. São eles:

1. Entendimento: Arquitetura de *Software* simplifica o entendimento de grandes sistemas apresentando-os em níveis de abstração, nos quais o desenho do sistema fica de fácil compreensão. Entretanto, quando se trata da implementação de novas funcionalidades ou correções de *bugs*, o desenvolvedor precisa ter o entendimento da arquitetura e das grandes estruturas que compõem a arquitetura implementada (BOUWERS et al., 2010), ou seja, quanto mais complexa a arquitetura, maior o esforço cognitivo para a compreensão do sistema;

2. Reutilização: Padrões de arquitetura podem dar apoio à reutilização de diversas formas, tanto à reutilização de grandes componentes (ou subsistemas), quanto de *frameworks* nos quais os componentes podem ser integrados. Esses *frameworks* reutilizáveis podem ser estilos arquitetônicos de *software* específicos de domínio, padrões de integração de componentes, e padrões de *design* arquitetônico;

3. Construção: Uma descrição arquitetural provê um desenho parcial para o desenvolvimento, indicando os componentes principais e as dependências entre eles, e facilitando a construção;

4. Evolução: Arquitetura de *Software* pode expor as direções, para as quais o sistema pode evoluir;

5. Análise: Descrições arquiteturais fornecem oportunidades para análises, como consistência do sistema; conformidade com as restrições impostas pelo padrão arquitetural; conformidade com os atributos de qualidade; análise de dependência, e análise de domínio específico, e

6. Gerenciamento: Experiências têm mostrado que uma avaliação crítica de uma arquitetura leva a um entendimento muito mais claro dos requisitos, das estratégias de implementação e dos potenciais riscos.

2.1.1 Arquitetura Monolítica

A Arquitetura Monolítica é um modelo tradicional unificado para o *design* de um *software*. Por monolítico, define-se que toda composição do *software* será feita em um único bloco, possuindo múltiplos componentes combinados em uma aplicação. Com esse tipo de arquitetura, pode-se ter uma maior simplicidade na definição do código a ser desenvolvido, além de facilidades em análises e menor complexidade na implementação de testes de integração e *end-to-end* (BLINOWSKI; OJDOWSKA; PRZYBYŁEK, 2022).

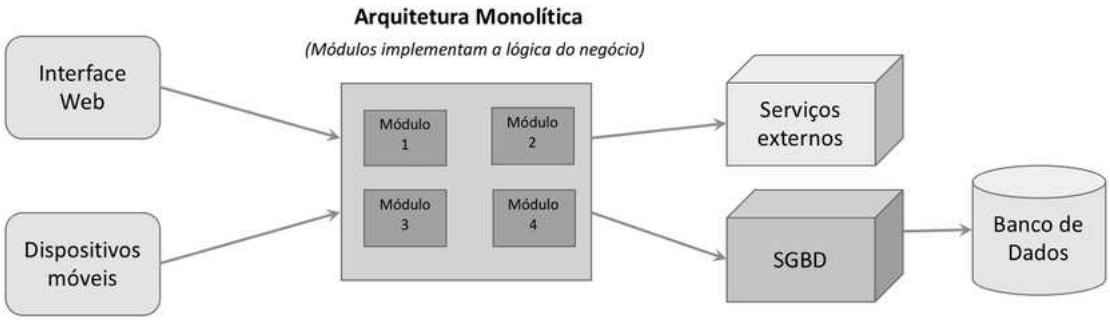
Segundo Conceição e Pinto (2021), utilizar o padrão de Arquitetura Monolítica pode trazer diversas vantagens na definição de um sistema, tornando-se benéfica para os requisitos e componentes do *software*. Dentre os benefícios mais relevantes, podem ser citados:

- Maior facilidade em testes de integração e *end-to-end* já que não há necessidade de configurar comunicações entre diferentes serviços;
- Maior facilidade na depuração da aplicação;
- Maior facilidade no desenvolvimento devido à definição dos componentes em visão unificada, e
- Definição de apenas um banco de dados, facilitando o desenvolvimento das estruturas de dados, bem como da manutenção.

Contudo, apesar de ser bastante utilizada e possuir diversos benefícios, a Arquitetura Monolítica pode se tornar um grande problema quando um sistema começa a crescer, pois a complexidade do sistema aumenta e traz consigo dificuldades no entendimento e na manutenção do código (CONCEIÇÃO; PINTO, 2021). Esse modelo de arquitetura também traz consigo uma dificuldade quando o assunto é a implementação de novas funcionalidades, consumindo tempo em modificações que serão necessárias e na mitigação de impactos que essa implementação poderá trazer ao sistema. Esse processo é complexo e arriscado (CONCEIÇÃO; PINTO, 2021).

Como definido na Figura 1, a Arquitetura Monolítica é apresentada através de um único bloco, onde todos os componentes estão integrados e são dependentes entre si. Nessa definição, há tanto as estruturas de dados quanto as regras de negócio, além de possuir a comunicação de entrada e saída das aplicações como interfaces web e dispositivos móveis.

Figura 1 – Arquitetura Monolítica



Fonte: (VILLACA; AZEVEDO; JR, 2018)

2.1.2 Arquitetura de Microsserviços

A Arquitetura de Microsserviços é um estilo arquitetural que busca estruturar um grande sistema em pequenas partes que possuem alta coesão e baixo acoplamento, permitindo pouca dependência entre serviços (RODRIGUES; PINTO, 2019). Microsserviços são serviços independentes que possuem sua modelagem definida na camada de domínio de um produto, encapsulando sua funcionalidade, e conferindo à mesma disponibilidade para outros serviços através de uma interface e um contrato bem definidos (NEWMAN, 2021).

Segundo Conceição e Pinto (2021), utilizar o padrão de Arquitetura de Microsserviços no desenvolvimento pode conferir diversas vantagens na definição de um sistema, tornando-se benéfica para os requisitos e componentes do *software*. Dentre os benefícios mais relevantes, podem ser citados:

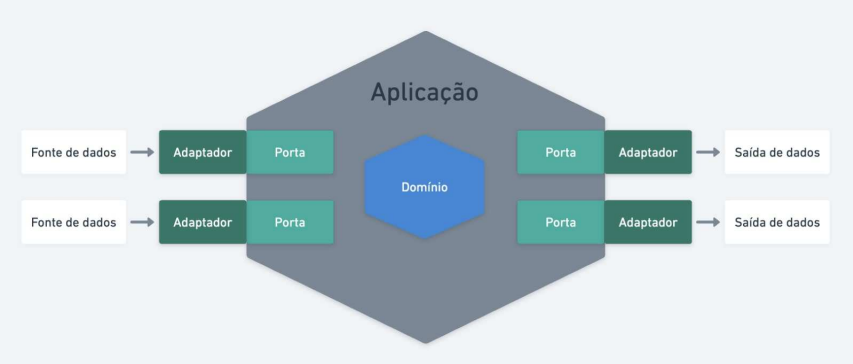
- Independência que os microsserviços possuem na aplicação como um todo, possuindo bancos de dados dedicados, o que ajuda a evitar a indisponibilidade, já que falhas em um determinado serviço não afetará diretamente a disponibilidade de outros serviços;
- Redução na sobrecarga de dados, já que cada banco de dados dedicado pode ser otimizado para atender as necessidades do serviço correspondente;
- Flexibilidade na disponibilização de serviços em diferentes regiões, de forma que a infraestrutura fique mais próxima da maioria dos usuários, e assim garantindo a disponibilidade do serviço com baixa latência;
- Facilidade no escalonamento das aplicações, replicando os serviços de maneira fácil e rápida, e
- Facilidade na manutenção das aplicações ao longo do tempo, pois cada serviço possui sua própria estrutura e regras, evitando interações indesejadas com terceiros.

Apesar de trazer tais benefícios, é importante lembrar que a Arquitetura de Microsserviços também possui desvantagens, principalmente relacionadas a sua natureza distribuída. A complexidade de *deploy*, a escalabilidade e o monitoramento em um sistema com múltiplos serviços tornam-se maiores do que em um sistema monolítico (BLINOWSKI; OJDOWSKA; PRZYBYŁEK, 2022).

2.1.3 Arquitetura de Portas e Adaptadores

A Arquitetura de Portas e Adaptadores é um estilo arquitetural que permite que uma aplicação seja utilizada por usuários; produtos de *software*; automatizada por testes ou scripts, e que possa ser desenvolvida e testada de maneira isolada de seus dispositivos e bancos de dados (NUNKESSER, 2022). A ideia desse tipo de arquitetura é definir o centro da aplicação através de um domínio que somente será acessado via portas, que definem a maneira que a aplicação será utilizada e quais componentes podem ser integrados à mesma. Nesse contexto, têm-se ainda os adaptadores, que são responsáveis por permitir que a aplicação se conecte com o mundo exterior, liberando funções como implementações de banco de dados, conexões com serviços externos, e recebimento de dados (NUNKESSER, 2022).

Figura 2 – Arquitetura de Portas e Adaptadores



Fonte: Autores

Como definido na Figura 2, a estrutura dessa arquitetura é apresentada através da sobreposição de duas camadas, os quais separam a aplicação do seu domínio, diferindo o que é regra de négocio das definições e comunicações necessárias através da tecnologia que está sendo empregada. Nessa definição, têm-se as seguintes estruturas:

- Domínio: Parte na qual a regra de negócio da aplicação está definida, sendo o centro da aplicação e o nível mais baixo da mesma, e
- Aplicação: Parte na qual a aplicação é definida, onde se têm as comunicações e definições de portas e adaptadores necessários.

Portas são definidas como elementos de entrada e saída de dados para o domínio da aplicação. Elas estabelecem a maneira que a aplicação será utilizada e quais componentes podem ser integrados à mesma (NUNKESSER, 2022). Através da definição de portas para a aplicação, torna-se possível integrar a mesma com diversos serviços externos existentes, tais como: bancos de dados; protocolos de comunicação e demais serviços que possam ser necessários para o funcionamento da aplicação.

Adaptadores são definidos como elementos que trazem consigo as aplicabilidades e os serviços, sendo responsáveis por permitir que a aplicação se conecte com o mundo exterior, liberando funções como implementações de banco de dados; conexões com serviços externos, e protocolos de comunicação (NUNKESSER, 2022).

Seguem algumas vantagens desse estilo arquitetural:

- Facilidade de integração da Arquitetura de Portas e Adaptadores com outros conceitos previamente definidos, apresentando adequada versatilidade que serve de base para outras arquiteturas (NUNKESSER, 2022);
- Sem distinção entre interfaces de usuário e interfaces para outros tipos de atores, como servidores e dispositivos. Tal estratégia permite a construção de uma *API* agnóstica de uma tecnologia específica de interface do usuário ou persistência de dados (JUNIOR et al., 2014);
- Facilidade na testabilidade da aplicação, visto que sua lógica e suas regras de negócio são completamente separadas das dependências e definições da camada de aplicação, assim, facilitando na utilização de *mocks* (MARTINEZ, 2021);
- Facilidade na implementação e na substituição de novos componentes, onde o modelo de comunicação ou persistência pode ser alterado sem que haja impacto na lógica da aplicação, possuindo diversas portas para adaptadores (BROWN, 2014), e
- Reutilização dos componentes, onde uma porta pode ser utilizada em diversos adaptadores, assim, proporcionando um código mais limpo e escalável (BROWN, 2014).

Entretanto, é importante lembrar que este estilo arquitetural também possui suas desvantagens, como por exemplo a complexidade que o desenvolvimento de uma aplicação com múltiplas camadas de abstração traz, o que requer um maior esforço de novos desenvolvedores para ter um entendimento da aplicação. Por este motivo é importante estudar a necessidade da implementação deste estilo arquitetural (BROWN, 2014).

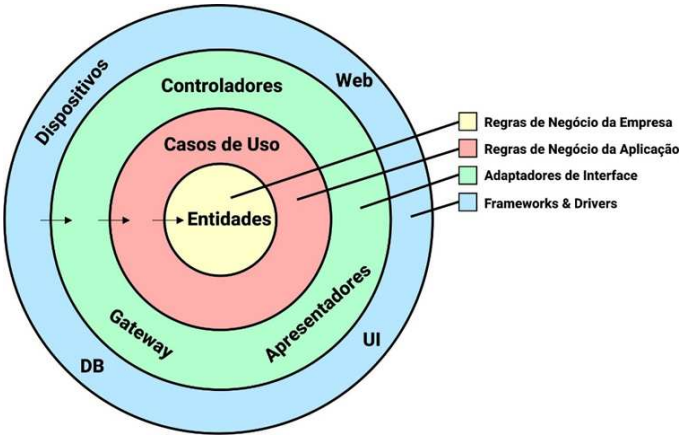
2.1.4 Arquitetura Limpa

Martin (MARTIN, 2017) propôs a ideia da Arquitetura Limpa como uma alternativa para incorporar ideias de outras arquiteturas, incluindo a Arquitetura de Portas e

Adaptadores, que tinham em comum a separação das preocupações, dividindo o *software* em camadas, separando regras de negócio e interfaces. De acordo com Martin (2017), cada uma dessas arquiteturas produz sistemas com as seguintes características:

- Independência de *frameworks*: A arquitetura não depende da existência de biblioteca de *software* carregada de recursos. Isso permite utilizar esses *frameworks* como ferramentas em vez de obrigar o sistema a se adaptar às restrições limitadas dos *frameworks*;
- Testabilidade: As regras de negócio podem ser testadas sem Interface de Usuário, banco de dados, servidor web ou qualquer outro elemento externo;
- Independência da Interface de Usuário: A Interface de Usuário pode mudar facilmente, sem alterar outros elementos do sistema. Uma Interface de Usuário web pode ser substituída por uma Interface de Usuário console, por exemplo, sem alterar as regras de negócio;
- Independência do banco de dados: Pode-se trocar Oracle ou SQL Server por um Mongo, BigTable ou CouchDB, entre outros sistemas gerenciadores de banco de dados, pois as regras de negócio não estão ligadas à base de dados, e
- Independência de aspectos externos: Na verdade, as regras de negócio não sabem sobre as interfaces do mundo externo.

Figura 3 – Arquitetura Limpa



Fonte: (MARTIN, 2017)

Como exemplificado na Figura 3, os círculos representam as diferentes camadas do *software*. Os círculos mais externos são mecanismos e os mais internos são políticas. Para aplicação desta arquitetura, é imprescindivel seguir a regra da dependência. Nessa

regra, as dependências de código fonte devem apontar apenas para dentro, na direção das políticas de nível mais alto (MARTIN, 2017).

Segundo esta regra, os elementos de um círculo interno não podem saber sobre os elementos de um círculo externo, sejam eles funções, classes, variáveis ou qualquer outra entidade de *software* nomeada. Pela mesma razão, os formatos de dados declarados em um círculo externo não devem ser usados em um círculo interno (MARTIN, 2017).

É possível perceber similaridades entre as Arquiteturas Portas e Adaptadores e Limpa. A escolha, nesse trabalho, em centrar o estudo na Arquitetura Portas e Adaptadores deu-se pelo viés de Reutilização de *Software*.

Nesse quesito de Reutilização de *Software*, a Arquitetura de Portas e Adaptadores tem diferenciais. O uso de adaptadores, por exemplo, pemite integrar mais facilmente a aplicação com o mundo exterior. Portanto, empacotar uma solução computacional e oferecê-la a terceiros tornam-se viáveis. Isso leva à maior reutilização de soluções prontas. Além disso, tais soluções podem ser mais facilmente testadas, o que também corrobora com suas reutilizações em diferentes cenários de uso. Esses diferenciais são, inclusive, incorporados em outras arquiteturas, como é o caso da Arquitetura Limpa, que faz uso de vários desses princípios.

Sendo assim, optou-se pelo estudo da Arquitetura de Portas e Adaptadores, que carrega em sua origem esses diferenciais de maior relevância para o presente trabalho. Complementarmente, a Arquitetura Limpa orienta-se por princípios *SOLID*, sendo também pertinente compreendê-los e incorporá-los no estudo exploratório. Demais aspectos a respeito das escolhas desse trabalho serão mais bem tratados no Capítulo 5 - Estudo Exploratório.

2.1.4.1 Princípios *SOLID*

A Arquitetura Limpa define vários princípios para o desenho de arquitetura e de componentes que abrangem vários tipos de sistema. O acrônimo *SOLID* é utilizado para os cinco seguintes princípios de desenho de componentes arquiteturais, de acordo com Lano (2023) e outros autores:

1. SRP - Princípio da Responsabilidade Única (*Single Responsibility Principle*) "Nunca deve haver mais de um motivo para uma classe mudar". O princípio da responsabilidade única defende que se houver mais de um motivo para mudar uma classe, é assumido que esta classe tem mais de uma responsabilidade, o que resulta em um alto acoplamento. Este tipo de alto acoplamento pode gerar fragilidades em caso de uma mudança inesperada de requisitos (SINGH; HASSAN, 2015);

2. OCP - Princípio Aberto/Fechado (*Open-Closed Principle*) "Entidades de *Software* como classes, módulos e funções devem ser abertas para extensão, mas fechadas para modificações". Este princípio defende que se uma única mudança feita em um programa resulta em uma cascata de mudanças de componentes dependentes, o programa exibe os indesejados atributos relacionados com uma arquitetura ruim. O programa torna-se frágil, rígido, imprevisível e não reutilizável. O princípio aberto/fechado recomenda que se desenhe módulos que nunca mudem. Caso um requisito mude, deve-se estender o comportamento do módulo adicionando novo código, respeitando e se atentando ao código existente que já funciona (SINGH; HASSAN, 2015).

3. LSP - Princípio da substituição de Liskov (*Liskov Substitution Principle*) Este princípio é uma forte limitação de como superclasses e subclasses podem se relacionar. O princípio da substituição de Liskov diz que os comportamentos de todas as subclasses devem satisfazer a especificação da superclasse (LANO, 2023).

4. ISP - Princípio da Segregação da Interface (*Interface Segregation Principle*) Este princípio afirma que componentes não devem depender de componentes ou interfaces que eles não utilizam. Cada uma destas dependências causa desperdício de esforço ao manter e realizar o *deploy* do sistema, além de complicar os processos de *build* e *deploy* (LANO, 2023).

5. DIP - Princípio da Inversão de Dependência (*Dependency Inversion Principle*) "Módulos de alto nível não devem depender de módulos de baixo nível. Os dois devem depender das Abstrações. Abstrações não devem depender de Detalhes. No entanto, detalhes devem depender das abstrações". Este príncipio afirma que se deve desacoplar módulos de alto nível de módulos de baixo nível, incluindo uma camada de abstração entre classes de alto nível e classes de baixo nível. Para estar em conformidade com este princípio, deve-se isolar essa abstração dos detalhes do problema (SINGH; HASSAN, 2015).

Seguem algumas vantagens adicionais dos principios *SOLID*:

- Manutenabilidade: Os princípios *SOLID* promovem a criação de código modular e de fácil compreensão. Isso facilita a manutenção do código ao longo do tempo, tornando mais simples realizar alterações, correções de *bugs* e melhorias, e
- Reusabilidade de código: Através do Princípio da Substituição de Liskov (LSP) e do Princípio da Segregação de Interfaces (ISP), os componentes do sistema tornam-se mais reutilizáveis. Classes podem ser substituídas ou combinadas de maneira mais eficiente, promovendo a reusabilidade do código.

Apesar das vantagens, assim como a Arquitetura de Portas e Adaptadores, a Arquitetura Limpa divide o *software* em camadas, o que aumenta a complexidade do sistema e traz um maior custo de tempo e esforço no desenvolvimento.

2.2 Aplicações *Back-end*

Aplicações *Back-end* são de suma importância para o funcionamento de grande parte das aplicações hospedadas na internet. Fluxo de acesso, controle de dados, requisitos de segurança e manutenção de funcionalidades, são alguns dos aspectos em que se encontram sob responsabilidade das mesmas (TRAJANO et al., 2022). Uma aplicação servidora ou aplicação de *Back-end* em computação é o sistema responsável por possibilitar a troca de informações entre os sistemas, permitindo a integração e o compartilhamento de funcionalidades e dados de forma controlada (Escola DNC, 2024). Uma aplicação servidora, geralmente, é aquela em que os dados de todo o sistema são armazenados através de bancos de dados e trabalhados por outros processos computacionais, para que possam gerar novas informações posteriormente (BACKENDLESS, 2017).

O foco desse trabalho está nesse contexto, ou seja, em aplicações *Back-end*. Foi conduzido o estudo sobre os comportamentos arquiteturais observando o desenvolvimento de uma aplicação *Back-end*, bem como aplicações *Back-end* já construídas, e que se orientem pela combinação da Arquitetura de Portas e Adaptadores em Microsserviços. Para isso, foram estabelecidas Provas de Conceito (em inglês, *Proof Of Concepts* - POCs). Outros detalhamentos de cunho metodológico, sobre as provas de conceito, foram tratados no Capítulo 4 - Metodologia. Adicionalmente, o principal olhar dos registros sobre o que está sendo observado concentra-se em Reutilização de *Software*.

2.3 Reutilização de *Software*

De acordo com Krueger (1992), a reutilização de *software* é um processo de criação de sistemas que derivam de um sistema já construído. Krueger (1992) ainda define que a reutilização de *software* é uma preocupação constante para organizações que buscam gerar um aumento na qualidade e na produtividade do processo de desenvolvimento. O desenvolvimento de um sistema focado em implementar a reutilização de *software* busca, desde seu conceito, produzir os artefatos que serão utilizados antes do sistema em si, incorporando os mesmos ao serviço final.

Segundo Frakes e Kang (2005), a reutilização de *software* tende a incorrer em benefícios ao desenvolvimento de sistema, tais como:

- Aumento na produtividade do time desenvolvedor;

- Redução de custos e tempo de desenvolvimento;
- Aumento na qualidade do *software*;
- Aumento da confiabilidade e da resiliência do *software*, e
- Facilidade na escalabilidade e na manutenção do *software*, uma vez que há maior padronização do código que está sendo implementado.

Diante do exposto, reutilizar é algo que a área de Engenharia de *Software* almeja em diferentes níveis de abstração, seja em termos de: regras de negócio; requisitos de *software*; desenho e projeto de *software*; componentes e conectores arquiteturais; testes de *software*, dentre outros (FILHO, 2019).

No escopo de Arquitetura de *software*, reutilizar demanda componentes reutilizáveis, capazes de prover soluções prontas ou de fácil adaptação, integráveis, e que foram testadas na resolução de uma preocupação ou problema arquitetural (FOWLER, 2006). Ao fazer uso de uma arquitetura, cuja literatura menciona facilitadores nesse sentido, caminha-se para uma solução, portanto, de maior qualidade, escalável, produzida em menor tempo, dentre outras vantagens (FOWLER, 2003).

2.4 Considerações Finais do Capítulo

Através desse capítulo, foi possível conferir uma visão acerca dos aspectos relacionados ao trabalho, onde inicialmente foi apresentado o conceito de Arquitetura de *Software*, levantando os principais pontos sobre Arquitetura Monolítica e da Arquitetura de Microsserviços, com vantagens e desvantagens de acordo com a literatura.

Dando continuidade, foram apresentados os conceitos de Arquitetura de Portas e Adaptadores e Arquitetura Limpa, focando em suas aplicabilidades, vantagens e desvantagens também de acordo com a literatura apresentada. Em seguida, foi apresentado o conceito de *SOLID*, que é um conjunto de princípios de desenvolvimento de *software* que visam facilitar a manutenção e a evolução do código, tornando-os mais legível e menos suscetível a *bugs*.

Adicionalmente, foi apresentado o contexto de Aplicações *Back-end*, que são aplicações que possuem a lógica de negócio executada no servidor, sendo o servidor responsável por gerir os dados do negócio e disponibilizar os mesmos através de interfaces. Por fim, tem-se um olhar sobre Reutilização de *Software*.

Parte III

Suporte Tecnológico

3 Suporte Tecnológico

Este capítulo apresenta as ferramentas e tecnologias que foram escolhidas para o desenvolvimento e a execução do trabalho. Este capítulo foi dividido em três seções, sendo elas: Ferramentas de Desenvolvimento, considerando o contexto de aplicações *Back-end*; Ferramentas de Apoio, e Considerações Finais do Capítulo.

As ferramentas foram escolhidas com base na pertinência para a demanda, na familiaridade dos autores, bem como na possibilidade de um desenvolvimento rápido, considerando o escopo e o tempo disponíveis para o projeto.

3.1 Ferramentas de Desenvolvimento

Essa seção apresenta as ferramentas e tecnologias escolhidas para condução das etapas de desenvolvimento do projeto, incluindo as provas de conceito centradas em aplicações *Back-end*.

3.1.1 Python 3.10

Python é uma linguagem de alto nível de abstração orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa (BORGES, 2014). O Python tem uma sintaxe clara e concisa, a qual favorece a legibilidade do código-fonte, tornando a linguagem mais produtiva. Possui, dentre outros recursos, diversas estruturas de alto nível e uma vasta coleção de módulos prontos para uso, além de *frameworks* de terceiros que podem ser adicionados ao projeto que está sendo desenvolvido (BORGES, 2014).

Python confere mais comodidade na hora de programar. Sendo assim, diferentemente das demais linguagens, utiliza poucas palavras, o que implica em menos código. Quando se começa a programar em Python, aprende-se que a indentação é primordial. Adicionalmente, percebe-se que um código organizado e limpo incorre em menos errors. Esta linguagem faz uso de palavras reservadas curtas da língua inglesa, facilitando assim o entendimento e o aprendizado, além de deter de adequado poder de processamento (flexibilidade e simplicidade), fator que pode ser utilizado tanto por aqueles que estão entrando no mundo da programação, como também para os mais experientes (SANTIAGO et al., 2020).

A linguagem Python foi escolhida pelos autores por conta da familiaridade dos mesmos com ela. Além disso, por ser uma linguagem de alto nível, permite-se que o foco seja direcionado mais para a resolução do problema do que para a implementação e a sintaxe do projeto em si, focando mais na lógica do que na linguagem.

3.1.2 RabbitMQ 3.12

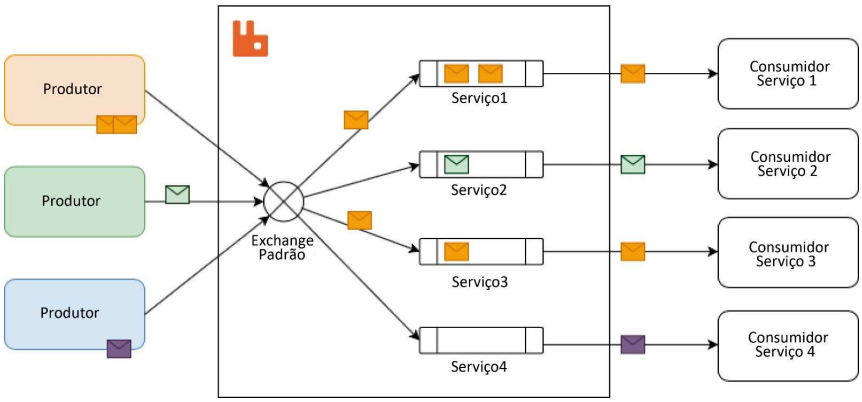
RabbitMQ é um servidor de mensageria de código aberto (*open source*) desenvolvido em Erlang, implementado para conferir apoio às mensagens em um protocolo denominado *Advanced Message Queuing Protocol* (AMQP) (MARQUES, 2018).

RabbitMQ é agnótisco quanto à linguagem, podendo ser utilizando em diversos sistemas independente de qual seja a linguagem que o *software* esteja implementado. Ele atende linguagens como .NET, Python, PHP, Ruby e dentre outras. Ele é escrito em Erlang, uma linguagem de programação funcional e concorrente, que é utilizada para construir sistemas distribuídos de alta disponibilidade e de tempo real que exigem alta escalabilidade e baixa latência. Ele é leve e pode ser implantado na nuvem (SHARVARI; NAG, 2019).

RabbitMQ utiliza filas de mensageria como método de comunicação, conforme ilustrado na Figura 4. Serviços externos podem publicar e consumir mensagens, o que remove a necessidade de um serviço estar disponível para que a mensagem seja entregue. Se um serviço consumidor estiver temporariamente indisponível, ao contrário do HTTP, a mensagem é armazenada de maneira segura no RabbitMQ, sendo entregue quando o serviço estiver disponível novamente (OLAH, 2020).

A escolha do RabbitMQ como servidor de mensageria foi dada pela familiaridade dos autores com a ferramenta, além da facilidade de implementação de serviços orientados a eventos e da escalabilidade que a ferramenta oferece. Através desse servidor, é possível abstrair complexas mensagens em filas, permitindo a comunicação assíncrona dos serviços.

Figura 4 – Diagrama Comunicação RabbitMQ



Fonte: (OLAH, 2020) (Tradução: Autores)

3.1.3 PostgreSQL 15

PostgreSQL é um sistema gerenciador de banco de dados com adequada confiabilidade e robustez. Ele é uma ferramenta de código aberto com reconhecida reputação de con-

fiabilidade, integridade de dados e correção de dados. Ele é um banco de dados avançado, sendo utilizado por grandes empresas em diversos tipos de aplicações (POSTGRESQL, 2024).

A escolha do PostgreSQL como banco de dados foi feita pela familiaridade dos autores e pela facilidade de implementação e desenvolvimento.

3.1.4 MongoDB 7

MongoDB é um sistema de gerenciamento de banco de dados (DBMS) não relacional, baseado em *software* livre, que utiliza documentos flexíveis em vez de tabelas e linhas para processar e armazenar várias formas de dados. Como uma solução de banco de dados NoSQL, o MongoDB não requerer um sistema de gerenciamento de banco de dados relacional (RDBMS), portanto, ele oferece um modelo de armazenamento de dados flexível/extensível, que permite aos usuários armazenar e consultar tipos de dados variados com facilidade (IBM, 2024).

A escolha do MongoDB como banco de dados NoSQL foi feita pela familiaridade dos autores e pela facilidade de implementação e desenvolvimento, além de ser gratuito e *open source*.

3.1.5 SonarQube 10.6

SonarQube é uma ferramenta de análise estática de código utilizada por diversas empresas e desenvolvedores para coletar métricas de qualidade em projetos e sistemas. Através da utilização dessa ferramenta, é possível identificar vulnerabilidades, *bugs* e *code smells* de forma eficiente (SONARSOURCE, 2024).

Pode-se ainda integrar *pipelines* e *workflows* para realizar a coleta de métricas, gerando assim checagens automatizadas através da integração da ferramenta com *branches* e *pull requests* de repositórios de código.

Através da coleta dessas métricas, torna-se praticável a avaliação periódica da qualidade do projeto. Com essas coletas, é possível identificar problemas e agir mais rapidamente para solucioná-los, além de evitar que novos problemas sejam introduzidos no código.

As métricas específicas escolhidas, dentre as oferidas pela plataforma, foram:

- Cobertura (*Coverage*): Refere-se à porcentagem do código que é exercitado pelos casos de teste (SonarSource, 2024c);
- Manutenibilidade (*Maintainability*): Refere-se à facilidade de reparar, melhorar e entender o código de *software* (SonarSource, 2024a);

- Confiabilidade (*Reliability*): Refere-se à medida de como o *software* é capaz de manter seu nível de desempenho sob condições específicas por um período de tempo determinado (SonarSource, 2024a);
- "Maus Cheiros no Código"(*Code Smells*): São definidos como categorias de problemas no código que podem levar a problemas mais profundos ao longo do tempo. Embora os maus cheiros de código não sejam erros ou *bugs* em si e não afetem a funcionalidade do software, eles indicam fraquezas no *design* que podem atrasar o desenvolvimento ou aumentar o risco de *bugs* ou falhas no futuro (SonarSource, 2024b); e
- Falhas Inesperadas (*Bugs*): São definidos como problemas que representam algo errado no código. Se ainda não ocasionaram uma quebra na aplicação, provavelmente irão (SonarSource, 2024b).

Insumos das coletas de cada métrica constam apresentados, em detalhes, no Capítulo 5: Estudo Exploratório, para cada Prova de Conceito desenvolvida nesse trabalho.

3.2 Ferramentas de Apoio

Essa seção apresenta as ferramentas de apoio utilizadas pelos autores no decorrer da elaboração do trabalho, sendo essas ferramentas de prototipação, versionamento do código, implantação e comunicação.

3.2.1 Lucidchart

Lucidchart (Lucidchart, 2024) é uma ferramenta de diagramação inteligente que permite visualizar ideias complexas com mais agilidade, clareza e de forma mais colaborativa. Permite trabalho síncrono dos autores, além de compartilhamento dos conteúdos elaborados com terceiros (ex. orientadores). No contexto do trabalho, essa ferramenta foi utilizada para a elaboração dos fluxogramas e das representações arquiteturais utilizadas pelos autores no intuito de conferir maior clareza na exposição das ideias contidas nessa monografia. Exemplos de uso do Lucidchart podem ser encontrados nos modelos apresentados no Capítulo 4 - Metodologia, na notação de modelagem de processos BPMN[1].

3.2.2 Git 2.24.1

Git (GIT, 2024) é um sistema de controle de versões distribuído, no qual é possível registrar quaisquer alterações feitas no código, armazenando as informações e permitindo

[1] Por exemplo, Fluxo de Atividades/Subprocessos - Segunda Etapa do TCC, disponível em: <https://lucid.app/lucidchart/bbf417bc-f630-475e-b592-eb7f331bbbcf>. Último acesso em: Agosto. 2024

o rastreamento, *rollback* e versionamento de uma aplicação de modo simples, objetivo e rápido. No contexto do trabalho, essa ferramenta foi utilizada para versionar tanto o documento latex gerado para essa monografia, como os recursos para as provas de conceito. Caso seja de interesse para consulta, considerar o repositório[2].

3.2.3 Docker 24.0.7

Docker (DOCKER, 2024) é um *software* de código aberto que permite criação, implantação e execução de aplicações dentro de contêineres. Os contêineres possibilitam que as aplicações sejam isoladas umas das outras, facilitando a execução rápida, confiável e consistente. Esse recurso foi relevante para o contexto desse projeto no intuito de viabilizar o trabalho colaborativo da dupla de autores nos insumos de interesse comuns, sendo principalmente: monografia e códigos das Provas de Conceito.

3.2.4 Docker Compose 2.21.0

Docker Compose (COMPOSE, 2024) é uma ferramenta para definição e a execução múltipla de contêineres. Com o Docker compose, é possível através de um arquivo YAML a configuração e a execução de diversos serviços. Isso facilitou o próprio uso do Docker, mencionado anteriormente.

3.2.5 Slack

Slack (SLACK, 2024) é um aplicativo de mensagens para empresas que conecta as pessoas às informações de que elas precisam. São permitidos: criação de canais dedicados, estabelecendo se são públicos ou privados; compartilhamento de mídias entre os participantes dos canais, sejam vídeos, áudios, documentos ou outros; envio de notificações para os envolvidos, dentre outras ações. No contexto do trabalho, essa ferramenta foi utilizada para manter rastros das comunicações e orientações durante o desenvolvimento do trabalho, sendo essencial para o alinhamento entre os autores e orientadores.

3.2.6 Telegram

Telegram (TELEGRAM, 2024) é um aplicativo que envia mensagens instantâneas de texto, voz, vídeo, foto e outros arquivos, de forma rápida e prática. No contexto do trabalho, essa ferramenta foi utilizada para troca de mensagens rápidas entre os autores e orientadores.

[2] Disponível em: <https://github.com/Matheusafonsouza/tcc_arquitetura_pocs>. Último acesso em: Agosto. 2024

3.2.7 Teams

O Teams (TEAMS, 2024) oferece opções de reunião por vídeo, áudio, também permitindo a integração de um participante via telefone fixo ou celular. No contexto do trabalho, essa ferramenta foi utilizada para reuniões semanais entre os autores e orientadores.

3.2.8 LaTeX

LaTeX (LATEX, 2024) é um *software* para editoração e confecção de documentos voltados para a escrita científica. No contexto do trabalho, essa ferramenta foi utilizada para a elaboração do presente documento.

3.3 Considerações Finais do Capítulo

Ao longo desse capítulo, foi apresentada uma breve descrição das ferramentas utilizadas tanto para desenvolvimento como apoio na elaboração do trabalho e de provas de conceito, além do gerenciamento de suas versões. Para conferir um resumo do que foi abordado durante o capítulo, segue a Tabela 1:

Tabela 1 – Principais tecnologias utilizadas no desenvolvimento do trabalho.

Nome	Descrição	*Link*
Python 3.10	Linguagem de programação utilizada durante o desenvolvimento do código.	<https://www.python.org/>
RabbitMQ 3.12	Ferramenta utilizada para comunicação através de mensageria.	<https://www.rabbitmq.com/>
PostgreSQL 15	Ferramenta utilizada para armazenamento de dados.	<https://www.postgresql.org/>
MongoDB	Ferramenta utilizada para armazenamento de dados.	<https://www.mongodb.com>
SonarQube	Ferramenta utilizada para coleta de métricas do projeto.	<https://www.sonarqube.org>
Figma	Ferramenta para modelagem dos fluxogramas.	<https://www.figma.com/>
Git 2.24.1	Ferramenta utilizada para versionamento do código.	<https://git-scm.com/>
Docker 24.0.7	Ferramenta utilizada para conteinerização do código.	<https://www.docker.com/>
Docker Compose 2.21.0	Ferramenta utilizada para conteinerização do código.	<https://docs.docker.com/compose/>
Slack	Ferramenta utilizada para comunicação com foco em facilitar o alinhamento entre autores e orientadores.	<https://slack.com/intl/pt-br>
Telegram	Ferramenta utilizada para comunicação rápida.	<https://web.telegram.org/>
Teams	Ferramenta utilizada para alinhamento em chamadas de vídeo.	<https://www.microsoft.com/pt-br/microsoft-teams/>
LaTeX	Linguagem de marcação voltada para escrita de artigos.	<https://www.latex-project.org/>

Parte IV

Metodologia

4 Metodologia

Este capítulo apresenta a metodologia adotada que guiou o trabalho como um todo. Desta forma foi definida a Classificação da Pesquisa, considerando a Abordagem adotada, a Natureza do estudo, os Objetivos propostos e os Procedimentos escolhidos. Em seguida, serão apresentados o Método Investigativo e o Método Orientado a Provas de Conceito, sendo o primeiro utilizado para pesquisar sobre os conceitos e outros tópicos de interesse junto às bases científicas, e o segundo visando a condução o trabalho de forma modularizada, passo a passo, apresentando os insumos cabíveis para cada problemática, ou seja, cada cenário observável em termos arquiteturais. Posteriormente, foi discutido o Método de Desenvolvimento, destacando as metodologias ágeis e sua aplicação prática no trabalho. Também foi abordado o Método de Análise de Resultados, descrevendo seus procedimentos e características. Ademais, serão apresentados os Fluxos de Atividades/Subprocessos e Cronogramas de Atividades/Subprocessos, para o escopo do TCC. Por fim, serão conferidas as Considerações Finais do Capítulo, resumindo os tópicos abordados no capítulo.

4.1 Classificação da Pesquisa

Conforme Gerhardt e Silveira (2009), o termo "metodologia" resulta da fusão de "*methodos*", que denota organização, com "*logos*", associado à pesquisa ou investigação. Nesse contexto, a metodologia pode ser descrita como o estudo da estrutura ou passos necessários para alcançar um objetivo específico. Nesta seção, seguindo as definições de Gerhardt e Silveira (2009), são abordadas a classificação da pesquisa deste trabalho em relação aos critérios: Abordagem, Natureza, Objetivos e Procedimentos.

4.1.1 Abordagem

A pesquisa conduzida neste trabalho adota uma abordagem **qualitativa**, conforme classificação de Gerhardt (GERHARDT; SILVEIRA, 2009). A pesquisa qualitativa difere-se por não se concentrar em representações numéricas, mas sim em aprofundar a compreensão de um tema específico.

Embora a abordagem seja predominantemente qualitativa, certas métricas coletadas por ferramentas como o SonarQube são apresentadas como dados quantitativos, seguidas de análises interpretativas, novamente, de viés mais qualitativo. Esses dados, juntamente com as observações dos comportamentos ao longo do desenvolvimento, servirão como insumos para uma análise qualitativa mais rica. Dentre as métricas quantificá-

veis, podem ser mencionadas, por exemplo: número de *bugs* e porcentagem de cobertura de teste. Detalhes da abordagem utilizada serão acordados no Capítulo 5 - Estudo Exploratório, para cada Prova de Conceito desenvolvida.

4.1.2 Natureza

A natureza deste trabalho é classificada como **pesquisa aplicada**. Pesquisas com esta classificação visam gerar conhecimentos para aplicação prática, e têm como objetivo solucionar problemas específicos (GERHARDT; SILVEIRA, 2009). Sendo assim, este trabalho soluciona a questão levantada na Questão de Pesquisa, procurando prover um estudo orientado a arquiteturas específicas, e conferindo incrementos de *software* via Provas de Conceito.

4.1.3 Objetivos

De acordo com Gerhardt e Silveira (2009), pode-se classificar pesquisas em três grupos: (i) a pesquisa exploratória; (ii) a pesquisa descritiva, e (iii) a pesquisa explicativa.

Com bases nos objetivos, esta pesquisa é classificada como **exploratória**. Este tipo de pesquisa tem como objetivo proporcionar maior familiaridade com o problema exposto nas seções Contexto e Justificativa, com vistas a torná-lo mais claro ou a formular hipóteses.

4.1.4 Procedimentos

Segundo Gerhardt e Silveira (2009), a pesquisa científica é o resultado de um exame minucioso, realizado com o objetivo de resolver um problema, recorrendo a procedimentos científicos. No contexto dos procedimentos adotados neste trabalho, destaca-se a **pesquisa bibliográfica**, que se inicia com a busca e o levantamento de referências teóricas e práticas já existentes e publicadas na comunidade especializada (GERHARDT; SILVEIRA, 2009).

Além disso, este trabalho conta com uma pesquisa orientada a **provas de conceito**. Essa abordagem pode ser uma ferramenta essencial para demonstrar a capacidade e a aptidão de um *software*, bem como o seu cumprimento em relação às necessidades dos clientes e usuários, conforme discutido por Prasanna et al. (2021).

4.2 Método Investigativo

Segundo Gil et al. (2002), uma vez estabelecido o tema, a pesquisa bibliográfica inicial visa fornecer ao pesquisador uma compreensão mais aprofundada da área de estudo em questão. Essa fase é relevante, pois auxilia na delimitação do escopo do trabalho a ser

realizado. Assim, após a definição do tema deste estudo, conduziu-se uma revisão teórica utilizando a plataforma acadêmica Google Scholar, uma fonte consolidada que possibilita a pesquisa não apenas em artigos, mas também em livros, revistas e outras fontes científicas e de mercado, oferecendo assim uma abordagem mais abrangente. A Tabela 2 exibe as *Strings* de Busca selecionadas para a pesquisa bibliográfica.

Tabela 2 – *Strings* de Busca

Base de dados	***String***	**Quantidade**
Google Scholar	'software architecture'	5.240.000
Google Scholar	'microservices architecture'	44.400
Google Scholar	'ports and adapters architecture'	34.100
Google Scholar	'hexagonal architecture pattern'	212.000
Google Scholar	'clean architecture'	2.350.000
Google Scholar	'ports and adapters' AND 'microservices'	1.840

Fonte: Autores

4.2.1 Critérios de Seleção

A escolha do material foi conduzida por meio de uma análise exploratória, a qual envolveu a revisão dos resumos e das palavras-chave dos trabalhos identificados. Os critérios de seleção adotados foram os seguintes:

- Relacionado à Arquitetura Portas e Adaptadores e/ou Microsserviços;
- Relacionado à Arquitetura de *Software*;
- Relacionado ao desenvolvimento de APIs, e
- Disponíveis gratuitamente.

Mesmo considerando *strings* de busca bem aderentes aos tópicos de interesse desse trabalho, constatou-se certa dificuldade em refinar o material obtido na Tabela 2. Acredita-se que essa dificuldade tenha ocorrido devido à falta de materiais que tratem sobre uma visão combinada de arquiteturas, no caso, Arquitetura Portas e Adaptadores e Arquitetura de Microsserviços. Os trabalhos tratam ou individualmente tais arquiteturas, ou ainda conferindo um *survey* sobre cada uma delas. Mas, uma visão combinada, de viés mais prático, é mais complicada de ser encontrada, o que justamente motivou a realização desse trabalho.

A dificuldade foi contornada, uma vez que o retorno em número de artigos foi razoavelmente grande, mesmo para a *string* de busca "'ports and adapters' AND 'microservices'", sendo possível, através de leituras mais dinâmicas:

- realizar uma pré-seleção de materiais bem detalhados sobre cada arquitetura, conferindo aprendizado adequado sobre cada uma delas;
- realizar uma pré-seleção de materiais de cunho mais técnico, que conferiram uma visão mais prática e embasada sobre as nuances de implementação de cada uma das arquiteturas de interesse, e
- realizar uma pré-seleção de materiais bem generalistas, que acordam uma visão mais abrangente sobre Arquitetura de *Software*.

Em um segundo momento, já selecionados materiais mais aderentes à pesquisa, os autores realizaram leituras mais criteriosas, visando selecionar os principais artigos que conferem a base da pesquisa, conforme constam especificados na próxima seção.

Ocorreu ainda um esforço adicional para combinar ambas as arquiteturas, orientando-se por Reutilização de *Software*. Entretanto, esse é o principal intuito do trabalho, ou seja, um Estudo sobre os Comportamentos Arquiteturais Observados na visão combinada das Arquiteturas Portas e Adaptadores e Microsserviços, orientando-se por um viés de Reutilização de *Software*.

Nesse sentido, optou-se por conhecer, via literatura, o conceito geral sobre Reutilização de *Software*. Isso permitiu aos autores compreenderem as particularidades da Reutilização de *Software*, provocando-os a identificar se essas facilidades foram ou não alcançadas com o estudo exploratório realizado.

Diante do exposto, foram consultados autores mais antigos, que revelam as origens sobre Reutilização de *Software*, como é o caso do autor Krueger (1992). Além disso, ocorreu a leitura de literaturas que focam nos benefícios da Reutilização de *Software*, cabendo destaque aos autores Frakes e Kang (2005). Por fim, optou-se por consultar um dos principais portais sobre Arquitetura de *Software*, no intuito de criar um sentimento sobre como uma Arquitetura pode influenciar em termos de Reutilização de *Software*. Isso remete aos artigos publicados pelo autor Martin Fowler, Fowler (2003) e Fowler (2006).

4.2.1.1 Resultados

Após uma análise prévia, principalmente considerando as Arquiteturas de Portas e Adaptadores e Microsserviços, os artigos selecionados para orientação deste trabalho foram:

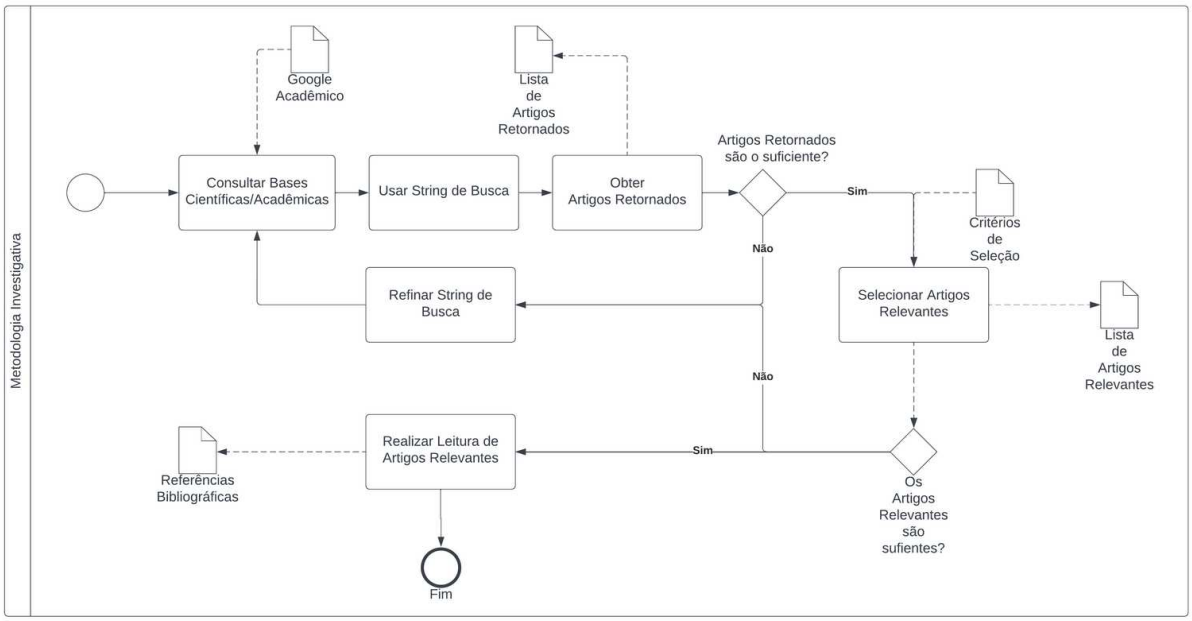
- *Building Microservices* (NEWMAN, 2021);
- *Using Hexagonal Architecture for Mobile Applications* (NUNKESSER, 2022);
- *Software architecture: a roadmap* (GARLAN, 2000), e

- *Arquitetura Limpa - O Guia do Artesão para Estrutura e Design de Software* (MARTIN, 2019).

Cabe mencionar que "*Hexagonal Architecture*"ou "Arquitetura Hexagonal"foi o primeiro nome dado à Arquitetura de Portas e Adaptadores. Portanto, também foi palavra chave utilizada como base nas pesquisas.

Adicionalmente, é essencial realizar um estudo complementar para abordar os desafios delineados nas provas de conceito, considerando Reutilização de *Software* como um diferencial. O processo investigativo deste estudo segue os mesmos passos da pesquisa bibliográfica anterior. Sendo assim, também foram utilizadas *strings* de busca específicas, com as palavras-chave: *Proof Of Concept*; *Software Reuse* e variações pertinentes. Detalhes sobre esses conceitos bem como os principais materiais bibliográficos de embasamento constam especificados no Capítulo 2 - Referencial Teórico. A Figura 5 ilustra, utilizando a modelagem BPMN, as atividades presentes no método investigativo.

Figura 5 – Diagrama da Metodologia Investigativa



Fonte: Autores

O processo começa na atividade Consultar Bases Científicas/Acadêmicas, com destaque para o Google Acadêmico. Na sequência, têm-se: Usar *String* de Busca e Obter Artigos Retornados. Com isso, obtem-se uma Lista de Artigos Retornados. Nesse ponto, ocorre uma decisão se os artigos retornados são ou não suficientes. Sendo suficientes, ocorre a atividade Selecionar Artigos Relevantes, considerando os Critérios de Seleção estabelecidos, e gerando uma Lista de Artigos Relevantes. Não sendo suficientes, então, ocorre

a atividade Refinar *String* de Busca, o que retoma a necessidade de executar Consultar Bases Científicas/Acadêmicas, ocorrendo mais um ciclo de atividades já explicado.

Caso já se esteja de posse da Lista de Artigos Relevantes, avalia se os mesmos são suficientes. Sendo suficientes, ocorre a atividade Realizar Leitura de Artigos Relevantes, levando às Referências Bibliográficas que embasam esse trabalho.

4.3 Método Orientado a Provas de Conceito

De acordo com Prasanna et al. (2021), o método orientado a provas de conceito é utilizado em diversas áreas, de marketing à medicina. Porém, na área de *software*, a aplicação deste método compreende um processo específico, que pode ocorrer no desenvolvimento de *hardware*, *websites* ou outro *software* para implementar uma solução. Ainda de acordo com Prasanna et al. (2021), esse processo é designado para determinar se um *software* pode ser criado no mundo real; qual tecnologia utilizar para o desenvolvimento, e se os usuários pretendidos provavelmente utilizarão o *software*.

Segundo Lab (2023), a metodologia POC (*Proof of Concept*, ou Prova de Conceito) é desenvolvida nas seguintes etapas:

- Definição de objetivos;
- Identificação de requisitos técnicos;
- Implementação da PoC;
- Teste e validação; e
- Análise de resultados.

4.3.1 Definição de Objetivos

A intenção dessa fase é definir claramente os objetivos que se espera alcançar com a prova. Nesta etapa, é elaborado o *roadmap* visando a implementação e um escopo bem definido.

4.3.2 Identificação de Requisitos Técnicos

Esta fase envolve a identificação dos requisitos técnicos específicos para a realização da prova de conceito, abrangendo aspectos como a seleção da plataforma de *hardware* e *software*; a configuração do ambiente de teste, e o estabelecimento das métricas desejadas.

4.3.3 Implementação da PoC

Com os objetivos e os requisitos técnicos bem definidos, inicia-se a implementação da prova de conceito. Isso pode envolver a criação de um protótipo ou modelo funcional; a configuração de *hardware* e *software*, e a coleta de dados para análise. Adicionalmente, para o caso desse trabalho em particular, essa etapa foi conduzida orientando-se pelo Método de Desenvolvimento, explicado mais adiante nesse capítulo.

4.3.4 Teste e Validação

Após a implementação da PoC, inicia-se esta etapa, onde a solução implementada é testada, bem como validada. Os testes podem ser para avaliar diferentes aspectos. No caso desse trabalho, foram utilizadas métricas aferidas via SonarQube, conferindo, na sequência, uma análise de abordagem parte quantitativa e, predominantemente, qualitativa. Ocorreu ainda a validação, junto ao público alvo, sendo esse composto por usuários de perfil mais técnico (i.e. conhecedores das arquiteturas de interesse desse trabalho), e que possam conferir opiniões sobre os comportamentos arquiteturais observados, concordando ou não com os mesmos. A ideia é obter pontos de vista de especialistas, permitindo, complementarmente, uma análise predominantemente qualitativa. Esses especialistas devem ser capazes de realizar uma revisão por pares sobre os pareceres de comportamentos observados pelos autores.

4.3.5 Análise de Resultados

Após finalizar a etapa de testes e validação, torna-se relevante analisar os resultados para concluir sobre a conformidade com os requisitos estabelecidos na fase inicial. Isso engloba análises de abordagem, predominantemente, qualitativa, mas com insumos também quantitativos, oriundos do uso do SonarQube. A intenção é indentificar possíveis áreas de aprimoramento, bem como a elaboração de um relatório de resultados. Essa etapa e a etapa anterior, Teste e Validação, foram conduzidas orientando-se pelo Método de Análise de Resultados, explicado mais adiante nesse capítulo.

4.4 Método de Desenvolvimento

A abordagem metodológica adotada para a execução prática deste trabalho foi híbrida, resultante da combinação dos principais elementos das metodologias ágeis *Scrum* e *Kanban*. Essa escolha considera a significativa relevância dessas metodologias no cenário atual de mercado e desenvolvimento de *software* (CARVALHO; MELLO, 2012).

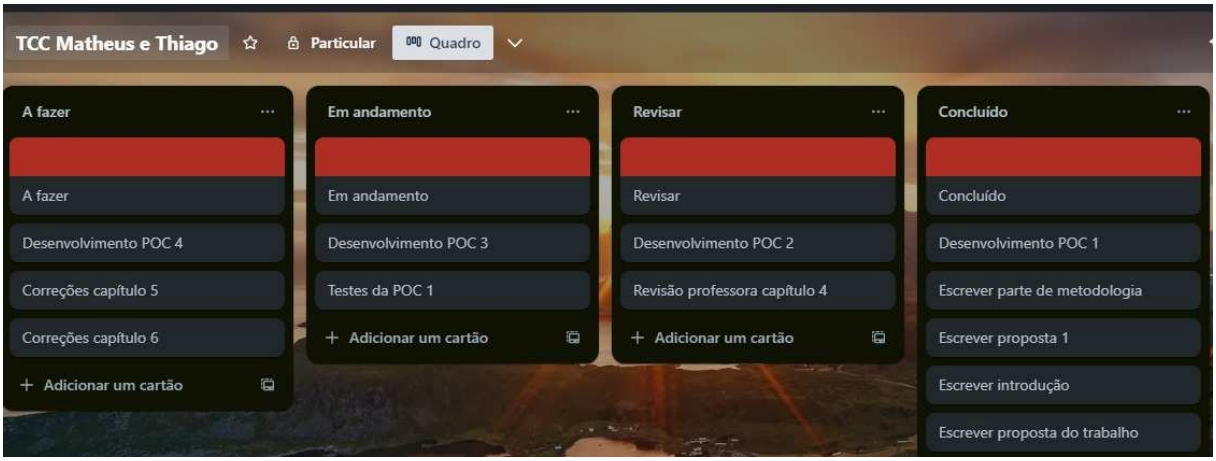
A adoção do Kanban foi realizada com o objetivo de proporcionar uma visualização transparente do avanço do projeto, facilitando a priorização de tarefas e a identificação

de possíveis gargalos. O Kanban é uma metodologia ágil de gestão de projetos que se fundamenta em um quadro de tarefas. Nesse quadro, as tarefas são movidas de uma coluna para outra conforme avançam em seu progresso. Algumas boas práticas que foram adotas, mencionadas por Ahmad, Markkula e Oivo (2013), são:

- Evitar tarefas externas ao quadro durante as fases de desenvolvimento;
- Considerar formas de limitar tarefas em progressos; e
- Encorajamento de *feedback* entre os membros.

Oriundo do Scrum, utiliza-se o *Product Backlog*, uma lista de requisitos que pode ser alterada à medida que o projeto avança. No entanto, preferiu-se uma abordagem menos formal do que o Scrum tradicional, sem papéis ou rituais definidos, como reuniões diárias ou retrospectivas. Mas, foram estabelecidos critérios de revisão de commits, também uma prática do Scrum, para que tudo transcorresse na implementação de forma mais tranquila. Na versão combinada com o Kanban, o Quadro Kanban, especificado via Trello, representou o *Sprint Backlog* em cada momento do projeto. Na Figura 6, é ilustrado um momento desse Quadro Kanban.

Figura 6 – Quadro Kanban



Fonte: Autores

4.5 Método de Análise de Resultados

Com o objetivo de avaliar os resultados alcançados ao longo do estudo, foi empregado o método de análise de resultados conhecido como pesquisa-ação. Esse método fundamenta-se na realização de uma ação ou resolução a partir da investigação de uma

problemática. No contexto da pesquisa-ação, os pesquisadores participam de maneira cooperativa e participativa (GERHARDT; SILVEIRA, 2009).

A pesquisa-ação envolve protocolos, cujas etapas podem ser configuradas conforme as exigências específicas de cada pesquisa. Para esta investigação, o fluxo da pesquisa-ação foi orientado seguindo as seguintes etapas:

- **Levantamento Investigativo:** Nesta fase, o foco principal concentra-se na obtenção de informações relevantes e significativas, visando justificar a elaboração da pesquisa. Para atender a esse propósito, foi conduzida uma Pesquisa Bibliográfica, conforme detalhado previamente na seção 4.2. Os resultados dessa pesquisa estão documentados nos Capítulos 2 e 3, referentes ao Referencial Teórico e ao Suporte Tecnológico, respectivamente. Adicionalmente, uma análise minuciosa de documentos disponíveis foi realizada para extrair informações pertinentes;

- **Planejamento:** Nesta fase, os objetivos da pesquisa são definidos de maneira clara, indicando as metas a serem alcançadas. Além disso, foram formuladas as questões de pesquisa que direcionam este trabalho, conforme abordado no Capítulo 1. A escolha de métodos e das técnicas foi cuidadosamente realizada para guiar as atividades ao longo do tempo, sem descartar a possibilidade de ajustes conforme necessidades ocorressem;

- **Acão:** A implementação efetiva das ações planejadas começa após a definição do plano de pesquisa. Isso inclui a aplicação dos métodos e das técnicas selecionados, bem como a coleta adicional de dados, se necessário. Durante essa fase, é importante registrar observações detalhadas para documentar o progresso e os *insights* obtidos (KRAFTA et al., 2007). Adicionalmente, é nessa etapa que atividades de desenvolvimento ocorrem. Portanto, as próprias implementações de cada PoC fazem parte do escopo de ações compreendidos nessa etapa de pesquisa-ação; e

- **Avaliação:** Nessa etapa, o foco principal é realizar a análise dos resultados obtidos, envolvendo a examinação dos dados coletados para verificar a confirmação ou refutação das hipóteses e uma reflexão crítica sobre o processo de pesquisa. Isso é essencial para a elaboração de conclusões sólidas que resumem os resultados. Em casos de ciclos adicionais de pesquisa-ação, é pertinente avaliar se ocorreram melhorias ou fragilidades, comparando os dados da iteração atual com os dados das iterações anteriores (KRAFTA et al., 2007). Adicionalmente, é nessa etapa que as atividades inerentes às análises de resultados de cada PoC ocorrem. Portanto, têm-se Teste e Validação, além da própria Análise de Resultados, sendo essas previstas para cada PoC no Método Orientado a Provas de Conceito, também já explicadas, respectivamente, nas seções 4.3.4 e 4.3.5.

4.6 Fluxos de Atividades/Subprocessos

Durante a elaboração deste trabalho, diversas atividades foram conduzidas com o propósito de gerenciar as demandas de forma coesa e dentro dos prazos estabelecidos. A primeira etapa do TCC abrange atividades desde a definição do tema até a apresentação dos resultados perante a banca examinadora. Desta maneira, um plano de atividades específico foi delineado, conforme detalhado a seguir, e ilustrado na Figura 7, na notação BPMN, para o caso da primeira etapa do TCC.

4.6.1 Primeira Etapa do TCC

1. **Definir o Tema:** Atividade que estabelece o tema central que orienta todo o trabalho. Para alcançar esse objetivo, é relevante obter uma compreensão mínima das lacunas existentes em termos de conhecimento, além de desenvolver a habilidade de formular uma questão de pesquisa clara e significativa.
   Status Atividade: Concluída.
   Tema do Trabalho: **Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de *Software*: Um Estudo sobre os Comportamentos Arquiteturais Observados.**

2. **Conduzir a Pesquisa Bibliográfica:** Atividade que envolve a seleção de material relevante sobre o tema escolhido, disponível em bases científicas e conforme os critérios apresentados na subseção 4.2. Além disso, tem como objetivo fornecer apoio aos pesquisadores por meio da revisão de trabalhos anteriores relacionados ao tema. O resultado desse processo contribuiu para o embasamento desta monografia, refletindo-se nas Referências Bibliográficas apresentadas ao final do documento.
   Status Atividade: Concluída.
   Insumos Gerados: Referências Bibliográficas obtidas orientando-se pelo Método Investigativo.

3. **Formular a Proposta Inicial:** Atividade realizada com o propósito de elaborar o Capítulo 1, abrangendo a contextualização, a justificativa, a formulação da questão de pesquisa e a definição dos objetivos.
   Status Atividade: Concluída.
   Insumos Gerados: Capítulo 1.

4. **Desenvolver o Referencial Teórico:** Atividade que resultou no Capítulo 2, compreendendo embasamento conceitual de tópicos como Arquitetura de *Software*, Arquitetura de Microsserviços, Arquitetura Portas e Adaptadores, Princípios *SOLID*, Aplicações *Back-end* e Reutilização de *Software*. Esse processo demandou a seleção criteriosa de literatura pertinente ao tema, juntamente com a combinação de infor-

mações relevantes associadas ao assunto.
Status Atividade: Concluída.
Insumos Gerados: Capítulo 2.

5. **Estabelecer Suporte Tecnológico:** Atividade com ênfase direcionada à seleção de ferramentas, visando viabilizar a pesquisa e alinhá-la estrategicamente aos objetivos do estudo. A descrição detalhada dessas ferramentas, fundamentais para o desenvolvimento prático do trabalho, está documentada no Capítulo 3.
Status Atividade: Concluída.
Insumos Gerados: Capítulo 3.

6. **Descrever a Metodologia:** Atividade dedicada à elaboração de um plano prático, que descreve os métodos e procedimentos escolhidos para condução do trabalho. O objetivo é garantir que a pesquisa seja realizada de forma coerente e confiável, para que os resultados sejam válidos. Esta atividade é descrita neste capítulo.
Status Atividade: Concluída.
Insumos Gerados: Presente Capítulo.

7. **Refinar Proposta:** Atividade para reavaliar o escopo previamente definido, com base na melhor compreensão do tema adquirida ao longo das atividades anteriores. O resultado dessa atividade foi o Capítulo 5, da monografia apresentada na primeira etapa do TCC. Entretanto, o conteúdo refinado do capítulo, já com a proposta elaborada encontra-se no Capítulo 5.
Status Atividade: Concluída.
Insumos Gerados: Capítulo 5 (primeira etapa do trabalho), cujo conteúdo refinado é apresentado no Capítulo 5.

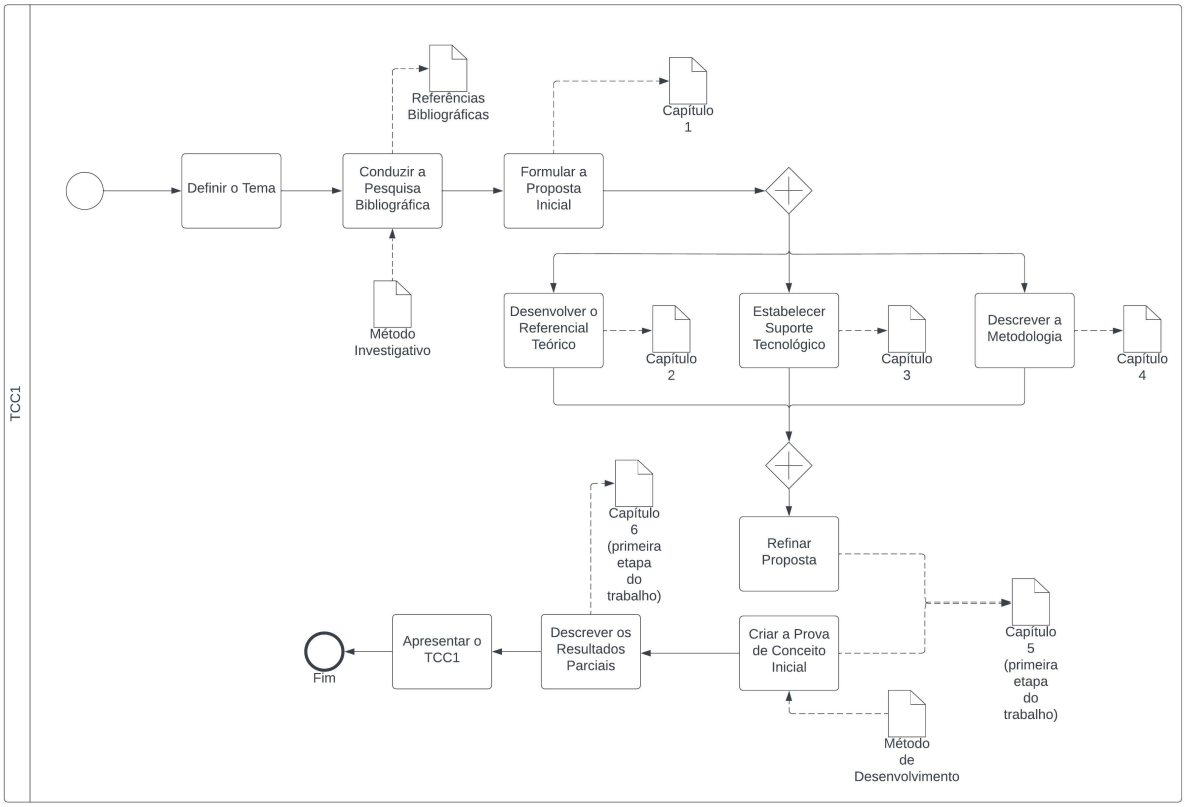
8. **Criar a Prova de Conceito Inicial:** Atividade para avaliação da viabilidade da proposta, sendo realizada por meio da implementação de uma prova de conceito. Tal abordagem permitiu identificar possíveis riscos e estabelecer as bases para a efetiva implementação da solução. Os resultados dessa atividade são apresentados no Capítulo 5.
Status Atividade: Concluída.
Insumos Gerados: Capítulo 5.

9. **Descrever os Resultados Parciais:** Atividade para apresentar o *status* atual do trabalho antes da apresentação da primeira etapa do TCC, descrevendo os avanços alcançados até aquele momento e delineando as perspectivas para a segunda etapa do TCC. Os resultados dessa análise foram documentados no Capítulo 6 da monografia da primeira etapa do TCC. Agora, esses resultados foram refinados, e encontram-se - na íntegra - no Capítulo 6.
Status Atividade: Concluída.

Insumos Gerados: Capítulo 6 (primeira etapa do trabalho), cujo conteúdo refinado é apresentado no Capítulo 6.

10. **Apresentar o TCC1:** Atividade que envolveu a apresentação para a banca do que foi desenvolvido durante a primeira etapa do TCC.
    Status Atividade: Concluída.

Figura 7 – Fluxo de Atividades/Subprocessos - Primeira Etapa do TCC



Fonte: Autores

4.6.2 Segunda Etapa do TCC

Complementarmente, um plano de atividades específico foi delineado, conforme detalhado a seguir, e ilustrado na Figura 8, na notação BPMN, para o caso da segunda etapa do TCC.

1. **Aplicar Correções:** Atividade que teve como objetivo efetuar as correções de acordo com os apontamentos da banca examinadora na apresentação da primeira etapada do TCC, e adaptar a monografia às mesmas.
   Status Atividade: Concluída.

2. **Realização das Atividades de Desenvolvimento:** Subprocesso para desenvolvimento do trabalho. O processo de desenvolvimento foi guiado pelo

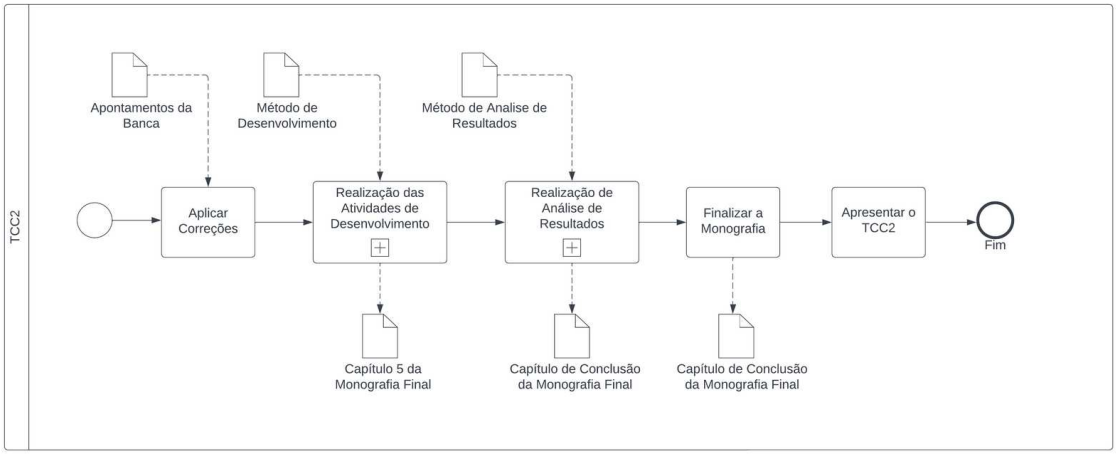
Método Orientado a Provas de Conceito combinado ao Método de Desenvolvimento.
Status Subprocesso: Concluída.
Insumos Gerados: Capítulo 5.

3. **Realização de Análise de Resultados:** Subprocesso para análise dos resultados obtidos durante a fase de desenvolvimento, buscando compreender e interpretar esses dados. O intuito foi extrair percepções que contribuíssem para a conclusão da pesquisa. Esse subprocesso orientou-se pelo Método de Análise de Resultados.
Status Subprocesso: Concluída.
Insumos Gerados: Capítulo 6.

4. **Finalizar a Monografia:** Atividade sobre o estado final da monografia, destacando os objetivos alcançados e descrevendo aspectos que podem ser abordados em trabalhos futuros.
Status Atividade: Concluída.
Insumos Gerados: Capítulo 7.

5. **Apresentar o TCC2:** Atividade voltada para a apresentação dos resultados finais conquistados à banca examinadora.
Status Atividade: Não iniciada.

Figura 8 – Fluxo de Atividades/Subprocessos - Segunda Etapa do TCC



Fonte: Autores

4.7 Cronograma de Atividades

Com base no que foi exposto na seção anterior, foi elaborado um cronograma para cada etapa do TCC, sendo apresentados nas Tabelas 3 e 4.

Tabela 3 – Cronograma de atividades/subprocessos da Primeira Etapa do TCC

Atividades/Subprocessos	**Ago**	**Set**	**Out**	**Nov**	**Dez**
Definir o Tema	X				
Conduzir a Pesquisa Bibliográfica		X			
Formular a Proposta Inicial		X			
Desenvolver o Referencial Teórico			X		
Estabelecer o Suporte Tecnológico				X	
Desenvolver a Metodologia				X	
Refinar a Proposta					X
Criar a Prova de Conceito Inicial					X
Descrever os Resultados Parciais					X
Apresentar o TCC1					X

Fonte: Autores

Tabela 4 – Cronograma de atividades/subprocessos da Segunda Etapa do TCC

Atividades/Subprocessos	**Mar**	**Abr**	**Mai**	**Jun**	**Jul**	**Ago**	**Set**
Implementar Ajustes	X						
Realização das Atividades de Desenvolvimento	X	X	X	X			
Realização da Análise de Resultados				X	X		
Finalizar a Monografia						X	
Apresentar o TCC2							X

Fonte: Autores

4.8 Considerações Finais do Capítulo

Este capítulo visou proporcionar clareza sobre as decisões metodológicas adotadas para a condução do trabalho. Desde o início, a pesquisa foi categorizada como majoritariamente qualitativa, de natureza aplicada, com objetivos exploratórios, sendo conduzida por meio dos procedimentos de pesquisa bibliográfica e provas de conceito.

A partir dessas considerações, foram definidas as seguintes abordagens metodológicas específicas para conduzir a pesquisa:

1. **Método Investigativo:** Utilizado para o desenvolvimento da pesquisa bibliográfica, este método visou uma análise aprofundada das fontes disponíveis.

2. **Método Orientado a Provas de Conceito:** Utilizado na criação dos desafios práticos do trabalho, esta abordagem visou demonstrar e validar conceitos por meio de implementações práticas.

3. **Método de Desenvolvimento:** Adotando uma abordagem híbrida, fundamentada na combinação de elementos das metodologias ágeis *Scrum* e Kanban, esse método foi aplicado ao processo de desenvolvimento do trabalho.

4. **Método de Análise de Resultados:** Baseado em pesquisa-ação, esse método permitiu a análise dos resultados obtidos, orientando-se por uma abordagem, predominantemente, qualitativa.

O capítulo é concluído apresentando fluxos de atividades e cronogramas para ambas as etapas do TCC, proporcionando uma visão geral do planejamento e da execução das diferentes fases do trabalho.

Parte V

Estudo Exploratório

5 Estudo Exploratório

Este capítulo apresenta o estudo exploratório desenvolvido nesse trabalho, conferindo a Motivação para a realização do mesmo. Conforme definido no Capítulo 4 - Metodologia, esse trabalho orienta-se por métodos específicos, sendo o maior foco em provas de conceito. A intenção é justamente conferir insumos bem focados em preocupações inerentes de Aplicações *Back-ends* implementadas combinando as Arquiteturas Portas e Adaptadores e Microsserviços, com viés de Reutilização de *Software*. Nesse sentido, foi realizada uma adaptação do Método Orientado a Provas de Conceito, baseando-se em Silva (2023) e Lab (2023), que conferiu um roteiro para exposição desse estudo. Diante do exposto, optou-se por apresentar cada prova de conceito, detalhando: Definição dos Objetivos, Identificação dos Requisitos Técnicos, Implementação da POC, e Revelação dos Resultados (com Teste & Validação e Análise de Resultados). Isso ocorre para todas as provas de conceito.

Seguem, portanto, as Provas de Conceito de forma mais detalhada (POC 1, POC 2, POC 3 e POC 4). Por fim, têm-se as Considerações Finais do Capítulo.

5.1 Motivação

A escolha de um tema no âmbito da Arquitetura de *Software* começou durante a jornada de ambos os autores no desenvolvimento de aplicações *Back-end*. Ao longo da trajetória profissional, os dois autores passaram por diversos sistemas desenvolvidos de acordo com diferentes padrões arquiteturais para viabilizar as necessidades levantadas nas atividades da Engenharia de Requisitos desses sistemas. Dentre esses projetos, desenvolveu-se uma aplicação com o objetivo de efetuar o pagamento de boletos através de diversos serviços, sendo esses isolados em seus contextos. No entanto, para cada um desses serviços, eram percebidos elementos comuns, tais como: protocolos de comunicação e métodos de retenção de dados. Diante desse desafio, a Arquitetura de Portas e Adaptadores apresentou-se como um apropriado modelo arquitetural para estabelecer as normas e os requisitos necessários para as integrações.

Com a maior proximidade para a definição do tema a ser abordado no TCC, e a inerente curiosidade dos autores e do mercado em arquiteturas mais modernas, percebeu-se uma oportunidade de explorar não apenas a Arquitetura Portas e Adaptadores, mas também a visão dessa combinada à Arquitetura de Microsserviços, que compreende a modularização do sistema em partes menores, mais desacopladas e coesas. Corroborando com essa visão exploratória do estudo almejado por esse trabalho, ainda foi estabelecido um viés de preocupação que guia o olhar dos autores no que foi observado de fato, ao

longo da pesquisa. No caso, foi escolhido o viés da Reutilização de *Software*.

Destaca-se sobre a relevância de Reutilização de *Software* com base na necessidade de mercado em prover soluções no menor prazo possível. Além disso, com Reutilização de *Software*, é possível abreviar os prazos de entrega, mitigando problemas de perdas de qualidade. Isso ocorre, pois deseja-se reutilização de soluções testadas e aprovadas previamente pela comunidade especializada. Sendo assim, quando utilizadas, tendem a conferir menor esforço e menor tempo de dedicação da equipe, além de não incorrer em partir para uma solução que não se sabe sobre ela. Em soluções testadas e mais consolidadas, como uma funcionalidade empacotada em um serviço, sendo esse oferecido por uma comunidade responsável (ex. validador de CEP dos Correios), há maior chance de sucesso na implementação.

As colocações apresentadas anteriormente são fundamentadas na literatura especializada, já previamente apresentada no Capítulo 2 - Referencial Teórico, na seção dedicada à Reutilização de *Software*.

Entretanto, para conduzir o trabalho de forma a conferir insumos que auxiliem outros interessados, não apenas no entendimento sobre cada arquitetura, mas também na implementação de determinadas necessidades inerentes ao desenvolvimento de aplicações *Back-end*, são apresentadas provas de conceito.

5.2 Provas de Conceito

Uma prova de conceito deve ser focada em uma preocupação, cujos objetivos são conhecidos e delineados. Portanto, para cada prova de conceito, serão definidos os objetivos. Além disso, uma prova de conceito tende a ser, em *software*, algo mais concreto, envolvendo levantamento de requisitos, implementações, testes e análises. Essas atividades permitem prover, via prova de conceito, insumos mais informativos e práticos sobre a preocupação em foco, orientando-se sempre pelos objetivos estabelecidos. Diante do exposto, as provas de conceito compreendem, além da Definição dos Objetivos, a Identificação dos Requisitos Técnicos; a Implementação da POC, e a Revelação dos Resultados. Nesse último caso, consta uma visão combinada das etapas Teste e Validação e Análise de Resultados.

5.2.1 POC 1 - Implementação de Serviços Orientados a Portas e Adaptadores

Essa seção descreve a prova de conceito relacionada à configuração do ambiente de desenvolvimento do trabalho, no que tange a implementação de serviços orientados a portas e adaptadores. A seção começa apresentando a Definição dos Objetivos, seguida da Identificação dos Requisitos Técnicos. Depois, tem-se a Implementação da POC, que precede a seção final, sendo essa Revelação dos Resultados.

5.2.1.1 Definição dos Objetivos

A primeira prova de conceito consistiu na implementação de três serviços utilizando a Arquitetura de Portas e Adaptadores, estabelecendo o ambiente de desenvolvimento do projeto. Esses serviços foram isolados em seus contextos, assim como a Arquitetura de Microsserviços propõe, apenas utilizando como meio comum entre eles as implementações dos adaptadores que serão consumidos.

5.2.1.2 Identificação dos Requisitos Técnicos

Com o intuito de alcançar os objetivos propostos para essa prova de conceito, foram definidos os seguintes requisitos técnicos, já especificados como histórias de usuário, correspondendo a um recorte do Backlog do Produto:

- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *software*, desejo que cada serviço possua um ambiente isolado em um microsserviço;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *software*, desejo que cada serviço implemente a Arquitetura de Portas e Adaptadores;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *software*, desejo que os serviços utilizem adaptadores comuns entre si, e
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *software*, desejo que os serviços sejam capazes de se comunicarem entre si sem dificuldades.

5.2.1.3 Implementação da POC

Para a implementação dessa prova de conceito, foi feita a configuração do ambiente de desenvolvimento que foi utilizado ao longo do trabalho e em todas as provas de conceito que serão apresentadas. Para isso, utilizou-se a linguagem de programação apresentada no Capítulo 3 - Suporte Tecnológico, Python, na elaboração do código e das entidades que foram utilizadas no projeto. Além disso, também foi utilizado Docker e Docker Compose para a configuração dos contêineres utilizados para a execução dos serviços.

Foram definidos três serviços para esta prova de conceito. Todos eles de implementação semelhante, a fim de permitir a reutilização de código entre eles e, dessa forma, levantar os pontos de observação orientando-se por Reutilização de *Software*. Todos os serviços possuem uma porta de entrada HTTP, que fornece uma rota para o *ping* da aplicação, servindo apenas como um meio de comunicação para que fosse possível implementar um adaptador que permitisse a execução das requisições entre os serviços.

Figura 9 – Estrutura de Entrada do Serviço.

```python
import uvicorn
from fastapi import FastAPI

from domain import service
from adapters.http import HTTPRequestAdapter

app = FastAPI()


@app.get("/ping")
def ping():
    return service.ping()


@app.get("/ping-server-2")
def ping_server_2():
    return service.ping_server(
        HTTPRequestAdapter,
        "http://server2:8000",
        "/ping",
    )


@app.get("/ping-server-3")
def ping_server_3():
    return service.ping_server(
        HTTPRequestAdapter,
        "http://server3:8000",
        "/ping",
    )


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)

```

Fonte: Autores

A Figura 9 apresenta a estrutura de entrada do serviço, onde são implementados o servidor HTTP e as rotas definidas, tanto de *ping*, quanto de requisições entre serviços. Nessa estrutura de entrada dos serviços, na qual consta a parte da camada de domínio da aplicação, foi possível passar o adaptador necessário através dos argumentos das funções definidas. Dessa forma, depende-se apenas da interface do adaptador, e não da sua implementação, o que facilita a alteração caso seja necessária. A Figura 10 apresenta de forma mais clara a definição da função empregada nas entradas.

Figura 10 – Função Auxiliadora de Entrada do Serviço.

```python
from fastapi.responses import JSONResponse

from ports.http import HTTPPort


def ping():
    return { "ping": True }


def ping_server(
    http_adapter: HTTPPort,
    server_url: str,
    route: str
):
    response = http_adapter(server_url).get(route)
    return JSONResponse(
        status_code=response.get("status_code"),
        content=response.get("content"),
    )

```

Fonte: Autores

Um adaptador foi criado para o protocolo HTTP, o qual foi usado para a comunicação entre os serviços. Esse adaptador implementa uma interface definida pela porta de requisições HTTP, onde é especificado que um adaptador de requisições deve implementar os métodos de requisição HTTP, como GET, POST, PUT, PATCH e DELETE. A interface é apresentada conforme ilustrado na Figura 11.

Figura 11 – Porta de Requisições HTTP.

```python
class HTTPPort:
    def __init__(self, base_url: str):
        self.base_url = base_url

    def request(self, method: str, route: str, data: dict = None):
        raise NotImplementedError

    def get(self, route: str):
        raise NotImplementedError

    def post(self, route: str, data: str):
        raise NotImplementedError

    def put(self, route: str, data: str):
        raise NotImplementedError

    def delete(self, route: str):
        raise NotImplementedError

    def patch(self, route: str, data: str):
        raise NotImplementedError

```

Fonte: Autores

Como demonstrado nas Figuras 10 e 11, através da porta de requisições HTTP, é possível criar um adaptador para realizar as requisições HTTP a partir da biblioteca *requests* do Python. Assim sendo, de acordo com os procedimentos estabelecidos na interface, foram empregados os recursos desta biblioteca, criando-se um novo adaptador, conforme ilustrado na Figura 12.

Figura 12 – Adaptador de Requisições HTTP.

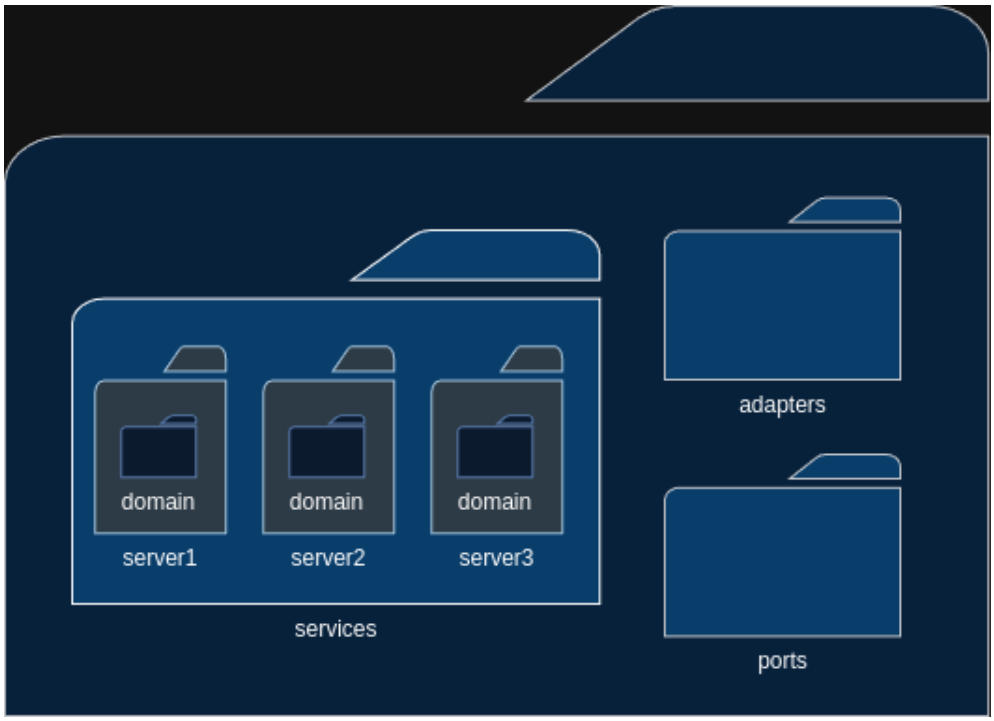
```python
import requests
from ports.http import HTTPPort

class HTTPRequestAdapter(HTTPPort):
    def request(self, method: str, route: str, data: dict = None):
        try:
            response = requests.request(method, f"{self.base_url}{route}",data=data)
            response.raise_for_status()
            return { "status_code": response.status_code, "content": response.json()}
        except (requests.exceptions.ConnectionError, requests.exceptions.Timeout) as e:
            return {"status_code": 500, "content": { "message": str(e)}}
        except requests.exceptions.HTTPError as e:
            return {"status_code": response.status_code, "content": {"message": str(e)}}

    def get(self, route: str):
        return self.request("GET", route)

    def post(self, route: str, data: dict):
        return self.request("POST", route, data=data)

    def put(self, route: str, data: dict):
        return self.request("PUT", route, data=data)

    def delete(self, route: str):
        return self.request("DELETE", route)

    def patch(self, route: str, data: dict):
        return self.request("PATCH", route, data=data)
```

Fonte: Autores

Com o objetivo de seguir os padrões arquitetônicos da Arquitetura de Portas e Adaptadores, foi estabelecida uma estrutura dos serviços com o objetivo de facilitar o uso das portas e adaptadores. Esta estrutura foi elaborada com o objetivo de permitir a reutilização de código entre os serviços definidos pela aplicação, onde seria necessário que adaptadores e portas fossem utilizados em diferentes aplicações. A Figura 13 apresenta a

organização estabelecida para os serviços, na qual há pacotes dedicados aos adaptadores (pacote *adapters*) e às portas (pacote *ports*).

Figura 13 – Estrutura da Arquitetura do Projeto.



Fonte: Autores

A Figura 13 apresenta a estrutura da arquitetura do projeto, onde é possível observar que há a implementação de múltiplos serviços dentro de um projeto só, conferindo a utilização dos adaptadores para todas as aplicações existentes. A estrutura desse projeto foi feita em inglês para adequar-se ao código, que também é desenvolvido em inglês. Isso é uma boa prática de projeto e implementação, segundo Martin (2008).

Têm-se como entidades principais dessa estrutura os adaptadores, onde há o código que foi reutilizado, seguido pelas portas que definem a interface necessária para implementação de um adaptador e os serviços, que implementam as portas de entrada e saída da aplicação através da utilização dos adaptadores. Os serviços implementados nessa prova de conceito são de caráter simples, onde cada um deles é uma replicação de uma aplicação simples, possuindo apenas rotas de *ping* e de requisições entre serviços.

Para desenvolvimento do projeto, escolheu-se a implementação de um único repositório, onde todos os serviços seriam implementados e a camada de adaptadores poderia ser utilizada por todos eles. Essa decisão foi tomada com o objetivo de facilitar a implementação dos serviços. Entretanto, os autores têm ciência da possibilidade de se ter repositórios dedicados, bem como banco de dados dedicado, para cada serviço. Dependendo do escopo do serviço, isso pode ser relevante. Para o caso dos serviços implementados na POC 1, os

mesmos são de menor escopo, bem microsserviços. Portanto, os autores optaram por uma abordagem mais simples. A seguir, encontra-se o endereço para acesso do repositório:

- Repositório: <https://github.com/Matheusafonsouza/tcc_arquitetura_pocs>

5.2.1.3.1 Contribuições

Durante o desenvolvimento desta primeira prova de conceito, foram observadas as seguintes contribuições relacionadas à utilização de Microsserviços e Arquitetura de Portas e Adaptadores em conjunto, tais como:

- Independência dos serviços, permitindo a separação de ambientes entre eles, e assim tendendo a facilitar aspectos como o escalonamento, a manutenção e o desenvolvimento, conforme colocado pela literatura especializada;
- Reutilização de código, permitindo que os serviços utilizem os mesmos adaptadores para realizar a comunicação entre eles, novamente, tendendo a facilitar o desenvolvimento e a manutenção dos mesmos, conforme apontado pela literatura especializada, e
- Facilidade na implementação e na manutenção de novos componentes e entidades, novamente, uma tendência, conforme colocado pela literatura especializada, uma vez que permite, através de uma porta previamente definida, utilizar novos adaptadores sem que incorra em "quebras"na aplicação.

5.2.1.3.2 Fragilidades

Ainda no desenvolvimento dessa primeira prova de conceito, também foram observados as seguintes fragilidades em relação à utilização de Microsserviços e à Arquitetura de Portas e Adaptadores em conjunto, sendo elas:

- Dificuldade acerca da organização dos serviços, onde as estruturas e definições do projeto estão muito separadas e, portanto, é necessária uma curva de aprendizado para entender o projeto como um todo, e
- Curva de aprendizado acerca das arquiteturas e tecnologias utilizadas, onde é necessário entender e definir bem as entidades para que não incorra em problemas futuros.

5.2.1.3.3 Breve Parecer

Com base na implementação da POC 1, percebeu-se que a combinação da Arquitetura de Portas e Adaptadores com a Arquitetura de Microsserviços tende a auxiliar na Reutilização de *Software*. Nesse sentido, observou-se que há apenas uma definição do que deve ser usado, sendo essa especificada em um adaptador, e reutilizada na implementação de diferentes serviços, evitando, dentre outras incorrências, a duplicação de código. Segundo a literatura, ao se evitar duplicação de código, há maiores chances de se mitigar situações indesejadas que, normalmente, ocorrem em tempo de manutenção evolutiva (ex. dificuldade de se evoluir um *software* devido ao fato de serem necessárias várias intervenções no código, que se encontra replicado em vários momentos). Os próprios princípios *SOLID*, tratados na seção 2.1.4.1, corroboram com essa afirmação.

Entretanto, mesmo diante dessa vantagem, para que essa solução seja implementada de forma adequada, é necessário que haja uma definição prévia da estrutura que foi utilizada, tanto para os adaptadores; quanto para as portas. Além disso, deve-se ter uma definição prévia dos serviços que serão utilizados, bem como da organização estrutural dos mesmos. Isso demanda conhecimento e mão de obra qualificada, o que pode dificultar o uso das arquiteturas.

Cabe colocar ainda que a combinação dessas arquiteturas tende a promover modularização da solução, uma vez que há necessidade de definir pequenos serviços, bem coesos. Por fim, a própria modularização acorda um menor acoplamento, o que tende a facilitar a manutenção evolutiva da solução (LARMAN, 2007), seja substituindo por completo um serviço que já não atende aos propósitos do *software*; seja melhorando o mesmo pontualmente, deixando-o mais atualizado às novas necessidades que surjam ao longo do ciclo de vida do *software*.

5.2.1.4 Revelação dos Resultados

Nessa seção, são tratadas, em conjunto, as etapas Teste e Validação e Análise de Resultados, uma vez que envolvem atividades intrinsecamente associadas.

Já procurando adiantar os principais resultados obtidos, a prova de conceito elaborada atendeu aos requisitos estabelecidos previamente, tais como:

- Cada serviço deve possuir um ambiente isolado em um microsserviço;
- Cada serviço deve implementar a Arquitetura de Portas e Adaptadores;
- Os serviços devem utilizar adaptadores comuns entre si; e
- Os serviços devem ser capazes de se comunicarem entre si sem dificuldades.

Os resultados obtidos através desta prova de conceito permitiram avançar em atendimento aos seguintes tópicos do trabalho:

- Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada, sendo uma preocupação inerente à essa arquitetura o fato de ter de isolar uma parte da aplicação em um microsserviço;

- Levantamento dos principais pontos relacionados à Arquitetura Portas e Adaptadores, usando como base a literatura especializada, sendo uma preocupação inerente à essa arquitetura o fato de ter de implementar portas e adaptadores específicos, e

- Estudo sobre Reutilização de *Software*, usando como base a literatura especializada, sendo inerente a possibilidade de reutilizar os microsserviços implementados em aplicações *Back-end* similares, além dos próprios adaptadores.

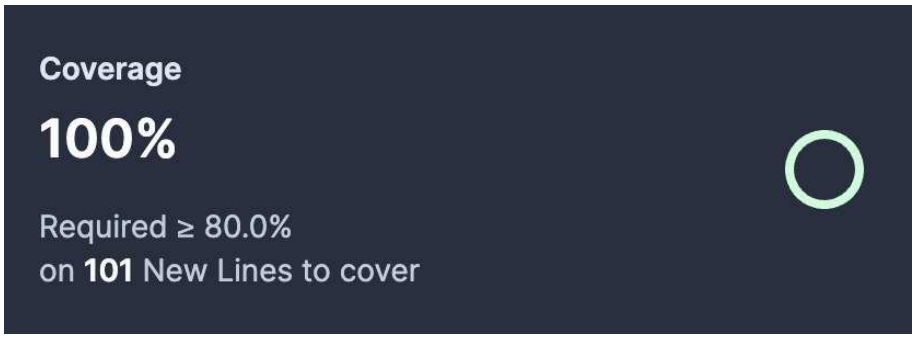
5.2.1.5 Análises Qualitativa e Quantitativa

A avaliação quantitativa das provas de conceito tem como objetivo estabelecer parâmetros concretos, com apresentação de escalas de referência, números e/ou porcentagens, sobre manutenibilidade, testabilidade (cobertura de teste) e confiabilidade, que são critérios qualitativos. Para isso, foi utilizado o SonarQube para analisar a POC 1. A análise foi feita para as métricas de Manutenibilidade e Confiabilidade, onde foram obtidos os resultado apresentados na Tabela 5. A nota A, obtida para ambos os casos, Manutenibilidade (*Maintainability*) e Confiabilidade (*Reliability*), representa a nota máxima atribuída pelo SonarQube aos critérios. Isso significa que o projeto, até o momento, encontra-se de acordo com os parâmetros de qualidade utilizados na comunidade especializada.

Analisou-se ainda a métrica Cobertura de Teste (*Coverage*) onde há uma porcentagem de 100% para as funcionalidades desenvolvidas, conforme a Figura 14. Isso significa a cobertura máxima que pode ser atribuída para uma programação, sendo assim, 100% das linhas de código foram executadas, enquanto os testes rodavam.

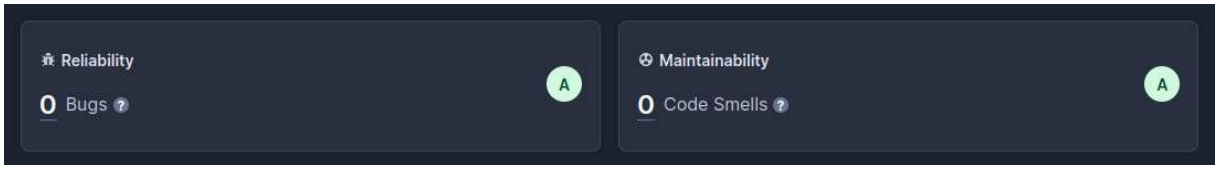
Há ainda uma breve análise de *Code Smells* e *Bugs*, vide Figura 15, ambos em 0. Sendo assim, sem registros de "Maus Cheiros no Código"e "Falhas".

Figura 14 – Cobertura de Teste Detectada pelo SonarQube - POC 1



Fonte: Autores

Figura 15 – *Bugs* e *Code Smells* Detectados pelo SonarQube - POC 1



Fonte: Autores

De acordo com SonarSource (2023), *bugs* representam um erro no código. Mesmo *bugs* que ainda não causaram um falha na aplicação representam um perigo de falha a qualquer momento. Desta forma, é recomendado sempre resolvê-los o mais brevemente possível. Como constata a Figura 15, o SonarQube detectou 0 *bugs* durante a análise da POC 1.

Ainda de acordo com SonarSource (2023), *code smells* remetem a um problema de manutenibilidade no código. Significam, portanto, que os desenvolvedores irão ter dificuldade em compreender e manter o código, podendo até mesmo introduzir novos erros, conforme novas mudanças sejam adicionadas. Durante a análise da POC 1, o SonarQube detectou 0 *code smells*, vide Figura 15.

Tabela 5 – Métricas entregues pelo SonarQube - POC 1

Métrica	**Nota**
Manutenibilidade	A
Cobertura de Teste	100%
Confiabilidade	A

Fonte: Autores

5.2.2 POC 2 - Implementação de um Adaptador para Retenção de Dados

Essa seção descreve a prova de conceito relacionada à configuração do ambiente de desenvolvimento do trabalho, no que tange a implementação de um adaptador para retenção de dados. A seção começa apresentando a Definição dos Objetivos, seguida da Identificação dos Requisitos Técnicos. Depois, tem-se a Implementação da POC, que precede a seção final, sendo essa Revelação dos Resultados.

5.2.2.1 Definição dos Objetivos

A segunda prova de conceito consiste na implementação de um adaptador, cujo proposito é a utilização de um banco de dados para retenção das estruturas que foram utilizadas na camada de dados do projeto.

5.2.2.2 Identificação dos Requisitos Técnicos

Com o intuito de alcançar os objetivos propostos para essa prova de conceito, foram definidos os seguintes requisitos técnicos, já especificados como histórias de usuário, correspondendo a um recorte do Backlog do Produto:

- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço utilize o mesmo adaptador para o banco de dados;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço utilize esquemas de dados em comum, e
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço utilize esquemas de dados isolados.

Sendo assim, o intuito dessa POC 2 foi analisar os comportamentos observáveis quando vários serviços utilizam o mesmo banco de dados, seja em esquemas em comum ou isolados. Foram registradas as principais nuances relacionadas à integração entre os serviços e o banco de dados, sendo essa uma relação comumente encontrada em aplicações *Back-end*.

5.2.2.3 Implementação da POC

Para a implementação dessa prova de conceito, foi desenvolvido um adaptador para retenção de dados. Para isso, utilizou-se o banco de dados apresentado no Capítulo 3 - Suporte Tecnológico, PostgreSQL, na elaboração do código e na implementação das estruturas de dados que foram utilizadas nessa POC. Além disso, também foi acrescentado ao Docker e Docker Compose a estrutura necessária para a utilização do banco de dados escolhido.

Um adaptador foi criado para a retenção de dados, o qual foi usado para a comunicação entre os serviços e o banco de dados utilizado pelos mesmos. Esse adaptador implementa uma interface definida pela porta de banco de dados, onde é especificado que um adaptador de retenção de dados deve implementar os métodos para criar, atualizar, remover e buscar dados das estruturas necessárias, conforme ilustrado na Figura 16, com os métodos: *create*(...), *update*(...), *delete*(...) e *get*(...).

Figura 16 – Adaptador de Retenção de Dados SQL

```python
class PostgresDatabase(DatabasePort):
    def __init__(self, database_uri: str, table: str, schema: str):
        self.table = self.get_table(table)
        engine = create_engine(database_uri)
        Session = sessionmaker(bind=engine)
        self.session = Session()

    def get_table(self, table: str):
        return {"users": User, "books": Book, "movies": Movie,"tv_shows":TvShow}.get(table)

    def create(self, data: dict):
        insert = self.table(**data)
        self.session.add(insert)
        self.session.commit()
        return insert.to_dict()

    def update(self, id: str, data: dict):
        record = self.session.query(self.table).filter(self.table.id == id).one_or_none()
        if record:
            for key, value in data.items():
                setattr(record, key, value)
            self.session.commit()
            return record.id
        return None

    def delete(self, id: str):
        record = self.session.query(self.table).filter(self.table.id == id).one_or_none()
        if record:
            self.session.delete(record)
            self.session.commit()

    def get(self, id: str):
        record = self.session.query(self.table).filter(self.table.id == id).one_or_none()
        if not record:
            return None
        return record.to_dict()

```

Fonte: Autores

Figura 17 – Porta de Retenção de Dados SQL

```python
class DatabasePort:
    def create(self, data: dict):
        raise NotImplementedError

    def update(self, data: dict):
        raise NotImplementedError

    def delete(self, id: str):
        raise NotImplementedError

    def get(self, where: dict):
        raise NotImplementedError
```

Fonte: Autores

Como demonstrado na Figura 16, através da porta de banco de dados definida, é possível criar um adaptador para realizar as operações necessárias com o objetivo de respeitar todas os métodos definidos na porta para retenção de dados. Com base no exposto na Figura 18, percebe-se que houve a definição de uma função auxiliadora para permitir a reutilização de um dado banco de dados utilizado através do adaptador desenvolvido anteriormente, onde foram definidos, para múltiplos serviços, um ponto comum para consumo de um recurso compartilhado.

Figura 18 – Função Auxiliadora da Conexão do Banco Relacional

```python
def get_postgres_database(schema: str, table: str) -> PostgresDatabase:
    return PostgresDatabase(
        (
            f"postgresql://{environ['PG_USER']}:"
            f"{environ['PG_PASS']}@{environ['PG_HOST']}:"
            f"{environ['PG_PORT']}/{environ['PG_DB']}"
        ),
        table,
        schema
    )
```

Fonte: Autores

Visando evidenciar o repositório utilizador do adaptador para o banco de dados relacional, foi definida a estrutura apresentada na Figura 19. Nela, constam métodos comuns do adaptador, estabelecidos pela porta. Além disso, o adaptador específico foi alterado, apenas modificando o campo obrigatório do construtor da classe, demonstrando ser fácil essa alteração, caso necessária em algum contexto.

Como elucidado na Figura 20, utilizou-se uma base de dados, definindo-se um adaptador de retenção de dados para um dado repositório referente à uma estrutura de dados da regra de negócio. No caso ilustrado na imagem, fez-se uso da estrutura de dados de usuário, onde, através da porta definida previamente, foram realizadas as operações necessárias com os dados existentes de usuários.

Figura 19 – Repositório Utilizador do Adaptador para Retenção de Dados

Fonte: Autores

Dessa forma, o esquema de dados contendo a Tabela *User* está disponível para todos os serviços. Para cada serviço, também foi definido um esquema de dados isolado, utilizando o mesmo adaptador, mas com outro esquema e outra tabela. A Figura 20 ilustra o uso do adaptador para o banco relacional no serviço 1, que consome um esquema de dados isolado e uma tabela específica sobre livros.

Figura 20 – Instanciação de Repositórios com o Adaptador de Retenção de Dados

Fonte: Autores

5.2.2.3.1 Contribuições

Durante o desenvolvimento desta segunda prova de conceito, foram observadas as seguintes contribuições relacionadas à utilização de Microsserviços e Arquitetura de Portas e Adaptadores em conjunto, tais como:

- Reutilização de código, permitindo que os serviços utilizem os mesmos adaptadores para realizar a retenção dos dados necessários, novamente, tendendo a facilitar o desenvolvimento e a manutenção dos mesmos, conforme apontado pela literatura especializada;
- Facilidade na utilização de diferentes esquemas de dados, tendendo a facilitar o desenvolvimento e a reutilização de código, e
- Independência da camada de domínio em relação à qualquer tecnologia de retenção de dados que se deseje utilizar, garantindo assim o isolamento da mesma e facilitando a manutenção e a escalabilidade do sistema, conforme apontado pela literatura especializada.

5.2.2.3.2 Fragilidades

Ainda no desenvolvimento dessa segunda prova de conceito, também foi observada a seguinte fragilidade em relação à utilização de Microsserviços e à Arquitetura de Portas e Adaptadores em conjunto, sendo ela:

- Aumento da complexidade, uma vez que para cada entidade se faz necessário criar o repositório utilizador do banco de dados, onde são definidos os métodos utilizados pela camada de domínio.

5.2.2.3.3 Breve Parecer

Com base na implementação da POC 2, observou-se que a combinação da Arquitetura de Portas e Adaptadores com a Arquitetura de Microsserviços tende a facilitar a Reutilização de *Software*. Esse benefício deve-se, em parte, à definição única dos métodos que serão utilizados, a qual é especificada em uma porta, implementada no adaptador e reutilizada na implementação de diferentes repositórios em diversos serviços. Isso evita, entre outras coisas, a duplicação de código.

Entretanto, mesmo diante dessa vantagem, para que essa solução seja implementada de forma adequada, é necessário que haja uma definição prévia da estrutura que foi utilizada, tanto para os adaptadores; quanto para as portas. Além disso, é fundamental ter um bom conhecimento da tecnologia de persistência de dados empregada no adaptador, permitindo abstrair a implementação dos métodos de forma a possibilitar sua reutilização por diferentes serviços, mesmo que utilizem esquemas de dados distintos.

É importante destacar que a combinação dessas arquiteturas promove a separação entre as regras de negócio e as interfaces. Como resultado, obtém-se um sistema com as seguintes características: independência de *framework*, testabilidade, independência da interface de usuário, independência do banco de dados e independência de aspectos externos. Essas características são alinhadas aos princípios da Arquitetura Limpa.

A seguir, encontra-se o endereço para acesso do repositório:

- Repositório: <https://github.com/Matheusafonsouza/tcc_arquitetura_pocs>

5.2.2.4 Revelação dos Resultados

A prova de conceito elaborada atendeu aos requisitos estabelecidos previamente, tais como:

- Cada serviço deve utilizar o mesmo adaptador para o banco de dados;
- Cada serviço deve utilizar esquemas de dados em comum, e
- Cada serviço deve utilizar esquemas de dados isolados.

Os resultados obtidos através desta prova de conceito permitiram avançar em atendimento aos seguintes tópicos do trabalho:

- Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada;
- Levantamento dos principais pontos relacionados à Arquitetura Portas e Adaptadores, usando como base a literatura especializada;
- Estudo sobre Reutilização de *Software*, usando como base a literatura especializada, e
- Especificação de componentes reutilizáveis, considerando um ou mais problema(s) recorrente(s) em termos arquiteturais.

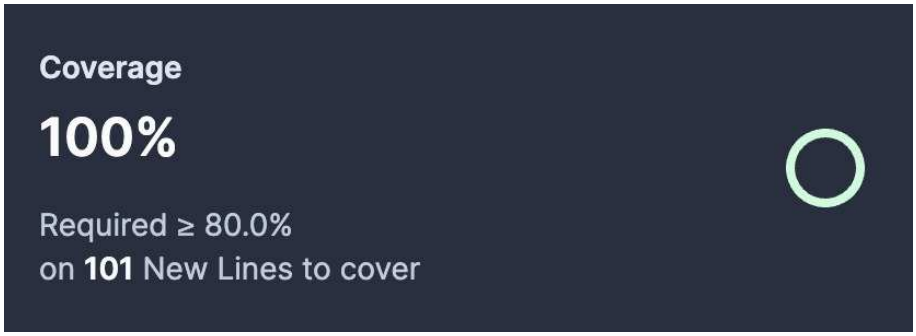
5.2.2.5 Análises Qualitativa e Quantitativa

A avaliação quantitativa das provas de conceito tem como objetivo estabelecer parâmetros concretos, com apresentação de escalas de referência, números e/ou porcentagens, sobre manutenibilidade, testabilidade (cobertura de teste) e confiabilidade, que são critérios qualitativos. Para isso, foi utilizado o SonarQube para analisar a POC 2. A análise foi feita para as métricas de Manutenibilidade e Confiabilidade, onde foram obtidos os resultado apresentados na Tabela 6. A nota A, obtida para ambos os casos, Manutenibilidade (*Maintainability*) e Confiabilidade (*Reliability*), representa a nota máxima atribuída pelo SonarQube aos critérios. Isso significa que o projeto, até o momento, encontra-se de acordo com os parâmetros de qualidade utilizados na comunidade especializada.

Analisou-se ainda a métrica Cobertura de Teste (*Coverage*) onde há uma porcentagem de 100% para as funcionalidades desenvolvidas, conforme a Figura 21. Isso significa a cobertura máxima que pode ser atribuída para uma programação, sendo assim, 100% das linhas de código foram executadas, enquanto os testes rodavam.

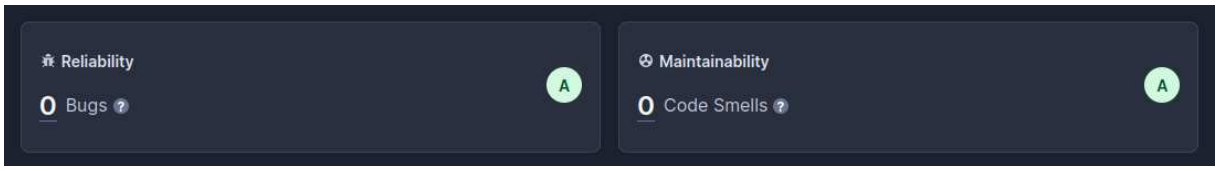
Há ainda uma breve análise de *Code Smells* e *Bugs*, vide Figura 22, ambos em 0. Sendo assim, sem registros de "Maus Cheiros no Código"e "Falhas".

Figura 21 – *Coverage* Detectado pelo SonarQube - POC 2



Fonte: Autores

Figura 22 – *Bugs* e *Code Smells* Detectados pelo SonarQube - POC 2



Fonte: Autores

Tabela 6 – Métricas entregues pelo SonarQube - POC 2

Métrica	Nota
Manutenibilidade	A
Cobertura de Teste	100%
Confiabilidade	A

Fonte: Autores

5.2.3 POC 3 - Implementação de um Adaptador para Retenção de Dados NoSQL

Essa seção descreve a prova de conceito relacionada à configuração do ambiente de desenvolvimento do trabalho, no que tange a implementação de um adaptador para retenção de dados utilizando uma ferramenta NoSQL. A seção começa apresentando a Definição dos Objetivos, seguida da Identificação dos Requisitos Técnicos. Depois, tem-se a Implementação da POC, que precede a seção final, sendo essa Revelação dos Resultados.

5.2.3.1 Definição dos Objetivos

A terceira prova de conceito consiste na implementação de um adaptador, cujo propósito é a utilização de um banco de dados NoSQL para retenção das estruturas que foram utilizadas na camada de dados do projeto. Além disso, ela demonstra a facilidade para alterar a ferramenta usada para retenção de dados dentro de uma aplicação através do adaptador e da porta especificados.

5.2.3.2 Identificação dos Requisitos Técnicos

Com o intuito de alcançar os objetivos propostos para essa prova de conceito, foram definidos os seguintes requisitos técnicos, já especificados como histórias de usuário, correspondendo a um recorte do Backlog do Produto:

- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço utilize o mesmo adaptador para o banco de dados;

- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço utilize esquemas de dados isolados, e
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que cada serviço seja capaz de alterar o adaptador para banco de dados sem dificuldade.

Sendo assim, o intuito dessa POC 3 foi analisar os comportamentos observáveis ao utilizar um banco de dados NoSQL através de diferentes serviços, empregando tanto esquemas de dados comuns, quanto isolados. Além disso, foram registradas as principais nuances relacionadas à substituição da tecnologia de retenção de dados, passando de um banco de dados SQL para um banco de dados NoSQL.

5.2.3.3 Implementação da POC

Para a implementação dessa prova de conceito, foi desenvolvido um adaptador para retenção de dados. Para isso, utilizou-se o banco de dados apresentado no Capítulo 3 - Suporte Tecnológico, MongoDB, na elaboração do código e na implementação das estruturas de dados que foram consumidas nessa POC. Além disso, também foi acrescentado ao Docker e Docker Compose a estrutura necessária para o uso do banco de dados escolhido.

Como demonstrado na Figura 24, através da porta de banco de dados definida, é possível criar um adaptador para realizar as operações necessárias com o objetivo de respeitar todas os métodos definidos na porta para retenção de dados. Com base na Figura 23, percebe-se que houve a definição de uma função auxiliadora para permitir a reutilização de um dado banco de dados utilizado através do adaptador desenvolvido anteriormente, no qual, para múltiplos serviços, foi definido um ponto comum para consumo de um recurso compartilhado.

Figura 23 – Função Auxiliadora da Conexão do Banco Não Relacional

Fonte: Autores

Figura 24 – Adaptador de Retenção de Dados NoSQL.

```python
class MongoDatabase(DatabasePort):
    def __init__(self, database_uri: str, database: str, collection: str):
        self.collection = MongoClient(database_uri)[database][collection]

    def create(self, data: dict):
        inserted_id = self.collection.insert_one({
            **data,"created_at": datetime.now(),
                "updated_at": datetime.now()}).inserted_id
        return self.get(inserted_id)

    def update(self, id: str, data: dict):
        self.collection.update_one(
            {"_id": ObjectId(id)}, { "$set": { **data, "updated_at": datetime.now()}})
        return self.get(id)

    def delete(self, id: str):
        self.collection.delete_one({"_id": ObjectId(id)})

    def get(self, id: str):
        entity = self.collection.find_one({"_id": ObjectId(id)})
        if not entity:
            return None
        entity["id"] = str(entity["_id"])
        del entity["_id"]
        return entity
```

Fonte: Autores

Assim, como elucidado na Figura 25, fez-se uso de uma base de dados, definindo-se um adaptador de retenção de dados para um dado repositório referente à uma estrutura de dados da regra de negócio. Neste contexto, reutilizou-se o repositório de usuários, apenas trocando a função auxiliadora para consumo do banco de dados não relacional, seguindo a regra de negócio definida previamente na porta de retenção de dados.

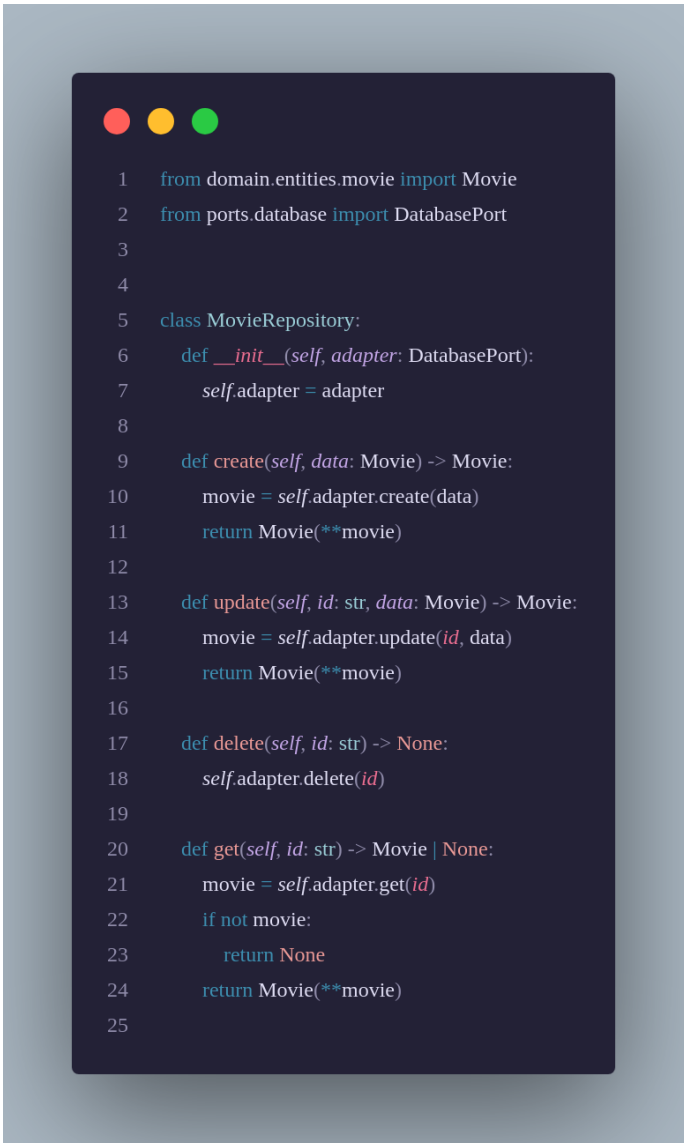
Figura 25 – Exemplo de Utilização da Função Auxiliadora

```python
from common.repositories.user_repository import UserRepository
from domain.repositories.movie_repository import MovieRepository

user_repository = UserRepository(
    adapter=get_mongo_database("commonSchema", "users"))

movie_repository = MovieRepository(
    adapter=get_mongo_database("serviceTwoSchema", "movies"))
```

Fonte: Autores

Ainda na Figura 25, é utilizado o repositório que se orienta pelo esquema de dados isolados do serviço 2, contendo nele uma tabela, ou coleção, relacionada a filmes. A tecnologia de retenção de dados pode ser escolhida por meio das funções auxiliadoras apresentadas, sem interferência no funcionamento da aplicação.

Figura 26 – Repositório que Utiliza Esquema de Dados Isolados do Serviço 2



```python
from domain.entities.movie import Movie
from ports.database import DatabasePort


class MovieRepository:
    def __init__(self, adapter: DatabasePort):
        self.adapter = adapter

    def create(self, data: Movie) -> Movie:
        movie = self.adapter.create(data)
        return Movie(**movie)

    def update(self, id: str, data: Movie) -> Movie:
        movie = self.adapter.update(id, data)
        return Movie(**movie)

    def delete(self, id: str) -> None:
        self.adapter.delete(id)

    def get(self, id: str) -> Movie | None:
        movie = self.adapter.get(id)
        if not movie:
            return None
        return Movie(**movie)

```

Fonte: Autores

5.2.3.3.1 Contribuições

Durante o desenvolvimento desta terceira prova de conceito, foram observadas as seguintes contribuições relacionadas à utilização de Microsserviços e Arquitetura de Portas e Adaptadores em conjunto, tais como:

- Reutilização de código, permitindo que os serviços utilizem os mesmos adaptadores

para realizar a retenção dos dados necessários, novamente, tendendo a facilitar o desenvolvimento e a manutenção dos mesmos, conforme apontado pela literatura especializada;

- Facilidade na implementação e na manutenção de novos componentes e entidades, novamente, uma tendência, conforme colocado pela literatura especializada, uma vez que permite, através de uma porta previamente definida, utilizar novos adaptadores sem que incorra em "quebras"na aplicação;

- Disponibilidade de uma alternativa para retenção de dados utilizando uma ferramenta diferente em um novo modelo de dados, e

- Facilidade na substituição na tecnologia de retenção de dados, sem a necessidade de alteração do código dentro do domínio de cada serviço.

5.2.3.3.2 Fragilidades

No desenvolvimento desta terceira prova de conceito, não foram observadas fragilidades na utilização de Microsserviços e da Arquitetura de Portas e Adaptadores em conjunto. Esse fato reforça a ideia de que, em situações onde é necessária a troca de sistemas de bancos de dados, a Arquitetura de Portas e Adaptadores é uma alternativa viável, conforme apontado pela literatura especializada. Além disso, quando utilizada em conjunto com a Arquitetura de Microsserviços, é possível a reutilização das portas e adaptadores em comum, aumentando a eficiência e a flexibilidade do sistema.

5.2.3.3.3 Breve Parecer

Com base na implementação da POC 3, ficou evidente que a combinação da Arquitetura de Portas e Adaptadores com a Arquitetura de Microsserviços continua a facilitar a Reutilização de *Software*. Esse benefício manifesta-se, neste caso, pela reutilização da porta de retenção de dados, que também é implementada pelo adaptador de retenção de dados NoSQL.

Além disso, destaca-se a facilidade de trocar a tecnologia de retenção de dados sem a necessidade de modificar o código dentro do domínio dos serviços. Isso reforça as vantagens discutidas na seção sobre Arquitetura de Portas e Adaptadores, evidenciando sua eficiência na abstração das tecnologias subjacentes e na manutenção da integridade do código de domínio.

A seguir, encontra-se o endereço para acesso do repositório:

- Repositório: <https://github.com/Matheusafonsouza/tcc_arquitetura_pocs>

5.2.3.4 Revelação dos Resultados

A prova de conceito elaborada atendeu aos requisitos estabelecidos previamente, tais como:

- Cada serviço deve utilizar o mesmo adaptador para o banco de dados;
- Cada serviço deve utilizar esquemas de dados isolados, e
- Cada serviço deve ser capaz de alterar o adaptador para banco de dados sem dificuldade.

Os resultados obtidos através desta prova de conceito permitiram avançar em atendimento aos seguintes tópicos do trabalho:

- Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada;
- Levantamento dos principais pontos relacionados à Arquitetura Portas e Adaptadores, usando como base a literatura especializada;
- Estudo sobre Reutilização de *Software*, usando como base a literatura especializada, e
- Especificação de componentes reutilizáveis, considerando um ou mais problema(s) recorrente(s) em termos arquiteturais.

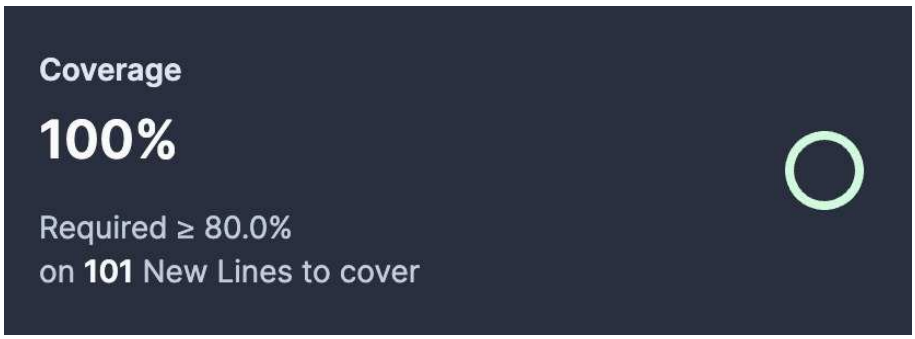
5.2.3.5 Análises Qualitativa e Quantitativa

A avaliação quantitativa das provas de conceito tem como objetivo estabelecer parâmetros concretos, com apresentação de escalas de referência, números e/ou porcentagens, sobre manutenibilidade, testabilidade (cobertura de teste) e confiabilidade, que são critérios qualitativos. Para isso, foi utilizado o SonarQube para analisar a POC 3. A análise foi feita para as métricas de Manutenibilidade e Confiabilidade, onde foram obtidos os resultado apresentados na Tabela 7. A nota A, obtida para ambos os casos, Manutenibilidade (*Maintainability*) e Confiabilidade (*Reliability*), representa a nota máxima atribuída pelo SonarQube aos critérios. Isso significa que o projeto, até o momento, encontra-se de acordo com os parâmetros de qualidade utilizados na comunidade especializada.

Analisou-se ainda a métrica Cobertura de Teste (*Coverage*) onde há uma porcentagem de 100% para as funcionalidades desenvolvidas, conforme a Figura 27. Isso significa a cobertura máxima que pode ser atribuída para uma programação, sendo assim, 100% das linhas de código foram executadas, enquanto os testes rodavam.

Há ainda uma breve análise de *Code Smells* e *Bugs*, vide Figura 28, ambos em 0. Sendo assim, sem registros de "Maus Cheiros no Código"e "Falhas".

Figura 27 – *Coverage* Detectado pelo SonarQube - POC 3



Fonte: Autores

Figura 28 – *Bugs* e *Code Smells* Detectados pelo SonarQube - POC 3

Fonte: Autores

Tabela 7 – Métricas entregues pelo SonarQube - POC 3

	Métrica	**Nota**
	Manutenibilidade	A
	Cobertura de Teste	100%
	Confiabilidade	A

Fonte: Autores

5.2.4 POC 4 - Implementação de um Novo Serviço Focado em *Logging*

Essa seção descreve a prova de conceito relacionada à configuração do ambiente de desenvolvimento do trabalho, no que tange a implementação de um novo serviço focado em *logging*. A seção começa apresentando a Definição dos Objetivos, seguida da Identificação dos Requisitos Técnicos. Depois, tem-se a Implementação da POC, que precede a seção final, sendo essa Revelação dos Resultados

5.2.4.1 Definição dos Objetivos

A quarta prova de conceito consiste na implementação de um novo serviço focado em *logging*, o qual pode ser utilizado pelos outros serviços existentes dentro do contexto do projeto, através de uma fila. Esse serviço de *logging* permitiu manter os *logs* em memória, dispondo os dados das aplicações. Além disso, ele também manteve os dados em um banco de dados não relacional. Isso incorreu na necessidade de uso do adaptador criado na POC 3.

5.2.4.2 Identificação dos Requisitos Técnicos

Com o intuito de alcançar os objetivos propostos para essa prova de conceito, foram definidos os seguintes requisitos técnicos, já especificados como histórias de usuário, correspondendo a um recorte do Backlog do Produto:

- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que o serviço possua um ambiente isolado em um microsserviço;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que o serviço implemente a Arquitetura de Portas e Adaptadores;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que o serviço seja capaz de ser utilizado pelos outros serviços;
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que o serviço implemente um adaptador para consumir mensagens de uma fila, e
- Eu, como desenvolvedor e conhecedor da área de Arquitetura de *Software*, desejo que o serviço implemente um adaptador para manter dados em memória.

Sendo assim, o intuito dessa POC 4 foi bem mais abrangente. Portanto, incluiu, dentre outros aspectos: reafirmar sobre o isolamento de um microsserviço; reafirmar sobre o alinhamento de um serviço à implementação de portas e adaptadores; perceber o quão fácil um serviço pode ser utilizado por outro serviço; experienciar aspectos de concorrência nas arquiteturas, usando um recurso compartilhado e o conceito de uma fila; compreender sobre requisições e protocolos envolvidos, e lidar com dados, mantendo-os em memória e na camada de persistência. Novamente, são cenários de uso comumente encontrados em aplicações *Back-end*.

5.2.4.3 Implementação da POC

Para a implementação dessa prova de conceito, foi desenvolvido um serviço focado em *logging* para reter qualquer informação necessária para *debug* gerada pelos serviços

principais. Para isso, utilizou-se o servidor de mensageria apresentado no Capítulo 3 - Suporte Tecnológico, RabbitMQ, na elaboração do código para envio e consumo dos dados de *logging* que foram utilizados nessa POC. Também foi acrescentado ao Docker e Docker Compose a estrutura necessária para o uso do banco de dados escolhido.

Um adaptador foi criado para a comunicação entre os serviços principais e o serviço responsável pelo *logging* dos dados utilizando a fila de mensagerias do RabbitMQ. Esse adaptador implementa uma interface definida pela porta de protocolo AMQP, onde é especificado que um adaptador de retenção de dados deve implementar os métodos de publicar e consumir mensagens do *broker*.

Figura 29 – Porta de Comunicação AMQP.

```python
class AMQPPort:
    def send_message(self, message: dict):
        raise NotImplementedError

    def receive_messages(self):
        raise NotImplementedError

```

Fonte: Autores

Como demonstrado na Figura 30, através da porta de comunicação AMQP, é possível criar um adaptador para realizar as operações necessárias com o objetivo de respeitar todas os métodos definidos na porta para comunicação AMQP. Com base na 31, percebe-se que houve a definição de uma função auxiliadora para permitir a reutilização da conexão do *broker* via adaptador desenvolvido anteriormente, onde foram definidos, para múltiplos serviços, um ponto comum para uso de um recurso compartilhado.

Figura 30 – Adaptador de Comunicação AMQP Utilizando RabbitMQ.

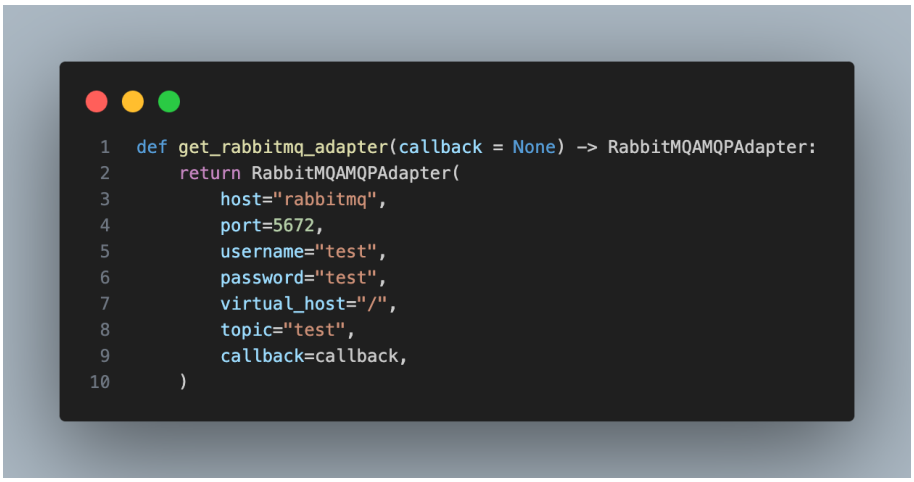
```python
class RabbitMQAMQPAdapter(AMQPPort):
    def __init__(self, host, port, username, password, virtual_host, topic, callback = None):
        self.host = host
        self.port = port
        self.username = username
        self.password = password
        self.virtual_host = virtual_host
        self.topic = topic
        self.callback = callback

    def get_session(self):
        return MQSession(
            host=self.host,
            port=self.port,
            username=self.username,
            password=self.password,
            virtual_host=self.virtual_host,
        )

    def send_message(self, message: dict):
        with self.get_session() as session:
            channel = session.channel()
            channel.exchange_declare(
                exchange="rabbitmq", exchange_type="topic")
            channel.basic_publish(
                exchange="rabbitmq",
                routing_key=self.topic,
                body=json.dumps(message),
            )

    def receive_messages(self):
        with self.get_session() as session:
            channel = session.channel()
            channel.exchange_declare(
                exchange="rabbitmq", exchange_type="topic")
            result = channel.queue_declare("", exclusive=True)
            queue = result.method.queue
            channel.queue_bind(
                exchange="rabbitmq", queue=queue, routing_key=self.topic)
            channel.basic_consume(
                queue=queue,
                on_message_callback=self.callback,
                auto_ack=True)
            channel.start_consuming()
```

Fonte: Autores

Assim, como elucidado na Figura 31, fez-se uso de um *broker* de mensageria, definindo-se um adaptador de comunicação AMQP para um dado servidor, e utilizando

o mesmo adaptador tanto para consumir mensagens existentes que foram publicadas de diversos serviços para o serviço de *logging*, como utilizar o adaptador para enviar as mensagens para o serviço de *logging*. Nesse cenário de uso, percebe-se um recurso comum para inúmeros outros servidores consumirem. Seguem detalhes desse cenários de uso, evidenciando ainda o envio de mensagem via adaptador, Figura 32, e uso da função auxiliadora, Figura 33.

Figura 31 – Função Auxiliadora da Conexão com o *Broker* RabbitMQ



```python
def get_rabbitmq_adapter(callback = None) -> RabbitMQAMQPAdapter:
    return RabbitMQAMQPAdapter(
        host="rabbitmq",
        port=5672,
        username="test",
        password="test",
        virtual_host="/",
        topic="test",
        callback=callback,
    )
```

Fonte: Autores

Figura 32 – Exemplo de Envio de Mensagem Através do Adaptador

```python
from ports.http import HTTPPort
from ports.amqp import AMQPPort
from fastapi.responses import JSONResponse


def ping():
    return { "ping": True }


def ping_server(
    http_adapter: HTTPPort,
    amqp_adapter: AMQPPort,
    server_url: str,
    route: str
):
    response = http_adapter(server_url).get(route)
    amqp_adapter.send_message(response)
    return JSONResponse(
        status_code=response.get("status_code"),
        content=response.get("content"),
    )

```

Fonte: Autores

Figura 33 – Exemplo de Utilização da Função Auxiliadora

```python
import json

from ports.amqp import AMQPPort
from common.database import get_mongo_database
from common.amqp import get_rabbitmq_adapter
from domain.repositories.log_repository import LogRepository


log_repository = LogRepository(
    adapter=get_mongo_database("logs")
)


def callback(channel, method, properties, body):
    message = json.loads(body.decode("utf-8"))
    print("INFO -> ", message)
    log_repository.create(({"message": message}))


class Worker:
    def __init__(self, adapter: AMQPPort):
        self.adapter = adapter

    def handle_message(self):
        self.adapter.receive_messages()


def main():
    Worker(
        adapter=get_rabbitmq_adapter(callback),
    ).handle_message()


if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

```

Fonte: Autores

5.2.4.3.1 Contribuições

Durante o desenvolvimento desta quarta prova de conceito, foram observadas as seguintes contribuições relacionadas à utilização de Microsserviços e Arquitetura de Portas e Adaptadores em conjunto, tais como:

- Independência dos serviços, permitindo a separação de ambientes entre eles, e assim tendendo a facilitar aspectos como o escalonamento, a manutenção e o desenvolvimento, conforme colocado pela literatura especializada;
- Reutilização de código, permitindo que os serviços utilizem os mesmos adaptadores para realizar a comunicação entre eles, novamente, tendendo a facilitar o desenvolvimento e a manutenção dos mesmos, conforme apontado pela literatura especializada, e
- Facilidade na implementação e na manutenção de novos componentes e entidades, novamente, uma tendência, conforme colocado pela literatura especializada, uma vez que permite, através de uma porta previamente definida, utilizar novos adaptadores sem que incorra em "quebras"na aplicação.

5.2.4.3.2 Fragilidades

No desenvolvimento desta quarta prova de conceito, não foram observadas fragilidades na utilização de Microsserviços e da Arquitetura de Portas e Adaptadores em conjunto. Esse fato reforça a ideia de que, em situações onde é necessária a troca de informações entre diferentes serviços, a Arquitetura de Portas e Adaptadores é uma alternativa viável. Além disso, é possível a reutilização das portas e adaptadores responsáveis tanto pelo envio e recepção de mensagens, quanto pela retenção dos dados em uma camada de persistência, aumentando a eficiência no desenvolvimento do sistema.

5.2.4.3.3 Breve Parecer

Com base na implementação da POC 4, observou-se que essa abordagem facilita a reutilização do serviço de *logging* sem a necessidade de acoplamento direto entre os serviços. A fila de mensagens permite que múltiplos serviços enviem *logs* simultaneamente, sem que isso impacte o desempenho ou a escalabilidade do sistema. A Arquitetura de Portas e Adaptadores, neste contexto, abstrai a complexidade de integração, garantindo que a comunicação entre os serviços seja consistente e independente da implementação específica do serviço de *logging*.

Entretanto, mesmo diante dessa vantagem, para que essa solução seja implementada de forma adequada, é necessário que haja uma definição prévia da estrutura que foi utilizada, tanto para os adaptadores; quanto para as portas. Além disso, é fundamental ter um bom conhecimento da tecnologia de mensageria empregada no adaptador, permitindo abstrair a implementação dos métodos de forma a possibilitar sua reutilização por diferentes serviços, mesmo que utilizem formatos de *logs* distintos.

A seguir, encontra-se o endereço para acesso do repositório:

- Repositório: <https://github.com/Matheusafonsouza/tcc_arquitetura_pocs>

5.2.4.4 Revelação dos Resultados

A prova de conceito elaborada atendeu aos requisitos estabelecidos previamente, tais como:

- O serviço deve possuir um ambiente isolado em um microsserviço;
- O serviço deve implementar a Arquitetura de Portas e Adaptadores;
- O serviço deve ser capaz de ser utilizado pelos outros serviços;
- O serviço deve implementar um adaptador para consumir mensagens de uma fila, e
- O serviço deve implementar um adaptador para manter dados em memória.

Os resultados obtidos através desta prova de conceito permitiram avançar em atendimento aos seguintes tópicos do trabalho:

- Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada;
- Levantamento dos principais pontos relacionados à Arquitetura Portas e Adaptadores, usando como base a literatura especializada;
- Estudo sobre Reutilização de *Software*, usando como base a literatura especializada;
- Especificação de componentes reutilizáveis, considerando um ou mais problema(s) recorrente(s) em termos arquiteturais;
- Implementação de um adaptador, por parte de um serviço, visando manter dados em memória, e
- Registro dos comportamentos de relevância - no contexto de Reutilização de *Software* - em uma aplicação orientada às Arquiteturas Microsserviços e Portas e Adaptadores.

5.2.4.4.1 Análises Qualitativa e Quantitativa

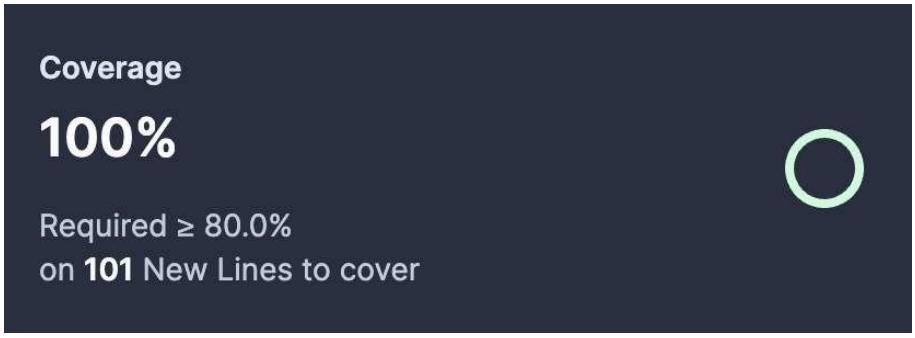
A avaliação quantitativa das provas de conceito tem como objetivo estabelecer parâmetros concretos, com apresentação de escalas de referência, números e/ou porcentagens, sobre manutenibilidade, testabilidade (cobertura de teste) e confiabilidade, que são critérios qualitativos. Para isso, foi utilizado o SonarQube para analisar a POC 4. A análise foi feita para as métricas de Manutenibilidade e Confiabilidade, onde foram obtidos os

resultado apresentados na Tabela 8. A nota A, obtida para ambos os casos, Manutenibilidade (*Maintainability*) e Confiabilidade (*Reliability*), representa a nota máxima atribuída pelo SonarQube aos critérios. Isso significa que o projeto, até o momento, encontra-se de acordo com os parâmetros de qualidade utilizados na comunidade especializada.

Analisou-se ainda a métrica Cobertura de Teste (*Coverage*) onde há uma porcentagem de 100% para as funcionalidades desenvolvidas, conforme a Figura 34. Isso significa a cobertura máxima que pode ser atribuída para uma programação, sendo assim, 100% das linhas de código foram executadas, enquanto os testes rodavam.

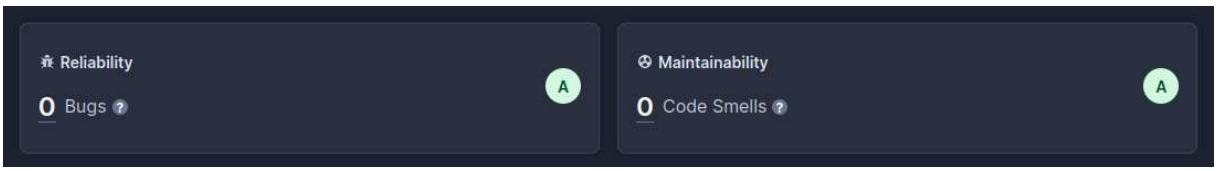
Há ainda uma breve análise de *Code Smells* e *Bugs*, vide Figura 35, ambos em 0. Sendo assim, sem registros de "Maus Cheiros no Código"e "Falhas".

Figura 34 – *Coverage* Detectado pelo SonarQube - POC 4



Fonte: Autores

Figura 35 – *Bugs* e *Code Smells* Detectados pelo SonarQube - POC 4



Fonte: Autores

Tabela 8 – Métricas entregues pelo SonarQube - POC 4

Métrica	**Nota**
Manutenibilidade	A
Cobertura de Teste	100%
Confiabilidade	A

Fonte: Autores

5.3 Considerações Finais do Capítulo

Este capítulo apresentou o estudo sobre as Arquiteturas de Microsserviços e Portas e Adaptadores, centradas em Reutilização de *Software*. O estudo foi apresentado como um processo, uma vez que demandou implementações para geração de insumos mais concretos, permitindo uma análise dos comportamentos arquiteturais de maior relevância.

Ao longo do processo, ocorreu a implementação de Microsserviços orientando-se pela Arquitetura de Portas e Adaptadores em aplicações *Back-end*. Para cada prova de conceito documentada, foram apresentados os objetivos; os requisitos técnicos; os detalhes da implementação; as contribuições e fragilidades observadas; a revelação dos resultados, bem como as análises realizadas com o SonarCloud.

Conclui-se, ao final do estudo, que as Arquiteturas de Microsserviços e Portas e Adaptadores, em uso combinado, são bem pertinentes para Reutilização de *Software*. Isso ocorre devido a diferentes aspectos, mas cabe mencionar a facilidade de integração, proporcionada pelos adaptadores e portas, e a própria modularização, promovida pela subdivisão de responsabilidades em asbstrações de microsserviços. Entretanto, conforme reportado no capítulo, fragilidades foram encontradas, com especial atenção à maior complexidade em termos de projeto e implementação, demandando conhecimento e uma curva de aprendizado mais acentuada por parte de quem desenvolve.

Parte VI

Análise de Resultados

6 Análise de Resultados

Este capítulo apresenta demais insumos oriundos das análises de resultados realizadas em cada prova de conceito, complementando as análises já documentadas no Capítulo 5 - Estudo Exploratório. Inicialmente, será fornecida uma tabela contendo os Principais Resultados das Provas de Conceito, sendo acompanhada por descrições sobre cada resultado. Depois, serão expostas as Contribuições e Fragilidades observadas durante o desenvolvimento das provas de conceito através do método de análise de resultados, pesquisa e ação (já esclarecido no Capítulo 4 - Metodologia, na seção 4.5 Método de Análise de Resultados). Em seguida, será apresentada a Revisão por Pares, orientando-se pelas respostas dos especialistas na área de interesse desse trabalho conferidas ao Questionário de Revisão por Pares das Provas de Conceito. Há ainda um um resumo das opiniões dos revisores sobre cada prova de conceito. Ao final, têm-se o Parecer dos Autores, concluindo o processo de Análise de Resultados do trabalho, bem como as Considerações Finais do Capítulo.

6.1 Principais Resultados das Provas de Conceito

Baseando-se na Pesquisa Bibliográfica, descrita no Capítulo 4 - Metodologia - seção 4.2 Método Investigativo, bem como apoiando-se no planejamento e no desenvolvimento orientados às práticas ágeis, ocorreu a apresentação das soluções para cada cenário de uso via provas de conceito, conforme detalhado ao longo do Capítulo 5 - Estudo Exploratório.

A intenção no momento, é conferir uma visão resumo, na qual os principais resultados são apresentados, conforme acordado na Tabela 9. Esses resultados foram obtidos com uso da ferramenta SonarCloud, sendo esse um apoio tecnológico que já promove uma análise prévia dos valores aferidos para cada métrica de interesse. Essa análise prévia leva em consideração boas práticas da área de Engenharia de *Software*, que permitem aferir a qualidade da métrica observada a partir do valor obtido. Sendo assim, a título de ilustração, pode-se considerar a métrica Manutenibilidade.

A Manutenibilidade é aferida com base em várias outras métricas. As mais utilizadas são: Complexidade Ciclomática (afere o quão acoplado o código está), Complexidade Cognitiva (afere o quão difícil é entender o fluxo de controle do código), Cobertura de Condição (afere a densidade de condição ao longo da execução dos testes de unidade), Cobertura de Condição no Novo Código (mesma intenção de aferição da anterior, mas centrada no código novo ou atualizado) e Cobertura de Testes (afere a cobertura total, compreendendo a combinação das coberturas anteriores). Para cada métrica, há cálculos

específicos, gerando resultados. Esses resultados são valores - análise quantificável. De posse de cada valor, bem como das boas práticas da Engenharia de *Sofware*, a ferramenta já estabelece uma análise de cunho qualitativo, atribuindo notas, especificadas com letras do alfabeto. A melhor nota em termos qualitativos é a NOTA A. Portanto, o quão mais próxima a análise qualitativa de uma métrica estiver da NOTA A, melhor, ou seja, mais de acordo aquela métrica está com as convenções e parâmetros estabelecidos pela comunidade da Engenharia de *Software*. Ressalta-se ainda que as coberturas são especificadas em porcentagens, sendo 100% a porcentagem máxima de cobertura.

Dentre as métricas escolhidas para esse trabalho, e que merecem destaque nessa análise, estão: Confiabilidade (*Reliability*), Cobertura de Testes (*Coverage*) e Manutenibilidade (*Maintainability*). Essas métricas foram escolhidas, uma vez que possibilitam obter uma percepção mais clara sobre a qualidade daquele cenário de uso (implementado) em aspectos como, por exemplo: se a Manutenibilidade é adequada, isso tende a facilitar a inserção de novas funcionalidades, bem como a evolução das já existentes, e até mesmo a remoção das que não atendem mais; ou ainda se há adequada Confiabilidade na programação, isso tende a facilitar a evolução de algo a partir dessa programação confiável; ou ainda se há Cobertura de Testes adequada, isso gera insumos mais concretos de que reutilizar essas "porções de soluções"em contextos similares é viável. Cabe reparar que essas "inferências"corroboram de uma forma ou de outra com a Reutilização de *Software*, que é um conceito de relevância para esse trabalho.

Tabela 9 – Resultados da análise quantitativa das Provas de Conceito

Prova de Conceito	**Critério 1**	**Critério 2**	**Critério 3**
POC 1	A	100%	A
POC 2	A	100%	A
POC 3	A	100%	A
POC 4	A	100%	A

Fonte: Autores

- Critério 1 representa o padrão de qualidade atingido no aspecto de Confiabilidade;
- Critério 2 representa a porcentagem de Cobertura de Testes, e
- Critério 3 representa o padrão de qualidade atingido no aspecto de Manutenibilidade.

Conforme esclarecido anteriormente, o SonarCloud utiliza um padrão de qualidade para avaliar o código das aplicações. Isso incorre na análise de vários fatores, incluindo: quantidade de *bugs*, vulnerabilidades, cobertura de código, duplicações, entre

outros. Quando uma aplicação atende aos padrões mais altos de qualidade definidos pelo SonarCloud para um critério específico, ela recebe a NOTA A nesse aspecto.

Diante do exposto, é importante ressaltar que todas as soluções desenvolvidas nesse trabalho (i.e. todas as provas de conceito) cumpriram os critérios de qualidade estabelecidos pelo SonarCloud.

6.2 Contribuições e Fragilidades

Durante o desenvolvimento da solução de cada prova de conceito, foram observadas contribuições e fragilidades. Esses aspectos são importantes para o trabalho, pois através deles pode-se realizar uma análise adicional sobre a pertinência ou não do uso combinado das Arquiteturas de Porta e Adaptadores e Microsserviços, além de identificar situações que beneficiem a Reutilização de *Software*.

- Independência dos serviços: pode-se implementar determinadas funcionalidades do sistema, mesmo que pequenas em escopo, usando o conceito de Microsserviço. Isso permite modularizar o código em componentes arquiteturais com responsabilidades mais bem definidas, tornandos-os mais coesos e, em sua totalidade, um código menos acoplado;

- Reutilização de código: uma vez que os componentes arquiteturais são coesos, independentes, e encapsulados em pequenos serviços, reutilizá-los torna-se uma tarefa menos custosa e mais viável de ser realizada;

- Facilidade da manutenção e desenvolvimento de novos componentes e entidades: podendo reutilizar mais facilmente os componentes arquiteturais, além de inserir novos componentes, remover os já existentes que encontram-se legados, e atualizar componentes para melhor adequação à realidade do *software*, são ações que descomplicam manter o *software* como um todo;

- Facilidade na utilização de diferentes esquemas de dados: usando portas e adaptadores, tem-se uma padronização nos protocolos de comunicação com sistemas terceiros, ou seja, externos à parte *core* da aplicação. Isso facilita a integraçào com esses sistemas terceiros, permitindo, adicionalmente, trocá-los quando necessário. Sistemas de gerenciamento de dados podem ser vistos como um desses sistemas terceiros. Portanto, trocar esquemas de dados torna-se uma tarefa mais tranquila quando usando uma Arquitetura como Portas e Adaptadores;

- Independência da camada de domínio em relação à qualquer tecnologia de retenção de dados que se deseje utilizar: aqui, tem-se uma facilidade oriunda da própria

facilidade de integração comentada no item anterior. Como dito, a camada de domínio fica protegida, alocada mais internamente na aplicação. A comunicação com terceiros é feita por camadas mais externas e via portas e adaptadores. A camada de domínio mantém-se encapsulada, mitingando impactos de terceiros nela, e

- Facilidade na substituição de ferramentas e interfaces, possibilitando uma alteração sem dificuldades: aqui, novamente, assim como visto para o caso dos esquemas de dados, que podem ser substituídos com facilidade, devido ao uso de portas e adaptadores que padronizam a comunicação e facilitam a integração com sistemas terceiros, o mesmo ocorre para o caso das interfaces gráficas e ferramentas adicionais.

As fragilidades mais significativas observadas foram:

- Dificuldade acerca da organização dos serviços: apesar de ambas as arquiteturas contribuírem para a obtenção de uma solução computacional mais bem modularizada, a tarefa de estruturar uma solução em módulos coesos, menos acoplados, cujas comunicações são padronizadas com portas e adaptadores, não são tarefas triviais de serem alcançadas. Isso ainda é mais complicado em se tratando de contextos que demandam soluções em curto espaço de tempo, sem histórico de desenvolvimento no domínio de interesse, bem como contando com mão de obra não qualificada nesse sentido. Desenvolver de forma mais criteriosa demanda tempo, responsabilidade e conhecimento. Portanto, as competências e habilidades inerentes nesse processo não são conquistadas de forma rápida pelas equipes de desenvolvimento, dificultando o uso dessas arquiteturas mais sofisticadas por equipes ou empresas mais novatas;

- Curva de aprendizado intensa acerca das arquiteturas e tecnologias utilizadas: foi perceptível para os autores, que já desenvolvem *software* em empresas bem sucedidas no mercado, que implementar algo usando essa visão combinada de Arquiteturas de Portas e Adaptadores e Microsserviços exigiu esforço, estudo e tempo para lidar com cada cenário de uso. Sendo assim, não são arquiteturas que podem ser sugeridas para desenvolvimento de uma solução, sem a devida recomendação de que a equipe precisa se capacitar antes nas nuances das arquiteturas, e

- Aumento da complexidade: esse ponto é muito interessante, pois é algo que precisa ser avaliado caso a caso. No geral, percebe-se que há inerente aumento da complexidade ao se implementar orientando-se por arquiteturas mais sofisticadas. De certa forma, isso é esperado. Essas arquiteturas demandam padronização, modularização, alta coesão nos componentes arquiteturais, baixo acoplamento geral no código, uso de interfaces e protocolos bem estabelecidos e especificados, dentre vários outros aspectos. Conferir esforço para cada um desses aspectos só vale quando isso representará uma solução computacional que será de longa vida, evoluída ao longo

de vários anos, cabendo investir um pouco mais desde de sua origem. Portanto, não é recomendado o uso dessa solução combinada para contextos simples, que não incorrerão na necessidade de reutilização, ou em manutenções a longo prazo. Nesses casos mais simples, o uso de algo mais sofisticado só representará aumento de complexidade, sem usufruto das contribuições mencionadas anteriormente.

6.3 Revisão por Pares

Visando uma análise adicional sobre os insumos obtidos nesse trabalho, com o desenvolvimento das provas de conceito, ocorreu a revisão por pares. Essa revisão demandou consultar especialistas na área de atuação desse trabalho, ou seja, especialistas em Arquitetura de *Software*, para que os mesmos respondessem um conjunto de perguntas. Essas perguntas foram formuladas com base nos comportamentos arquiteturais observados e reportados ao longo do Capítulo 5 - Estudo Exploratório. Dentre os comportamentos, destacam-se: baixo acoplamento dos componentes arquiteturais; facilidade de reutilização de componentes arquiteturais, e facilidade de inserção, atualização e remoção dos componentes arquiteturais.

Os autores entraram em contato com os especialistas, uma vez que atuam em empresas conhecidas no mercado, nas quais há boas equipes de arquitetos de *software*. Para evitar exposição de dados não necessários para a avaliação desse trabalho, algumas informações são tratadas de forma anônima. Cabe mencionar que todos os participantes concordaram em participar, sendo apresentado a eles o Termo de Consentimento Livre e Esclarecido (TCLE), conforme consta no Apêndice A.

O perfil do revisor é acordado na persona, ilustrada na Figura 36. Com base em Ferreira, Barbosa e Conte (2018), persona é uma representação fictícia de um interessado. No caso do contexto em questão, esse interessado representa o revisor. São acrescentados ainda para cada revisor um nome genérico (Revisor A e Revisor B), conforme consta na Tabela 10. Um dos revisores trabalha no Distrito Federal, enquanto o outro está em Ontário, no Canadá, ambos atuando em empresas na área de Engenharia de *Software* com portfólios de projetos em diferentes domínios cognitivos, bem como com mais de 10 funcionários.

Na Tabela 10, consta o perfil da persona instanciado para dois revisores.

Figura 36 – Perfil do Revisor - Persona Revisor

Fonte: Autores

Tabela 10 – Revisores

Nome	**Revisor A**	**Revisor B**
Faixa Etária	Atendida	Atendida
Educação e Experiência		
Grau de Instrução	Graduado	Graduado
Experiência	CTO (*Chief Technology Officer*), com 5 anos de atuação em Arquitetura de *Software*	Desenvolvedor Sênior, com 8 anos de atuação em Arquitetura de *Software*
Habilidades & Competências Desejadas		
Conhecimento Tecnológico	Atendido	Atendido
Conhecimentos Adicionais	Atendido	Atendido

Fonte: Autores

As Tabelas 11 e 12 apresentam as respostas dos revisores, de acordo com o Capítulo 4 (seção 4.3.4), sobre a solução e os comportamentos apresentados em cada prova de conceito. As perguntas foram:

- Pergunta 1: Há de fato baixo acoplamento entre os componentes arquiteturais?

- Pergunta 2: Há facilidade em termos de reutilização de componentes entre os microsserviços?
- Pergunta 3: Há facilidade de inserção de novos componentes?
- Pergunta 4: Há facilidade de remoção de componentes que não atendem mais as necessidades?
- Pergunta 5: Você concorda com os comportamentos relatados pelos autores? Se não concordar, explique quais são esses comportamentos e o motivo.

Para as perguntas de 1 a 4, eram esperadas respontas "Sim"ou "Não". Já para o caso da Pergunta 5, optou-se, além da resposta "Sim"ou "Não", por um esclarecimento textual complementar, caso a resposta fosse "Não". Levou-se em consideração a necessidade do questionário ser respondido em menor tempo, uma vez que isso foi solicitado pelos respondentes, como condição de participação da pesquisa.

Tabela 11 – Resumo das respostas do Revisor A

Prova de Conceito	**Pergunta 1**	**Pergunta 2**	**Pergunta 3**	**Pergunta 4**	**Pergunta 5**
Questionário da POC 1	Sim	Sim	Sim	Sim	Sim*
Questionário da POC 2	Sim	Sim	Sim	Sim	Sim*
Questionário da POC 3	Sim	Sim	Sim	Sim	Sim*
Questionário da POC 4	Sim	Sim	Sim	Sim	Sim

Fonte: Autores

Tabela 12 – Resumo das respostas do Revisor B

Prova de Conceito	**Pergunta 1**	**Pergunta 2**	**Pergunta 3**	**Pergunta 4**	**Pergunta 5**
Questionário da POC 1	Sim	Sim	Sim	Sim	Sim
Questionário da POC 2	Sim	Sim	Sim	Sim	Sim
Questionário da POC 3	Sim	Sim	Sim	Sim	Sim
Questionário da POC 4	Sim	Sim	Sim	Sim	Sim

Fonte: Autores

6.3.1 Adendos às Respostas

Pelo fato da Pergunta 5 possuir uma natureza mais aberta, o Revisor A, em particular, adicionou algumas observações mesmo respondendo "Sim". Portanto, segue esse complemento descritivo, conferido à cada prova de conceito (PoC 1, 2 e 3) pelo Revisor A.

Ressalta-se ainda que foi mantida a escrita conferida pelo revisor, na íntegra, sem ajustes (nem em questões ortográficas).

- POC 1: "Sim, eu concordo que ficou muito simples a criação e remoção de novos componentes, porem isso também aumentou a complexidade do código, sendo menos trivial entender o funcionamento do serviço 'por trás dos panos'."
- POC 2: "Sim, com a adição de que o serviço e acesso ao banco de dados ficou muito limitado, possibilitando apenas usos específicos do banco de dados, como um *get* por *id*, porem não permitindo uma *query* mais complexa, assim como outros tipos de operações menos triviais."
- POC 3: "Sim, porém gostaria de acrescentar sobre uma fragilidade observada. De fato a modificação de trocar de uma tecnologia de banco de dados para outra ficou muito simples de ser feita, porem o uso de cada uma é quase idêntico, o que dificulta a utilização das especialidades e vantagens de uso de uma tecnologia específica."

6.3.2 Comentários Adicionais

No questionário aplicado para cada prova de conceito, além das perguntas, foi dedicado um espaço para os revisores adicionarem algum comentário com base em suas próprias experiências ao usarem as arquiteturas de interesse desse trabalho. Nesse sentido, o Revisor A não encontrou algo relevante para acrescentar, até mesmo pelo fato dele ter utilizado o espaço da Pergunta 5 para complementar sobre suas impressões quanto aos tópicos cobertos no questionário. Entretanto, no caso do Revisor B, o mesmo adicionou comentários relevantes para as POCs 3 e 4, nesses espaços adicionais. Seguem as principais colocações:

- POC 3: "Não é muito comum precisar alterar o componente de persistência de dados, e quando isso ocorre, geralmente é necessário um esforço significativo para migrar os dados entre as duas tecnologias, o que ja torna o procedimento acoplado."
- POC 4: "Parece funcionar bem para abstrair a comunicação entre o componente que cria mensagens e o que consome."

A primeira colocação, sobre a POC 3, revela um aspecto interessante para reflexão, sendo ele o fato de não ser muito comum a necessidade de alterar a camada de persistência de dados. Nesse caso, uma das facilidades reportadas nos resultados desse Estudo Exploratório, como contribuição no uso da combinação das Arquiteturas Portas e Adaptadores e Microsserviços, é justamente permitir essa troca dos sistemas de gerenciamento de dados com mais tranquilidade. Com esse retorno do Revisor B, cabe uma ponderação adicional sobre essa questão ao se tomar a decisão sobre usar ou não essas

arquiteturas em um projeto em particular. Entretanto, é difícil prever esse aspecto em se tratando de manutenibilidade de longa vida. Por exemplo, com o conhecimento sobre Banco de Dados Não Relacional avançando, muitas empresas estão migrando suas bases de dados, de forma completa ou parcial, para esse modelo de gerenciamento de dados. Nesses casos, seria pertinente que os sistemas dessas empresas estivessem projetados para essa alteração, mitingando os impactos em outros aspectos do sistema. De toda forma, esse comentáro do Revisor B conferiu essa questão que precisa ser mais bem avaliada, até mesmo em trabalhos futuros, que complementem o Estudo Exploratório realizado até o momento.

A segunda colocação, sobre a POC 4, reforça uma das contribuições já esclarecidas com base nos resultados obtidos nesse trabalho, sendo o fato das portas e dos adaptadores padronizarem a comunicação entre os componentes arquiteturais, facilitando a interação e a intregração desses componentes.

6.4 Parecer dos Autores

Com base na implementaçao das provas de conceito e com a revisão por pares junto aos especialistas, percebeu-se que a combinação da Arquitetura de Portas e Adaptadores com a Arquitetura de Microsserviços tende a auxiliar na Reutilização de *Software*. Nesse sentido, observou-se que há a definição do que deve ser usado, sendo essa definição especificada por uma porta e implementada por um adaptador. Esse último é então reutilizado na implementação de diferentes serviços, evitando, dentre outras incorrências, a duplicação de código.

Segundo a literatura, ao se evitar duplicação de código, há maiores chances de se mitigar situações indesejadas que, normalmente, ocorrem em tempo de manutenção evolutiva (ex. dificuldade de se evoluir um *software* devido ao fato de serem necessárias várias intervenções no código, que se encontra replicado em vários momentos). Os próprios princípios *SOLID*, tratados no Capítulo 2 (seção 2.1.4.1), corroboram com essa afirmação.

Além disso, foi possível observar que a combinação dessas duas arquiteturas promove sistemas com baixo acoplamento entre seus microsserviços e os componentes que os constituem. Isso significa que a camada de domínio de cada serviço é isolada, e as regras de negócio são independentes das tecnologias empregadas, seja para comunicação, armazenamento de dados ou outros aspectos. Nesse contexto, verificou-se que, através das interfaces definidas pelas portas, qualquer alteração de código necessária devido a mudanças nas tecnologias utilizadas ocorre exclusivamente nos adaptadores. Isso garante o isolamento da camada de domínio. Conforme acordado na literatura, ao isolar a camada de domínio, o sistema adquire características importantes, tais como: independência de *frameworks*; testabilidade; independência da interface de usuário; independência do banco

de dados; e independência de aspectos externos, como tratado no Capítulo 2 (seção 2.1.4).

Perante o exposto, cabe colocar que a combinação dessas arquiteturas tende a promover modularização da solução, uma vez que há necessidade de definir pequenos serviços, bem coesos. Por fim, a própria modularização acorda um menor acoplamento, o que tende a facilitar a manutenção evolutiva da solução (LARMAN, 2007), seja substituindo por completo um serviço que já não atende aos propósitos do *software*; seja melhorando o mesmo pontualmente, deixando-o mais atualizado às novas necessidades que surjam ao longo do ciclo de vida do *software*.

Entretanto, mesmo diante dessas vantagens, para que essa solução seja implementada de forma adequada, é necessário que haja uma definição prévia da estrutura que foi utilizada, tanto para os adaptadores; quanto para as portas. Além disso, deve-se ter uma definição prévia dos serviços que serão utilizados, bem como da organização estrutural dos mesmos. Isso demanda conhecimento e mão de obra qualificada, o que pode dificultar o uso das arquiteturas.

Também é pertinente avaliar a necessidade de aplicar essas arquiteturas considerando as particularidades de projeto, como abordado no Capítulo 2 (seção 2.1.3), uma vez que, conforme identificado durante o desenvolvimento da aplicação e destacado pelos revisores, tal combinação pode levar a desvantagens significativas. Entre elas, destacam-se o aumento da complexidade do código e a limitação no uso das especificidades das ferramentas empregadas. Essas fragilidades podem surgir da necessidade de abstrair os adaptadores para que estejam alinhados com as interfaces definidas pelas portas.

6.5 Considerações Finais do Capítulo

Este capítulo apresentou uma perpectiva complementar sobre a análise dos resultados obtidos no Estudo Exploratório realizado, fornecendo uma visão detalhada da análise quantitativa e qualitativa de cada prova de conceito. Foram discutidos os resultados da análise do SonarCloud, abrangendo aspectos como Confiabilidade, Manutenibilidade e Cobertura de Testes de cada prova de conceito. Além disso, foram reportadas as contribuições e fragilidades observadas pelos autores em cada prova de conceito, conferindo esclarecimentos adicionais sobre os principais comportamentos arquiteturais observados ao longo do desenvolvimento de cada POC.

Ocorreu ainda a apresentação da revisão por pares, a qual contou com a participação de dois especialistas da área de Arquitetura de *Software*. Esses revisores responderam, via questionário, perguntas orientadas aos principais comportamentos observados no Estudo Exploratório, tendo a oportunidade de concordar ou não com cada comportamento, bem como prover comentários adicionais sobre as Arqutieturas Portas e Adaptadores e Microsserviços com bases em suas experiências profissionais.

Por fim, os próprios autores do trabalho conferiram um parecer com reflexões baseadas nas experiências proporcionadas com a realização desse trabalho, bem como na revisão por pares. Concluí-se que o uso combinado dessas arquiteturas confere contribuições, em especial no tange à Reutilização de *Software* e aspectos correlatos. Entretanto, esse uso deve ser ponderado para cada projeto, considerando principalmente que: (i) há dificuldade de estruturar os componentes arquiteturais usando as boas práticas de cada arquitetura; (ii) agrega complexidade ao projeto, e (iii) demanda mão de obra qualificada.

Parte VII

Conclusão

7 Conclusão

Este capítulo apresenta a conclusão do Trabalho de Conclusão de Curso, bem como os resultados alcançados durante seu desenvolvimento. Inicialmente, será retomado o Status do Trabalho, o qual inclui as atividades realizadas nas duas etapas do trabalho. Em sequência, será demonstrado o Status de Cumprimento dos Objetivos, buscando retomar os objetivos que foram definidos para o trabalho. Em seguida, busca-se conferir uma Resposta à Questão de Pesquisa e, por fim, destacar Trabalhos Futuros para o projeto.

7.1 Status do Trabalho

A primeira etapa do vigente trabalho teve como objetivo a elaboração das atividades que fundamentam e trazem base para a temática da proposta, que é entender os principais pontos positivos e negativos acerca da implementação da Arquitetura de Portas e Adaptadores de forma combinada com a Arquitetura de Microsserviços, buscando a Reutilização de Software. Com base no Fluxo de Atividades e nos Cronogramas de Atividades estabelecidos para esse trabalho, a Tabela 13 apresenta o andamento das atividades e subprocessos da etapa inicial deste Trabalho de Conclusão de Curso.

Tabela 13 – Andamento das Atividades e dos Subprocessos da Etapa Inicial

Atividade/Subprocesso	**Andamento**
Definir o Tema	Concluída
Conduzir a Pesquisa Bibliográfica	Concluída
Formular a Proposta Inicial	Concluída
Desenvolver o Referencial Teórico	Concluída
Estabelecer Suporte Tecnológico	Concluída
Descrever a Metodologia	Concluída
Refinar Proposta	Concluída
Criar a Prova de Conceito Inicial	Concluída
Descrever os Resultados Parciais	Concluída
Apresentar o TCC1	Concluída

Fonte: Autores

Na etapa final deste trabalho, o foco foi o desenvolvimento das provas de conceito propostas, buscando trazer para a prática e elucidar as questões levantadas na etapa inicial do trabalho, utilizando-se da pesquisa-ação descrita no Método de Análise de Resultados para realizar a coleta e a análise dos resultados. A Tabela 14 apresenta o andamento das atividades e subprocessos da etapa final deste Trabalho de Conclusão de Curso.

Tabela 14 – Andamento das Atividades e dos Subprocessos da Etapa Final

Atividade/Subprocesso	**Andamento**
Aplicar Correções	Concluída
Realização das Atividades de Desenvolvimento	Concluída
Realização de Análise de Resultados	Concluída
Finalizar a Monografia	Concluída
Apresentar o TCC2	Em Andamento

Fonte: Autores

7.2 Status de Cumprimento dos Objetivos

Para trazer à conclusão deste trabalho, vale ressaltar os objetivos que foram especificados no Capítulo de Introdução, permitindo compreender de maneira mais completa os mesmos agora que as provas de conceito e a análise dos resultados foram realizadas. Ressalta-se ainda sobre quais objetivos foram cumpridos ou não. Conferindo o status atual quanto ao cumprimento dos mesmos, seguem os apontamentos:

- **Levantamento dos principais pontos relacionados à Arquitetura de Microsserviços, usando como base a literatura especializada.** Status: Concluído no Referencial Teórico - Arquitetura de Microsserviços;
- **Levantamento dos principais pontos relacionados à Arquitetura de Portas e Adaptadores, usando como base a literatura especializada.** Status: Concluído no Referencial Teórico - Arquitetura de Portas e Adaptadores;
- **Estudo sobre Reutilização de *Software*, usando como base a literatura especializada.** Status: Concluído no Referencial Teórico - Reutilização de *Software*;
- **Especificação de componentes reutilizáveis, considerando um ou mais problema(s) recorrente(s) em termos arquiteturais.** Status: Concluído no Estudo Exploratório;
- **Registro dos comportamentos de relevância - no contexto de Reutilização de *Software* - em uma aplicação orientada às Arquiteturas Microsserviços e Portas e Adaptadores.** Status: Concluído na Análise de Resultados.

7.3 Resposta à Questão de Pesquisa

Para a conclusão deste estudo, foi levantada uma questão de pesquisa que fundamentou os passos e o caminho para as tomadas das decisões posteriores. A partir de todos

os expostos e das evidências presentes nesse Trabalho de Conclusão de Curso, conclui-se que:

1. Quais são os comportamentos de relevância observados ao longo do desenvolvimento de uma aplicação que combina **Arquitetura de Microsserviços e Arquitetura Portas e Adaptadores, considerando como foco Reutilização de *Software***?

   Com base nos resultados apresentados orientando-se pelo Estudo Exploratório realizado, pode-se mencionar os principais comportamentos de relevância observados no contexto, organizando-os como Comportamentos Desejados e Comportamentos Indesejados.

   Em termos de Comportamentos Desejados, têm-se: Independência dos serviços; Reutilização de código; Facilidade da manutenção e desenvolvimento de novos componentes e entidades; Facilidade na utilização de diferentes esquemas de dados; Independência da camada de domínio em relação à qualquer tecnologia de retenção de dados que se deseje utilizar, e Facilidade na substituição de ferramentas e interfaces, possibilitando uma alteração sem dificuldades.

   Em termos de Comportamentos Indesejados, têm-se: Dificuldade acerca da organização dos serviços; Curva de aprendizado intensa acerca das arquiteturas e tecnologias utilizadas, e Aumento da complexidade.

   Demais colocações são apresentadas em detalhes na seção 6.2 Contribuições e Fragilidades.

   Adicionalmente, os autores gostariam de acrescentar que a implementação da Arquitetura de Microsserviços combinada à Arquitetura de Portas e Adaptadores, considerando como foco a Reutilização de Software, é algo que agrega valor ao desenvolvimento de aplicações e sistemas, gerando uma base consistente e sólida para lidar com a alta mutabilidade dos serviços, seja por adequações tecnológicas, seja por adequações funcionais ou qualitativas. Sendo assim, essa combinação de arquiteturas permite lidar com a necessidade de contratos de serviços abertos e não fechados e de difícil adaptação/adequação. Essa necessidade é inerente nas demandas de *software* (CANCIAN; ALMEIDA; ALVARO, 2009).

7.4 Trabalhos Futuros

Com base no que foi apresentado ao decorrer do Trabalho de Conclusão de Curso, torna-se relevante destacar alguns pontos de melhoria para que se possa atender de forma ainda mais adequada e abrangente a questão de pesquisa. Alguns desses pontos que merecem menção são:

1. Adicionar adaptadores e portas que sirvam como protocolo de comunicação base da aplicação, podendo alterar de maneira fácil o ferramental utilizado para a comunicação do cliente com o servidor;

2. Adicionar os adaptadores e portas criadas em uma biblioteca compartilhada, podendo assim utilizar a estrutura apresentada em repositórios separados, e

3. Adicionar adaptadores de serviços externos como provedores de computação e serviços terceiros, buscando entender como a reutilização irá se comportar com dados externos.

Referências

AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: . [S.l.: s.n.], 2013. p. 9–16. Citado na página 63.

BACKENDLESS. *WHAT IS BACKEND AS A SERVICE?* 2017. Disponível em: <https://backendless.com/what-is-backend-as-a-service>. Citado na página 42.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software architecture in practice*. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 34.

BERGMAN, K. et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, v. 15, p. 181, 2008. Citado na página 25.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYŁEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, IEEE, v. 10, p. 20357–20374, 2022. Citado na página 35.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYłEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, v. 10, p. 20357–20374, 2022. Citado na página 37.

BORGES, L. E. *Python para desenvolvedores: aborda Python 3.3*. [S.l.]: Novatec Editora, 2014. Citado na página 47.

BOUWERS, E. et al. A cognitive model for software architecture complexity. p. –, 2010. Citado na página 34.

BROWN, P. *What is Hexagonal Architecture?* 2014. Disponível em: <https://culttt.com/2014/12/31/hexagonal-architecture#what-are-the-benefits-of-hexagonal-architecture>. Citado na página 38.

CANCIAN, M.; ALMEIDA, J. R. d. C.; ALVARO, A. Uma proposta para elaboração de contrato de nível de serviço para software-as-a-service (saas). In: *VIII Simpósio Brasileiro de Qualidade de Software*. Maringá, PR: SBC, 2009. p. 153–166. Citado na página 127.

CARVALHO, B. V. d.; MELLO, C. H. P. Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. *Gestão & Produção*, SciELO Brasil, v. 19, p. 557–573, 2012. Citado na página 63.

COMPOSE, D. 2024. Disponível em: <https://docs.docker.com/compose/>. Citado na página 51.

CONCEIÇÃO, M. T. da; PINTO, G. S. Arquitetura de microsserviços. *Revista Interface Tecnológica*, v. 18, n. 2, p. 53–64, 2021. Citado 2 vezes nas páginas 35 e 36.

DOCKER. 2024. Disponível em: <https://www.docker.com/>. Citado na página 51.

Escola DNC. *Fundamentos de Desenvolvimento Back-end - Guia Completo*. 2024. Accessed: 2024-06-29. Disponível em: <https://www.escoladnc.com.br/blog/fundamentos-de-desenvolvimento-backend-guia-completo/>. Citado na página 42.

FERREIRA, B.; BARBOSA, S.; CONTE, T. Creating personas focused on representing potential requirements to support the design of applications. In: *Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2018. (IHC '18). ISBN 9781450366014. Disponível em: <https://doi.org/10.1145/3274192.3274207>. Citado na página 115.

FILHO, W. d. P. P. *Engenharia de software*. 4. ed. Rio de Janeiro: LTC, 2019. Citado na página 43.

FOWLER, M. *Enterprise Architecture*. 2003. Note in martinFowler.com. Disponível em: <https://martinfowler.com/bliki/EnterpriseArchitecture.html>. Citado 2 vezes nas páginas 43 e 60.

FOWLER, M. *GUI Architectures*. 2006. Note in martinFowler.com. Disponível em: <https://martinfowler.com/eaaDev/uiArchs.html>. Citado 2 vezes nas páginas 43 e 60.

FRAKES, W.; KANG, K. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, v. 31, n. 7, p. 529–536, 2005. Citado 2 vezes nas páginas 42 e 60.

GARLAN, D. Software architecture: a roadmap. In: *Proceedings of the Conference on the Future of Software Engineering*. [S.l.: s.n.], 2000. p. 91–101. Citado 2 vezes nas páginas 34 e 60.

GERHARDT, T.; SILVEIRA, D. *Métodos de Pesquisa*. PLAGEDER, 2009. (Série Educação a Distância - UFRGS). ISBN 9788538600718. Disponível em: <https://books.google.com.br/books?id=dRuzRyEIzmkC>. Citado 3 vezes nas páginas 57, 58 e 64.

GIL, A. C. et al. *Como elaborar projetos de pesquisa*. [S.l.]: Atlas São Paulo, 2002. v. 4. Citado na página 58.

GIT. 2024. Disponível em: <https://www.git-scm.com/>. Citado na página 50.

GRIFFIN, J. Hexagonal-driven development. *Domain-Driven Laravel: Learn to Implement Domain-Driven Design Using Laravel*, Springer, p. 521–544, 2021. Citado na página 26.

IBM. *MongoDB*. 2024. <https://www.ibm.com/br-pt/topics/mongodb>. Accessed: 2024-07-21. Citado na página 49.

JAISWAL, M. Software architecture and software design. *International Research Journal of Engineering and Technology (IRJET) e-ISSN*, p. 2395–0056, 2019. Citado na página 25.

JUNIOR, J. d. J. N. d. S. et al. Uma arquitetura orientada a serviços para visualização de dados em dispositivos inteligentes. Universidade Federal do Pará, 2014. Citado na página 38.

KRAFTA, L. et al. O método da pesquisa-ação: um estudo em uma empresa de coleta e análise de dados. *Revista Quanti & Quali*, 2007. Citado na página 65.

KRUEGER, C. W. Software reuse. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 24, n. 2, p. 131–183, jun 1992. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/130844.130856>. Citado 2 vezes nas páginas 42 e 60.

LAB, F. *Tipos de Provas de Conceito*. 2023. Acesso em: 08 dez. 2023. Disponível em: <https://fiemglab.com.br/tipos-de-provas-de-conceito/>. Citado 2 vezes nas páginas 62 e 75.

LANO, S. Y. T. K. *Introduction to Software Architecture: Innovative Design using Clean Architecture and Model-Driven Engineering*. Springer, 2023. (Undergraduate topics in computre science). ISBN 9783031441424,9783031441431. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=6520446BDC2BC41D417C36ADBF3E3884>. Citado 2 vezes nas páginas 40 e 41.

LARMAN, C. *Utilizando UML e padrões*. [S.l.]: Bookman Editora, 2007. Citado 3 vezes nas páginas 27, 83 e 120.

LATEX. 2024. Disponível em: <https://www.latex-project.org/>. Citado na página 52.

Lucidchart. *Lucidchart: Software de Diagramas*. 2024. Acessado em: 7 de agosto de 2024. Disponível em: <https://www.lucidchart.com/pages/pt>. Citado na página 50.

LUCRéDIO, D. *Uma abordagem orientada a modelos para reutilização de software*. Tese (Doutorado) — Universidade de São Paulo, 2009. Citado na página 27.

MARQUES, R. *RabbitMQ: o que é e como utilizar*. 2018. Disponível em: <https://cedrotech.com/blog/rabbitmq-o-que-e-e-como-utilizar/>. Citado na página 48.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st. ed. [S.l.]: Pearson, 2008. ISBN 978-0132350884. Citado na página 81.

MARTIN, R. C. Clean architecture: A craftsman's guide to software structure and design. Prentice Hall; Auflage: 01 (20. September, 2017. Citado 4 vezes nas páginas 26, 38, 39 e 40.

MARTIN, R. C. *Arquitetura Limpa - O Guia do Artesão para Estrutura e Design de Software*. Alta Books, 2019. ISBN 9788550804002. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=0BC6969AD1129EED85C07A20D25A8E42>. Citado na página 61.

MARTINEZ, P. *Hexagonal Architecture, there are always two sides to every story*. 2021. Disponível em: <https://medium.com/ssense-tech/hexagonal-architecture-there-are-always-two-sides-to-every-story-bc0780ed7d9c>. Citado na página 38.

NEWMAN, S. *Building microservices*. [S.l.]: "O'Reilly Media, Inc.", 2021. Citado 2 vezes nas páginas 36 e 60.

NUNKESSER, R. Using hexagonal architecture for mobile applications. 2022. Citado 3 vezes nas páginas 37, 38 e 60.

OLAH, G. *An introduction to RabbitMQ – What is RabbitMQ?* 2020. Disponível em: <https://www.erlang-solutions.com/blog/an-introduction-to-rabbitmq-what-is-rabbitmq/>. Citado na página 48.

PONCE, F.; MÁRQUEZ, G.; ASTUDILLO, H. Migrating from monolithic architecture to microservices: A rapid review. In: IEEE. *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. [S.l.], 2019. p. 1–7. Citado 2 vezes nas páginas 26 e 27.

POSTGRESQL. 2024. Disponível em: <https://www.postgresql.org/>. Citado na página 49.

PRASANNA, K. et al. Poc design: A methodology for proof-of-concept (poc) development on internet of things connected dynamic environments. *Security and Communication Networks*, Hindawi, v. 2021, p. 7185827, 2021. Disponível em: <https://doi.org/10.1155/2021/7185827>. Citado 2 vezes nas páginas 58 e 62.

RODRIGUES, C. S. C.; WERNER, C. M. L. Uma revisão sistemática sobre as iniciativas realizadas no ensino de arquitetura de software. *Relatório técnico ES-728*, v. 9, 2009. Citado na página 33.

RODRIGUES, J. L.; PINTO, G. S. Análise da arquitetura de microserviços. 2019. Citado na página 36.

SANTIAGO, C. P. et al. Desenvolvimento de sistemas web orientado a reuso com python, django e bootstrap. *Sociedade Brasileira de Computação*, 2020. Citado na página 47.

SHARVARI, T.; NAG, K. S. A study on modern messaging systems-kafka, rabbitmq and nats streaming. *CoRR abs/1912.03715*, 2019. Citado na página 48.

SILVA, P. V. d. *Microfrontends e Arquitetura Limpa: Um Estudo Exploratório Orientado a Provas de Conceito*. 98 p. Dissertação (Monografia de TCC) — Universidade de Brasília - Campus Gama, 2023. Citado na página 75.

SINGH, H.; HASSAN, S. I. Effect of solid design principles on quality of software: An empirical assessment. *International Journal of Scientific & Engineering Research*, v. 6, n. 4, p. 1321–1324, 2015. Citado 2 vezes nas páginas 40 e 41.

SLACK. 2024. Disponível em: <https://www.slack.com/>. Citado na página 51.

SonarSource. *SonarQube Documentation*. 2023. Acesso em: 2023-12-14. Disponível em: <https://docs.sonarsource.com/sonarqube/9.8/user-guide/concepts/>. Citado na página 85.

SONARSOURCE. 2024. Disponível em: <https://www.sonarqube.org>. Citado na página 49.

SonarSource. *Software Qualities*. 2024. Acessado em: 7 de agosto de 2024. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/user-guide/clean-code/software-qualities/>. Citado 2 vezes nas páginas 49 e 50.

SonarSource. *SonarQube Concepts*. 2024. Acessado em: 7 de agosto de 2024. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/user-guide/concepts/>. Citado na página 50.

SonarSource. *Test Coverage Overview*. 2024. Acessado em: 7 de agosto de 2024. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/analyzing-source-code/test-coverage/overview/>. Citado na página 49.

SORDI, J. O. D.; MARINHO, B. d. L.; NAGY, M. Benefícios da arquitetura de software orientada a serviços para as empresas: análise da experiência do abn amro brasil. *JISTEM-Journal of Information Systems and Technology Management*, SciELO Brasil, v. 3, p. 19–34, 2006. Citado na página 33.

TEAMS. 2024. Disponível em: <https://www.microsoft.com/pt-br/microsoft-teams/>. Citado na página 51.

TELEGRAM. 2024. Disponível em: <https://web.telegram.org/>. Citado na página 51.

TRAJANO, M. A. et al. Mocktests: uma ferramenta para modelagem eficiente de testes envolvendo end-points de aplicações back-end.. Universidade Federal de Campina Grande, 2022. Citado na página 42.

VAROTO, A. C. *Visões em arquitetura de software*. Tese (Doutorado) — Universidade de São Paulo, 2002. Citado 2 vezes nas páginas 26 e 27.

VILLACA, L.; AZEVEDO, L.; JR, A. Construindo aplicações distribuídas com microsserviços. In: _____. [S.l.: s.n.], 2018. p. 1–40. ISBN 978-85-7669-452-6. Citado na página 36.

Apêndices

APÊNDICE A – Termo de Consentimento

Este apêndice tem como objetivo apresentar o Termo de Consentimento Livre e Esclarecido assinado pelos participantes da Revisão por Pares. As Figuras 37 e 38 mostram, respectivamente, as páginas 1 e 2 do termo de consentimento.

Figura 37 – Termo de Consentimento - Página 1

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Título do Projeto de pesquisa: **Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados**

Pesquisadores Responsáveis: **Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho**

Nome do participante:

Data de nascimento:

Você está sendo convidado (a) para ser participante do Projeto de pesquisa intitulado "**Arquitetura Portas e Adaptadores em Microsserviços Orientando-se por Reutilização de Software: Um Estudo sobre os Comportamentos Arquiteturais Observados**" de responsabilidade dos pesquisadores **Matheus Afonso de Souza e Thiago Mesquita Peres Nunes de Carvalho.**
Leia cuidadosamente o que se segue e pergunte sobre qualquer dúvida que você tiver. Caso se sinta esclarecido (a) sobre as informações que estão neste Termo e aceite fazer parte do estudo, peço que assine ao final deste documento, em duas vias, sendo uma via sua e a outra dos pesquisadores responsáveis pela pesquisa. Saiba que você tem total direito de não querer participar.

1. O trabalho tem por objetivo realizar um levantamento de comportamentos de relevância observados ao longo do desenvolvimento de uma aplicação que combina Arquitetura de Microsserviços e Arquitetura Portas e Adaptadores, considerando como foco Reutilização de Software.

2. A participação nesta pesquisa consistirá em uma revisão por pares com o objetivo de avaliar as provas de conceito propostas no trabalho e desenvolvidas em uma aplicação *back-end*. A partir da revisão será passado um formulário para a realização da avaliação de aspectos de cunho arquitetural e que permitam reutilização de *software* em soluções similares.

3. Os participantes não terão nenhuma despesa ao participar da pesquisa e poderão retirar sua concordância na continuidade da pesquisa a qualquer momento.

4. Não há nenhum valor econômico a receber ou a pagar aos voluntários pela participação, no entanto, caso haja qualquer despesa decorrente desta participação haverá o seu ressarcimento pelos pesquisadores.

Rubrica do pesquisador 1: _____. Rubrica do participante: _____.
Rubrica do pesquisador 2: _____.

Fonte: Autores

Figura 38 – Termo de Consentimento - Página 2

Universidade de Brasília

5. Caso ocorra algum dano comprovadamente decorrente da participação no estudo, os voluntários poderão pleitear indenização, segundo as determinações do Código Civil. (Lei nº 10.406 de 2002) e das Resoluções 466/12 e 510/16 do Conselho Nacional de Saúde.

6. O nome dos participantes será mantido em sigilo, assegurando assim a sua privacidade, e se desejarem terão livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências, enfim, tudo o que queiram saber antes, durante e depois da sua participação.

7. Os dados coletados serão utilizados única e exclusivamente para fins desta pesquisa, e os resultados poderão ser publicados.

Qualquer dúvida, pedimos a gentileza de entrar em contato com Thiago Mesquita Peres Nunes de Carvalho, pesquisador responsável pela pesquisa, telefone: (61) 98102-3632, e-mail: *thiago099carvalho@gmail.com*

Eu, _____, RG nº _____ declaro ter sido informado e concordo em ser participante do Projeto de pesquisa acima descrito.

____ de _____ de 2024.

_____
Assinatura do participante

_____
Nome e assinatura do responsável por obter o consentimento

Rubrica do pesquisador 1: _____. Rubrica do participante: _____.
Rubrica do pesquisador 2: _____.

Fonte: Autores