RESIGeT On-Line
Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica
Vol. 15 Nro. 1 2025
Uni-FACEF
Centro Universitário Municipal de Franca

Análise comparativa entre arquiteturas de *software*: monolítica, modular e microsserviços

Lucas Rodrigues Cintra
Graduando em Engenharia de *Software* – Uni-FACEF
lucasrcintra10@hotmail.com

Pedro Paulino Martiniano
Graduando em Engenharia de *Software* – Uni-FACEF
pedropmartiniano@gmail.com

Leandro Borges
Doutor em Computação – Uni-FACEF
leandro.borges@facef.br

Resumo

A crescente complexidade dos sistemas de *software* contemporâneos tem intensificado a busca por modelos arquiteturais capazes de assegurar robustez, escalabilidade e sustentabilidade. Este Trabalho de Conclusão de Curso realiza uma análise comparativa entre três abordagens predominantes: arquitetura monolítica, monólito modular e microsserviços. A pesquisa, fundamentada em revisão bibliográfica sistemática, investigou diferentes parâmetros técnicos, organizacionais e de negócio, como escalabilidade, manutenibilidade, custo operacional, resiliência, segurança e adequação ao tamanho da equipe, com o objetivo de identificar vantagens, limitações e contextos de aplicação ideais de cada paradigma. Os resultados indicam que a arquitetura monolítica é mais adequada para projetos de pequeno e médio porte, oferecendo simplicidade inicial, mas apresentando limitações de escalabilidade e flexibilidade. O monólito modular equilibra simplicidade e organização, promovendo modularidade interna e facilitando uma possível evolução para microsserviços. Já a arquitetura de microsserviços mostra-se mais apropriada a sistemas de grande escala e equipes distribuídas, proporcionando maior resiliência e liberdade tecnológica, porém ao custo de maior complexidade e demanda por infraestrutura especializada. Como contribuição prática, este estudo apresenta um quadro comparativo e um guia de apoio à decisão, visando auxiliar profissionais e estudantes na seleção da arquitetura mais adequada a diferentes contextos.

Palavras-chave: arquitetura de *software*; monólito; monólito modular; microsserviços; análise comparativa; engenharia de *software*.

Abstract

The increasing complexity of contemporary software systems has intensified the search for architectural models capable of ensuring robustness, scalability, and sustainability. This Course Conclusion Paper presents a comparative analysis of three predominant approaches: monolithic architecture, modular monolith, and microservices. The research, based on a systematic literature review, investigated technical, organizational, and business parameters, such as scalability, maintainability,

operational cost, resilience, security, and team size suitability, in order to identify the advantages, limitations, and ideal application contexts of each paradigm. The results indicate that the monolithic architecture is most suitable for small to medium-sized projects, offering initial simplicity but presenting limitations in scalability and flexibility. The modular monolith provides a balance between simplicity and organization, promoting internal modularity and facilitating a potential evolution towards microservices. Meanwhile, the microservices architecture is more appropriate for large-scale systems and distributed teams, delivering greater resilience and technological freedom, albeit at the cost of higher complexity and specialized infrastructure requirements. As a practical contribution, this study presents a comparative table and a decision support guide, aiming to assist professionals and students in selecting the most appropriate architecture for different contexts.

Keywords: *software architecture; monolithic; modular monolith; microservices; comparative analysis; software engineering.*

1.Introdução

O desenvolvimento de sistemas de *software* tem enfrentado desafios cada vez mais significativos em função da crescente complexidade dos requisitos de negócio e da necessidade de soluções escaláveis e sustentáveis. A definição da arquitetura de *software* constitui uma das decisões mais críticas nesse processo, uma vez que influencia diretamente atributos como desempenho, manutenibilidade, segurança e custo operacional. Nesse cenário, compreender as implicações de diferentes estilos arquiteturais torna-se essencial para orientar escolhas mais conscientes e estratégicas.

Durante décadas, a arquitetura monolítica foi o padrão predominante, oferecendo simplicidade no desenvolvimento e implantação. Contudo, à medida que aplicações crescem, surgem dificuldades relacionadas à escalabilidade granular, à evolução tecnológica e à flexibilidade de manutenção. Em resposta a essas limitações, emergiram alternativas como o monólito modular, que busca manter a coesão de um sistema único, mas com maior organização interna, e a arquitetura de microsserviços, caracterizada pela decomposição em serviços independentes, que ampliam a escalabilidade e a resiliência.

Este artigo tem como objetivo comparar sistematicamente essas três arquiteturas, destacando suas características, vantagens, desvantagens e contextos de aplicação. A metodologia adotada consiste em uma pesquisa bibliográfica comparativa, baseada em revisão sistemática da literatura em bases acadêmicas e técnicas de referência. A análise foi conduzida por meio de uma matriz de parâmetros, abrangendo aspectos técnicos, operacionais e organizacionais.

A principal contribuição deste estudo é oferecer um panorama estruturado e crítico que auxilie profissionais e estudantes na seleção de arquiteturas adequadas a diferentes contextos de desenvolvimento, ao mesmo tempo em que fomenta a reflexão sobre os trade-offs envolvidos em cada decisão arquitetural. Para tal, este trabalho está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico

sobre os estilos arquiteturais; a Seção 3 descreve a metodologia de pesquisa; a Seção 4 justifica a abordagem adotada; a Seção 5 expõe os resultados comparativos; e, por fim, são apresentadas as conclusões, limitações e sugestões de trabalhos futuros.

2. Referencial Teórico

2.1 Arquitetura de *Software*

A complexidade crescente dos sistemas de *software* modernos exige abordagens estruturadas para seu desenvolvimento e manutenção. Nesse contexto, a arquitetura de *software* emerge como um pilar fundamental, oferecendo a base conceitual e estrutural sobre a qual sistemas robustos, escaláveis e sustentáveis são construídos. Compreender o que é arquitetura de *software*, sua importância e sua evolução histórica é crucial para qualquer profissional ou estudante da área, especialmente ao se analisar diferentes abordagens arquiteturais como a monolítica, a modular e a de microsserviços.

Definir arquitetura de *software* não é uma tarefa trivial, e diversas perspectivas coexistem na literatura e na prática. Uma definição abrangente e amplamente aceita é fornecida pela norma ISO/IEC/IEEE 42010:2011, que a descreve como os "conceitos ou propriedades fundamentais de um sistema em seu ambiente, incorporados em seus elementos, relacionamentos e nos princípios de seu projeto e evolução" (ISO/IEC/IEEE 42010:2011, 3.2). Essa definição enfatiza que a arquitetura não se limita a uma visão estática dos componentes, mas engloba também os princípios que guiam o desenvolvimento e a adaptação do sistema ao longo do tempo.

Bass, Clements e Kazman (2013, p.4), em sua obra *"Software Architecture in Practice"*, oferecem uma perspectiva complementar e prática, definindo arquitetura como "o conjunto de estruturas necessárias para raciocinar sobre o sistema, que compreendem elementos de *software*, as relações entre eles e as propriedades de ambos". Eles destacam que um sistema possui múltiplas estruturas (como estruturas de módulos, estruturas de componentes e conectores em tempo de execução, e estruturas de alocação) e que nenhuma delas, isoladamente, representa a totalidade da arquitetura. O que torna uma estrutura "arquitetural" é sua capacidade de suportar o raciocínio sobre atributos importantes do sistema, como desempenho, modificabilidade ou segurança, relevantes para os *stakeholders*.

É crucial distinguir a arquitetura em si da sua representação, a descrição da arquitetura (*AD - Architecture Description*), que é o artefato concreto (documento, modelo) usado para expressar a arquitetura (ISO/IEC/IEEE 42010:2011). A arquitetura é uma abstração que foca nos aspectos públicos e essenciais dos elementos e suas interações, omitindo detalhes internos de implementação que não impactam o raciocínio sobre o sistema como um todo. Todo sistema de *software* possui uma arquitetura, mesmo que não documentada ou conscientemente projetada, mas nem toda arquitetura é adequada para os objetivos do sistema (Bass, Clements e Kazman, 2013).

2.1.1 A Importância da Arquitetura de *Software*

A arquitetura de *software* não é um mero exercício acadêmico ou um diagrama superficial; ela desempenha um papel crítico no sucesso ou fracasso de um projeto de *software*. Sua importância manifesta-se de diversas formas ao longo do ciclo de vida do sistema.

Primeiramente, a arquitetura é o principal veículo para habilitar ou inibir os atributos de qualidade de um sistema. Atributos como desempenho, segurança, disponibilidade, modificabilidade e usabilidade não surgem espontaneamente; eles são diretamente influenciados pelas decisões arquiteturais tomadas no início do projeto. Uma arquitetura bem definida permite raciocinar sobre como esses atributos serão alcançados e prever o comportamento do sistema em relação a eles (Bass, Clements e Kazman, 2013).

Além disso, a arquitetura facilita o gerenciamento da mudança. Ao definir as estruturas e as interfaces entre os componentes, ela permite analisar o impacto de alterações propostas e localizar os pontos de modificação, tornando a evolução do sistema mais controlada e menos custosa. Ela também serve como um meio de comunicação essencial entre os diversos *stakeholders* (desenvolvedores, gerentes, clientes), fornecendo uma visão comum e abstrata do sistema que transcende o código (Bass, Clements e Kazman, 2013; ISO/IEC/IEEE 42010:2011).

As decisões arquiteturais são as primeiras decisões de design tomadas e as mais difíceis de reverter. Elas carregam consigo as restrições fundamentais que guiarão a implementação subsequente. Uma arquitetura bem definida pode influenciar positivamente a estrutura organizacional da equipe de desenvolvimento, permitir a prototipagem evolutiva, melhorar a precisão das estimativas de custo e cronograma, e fornecer uma base sólida para o treinamento de novos membros da equipe. Ela também possibilita a incorporação de componentes desenvolvidos independentemente e restringe o vocabulário de alternativas de design, focando os esforços da equipe (Bass, Clements e Kazman, 2013).

Em suma, investir tempo e esforço na definição e documentação da arquitetura de *software* é fundamental para mitigar riscos, gerenciar a complexidade e garantir que o sistema final atenda não apenas aos requisitos funcionais, mas também aos atributos de qualidade essenciais e aos objetivos de negócio da organização.

2.1.2 A Evolução Histórica e Fundamentos

A disciplina de arquitetura de *software*, embora com raízes em práticas de design anteriores, consolidou-se como um campo de estudo e prática distinto principalmente a partir da década de 1990. Antes disso, o foco da engenharia de *software* estava mais voltado para o design detalhado e a implementação, muitas vezes obscurecendo as decisões estruturais de mais alto nível que impactam fundamentalmente o sistema (Perry e Wolf, 1992).

Perry e Wolf (1992, p. 40), em seu influente artigo *"Foundations for the Study of Software Architecture"*, argumentaram que a década de 1990 seria a "década da arquitetura de *software*", destacando a necessidade de codificação, abstração, padrões e estilos arquiteturais, em contraste com o foco anterior em design e implementação. Eles propuseram um modelo seminal para a arquitetura de *software* composto por três componentes essenciais: elementos (de processamento, dados ou conexão), forma (as propriedades e relações entre os elementos, ou seja, as restrições) e *rationale* (a justificativa subjacente para as escolhas arquiteturais, derivada dos requisitos e restrições do sistema) (Perry e Wolf, 1992).

A motivação para elevar a arquitetura a uma disciplina própria surgiu da necessidade de lidar com problemas recorrentes no desenvolvimento de *software* em larga escala, como a evolução e a customização. Sistemas tendem a se tornar mais "frágeis" com o tempo devido à "erosão arquitetural" (violações da arquitetura original) e ao "desvio arquitetural" (insensibilidade à arquitetura, levando à incoerência). A falta de estilos arquiteturais padronizados e reutilizáveis também contribuía para a recriação constante de soluções, indicando uma necessidade de codificação e *templates* (Perry e Wolf, 1992, p. 43).

Assim, a arquitetura de *software* emergiu como o nível de abstração onde as decisões críticas sobre a estrutura geral do sistema, suas propriedades de qualidade e sua evolução futura são tomadas. Ela fornece o *framework* para satisfazer os requisitos, a base técnica para o design e a implementação, e um fundamento essencial para a análise de dependências, consistência e reuso. A distinção entre arquitetura e design detalhado reside no nível de abstração e no foco: a arquitetura seleciona os elementos essenciais, suas interações e as restrições globais, enquanto o design detalha a modularização, interfaces, algoritmos e tipos de dados dentro desse *framework* arquitetural (Perry e Wolf, 1992).

Compreender essa evolução e os fundamentos conceituais é vital para apreciar o papel da arquitetura na engenharia de *software* moderna e para analisar criticamente as diferentes abordagens arquiteturais que serão exploradas neste trabalho.

2.2 Arquitetura Monolítica

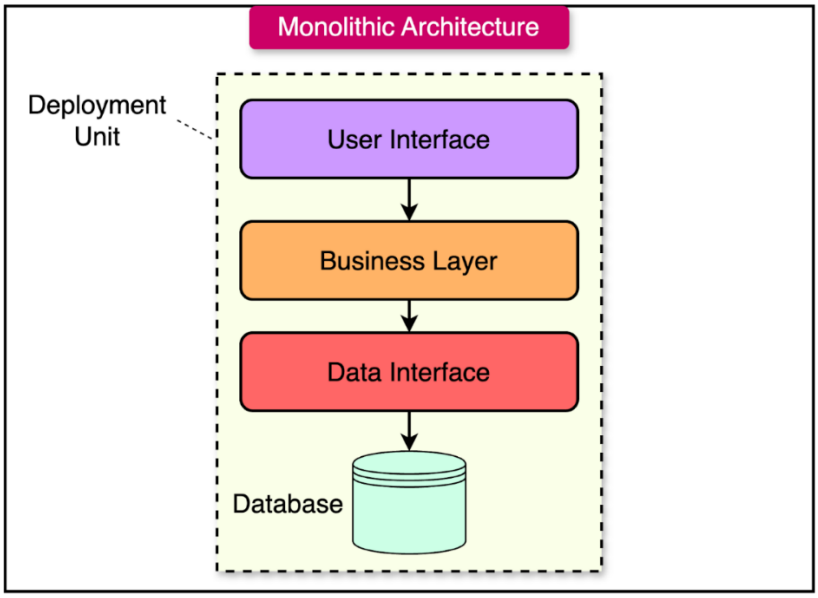
Dando continuidade à exploração dos fundamentos da arquitetura de *software*, adentramos agora em um dos estilos arquiteturais mais tradicionais e, por muito tempo, predominante: a arquitetura monolítica. Esta abordagem, embora frequentemente contrastada com estilos mais modernos, possui características, vantagens e desvantagens que merecem uma análise aprofundada, pois estabelece a base para a compreensão da evolução arquitetural e a análise comparativa proposta neste trabalho.

2.2.1 Definição e Características

A arquitetura monolítica, derivada da palavra grega *monos* (único) e *lithos* (pedra), refere-se a um sistema de *software* construído como uma única unidade coesa e indivisível (Kalske, 2017; Elgheriani e Ahmed, 2022). Neste modelo, todas as funcionalidades e componentes da aplicação – como a interface do usuário (UI), a lógica de negócios (regras de negócio) e a camada de acesso a dados (DAL) – são desenvolvidos, empacotados e implantados juntos como um único artefato executável ou biblioteca (Tapia et al., 2020; Amaral, 2018). Tipicamente, uma aplicação monolítica opera como um único processo no servidor ou, em algumas variações conhecidas como "monolitos distribuídos", como um conjunto de processos fortemente acoplados que, apesar de distribuídos, ainda precisam ser implantados e gerenciados como uma unidade coesa (Elgheriani e Ahmed, 2022; Bass, Clements e Kazman, 2013).

A figura 1 demonstra a estrutura básica de uma aplicação usando arquitetura monolítica.

Figura 1 - Exemplo de uma aplicação usando arquitetura monolítica.



Fonte: XU, 2025

Internamente, um monolito pode (e idealmente deveria) possuir uma estrutura modular bem definida, frequentemente organizada em camadas lógicas (por exemplo, apresentação, negócios, persistência). No entanto, a característica definidora é que essas camadas e módulos lógicos residem na mesma base de código e são compilados e implantados como um todo único. A comunicação entre esses componentes internos geralmente ocorre por meio de chamadas de método diretas dentro do mesmo espaço de processo, o que contribui para um forte acoplamento em tempo de compilação e execução (Newman, 2015; Kalske, 2017).

2.2.2 Contexto Histórico e Vantagens

Historicamente, a abordagem monolítica foi o padrão de fato para o desenvolvimento de *software* por décadas, sendo a forma natural de construir aplicações antes da ampla adoção de técnicas avançadas de componentização, virtualização e computação distribuída (Kalske, 2017; Tapia et al., 2020). Sua longevidade e prevalência inicial derivam de vantagens significativas, especialmente em contextos específicos:

- **Simplicidade Inicial**: Para aplicações de escopo pequeno a médio, ou em fases iniciais de desenvolvimento, o modelo monolítico oferece uma simplicidade conceitual e prática. Ter uma única base de código facilita o raciocínio sobre o fluxo de execução, a configuração do ambiente de desenvolvimento, a depuração (*debugging*) através de ferramentas padrão e a realização de testes *end-to-end*, já que toda a lógica reside em um único contexto de processo (Elgheriani e Ahmed, 2022; Kalske, 2017; Bass, Clements e Kazman, 2013).
- **Desempenho**: Em muitos cenários, a comunicação intra-processo via chamadas de método diretas em um monolito podem ser significativamente mais rápida do que a comunicação inter-processo ou via rede, comum em arquiteturas distribuídas. Isso pode resultar em menor latência para certas operações, especialmente se não houver gargalos internos (Elgheriani e Ahmed, 2022; Al-Debagy e Martinek apud Tapia et al., 2020).
- **Gerenciamento de Transações**: Operações que exigem atomicidade e consistência transacional envolvendo múltiplos componentes são inerentemente mais simples de gerenciar dentro de um único processo e, frequentemente, com um único banco de dados relacional, utilizando os mecanismos de transação *ACID* padrão (Newman, 2015).
- **Implantação Simplificada (Inicialmente)**: Implantar uma única aplicação pode ser mais direto do que orquestrar a implantação de múltiplos serviços distribuídos, especialmente quando a infraestrutura de automação ainda não está madura (Kalske, 2017).

2.2.3 Desvantagens e Desafios

Entretanto, à medida que as aplicações crescem em complexidade e escala, as desvantagens da arquitetura monolítica tornam-se mais evidentes e problemáticas. A natureza fortemente acoplada dos componentes significa que uma alteração em uma pequena parte do sistema exige a recompilação e reimplantação de toda a aplicação, aumentando o risco de regressões e tornando o ciclo de desenvolvimento mais lento e custoso (Kalske, 2017; Tapia et al., 2020). A escalabilidade também se torna um desafio significativo; se apenas uma funcionalidade específica demanda mais recursos, é necessário escalar toda a aplicação, levando a um uso ineficiente de recursos (Amaral, 2018; Elgheriani e Ahmed, 2022).

A adoção de novas tecnologias ou a atualização de *frameworks* torna-se complexa e arriscada em um monolito, pois toda a aplicação geralmente depende de uma única pilha tecnológica. A baixa tolerância a falhas é outra desvantagem crítica: um erro em um único componente pode comprometer a disponibilidade de toda a aplicação. Além disso, a deterioração da arquitetura (erosão e desvio, como mencionado por Perry e Wolf, 1992) é um risco comum em monolitos grandes e

antigos, dificultando a manutenção, a evolução e a integração de novos desenvolvedores na equipe (Elgheriani e Ahmed, 2022; Kalske, 2017).

É crucial entender que a arquitetura monolítica não é inerentemente falha. Muitos sistemas de sucesso foram e continuam sendo construídos como monolitos. Os desafios descritos acima tornam-se mais pronunciados com o aumento da escala e da complexidade. Um "monolito bem-estruturado", com forte modularidade interna, baixo acoplamento entre módulos e interfaces bem definidas, pode mitigar alguns desses problemas e ser uma escolha arquitetural válida para muitos cenários (Newman, 2015).

No entanto, os desafios intrínsecos relacionados à implantação, escalabilidade granular, adoção de tecnologia e resiliência, especialmente em face das demandas modernas por entrega contínua, agilidade e sistemas distribuídos em nuvem, impulsionaram fortemente a busca por alternativas. A análise detalhada do monolito, com suas forças e fraquezas, estabelece, portanto, a linha de base essencial para compreendermos as motivações, os *trade-offs* e os benefícios buscados por estilos arquiteturais como o modular e, principalmente, o de microsserviços, que serão explorados a seguir.

2.3 Arquitetura Modular

Em contraste com a simplicidade no desenvolvimento e implantação inicial, mas desafios de escalabilidade e flexibilidade dos monólitos tradicionais, e a alta escalabilidade e autonomia da equipe, mas complexidade operacional e os desafios de manutenção dos microsserviços, a arquitetura de monólito modular (*Modular Monolith*) surge como uma alternativa pragmática para esse impasse (Skalický, 2023; Su e Li, 2024). Neste cenário de *trade-offs*, este estilo surge como alternativa pragmática e atraente, buscando equilibrar a simplicidade operacional de um monólito com os benefícios organizacionais da modularidade, estruturando uma única unidade implantável em módulos internos distintos e bem encapsulados (Barde, 2023; Tsechelidis et al., 2023).

Como dito, este estilo arquitetônico estrutura uma única unidade implantável (um monólito) em módulos distintos e bem encapsulados internamente. Tal abordagem promove uma melhor organização do código, facilita a manutenibilidade e oferece um caminho potencial para a futura extração de microsserviços, caso a necessidade surja, sem incorrer prematuramente nos custos e complexidades da distribuição (Tsechelidis et al., 2023; Newman, 2020).

2.3.1 Definição e Conceitos

Um monólito modular é, em sua essência, uma aplicação monolítica projetada com uma forte ênfase na modularidade interna. Diferencia-se fundamentalmente de um monólito tradicional não estruturado, frequentemente descrito pejorativamente como "grande bola de lama" (*big ball of mud*), onde os componentes são fortemente acoplados e os limites entre as funcionalidades são pouco claros (Newman, 2020).

Nesta abordagem, a base de código é organizada em módulos distintos e logicamente independentes, onde cada módulo representa uma capacidade de negócio específica ou um domínio delimitado (*Bounded Context*), conforme os

princípios do *Domain-Driven Design (DDD)* (Tsechelidis et al., 2023; Newman, 2020). Su e Li (2024) o definem como um padrão de arquitetura que organiza sistemas em módulos fracamente acoplados, cada um delineando limites bem definidos e com dependências explícitas em relação a outros módulos.

2.3.2 Princípios Fundamentais: Coesão e Acoplamento

Dois princípios de design de *software* são cruciais para o sucesso de um monólito modular: alta coesão e baixo acoplamento. A alta coesão significa que os elementos dentro de um módulo devem estar fortemente relacionados e focados em uma única responsabilidade ou domínio de negócio bem definido (Skalický, 2023; Newman, 2020). Isso torna o módulo mais compreensível e fácil de manter.

O baixo acoplamento, por sua vez, refere-se à minimização das dependências entre os módulos. As interações entre eles devem ocorrer através de interfaces públicas bem definidas e estáveis, ocultando os detalhes internos de implementação de cada módulo (*information hiding*) (Su e Li, 2024; Newman, 2020). Reduzir o acoplamento diminui o impacto das mudanças: uma alteração interna em um módulo tem menor probabilidade de afetar outros módulos, facilitando a evolução independente e a refatoração.

2.3.3 Comunicação entre Módulos

Uma característica distintiva do monólito modular em comparação com os microsserviços é a forma como os módulos se comunicam. Diferentemente da arquitetura em microsserviços, onde os módulos são unidades de implantação separadas e independentes, em modular, todos os módulos residem na mesma unidade de implantação e executam no mesmo processo, a comunicação entre eles é tipicamente intraprocesso (Barde, 2023; Skalický, 2023).

Essa comunicação geralmente ocorre através de chamadas diretas de método ou função entre as interfaces públicas dos módulos, ou através de mecanismos de eventos internos (*in-memory events*). Isso evita a sobrecarga (*overhead*) e a latência associadas às chamadas de rede, comuns em arquiteturas distribuídas como microsserviços (Newman, 2020). A comunicação intraprocesso simplifica o fluxo de controle e o gerenciamento de transações dentro do sistema, embora a necessidade de manter transações atômicas entre módulos ainda possa exigir atenção.

2.3.4 Características Principais

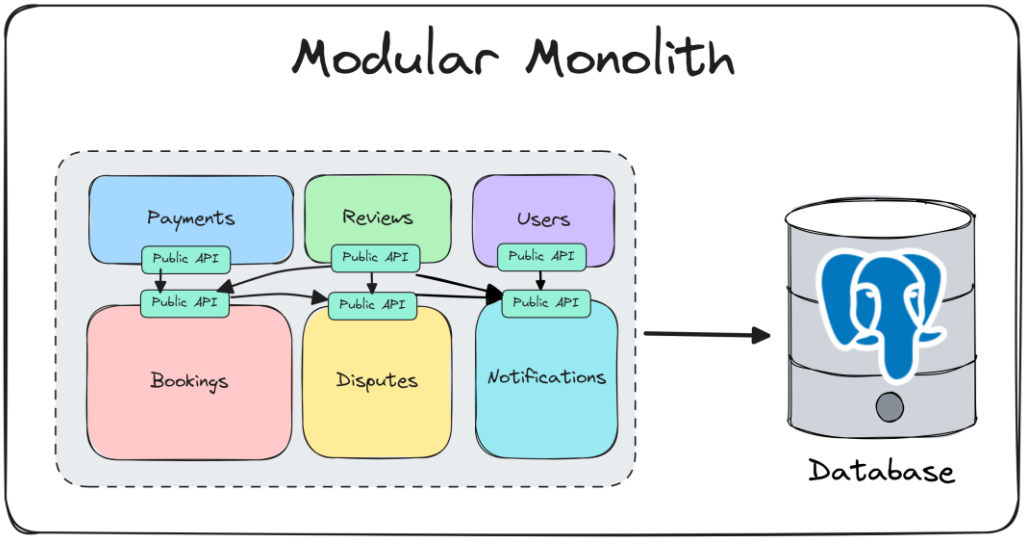
Além dos princípios de coesão e acoplamento e da comunicação intraprocesso, outras características definem a arquitetura de monólito modular:

- **Modularidade Forte:** O sistema é rigorosamente dividido em módulos bem definidos, com limites claros e responsabilidades distintas, frequentemente alinhados aos domínios de negócio (Tsechelidis et al., 2023; Newman, 2020). A estrutura de diretórios, *namespaces* ou pacotes no código-fonte reflete essa modularidade.

- **Dependências Explícitas:** As interações entre módulos são gerenciadas exclusivamente através de interfaces explícitas, minimizando dependências ocultas e reforçando o baixo acoplamento (Su e Li, 2024).
- **Unidade Única de Implantação:** Apesar da modularidade interna, toda a aplicação é construída, testada e implantada como uma única unidade (um único artefato ou processo) (Skalický, 2023; Newman, 2020). Isso simplifica significativamente o pipeline de *CI/CD* e as operações de implantação em comparação com microsserviços.
- **Infraestrutura Compartilhada (Frequentemente):** É comum que os módulos dentro de um monólito modular compartilhem a mesma instância de banco de dados e outros recursos de infraestrutura (Barde, 2023). Embora seja possível, e por vezes desejável, separar logicamente os dados por módulo (por exemplo, usando esquemas ou prefixos de tabela distintos), o compartilhamento físico é uma característica comum que simplifica a gestão, mas pode levar a acoplamento indesejado se não for bem gerenciado (Newman, 2020).

A Figura 2 ilustra a estrutura típica de um monólito modular, destacando os módulos internos com suas APIs públicas e o acesso a um banco de dados compartilhado.

Figura 2 - Exemplo de estrutura de um Monólito Modular.



Fonte: Jovanović (2023)

2.3.5 Vantagens da Abordagem

A arquitetura de monólito modular oferece um conjunto equilibrado de vantagens:

- **Simplicidade Operacional**: Mantém a simplicidade de implantação, monitoramento e gerenciamento de um monólito tradicional, com uma única base de código e um único artefato a ser implantado (Skalický, 2023; Newman, 2020).

- **Organização e Manutenibilidade Aprimoradas:** A divisão em módulos coesos e fracamente acoplados facilita a compreensão, a modificação e a manutenção do código ao longo do tempo, permitindo que equipes trabalhem em paralelo em diferentes módulos com menor interferência (Su e Li, 2024; Barde, 2023).
- **Facilidade de Refatoração e Teste:** Alterações dentro de um módulo têm menor probabilidade de impactar outros módulos. Os módulos podem ser testados de forma mais isolada (unitária e de integração), embora testes de ponta a ponta ainda sejam necessários (Skalický, 2023; Newman, 2020).
- **Performance da Comunicação:** A comunicação intraprocesso entre módulos é significativamente mais rápida e confiável do que as chamadas de rede em microsserviços, evitando latência e complexidade de comunicação distribuída (Barde, 2023; Skalický, 2023).
- **Caminho de Migração Flexível:** Oferece um ponto de partida menos complexo que microsserviços. Módulos bem definidos podem ser extraídos e transformados em microsserviços independentes no futuro, caso a necessidade de escalabilidade independente, implantação separada ou tecnologia específica surja (Su e Li, 2024; Barde, 2023; Newman, 2020).

2.3.6 Desafios e Considerações

Apesar de suas vantagens, a abordagem do monólito modular não está isenta de desafios e requer disciplina:

- **Disciplina de Modularidade:** Manter limites claros entre os *bounded contexts* e baixo acoplamento entre os módulos exige disciplina contínua da equipe de desenvolvimento. É fácil introduzir acoplamentos acidentais que degradam a arquitetura ao longo do tempo se as regras de comunicação entre módulos não forem rigorosamente aplicadas e fiscalizadas (Skalický, 2023; Newman, 2020).
- **Acoplamento de Banco de Dados:** O compartilhamento de um único banco de dados físico, mesmo com separação lógica, pode criar dependências indesejadas nos dados e dificultar a evolução independente dos módulos ou a futura extração para microsserviços (Barde, 2023; Newman, 2020). Estratégias cuidadosas de design de banco de dados são necessárias.
  Escalabilidade Global: Como a aplicação é uma unidade única de implantação, ela deve ser escalada como um todo (escalabilidade horizontal replicando a instância inteira), mesmo que apenas um módulo específico esteja sob alta carga (Skalický, 2023). Isso pode levar a um uso menos eficiente de recursos em comparação com a escalabilidade seletiva dos microsserviços.
- **Restrição Tecnológica:** Todos os módulos geralmente compartilham a mesma *stack* tecnológica (linguagem, *frameworks*) dentro do monólito. Isso limita a possibilidade de usar a ferramenta ou linguagem mais adequada para cada módulo específico, uma vantagem frequentemente citada dos microsserviços (Barde, 2023; Newman, 2020).
- **Consistência e Transações:** Garantir a consistência dos dados e gerenciar transações que abranjam múltiplos módulos dentro do mesmo processo pode ser mais simples do que em sistemas distribuídos, mas ainda requer mecanismos cuidadosos, especialmente se não houver suporte a transações distribuídas leves dentro do framework utilizado (Skalický, 2023).

2.4 Arquitetura de Microsserviços

O conceito de microsserviços surge como uma evolução das arquiteturas orientadas a serviços (*SOA*), impulsionado pela necessidade de criar sistemas mais ágeis, escaláveis e resilientes frente às demandas modernas de negócio. De acordo com Newman (2015), o termo começou a ganhar tração em meados da década de 2010, quando organizações como Netflix e Amazon passaram a relatar benefícios significativos ao dividir aplicações monolíticas em serviços menores e independentes. Análises comparativas recentes (Skalický, 2023) reforçam que essa abordagem permite alinhar a estrutura do *software* aos limites do negócio, promovendo uma maior flexibilidade e facilitando a evolução contínua dos sistemas.

A crescente demanda por sistemas distribuídos e a necessidade de respostas ágeis às mudanças de negócio impulsionaram a adoção de arquiteturas baseadas em microsserviços. Namiot e Sneps-Sneppe (2014) apontam que essa mudança de paradigma reflete uma tendência geral de descentralização e de adoção de práticas de desenvolvimento contínuo, viabilizadas pelos avanços em virtualização, containers e práticas *DevOps*. Essa abordagem propõe a decomposição de uma aplicação em um conjunto de serviços pequenos, autônomos e especializados, os quais interagem por meio de interfaces bem definidas, diferentemente das soluções monolíticas tradicionais.

2.4.1 Definição e Conceitos

Os microsserviços podem ser definidos como componentes independentes que executam funções específicas e que, ao interagir, compõem a aplicação global. Newman (2015) aponta que essa arquitetura se fundamenta na ideia de que o desenvolvimento e a implantação isolada de serviços facilitam a evolução e a manutenção do sistema. De acordo com Namiot e Sneps-Sneppe (2014), cada serviço opera em seu próprio processo, possibilitando o uso de diferentes *stacks* tecnológicas e promovendo o isolamento funcional e operacional.

2.4.2 Características Principais

Os microsserviços apresentam características essenciais que os diferenciam de outras abordagens arquiteturais. Uma delas é a independência e o desacoplamento, permitindo que cada serviço seja desenvolvido, testado e implantado de forma autônoma, reduzindo a complexidade e evitando impactos diretos no sistema global (Newman, 2015; Namiot e Sneps-Sneppe, 2014).

Outro aspecto importante é a escalabilidade seletiva, que possibilita escalar apenas os serviços que apresentam maior demanda, otimizando o uso de recursos computacionais e melhorando o desempenho geral do sistema (Christudás, 2019). Além disso, a diversidade tecnológica é um fator relevante, pois permite que cada componente utilize o conjunto de tecnologia mais adequadas à sua função, sem a imposição de uma única *stack* para toda a aplicação (Newman, 2015; Skalický, 2023).

Por fim, a comunicação via APIs padronizadas garante interoperabilidade e integração facilitada entre os serviços, utilizando protocolos como REST ou gRPC,

que são amplamente utilizados para a comunicação eficiente entre serviços. (Fowler, 2023).

2.4.3 Vantagens da Abordagem

A adoção da arquitetura de microsserviços proporciona diversas vantagens. A agilidade no desenvolvimento é um dos principais benefícios, pois a independência dos serviços permite que diferentes equipes trabalhem simultaneamente, acelerando a entrega de novas funcionalidades e favorecendo práticas de integração contínua (Newman, 2015).

Outro benefício significativo é a resiliência e disponibilidade, já que a falha de um único serviço não compromete o funcionamento do sistema como um todo, tornando-o mais robusto e tolerante a falhas (Namiot e Sneps-Sneppe, 2014). Além disso, a facilidade de manutenção torna possível realizar atualizações e correções de forma localizada, sem a necessidade de reimplantar toda a aplicação, contribuindo para a evolução contínua do *software* (Christudás, 2019; Skalický, 2023).

2.4.4 Desafios e Considerações Técnicas

Apesar das vantagens, a implementação de microsserviços impõe desafios significativos. Um dos principais é a complexidade na comunicação, uma vez que a dependência de chamadas remotas pode aumentar a latência e exigir a implementação de mecanismos de resiliência, como os padrões de *Circuit Breaker*, para evitar falhas sistêmicas (Namiot e Sneps-Sneppe, 2014).

Além disso, o gerenciamento e monitoramento distribuído se tornam mais complexos, exigindo ferramentas sofisticadas para rastreamento de interações, monitoramento e *logging*, para garantir a identificação precoce de problemas e a estabilidade do sistema (Fowler, 2023).

O gerenciamento da infraestrutura em sistemas baseados em microsserviços é também um dos grandes desafios enfrentados devido à necessidade de orquestrar múltiplas instâncias distribuídas. Ferramentas como Kubernetes e Docker Swarm são essenciais para automação de *deploys* e escalonamento, mas exigem conhecimento avançado. Além disso, a comunicação entre serviços requer *service discovery* e balanceamento de carga, enquanto o monitoramento distribuído demanda soluções como Prometheus e Jaeger. Essa complexidade aumenta os custos operacionais e a curva de aprendizado da equipe (Skalický, 2023).

A segurança e governança também representam desafios, pois cada microsserviço deve implementar seus próprios mecanismos de autenticação e autorização. Isso aumenta a superfície de ataque do sistema e demanda uma estratégia robusta para garantir a proteção dos dados trocados entre os serviços (Christudás, 2019).

Por fim, a definição de limites e escalabilidade horizontal deve ser cuidadosamente planejada. A escolha de uma estratégia eficiente de particionamento, como segmentação por caso de uso ou por recursos, é essencial para evitar dependências excessivas e garantir a escalabilidade adequada do sistema (Newman, 2015; Skalický, 2023).

2.4.5 Padrões de Comunicação

A comunicação entre microsserviços pode ser realizada por diferentes padrões. O modelo de comunicação direta (*Point-to-Point*) utiliza chamadas síncronas via REST ou gRPC, que consiste em chamadas que esperam o retorno de uma resposta, seja de sucesso ou de falha, para continuar seu fluxo de execução, sendo uma abordagem flexível, porém suscetível a latências elevadas em cenários de alta demanda (Skalický, 2023).

Uma alternativa amplamente adotada são os modelos assíncronos, como o *Message Bus* ou o *Publish-Subscribe*, que permitem comunicação desacoplada entre serviços, favorecendo a escalabilidade e a tolerância a falhas. Esse modelo utiliza um broker de mensageria responsável por receber as mensagens e enviá-las para os serviços corretos, sendo um modelo amplamente utilizado por sua resiliência na garantia de envio para seu destino. Vantagens estas, que facilitam a integração de novos serviços sem a necessidade de grandes mudanças na arquitetura existente (Fowler, 2023).

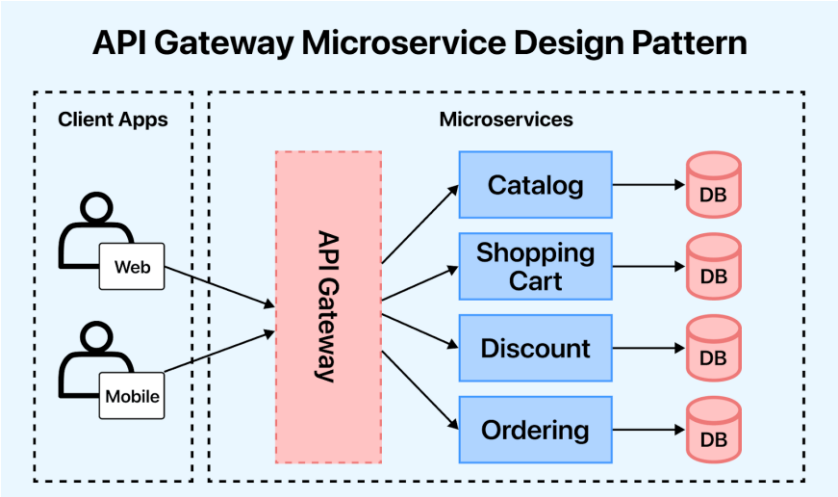
2.4.6 Comunicação de Clients via API Gateway

Uma abordagem consolidada na arquitetura de microsserviços é a utilização de um *API Gateway* como ponto único de entrada para os clientes. Esse padrão simplifica a comunicação, pois os clientes (como aplicativos web e mobile) não precisam conhecer diretamente cada microsserviço, mas apenas o Gateway, que encaminha as requisições para os serviços apropriados.

Além de simplificar o acesso, essa abordagem centraliza a aplicação de políticas de segurança, permitindo que autenticação e autorização sejam gerenciadas antes que a requisição chegue aos microsserviços (Newman, 2015). O Gateway também pode agregar respostas de múltiplos serviços e aplicar balanceamento de carga e *circuit breakers*, garantindo a escalabilidade e a resiliência do sistema (Namiot, e Sneps-Sneppe, 2014).

A Figura 3 ilustra um exemplo de padrão de design com *API Gateway* na arquitetura de microsserviços, onde clientes (web e mobile) acessam os serviços de *Catalog, Shopping Cart, Discount e Ordering*, cada um com seu próprio banco de dados.

Figura 3 – Exemplo de comunicação via API Gateway em uma arquitetura de microsserviços.



Fonte: Nair, H. 2025

3. Metodologias

3.1 Introdução a Metodologia

O presente artigo consiste em uma pesquisa bibliográfica comparativa sobre arquiteturas de *software*, com foco específico nas arquiteturas Monolítica, Modular e de Microsserviços. A escolha deste tema justifica-se pela crescente complexidade dos sistemas de *software* contemporâneos e pela necessidade de compreender as vantagens e desvantagens de cada abordagem arquitetural, permitindo assim decisões mais fundamentadas no desenvolvimento de novos sistemas.

Esta pesquisa não contempla o desenvolvimento de um sistema prático, concentrando se exclusivamente na análise comparativa das arquiteturas mencionadas através de uma revisão sistemática da literatura. O objetivo principal é estabelecer um conjunto de parâmetros de comparação que permitam avaliar de forma objetiva e abrangente as diferentes arquiteturas, considerando aspectos técnicos, organizacionais e de negócio.

Esta seção detalha os procedimentos metodológicos empregados, os parâmetros de comparação estabelecidos e as ferramentas utilizadas para organização e análise dos dados coletados.

3.2 Bases de Dados e Fontes Bibliográficas

A pesquisa bibliográfica foi realizada utilizando as seguintes bases de dados e fontes:

- **IEEE Xplore Digital Library:** Para artigos científicos e conferências na área de engenharia de *software*.
- **Google Scholar:** Para ampliar a cobertura e identificar citações relevantes.

- **Blogs e publicações técnicas:** Martin Fowler, ThoughtWorks, Microsoft Architecture Center e outras fontes reconhecidas na indústria.

Os critérios de seleção de material incluíram: - Relevância para o tema de arquiteturas de *software* - Publicações dos últimos 10 anos (com exceção de obras seminais) Citações por outros trabalhos (indicador de impacto) - Diversidade de perspectivas (acadêmica e industrial) - Cobertura dos parâmetros de comparação estabelecidos.

3.3 Procedimentos Metodológicos

3.3.1 Etapas da Pesquisa

A pesquisa foi conduzida seguindo as seguintes etapas:

1. **Definição do escopo:** Delimitação precisa do objeto de estudo, incluindo as arquiteturas a serem comparadas e os parâmetros de avaliação.
2. **Levantamento bibliográfico inicial:** Identificação de obras seminais e revisões de literatura existentes sobre o tema.
3. **Elaboração dos parâmetros de comparação:** Definição e refinamento dos critérios para análise comparativa.
4. **Pesquisa bibliográfica sistemática:** Busca estruturada nas bases de dados utilizando palavras-chave como "*software architecture comparison*", "*monolithic vs microservices*", "*modular architecture*", entre outras.
5. **Seleção e classificação do material:** Aplicação dos critérios de inclusão e exclusão, seguida de categorização do material por tipo de arquitetura e parâmetros abordados.
6. **Análise do conteúdo:** Leitura aprofundada e extração sistemática de informações relevantes para cada parâmetro de comparação.
   Validação da análise: Revisão crítica dos resultados, identificação de vieses e lacunas, e complementação da pesquisa quando necessário.
7. **Documentação dos resultados:** Redação da análise comparativa, incluindo fundamentação teórica, metodologia e resultados

3.3.2 Critérios de Inclusão e Exclusão

Para garantir a qualidade e relevância do material analisado, foram estabelecidos os seguintes critérios:

Critérios de Inclusão: - Publicações que abordam explicitamente pelo menos uma das arquiteturas estudadas - Estudos que apresentam comparações entre diferentes arquiteturas - Casos de estudo com implementações reais das arquiteturas - Publicações com análise de pelo menos um dos parâmetros de comparação definidos - Material publicado por autores ou organizações reconhecidos na área.

Critérios de Exclusão: - Publicações sem revisão por pares (exceto documentação técnica de referência) - Material com foco exclusivo em tecnologias específicas sem discussão arquitetural - Estudos sem fundamentação metodológica clara - Publicações com mais de 10 anos, exceto obras seminais ou de referência

fundamental - Material duplicado ou com conteúdo substancialmente similar a outros já incluídos

3.3.3 Método de Análise Comparativa

A análise comparativa foi realizada através de uma abordagem matricial, onde cada arquitetura foi avaliada em relação a cada um dos parâmetros estabelecidos. Para cada combinação arquitetura-parâmetro, foram identificados:

- Pontos fortes e vantagens
- Limitações e desafios
- Contextos de aplicação mais adequados
- *Trade-offs* e considerações importantes
- Tendências e evolução ao longo do tempo

Esta abordagem permitiu uma visualização clara das diferenças entre as arquiteturas e facilitou a identificação de padrões e relações entre os diferentes parâmetros. Além disso, foi elaborado um quadro comprativo, e um guia prático para auxiliar a decisão arquitetural de acordo com as especificações individuais de cada projeto.

4. Justificativa da Abordagem Metodológica

O presente artigo adota uma abordagem metodológica pautada exclusivamente na pesquisa bibliográfica para a análise comparativa das arquiteturas de *software* Monolítica, Modular e de Microsserviços. Esta escolha metodológica é intrínseca à natureza e aos objetivos deste estudo, que visa aprofundar o conhecimento teórico, contrastar paradigmas arquiteturais e consolidar informações existentes na literatura especializada, em vez de desenvolver uma implementação prática.

A complexidade inerente à concepção, desenvolvimento e validação de sistemas representativos para cada uma das três arquiteturas, mesmo que em escala reduzida, extrapolaria significativamente o escopo e o cronograma de um trabalho de graduação. Tal empreendimento desviaria o foco da análise conceitual e comparativa, que constitui o objetivo deste artigo. A pesquisa bibliográfica sistemática, conforme detalhado na Seção 3, permite uma abrangência e profundidade analítica que seriam inviáveis em um projeto de desenvolvimento prático, possibilitando a exploração de múltiplos casos de estudo e perspectivas teóricas sem as restrições de recursos e tempo associadas à implementação.

4.1 Método de Análise Comparativa e Construção do Guia de Seleção

A análise comparativa das arquiteturas foi conduzida por meio de uma abordagem matricial, onde cada arquitetura (Monolítica, Modular e Microsserviços) foi avaliada em relação a um conjunto predefinido de parâmetros. Estes parâmetros, derivados de uma revisão extensiva da literatura e de frameworks de avaliação de arquitetura de *software*, abrangem aspectos técnicos, operacionais, organizacionais e de negócio, essenciais para uma compreensão holística das implicações de cada escolha arquitetural. Exemplos de parâmetros incluem: escalabilidade,

manutenibilidade, complexidade de desenvolvimento, custo de implantação, resiliência, flexibilidade tecnológica e adequação a diferentes tamanhos de equipe.

Um dos resultados primordiais desta pesquisa será a elaboração de um guia prático, apresentado em formato de quadro na seção de Resultados. Este guia visa auxiliar profissionais e estudantes na tomada de decisão sobre a arquitetura mais adequada para um determinado projeto de *software*. O quadro consolidará os achados da análise comparativa, mapeando cada arquitetura aos parâmetros relevantes e indicando seus pontos fortes, limitações e contextos de aplicação ideais. A construção deste guia reflete o objetivo de traduzir a pesquisa teórica em um artefato de valor prático, oferecendo uma ferramenta de consulta rápida e objetiva para a seleção arquitetural. Este guia não pretende ser prescritivo, mas sim um facilitador para a discussão e a tomada de decisão informada, reconhecendo que a escolha arquitetural ideal é sempre contextual e depende de múltiplos fatores do projeto.

4.2 Contribuições e Relevância do Estudo

A relevância deste trabalho reside em sua capacidade de sintetizar e organizar o vasto conhecimento sobre arquiteturas de *software*, oferecendo uma visão clara e comparativa dos modelos Monolítico, Modular e Microsserviços. Ao focar na pesquisa bibliográfica, o estudo contribui para a literatura existente ao:

- **Consolidar o conhecimento:** Reunir e analisar criticamente as informações dispersas sobre as três arquiteturas, destacando suas características, vantagens e desvantagens sob diferentes perspectivas.
- **Fornecer um guia prático:** Desenvolver uma ferramenta de apoio à decisão (o guia de seleção) que simplifica o processo de escolha arquitetural, baseando-se em critérios objetivos e bem fundamentados.
- **Identificar trade-offs**: Aprofundar a compreensão dos compromissos (*trade-offs*) inerentes a cada arquitetura, auxiliando na ponderação de fatores críticos para o sucesso de um projeto.
- **Educar e informar:** Servir como um recurso valioso para estudantes e profissionais de engenharia de *software*, promovendo uma compreensão mais aprofundada das opções arquiteturais disponíveis e suas implicações.
- **Estimular a reflexão crítica:** Ao apresentar uma análise comparativa estruturada, o trabalho incentiva a reflexão sobre a adequação de cada arquitetura a diferentes contextos de negócio e requisitos técnicos, promovendo uma cultura de design arquitetural mais consciente e estratégica.

Esta abordagem metodológica, portanto, não apenas justifica a ausência de um desenvolvimento prático, mas também estabelece as bases para uma contribuição significativa ao campo da engenharia de *software*, fornecendo uma análise comparativa robusta e um guia prático para a seleção de arquiteturas.

5. Resultados

Esta seção apresenta os resultados da análise comparativa entre as arquiteturas Monolítica, Monólito Modular e Microsserviços. Os achados aqui expostos derivam da aplicação da matriz de avaliação descrita na Seção de Métodos e da revisão bibliográfica realizada ao longo do trabalho.

5.1 Síntese dos Achados

A análise evidencia diferenças robustas entre as três abordagens arquiteturais, ressaltando *trade-offs* relevantes para decisões de engenharia de *software*. A seguir, apresentam-se, de forma sintética, as características observadas para cada arquitetura.

- **Monolítico:** melhor adequação a projetos de escopo pequeno a médio, com equipes reduzidas e prazos curtos. Oferece simplicidade inicial e menor custo operacional, mas tende a apresentar dificuldades para escalabilidade granular e aumento de acoplamento à medida que o sistema cresce.
- **Monólito Modular:** representa um equilíbrio entre simplicidade operacional e organização interna. A modularização favorece manutenção, testes e evolução, possibilitando uma futura extração de componentes sem incorrer imediatamente na complexidade de uma arquitetura distribuída.
- **Microsserviços:** proporciona maior escalabilidade, tolerância a falhas e liberdade tecnológica, sendo indicada para sistemas de larga escala e organizações com equipes distribuídas. Entretanto, impõe custos operacionais e complexidade acrescida (orquestração, observabilidade, comunicação entre serviços).

5.2 Quadro de Resultados

O quadro de resultados deste estudo sintetiza a avaliação comparativa das arquiteturas monolítica, monólito modular e microsserviços, operacionalizando a matriz de parâmetros desenvolvida. Neste quadro, as avaliações de cada arquitetura frente aos diversos parâmetros são expressas em termos como ‘baixa’, ‘média’ e ‘alta’. É crucial ressaltar que a interpretação desses termos varia conforme o parâmetro analisado: por exemplo, uma avaliação ‘baixa’ para a escalabilidade de uma arquitetura pode indicar uma limitação, enquanto uma avaliação ‘baixa’ para a complexidade representa uma vantagem intrínseca. Essa nuance na interpretação, fundamentada na revisão teórica do trabalho, permite uma compreensão mais aprofundada de como cada arquitetura se comporta e quais são seus *trade-offs* inerentes em diferentes contextos de aplicação.

Parâmetro	Monolítico	Monólito Modular	Microsserviços
Escalabilidade	Baixa — escala global replicando toda a aplicação.	Média — modularidade facilita extrações futuras.	Alta — escalabilidade seletiva por serviço.

Manutenibilidade	Baixa/Média — tende a piorar sem disciplina.	Alta — alta coesão entre módulos.	Média/Alta — demanda governança e automação.
Complexidade de desenvolvimento	Baixa — inicial simples.	Média — exige disciplina de modularização.	Alta — contratos, testes e comunicação distribuída.
Custo operacional	Baixo — pipeline simples.	Médio — pipeline parecido com monolito, porém com regras de modularidade.	Alto — orquestração, observabilidade, infraestrutura.
Resiliência	Baixa — falha em componente pode derrubar tudo.	Média — melhor isolamento lógico, mas único processo.	Alta — falhas isoladas por serviço (quando bem projetado).
Flexibilidade tecnológica	Baixa — mesma *stack* para toda a aplicação.	Média — geralmente *same stack*, mas organização facilita mudança interna.	Alta — cada serviço pode usar *stack* própria.
Adequação ao tamanho da equipe	Pequenas equipes (1–5): ideal.	Equipes médias (2–10): muito adequado.	Grandes equipes/organizações: ideal, com times por serviço.
Performance (latência/*throughput*)	Média — bom desempenho em contextos sem alta concorrência; latências internas baixas por ausência de rede.	Média/Alta — modularização ajuda otimizar caminhos críticos; ainda sem sobrecarga de rede.	Alta — pode alcançar alto *throughput*, porém sujeito a latências de rede e overhead de serialização/deserialização.
Testabilidade	Média — testes integrados simples, porém difícil isolar comportamentos específicos.	Alta — testes de unidade e integração por módulo são mais fáceis; permite testes *end-to-end* controlados.	Média — testes unitários por serviço são fáceis; testes de integração e *end-to-end* são mais complexos e custosos.
Observabilidade e monitoramento	Baixa/Média — logs e métricas centralizados, porém menos	Média/Alta — facilita instrumentação por módulo	Alta — requer *stack* de observabilidade abrangente (*tracing*

	granularidade por componente.	dentro do mesmo processo.	distribuído, métricas e logs centralizados).
Segurança e conformidade	Média — superfície de ataque unificada; controles centralizados simples.	Média — possibilidade de políticas internas por módulo; ainda com superfície consolidada.	Alta — maior superfície distribuída, exige governança, autenticação/autorizações entre serviços e gestão de segredos.
Consistência de dados	Alta (forte) — transações ACID centralizadas simples de implementar.	Alta (quando bem projetado) — mantém consistência dentro do processo; extrações exigem cuidado.	Variável — frequentemente adota-se eventual *consistency*; coordenação transacional distribuída é complexa.
Isolamento de falhas detalhado	Baixo — falhas afetam todo o *runtime*; isolamento por processo não existe.	Média — falhas podem ser limitadas logicamente, mas o processo único ainda é ponto único de falha.	Alta — falhas isoladas por serviço; degradação localizada é possível quando bem projetado.
Complexidade de *deploy*	Baixa — pipeline e *deploy* simples (uma unidade).	Média — *deploy* único com necessidade de coordenação entre módulos.	Alta — múltiplos pipelines, versões e coordenação de *deploys*, além de orquestração.
***Time-to-market* (velocidade de entrega)**	Alta — rápido para lançar MVPs e novas features sem overhead de integração entre serviços.	Alta/Média — bom equilíbrio entre velocidade e qualidade arquitetural.	Média — velocidade depende de maturidade da equipe; overhead inicial pode retardar entregas.

5.2.1 Guia para aplicação do quadro na avaliação de um projeto real

Objetivo

O presente guia tem como objetivo propor um procedimento sistemático e reprodutível para a aplicação da matriz de parâmetros (Quadro de Resultados) no processo de avaliação de projetos reais. Dessa forma, busca-se oferecer um

instrumento de apoio à decisão quanto à adoção da arquitetura de *software* mais adequada entre os modelos Monolítico, Monólito Modular e Microsserviços, considerando os fatores analisados no referencial teórico (Seção 2) e os resultados obtidos na pesquisa (Seção 5).

Pré-requisitos

Para a aplicação correta do método, são necessários os seguintes elementos:

- Disponibilidade do Quadro de Resultados consolidada na Seção 5.2;
- Informações do projeto em análise, incluindo estimativas de carga, requisitos não funcionais, restrições regulatórias, tamanho e maturidade da equipe, prazos e orçamento;
- Acesso ao referencial teórico apresentado neste trabalho, de modo a fundamentar a atribuição de notas e pesos.

Procedimento proposto

1. Cada parâmetro listado no Quadro de Resultados deve receber um peso proporcional à sua importância para o projeto em questão. A escala recomendada varia de 0,00 a 1,00, de forma que a soma total seja 1,00. Os pesos devem ser definidos em conjunto com os principais stakeholders (gestores, arquitetos de *software*, equipe de operações e segurança), conforme orientações teóricas sobre priorização de atributos de qualidade.
2. Para cada parâmetro, atribuir uma nota de 1 a 5:

- 1 representa baixa relevância para o projeto;
- 3 indica relevância moderada;
- 5 indica requisito crítico.

As notas devem ser fundamentadas em dados ou requisitos documentados, sempre com base nos conceitos discutidos na Seção 2.

3. A nota de cada parâmetro deve ser multiplicada pelo peso atribuído. A soma das pontuações resultará em um escore final para cada arquitetura.

Fórmula:

$$\text{Pontuação Final} = \sum(\text{peso} \times \text{nota})$$

4. Interpretação dos resultados

- Escore ≤ 2,5 → Arquitetura Monolítica, priorizando simplicidade e *time-to-market*.
- 2,5 < Escore ≤ 3,5 → Monólito Modular, conciliando flexibilidade e menor custo operacional.
- Escore > 3,5 → Microsserviços, quando os ganhos de escalabilidade, isolamento de falhas e autonomia das equipes justificam a complexidade adicional.

Justificativa dos limiares

Os limiares adotados (≤ 2,5 Monolítico; > 2,5–≤ 3,5 Monólito Modular; > 3,5 Microsserviços) são uma regra heurística que transforma a média ponderada de notas

(1–5) em uma recomendação prática. Eles criam uma zona intermédia em torno do ponto neutro para reduzir sensibilidade a pequenas variações nas avaliações, facilitando uma decisão inicial rápida. Ressalta-se que os limiares têm caráter orientador: recomenda-se complementar a decisão com análise qualitativa, prova de conceito e testes de sensibilidade antes de implementar a arquitetura escolhida.

Análise de sensibilidade

Recomenda-se a repetição do processo considerando diferentes cenários (pessimista, base e otimista). Essa prática auxilia na identificação de parâmetros críticos que podem alterar significativamente a decisão final, tornando a análise mais robusta.

Formalização da recomendação

O resultado da análise deve ser registrado em um relatório técnico contendo:

a. A arquitetura recomendada;
b. Justificativa baseada no escore ponderado;
c. Plano de mitigação de riscos e *roadmap* técnico;
d. Recursos necessários (infraestrutura, ferramentas, observabilidade, segurança);
e. Indicadores de desempenho esperados (SLA, MTTR, TCO).

Checklist de evidências necessárias

- Projeção de tráfego (requisições por segundo, usuários simultâneos) e previsão de crescimento;
- Requisitos de consistência transacional;
- Restrições legais e regulatórias;
- Estrutura e maturidade das equipes;
- Orçamento de infraestrutura e operações;
- Prazos críticos e metas de entrega;
- Necessidade de heterogeneidade tecnológica entre domínios;
- Nível atual de automação (CI/CD, testes automatizados) e observabilidade.

Riscos e limitações

O método não substitui o julgamento qualitativo da equipe de arquitetura e gestão. A atribuição de pesos pode refletir vieses individuais, sendo essencial a validação em conjunto com diferentes áreas da organização.

5.2.2 Exemplo de aplicação do guia prático

Objetivo

Apresenta-se um estudo de caso fictício, porém realista, que ilustra passo a passo a aplicação do procedimento proposto na Seção 5.2.1. O objetivo é demonstrar a operacionalização prática da matriz de parâmetros para apoiar a escolha entre Arquitetura Monolítica, Monólito Modular e Microsserviços.

Descrição do projeto em análise

Projeto: Plataforma de comércio eletrônico regional (B2C)
Contexto: PME de varejo que pretende lançar uma plataforma web e mobile para vendas, com previsão de crescimento de usuários e picos sazonais (ex.: campanhas).
Requisitos e restrições relevantes:

- Pico esperado: 1.200 requisições por segundo (RPS) em eventos promocionais.
- Requisitos de alta disponibilidade (SLA 99,9%) e baixa latência nas páginas críticas (checkout).
- Conformidade com normas de pagamento (PCI-DSS) e proteção de dados de clientes.
- Equipe inicial: 12 desenvolvedores (*frontend* + *backend* + *devops*).
- Orçamento operacional moderado — disponibilidade para investimento, porém com limite.
- Prazo de entrega do MVP: 3 meses (pressão de *time-to-market*).

Definição de pesos

Os pesos refletem a importância relativa de cada parâmetro para este projeto (definidos com stakeholders fictícios: produto, operações, segurança):

- Escalabilidade: 0,18
- Resiliência: 0,15
- Performance: 0,12
- Manutenibilidade: 0,12
- Segurança / Conformidade: 0,10
- Observabilidade e monitoramento: 0,08
- Custo operacional: 0,08
- Flexibilidade tecnológica: 0,07
- Adequação ao tamanho da equipe: 0,06
- Testabilidade: 0,04

Pontuação

Monolítico

- Escalabilidade: 2 (escala global; ineficiente para cargas seletivas).
- Resiliência: 2 (único processo; ponto único de falha).
- Performance: 3 (boa latência intraprocesso, mas limitações sob alta concorrência).
- Manutenibilidade: 2 (dificuldade com código crescente).
- Segurança: 3 (controle centralizado facilita algumas políticas).

- Observabilidade: 3 (menos granularidade).
- Custo: 4 (pipeline e infraestrutura mais simples).
- Flexibilidade tecnológica: 2 (mesma *stack*).
- Tamanho da equipe: 3 (adequado para equipes pequenas/médias).
- Testabilidade: 3 (testes integrados simples, isolamento limitado).

Monólito Modular

- Escalabilidade: 3 (melhor organização; ainda escala global).
- Resiliência: 3 (isolamento lógico; mas processo único).
- Performance: 4 (com módulos bem desenhados, caminhos críticos otimizados).
- Manutenibilidade: 4 (alta coesão e baixo acoplamento internos).
- Segurança: 3 (políticas por módulo, mas infraestrutura única).
- Observabilidade: 4 (instrumentação mais fácil por módulo).
- Custo: 3 (similar ao monolito, com exigência de disciplina).
- Flexibilidade tecnológica: 3 (alguma facilidade para futuras extrações).
- Tamanho da equipe: 4 (bom para equipes médias).
- Testabilidade: 4 (testes por módulo eficientes).

Microsserviços

- Escalabilidade: 5 (escalabilidade seletiva por serviço).
- Resiliência: 5 (falhas isoladas; degradação localizada).
- Performance: 4 (overhead de rede, mas alta capacidade de *throughput*).
- Manutenibilidade: 4 (serviços menores, times autônomos).
- Segurança: 3 (governança e autenticação distribuída exigidas).
- Observabilidade: 5 (metricas/*tracing* necessários e eficazes).
- Custo: 2 (infraestrutura e operação mais custosas).
- Flexibilidade tecnológica: 5 (cada serviço escolhe *stack*).
- Tamanho da equipe: 5 (adequado para times por domínio).
- Testabilidade: 3 (unitários fáceis; integração distribuída complexa).

Cálculo das pontuações ponderadas

Para cada arquitetura, multiplicou-se a nota (1–5) pelo peso do parâmetro e somaram-se as contribuições. Os resultados são apresentados com duas casas decimais.

- **Monolítico — pontuação final (∑ peso × nota): 2,56** (ex.: Escalabilidade 0,18×2 = 0,36; Resiliência 0,15×2 = 0,30; ... total = 2,56)
- **Monólito Modular — pontuação final: 3,42** (ex.: Escalabilidade 0,18×3 = 0,54; Resiliência 0,15×3 = 0,45; ... total = 3,42)

- **Microsserviços — pontuação final: 4,24** (ex.: Escalabilidade 0,18×5 = 0,90; Resiliência 0,15×5 = 0,75; ... total = 4,24)

Interpretação segundo limiares do guia

- Escore ≤ 2,5 → Monolítico
- 2,5 < Escore ≤ 3,5 → Monólito Modular
- Escore > 3,5 → Microsserviços

Aplicando os limiares, a arquitetura recomendada para este projeto é Microsserviços (pontuação 4,24). A vantagem principal deriva da forte exigência de escalabilidade seletiva, resiliência e observabilidade para suportar picos promocionais e requisitos de disponibilidade.

Formalização da recomendação

a. Arquitetura recomendada: Microsserviços.

b. Justificativa: pontuação global superior (4,24) e alinhamento com requisitos críticos: escalabilidade em picos, resiliência e observabilidade detalhada para monitoramento de checkout e pagamentos.

c. Plano de mitigação de riscos e *roadmap* técnico (sumário):

- **Fase 0 — MVP (0–3 meses):** construir MVP como *monólito modular* estritamente organizado por *bounded contexts* (permitindo *time-to-market* e disciplina na modularidade).
- **Fase 1 — Extração incremental (3–12 meses):** identificar módulos críticos (checkout, catálogo, pagamentos, carrinho) e extrair como microsserviços em iterações; implantar API Gateway e autenticação centralizada.
- **Fase 2 — Operacionalização (paralela):** implementar CI/CD por serviço, observabilidade (Prometheus + Grafana + Jaeger), logging centralizado (ELK/EFK), e estratégias de resiliência (Circuit Breaker, *retries* com *backoff*).
- **Fase 3 — Otimização (12+ meses):** particionamento de dados por serviço, tuning de autoscaling, e revisão de custos operacionais.

d. Recursos necessários: orquestração (Kubernetes), registry de serviços, message broker (RabbitMQ/Kafka) para comunicações assíncronas, CDN para *assets*, ferramentas de *tracing*, e time *DevOps* para automação. e) Indicadores esperados: SLA 99,9%, tempo médio de resposta do checkout < 500 ms, MTTR < 30 min para falhas críticas.

Checklist de evidências necessárias:

- Projeção de tráfego: **sim** (1.200 RPS em pico)
- Requisitos de consistência transacional: **sim** (pagamentos exigem consistência forte) — mitigação: transações locais + orquestração via sagas para processos distribuídos.

- Restrição regulatória: **sim** (PCI-DSS) — mitigação: tokenização, gateway de pagamento terceirizado.
- Estrutura/maturidade da equipe: **sim** (12 desenvolvedores; há capacidade para times por serviço).
- Orçamento de infraestrutura: **moderado** — mitigação: iniciar com *modular monolith* e migrar incrementalmente.
- Prazos críticos: **sim** (MVP 3 meses) — justificativa do *roadmap* híbrido (modular → microsserviços).
- Necessidade de heterogeneidade tecnológica: **média** — útil à medida que a organização cresce.
- Nível de automação: **atualizar** (necessidade de investir em CI/CD e observabilidade desde fase inicial).

Análise de sensibilidade

Realizaram-se simulações (cenários base/pessimista/otimista) alterando pesos de Escalabilidade e Custo. Observou-se que, mesmo quando o peso de Custo sobe substancialmente (por exemplo, Custo = 0,18 e Escalabilidade = 0,10), o Monólito Modular torna-se competitivo; logo, a decisão final deve sempre considerar restrições orçamentárias estritas. Recomenda-se a realização da análise de sensibilidade com stakeholders antes da formalização definitiva.

Conclusão do exemplo

O estudo de caso demonstra a aplicabilidade do guia: ao sistematizar pesos e notas, torna-se possível justificar tecnicamente a escolha arquitetural. Para o projeto em análise, a recomendação por Microsserviços é consistente com os requisitos críticos de escalabilidade, resiliência e observabilidade. Entretanto, recomenda-se uma abordagem incremental (iniciar como monólito modular e extrair serviços prioritários), reduzindo riscos técnicos e financeiros.

5.3 Limitações do estudo

Este estudo baseou-se majoritariamente em revisão bibliográfica e análise qualitativa. Não foram realizados benchmarks empíricos comparativos dentro do escopo deste trabalho, o que limita a generalização quantitativa dos achados.

Outra limitação refere-se ao viés de publicação: estudos de casos bem-sucedidos tendem a ser mais divulgados, enquanto relatos de falhas operacionais podem estar sub-representados. Ademais, as recomendações são heurísticas e devem ser contextualizadas conforme o setor, requisitos não-funcionais e maturidade organizacional.

5.4 Sugestões para trabalhos futuros

• Conduzir estudos empíricos comparativos (benchmarks de desempenho, latência e custos) entre implementações representativas das três arquiteturas.

• Desenvolver estudos de caso em empresas locais para validar e calibrar o guia de seleção arquitetural.

• Investigar arquiteturas híbridas práticas (por exemplo, monólito modular complementado por filas assíncronas) que possam mitigar custos sem perder escalabilidade.

6. Conclusão

Este artigo teve como objetivo principal realizar uma análise comparativa sistemática entre as arquiteturas de *software* monolítica, monólito modular e microsserviços. A pesquisa, fundamentada em uma revisão bibliográfica abrangente, buscou identificar as características, vantagens, desvantagens e contextos de aplicação ideais para cada um desses paradigmas arquiteturais, fornecendo um panorama crítico para auxiliar na tomada de decisão em projetos de desenvolvimento de *software*.

Ao longo deste estudo, foi possível constatar que cada arquitetura possui um conjunto distinto de atributos que as tornam mais ou menos adequadas a diferentes cenários. A arquitetura monolítica, embora ofereça simplicidade inicial no desenvolvimento e implantação, demonstrou limitações significativas em termos de escalabilidade granular e flexibilidade de manutenção à medida que os sistemas crescem. Seus pontos fortes residem na facilidade de gerenciamento para projetos de pequeno e médio porte, onde a complexidade inicial é menor e a equipe é mais coesa.

O monólito modular emergiu como uma abordagem intermediária, buscando equilibrar a simplicidade do monolito com a organização interna e a modularidade. Esta arquitetura provou ser eficaz para promover a manutenibilidade e facilitar uma eventual transição para microsserviços, ao mesmo tempo em que mitiga alguns dos desafios de escalabilidade e flexibilidade inerentes aos monolitos puros. Sua adequação é notável em projetos que preveem crescimento futuro, mas que ainda não justificam a complexidade total dos microsserviços.

Por outro lado, a arquitetura de microsserviços revelou-se a mais apropriada para sistemas de grande escala, complexos e com equipes de desenvolvimento distribuídas. Sua principal vantagem reside na capacidade de proporcionar alta escalabilidade, resiliência e liberdade tecnológica, permitindo que diferentes serviços sejam desenvolvidos, implantados e escalados independentemente. No entanto, essa flexibilidade e poder vêm acompanhados de um custo considerável em termos de complexidade operacional, necessidade de infraestrutura especializada e desafios de gerenciamento distribuído.

A contribuição prática deste trabalho materializa-se na apresentação de um quadro comparativo detalhada e um guia de apoio à decisão. Esses artefatos visam auxiliar profissionais e estudantes de Engenharia de *Software* na seleção da arquitetura mais alinhada às necessidades específicas de seus projetos, considerando fatores técnicos, organizacionais e de negócio, como escalabilidade, manutenibilidade, custo operacional, resiliência, segurança e adequação ao tamanho da equipe.

É importante ressaltar que este estudo, embora abrangente, possui algumas limitações. A análise foi predominantemente teórica, baseada em revisão bibliográfica sistemática, e não incluiu estudos de caso empíricos ou experimentos práticos que

pudessem validar as conclusões em cenários reais de desenvolvimento. Além disso, a rápida evolução do cenário tecnológico pode introduzir novas abordagens ou refinar as existentes, o que requer atualização contínua do conhecimento.

Para trabalhos futuros, sugere-se a realização de estudos de caso em empresas que adotaram diferentes arquiteturas, a fim de coletar dados empíricos sobre os desafios e benefícios reais de cada abordagem. A exploração de ferramentas e metodologias para auxiliar na transição entre arquiteturas, bem como a análise do impacto de novas tecnologias (como computação sem servidor e *edge computing*) na escolha arquitetural, também representam áreas promissoras para pesquisa. Espera-se que este artigo sirva como um ponto de partida sólido para futuras investigações e como um recurso valioso para a comunidade de desenvolvimento de *software*.

Referências

AMARAL, J. D. C. A evolução das arquiteturas monolíticas para as arquiteturas baseadas em microsserviços. 2018. Dissertação (Mestrado em Engenharia Informática) – Instituto Superior de Engenharia do Porto. Repositório Científico do Instituto Politécnico do Porto. Disponível em: https://recipp.ipp.pt/bitstream/10400.22/11920/1/DM_JoseAmaral_2018_MEI.pdf. Acesso em: 21 out. 2025.

BARDE, K. Modular Monoliths: Revolutionizing Software Architecture for Efficient Payment Systems in Fintech. *International Journal of Computer Trends and Technology*, v. 71, n. 10, p. 20–27, 2023.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice.* 3. ed. Addison-Wesley Professional, 2013.

CHRISTUDÁS, B. *Practical Microservices Architectural Patterns.* Apress, 2019.

ELGHERIANI, N. S.; AHMED, N. A. S. Microservices vs. Monolithic Architectures: The Differential Structure Between Two Architectures. *International Journal of Applied Sciences and Technology*, v. 4, n. 3, p. 509–514, 2022. DOI: 10.47832/2717-8234.12.47. Disponível em: http://dx.doi.org/10.47832/2717-8234.12.47. Acesso em: 21 out. 2025.

FOOTE, B.; YODER, J. Big Ball of Mud. In: *Fourth Conference on Pattern Languages of Programs (PLoP'97/EuroPLoP'97)*, Monticello, Illinois, set. 1997. University of Illinois at Urbana-Champaign, 1999. Disponível em: http://www.laputan.org/mud/mud.html. Acesso em: 21 out. 2025.

FOWLER, M. *Patterns of Distributed Systems.* Addison-Wesley Professional, 2023.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). *Systems and software engineering — Architecture description* (ISO/IEC/IEEE 42010:2011(E)). Geneva: ISO, 2011.

JOVANOVIĆ, M. What Is a Modular Monolith? *Milan Jovanović Tech Blog*, 2023. Disponível em: https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith. Acesso em: 21 out. 2025.

KALSKE, M. Transforming monolithic architecture towards microservice architecture. 2017. Dissertação (Mestrado em Ciências da Computação) – University of Helsinki. University of Helsinki Helda Repository.

NAIR, H. Top Microservices Design Patterns for Microservices Architecture in 2025. *LambdaTest Blog*, 2025. Disponível em: https://www.lambdatest.com/blog/microservices-design-patterns/. Acesso em: 21 out. 2025.

NAMIOT, D.; SNEPS-SNEPPE, M. On Micro-Services Architecture. *International Journal of Open Information Technologies*, v. 2, n. 9, p. 24–27, 2014.

NEWMAN, S. *Building Microservices.* O'Reilly Media, 2015.

NEWMAN, S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, 2020.

PERRY, D. E.; WOLF, A. L. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, v. 17, n. 4, p. 40–52, 1992.

SKALICKÝ, M. Analysis and Comparison of Application Architecture: Monolith, Microservices and Modular Approach. Prague: Czech Technical University in Prague, 2023.

SU, R.; LI, X. Modular Monolith: Is This the Trend in Software Architecture? In: *IEEE/ACM International Workshop New Trends in Software Architecture (SATrends)*, 2024. IEEE/ACM.

TAPIA, F. et al. From Monolithic Systems to Microservices: A Comparative Study of Performance. *Applied Sciences*, v. 10, n. 17, p. 5797, 2020. DOI: 10.3390/app10175797. Disponível em: https://doi.org/10.3390/app10175797. Acesso em: 21 out. 2025.

TSECHELIDIS, M.; NIKOLAIDIS, N.; MAIKANTIS, T.; AMPATZOGLOU, A. Modular Monoliths the way to Standardization. 2023.

XU, Alex. Monolith vs Microservices vs Modular Monoliths: What's the Right Choice. [S. l.], mar. 2025. Disponível em: https://blog.bytebytego.com/p/monolith-vs-microservices-vs-modular. Acesso em: 22 out. 2025.