ARQUITETURA DE MICRO-FRONTEND: ANÁLISE DE ABORDAGENS DE IMPLEMENTAÇÃO NO DESENVOLVIMENTO WEB

Esthéfani Possamai[1], Luciano Antunes[2]

Resumo: A crescente complexidade do *frontend* de aplicações *web* modernas, aliada à dificuldade de escolher a arquitetura de *software* mais adequada, representa o problema central desta pesquisa. Para endereçar essa lacuna, este trabalho propôs a avaliação comparativa das abordagens de implementação da arquitetura de *micro-frontend* (Single-SPA e *Web Components*) por meio de provas de conceito. A metodologia empregada incluiu a fundamentação teórica da arquitetura, o desenvolvimento prático de protótipos e a análise e síntese dos resultados. A análise demonstrou que o Single-SPA é robusto para orquestração de frameworks heterogêneos, enquanto os *Web Components*, potencializados pelo Stencil, oferecem encapsulamento superior e maior interoperabilidade. A escolha entre eles depende do alinhamento estratégico com o contexto do projeto. Os principais avanços residem na validação prática e comparativa dessas abordagens, gerando um conhecimento que auxilia empresas e desenvolvedores na tomada de decisões mais assertivas, facilitando a adoção eficaz dessa arquitetura e o alcance dos objetivos de negócio.

Palavras-chave: micro-frontends; arquitetura de software; desenvolvimento web; microsserviços.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), esthefani_possamai@unesc.net

[2]Orientador. Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), luc@unesc.net

ABSTRACT: The increasing complexity of modern web application frontends, coupled with the difficulty of choosing the most suitable software architecture, represents the central problem of this research. To address this gap, this work proposed the comparative evaluation of micro-frontend architecture implementation approaches (Single-SPA and Web Components) through proofs of concept. The methodology employed included the theoretical foundation of the architecture, the practical development of prototypes, and the analysis and synthesis of the results. The analysis demonstrated that Single-SPA is robust for orchestrating heterogeneous frameworks, while Web Components, enhanced by Stencil, offer superior encapsulation and greater interoperability. The choice between them depends on strategic alignment with the project context. The main advances lie in the practical and comparative validation of these approaches, generating knowledge that assists companies and developers in making more assertive decisions, facilitating the effective adoption of this architecture and the achievement of business objectives.

Keywords: micro-frontends; software architecture; web development; microservices.

1 INTRODUÇÃO

No cenário atual, desenvolver de forma eficaz a camada de apresentação de uma aplicação *web* moderna, conhecida como *frontend*, tornou-se uma tarefa essencial e desafiadora para as empresas (Geers, 2020). Essa camada, responsável pela interação direta com os usuários e pela exibição de informações de forma clara e responsiva em diversos dispositivos, deve entregar valor de maneira rápida e eficaz (Peltonen; Mezzalira; Taibi, 2021).

Adicionalmente, um fator que agrava esse desafio e contribui para o declínio de um *software* é a escolha inadequada de uma arquitetura que não se adapta ao contexto da aplicação, assim como a adoção de abordagens de implementação ineficientes (Santos, 2024). Segundo a ISO/IEC/IEEE 42010:2022 (2022), a arquitetura de *software* é o alicerce fundamental de um sistema, que define seus componentes, as relações entre eles e os princípios que orientam seu design e evolução. Dessa forma, uma arquitetura mal escolhida pode comprometer o desempenho, a manutenção e a evolução do sistema (Montelius, 2021).

Diante desse desafio, a escolha da arquitetura de *software* é um dos aspectos mais importantes no planejamento de um sistema, uma vez

que, à medida que o desenvolvimento avança, a aplicação pode se tornar rapidamente complexa e difícil de manter (Sommerville, 2019).

Para essa escolha, as arquiteturas de *software* podem ser classificadas em dois tipos principais: monolítica e distribuída (Richards; Ford, 2020). O estilo monolítico é baseado em uma única base de código e uma unidade de execução, enquanto a arquitetura distribuída consiste em unidades de execução descentralizadas e fracamente acopladas (Nallainathan, 2024).

Devido à sua simplicidade, o modelo monolítico torna-se uma abordagem tradicional para o desenvolvimento da camada de apresentação na *web* e frequentemente é a escolha inicial de arquitetura. No entanto, essa abordagem apresenta problemas relacionados à manutenção, colaboração e escalabilidade. Dessa forma, aplicações mais complexas tendem a adotar uma arquitetura distribuída (Richards; Ford, 2020).

Nesse contexto, com a complexidade crescente dos sistemas nos últimos anos, uma arquitetura distribuída ganhou destaque: a de microsserviços (Richards; Ford, 2020). Considerando a evolução e a complexidade das aplicações do lado do cliente, ou seja, as aplicações de *frontend*, a arquitetura de microsserviços para essa camada desempenha um papel ainda mais significativo, pois esse modelo oferece escalabilidade e permite que diferentes partes sejam desenvolvidas simultaneamente por várias equipes de desenvolvedores (Pavlenko et al., 2020).

Em face desse crescimento e da expansão da arquitetura de microsserviços, o *micro-frontend* foi apresentado no ThoughtWorks Technology Radar (2016) (Geers, 2020). A arquitetura estende os conceitos de microsserviços ao *frontend*, tratando-o como uma composição de recursos, onde cada serviço é mantido por uma equipe independente. Mezzalira (2021) define a abordagem como uma arquitetura do lado do cliente que surgiu para acompanhar o crescimento dos microsserviços.

No entanto, embora a arquitetura de *micro-frontend* seja uma referência para os desenvolvedores atualmente, muitas empresas ainda enfrentam desafios na sua implementação, decorrentes da falta de uma visão ampla e voltada à prática em termos de abordagens de implementação (Jackson, 2019). Além disso, devido à falta de conhecimento em ferramentas eficazes, os desenvolvedores acabam consumindo muito tempo para gerenciar o projeto, o que reduz o tempo disponível para o desenvolvimento efetivo das funcionalidades (Peltonen; Mezzalira; Taibi, 2021).

Considerando essa lacuna, esta pesquisa propõe uma análise

estrutural da arquitetura de *micro-frontend*, juntamente com uma análise avaliativa de forma comparativa entre duas abordagens de implementação, utilizando protótipos práticos de desenvolvimento *web*, validados por meio de provas de conceito. Assim, espera-se auxiliar empresas e desenvolvedores na tomada de decisões mais assertivas, facilitando a adoção eficaz dessa arquitetura. Esta pesquisa não apenas contribuirá para avanços tecnológicos, mas também fornecerá embasamento para que as equipes analisem de maneira mais consciente o contexto em que os projetos estão inseridos e como a escolha do *micro-frontend* pode influenciar, positiva ou negativamente, o alcance dos objetivos de negócio.

Para preencher a lacuna identificada, é fundamental compreender que a arquitetura de *micro-frontend* estende os princípios dos microsserviços para a interface do usuário, fragmentando o *frontend* em componentes independentes e especializados (Peltonen; Mezzalira; Taibi, 2021). Essa abordagem permite que cada funcionalidade seja desenvolvida, testada e implantada de forma autônoma por equipes distintas, que podem utilizar diferentes tecnologias, como Angular ou React, para cada parte do sistema.

Dentre as diversas técnicas disponíveis, este estudo se concentra em duas abordagens proeminentes: Web Components, que utilizam um conjunto de APIs nativas da *web* para criar elementos HTML reutilizáveis e encapsulados, ideais para componentes compartilhados; e o *framework* Single-SPA, uma solução robusta para orquestrar múltiplos *micro-frontends* desenvolvidos em diferentes tecnologias, gerenciando seus ciclos de vida em uma única aplicação *web*.

Nesse contexto, o objetivo geral deste trabalho foi avaliar as abordagens de implementação da arquitetura de *micro-frontend* no desenvolvimento *web*, utilizando provas de conceito para revelar as características e comportamentos observados ao longo do desenvolvimento de dois protótipos de *frontend*.

Para alcançar tal propósito, a pesquisa primeiramente fundamentou teoricamente os conceitos de arquitetura *web* e as abordagens de implementação de *micro-frontends*, com o intuito de estabelecer o referencial que subsidiou a análise. Em seguida, foram desenvolvidos dois protótipos práticos, utilizando as abordagens de Single-SPA e Web Components.

Posteriormente, esses protótipos foram validados por meio de provas de conceito para examinar as características e comportamentos de cada abordagem. Por fim, os resultados obtidos foram analisados para sin-

tetizar as conclusões finais, destacando as vantagens e desvantagens das abordagens estudadas e contribuindo para a tomada de decisão técnica em projetos de *software*.

2 TRABALHOS CORRELATOS

A literatura recente sobre a arquitetura de *micro-frontends* tem se concentrado em explorar diferentes estratégias de implementação e integração, visando resolver desafios práticos como a dependência de frameworks, a complexidade arquitetural e a manutenibilidade de sistemas *web*. Estudos recentes, como os de Bastos (2020), Silva (2023) e Queiroz (2023), evidenciam essa tendência, empregando provas de conceito para validar abordagens distintas.

A pesquisa realizada por Queiroz (2023) teve como objetivo propor uma abordagem para implementação de *micro-frontends* agnóstica de *framework*, utilizando as tecnologias Web Components e Webpack Module Federation. Para tanto, foram elaboradas quatro provas de conceito (PoCs): a primeira demonstrou a integração entre os *micro-frontends*; a segunda explorou implementações com diferentes *frameworks* (React, Angular e Vue); a terceira focou no uso do Shadow DOM para isolamento de estilos; e a quarta validou a abordagem de forma nativa.

Os resultados da pesquisa de Queiroz (2023) indicaram que a combinação das tecnologias foi eficiente, garantindo o encapsulamento, o isolamento de estilos e o carregamento dinâmico dos *micro-frontends*.

Silva (2023), por sua vez, investigou a aplicação combinada das arquiteturas de *micro-frontends* e arquitetura limpa no desenvolvimento de uma loja online. A metodologia empregada consistiu na elaboração de cinco provas de conceito, cada uma voltada para um desafio específico da aplicação, como a integração dos módulos e a construção do carrinho de compras, utilizando tecnologias como React, TypeScript, Module Federation e a ferramenta SonarQube.

Dentre os principais resultados de Silva (2023), destacam-se a separação clara de responsabilidades e a facilidade de manutenção. Por outro lado, foram identificadas limitações, tais como a complexidade inicial na configuração do ambiente e conflitos de estilos CSS.

Em sua dissertação, Bastos (2020) estudou a aplicação da arquitetura de *micro-frontends* para mitigar problemas de acoplamento e escalabilidade em aplicações *web*. O trabalho implementou uma prova de conceito de comércio eletrônico, na qual foram utilizadas e comparadas di-

ferentes estratégias de integração (*client-side*, *server-side* e *universal composition*).

Os resultados obtidos por Bastos (2020) apontaram como benefícios a possibilidade de desenvolvimento independente e a redução do acoplamento, mas também identificaram desafios como o risco de inconsistência na interface e o aumento do tamanho dos *bundles*.

3 MATERIAIS E MÉTODOS

A presente pesquisa é de natureza qualitativa, exploratória e aplicada, com o objetivo de analisar e comparar, através de Provas de Conceito (PoCs), duas abordagens distintas para a implementação de *micro-frontends*: Single-SPA e Web Components. As PoCs foram desenvolvidas em um ambiente controlado e com funcionalidades equivalentes para garantir uma comparação justa em relação à orquestração, isolamento, encapsulamento e complexidade estrutural. A metodologia de PoCs foi escolhida por seu caráter experimental, permitindo a validação de hipóteses técnicas em um cenário de desenvolvimento real.

3.1 AMBIENTE DE DESENVOLVIMENTO

O ambiente de desenvolvimento foi configurado para garantir estabilidade, reprodutibilidade e isolamento das variáveis de teste. Para tanto, utilizou-se uma estação de trabalho com o sistema operacional Linux e distribuição Ubuntu 22.04.3 LTS, 16 GB de memória RAM e unidade de armazenamento SSD.

Esta configuração foi selecionada pela sua compatibilidade nativa com ferramentas modernas de desenvolvimento, robustez e pela facilidade de gerenciamento de dependências do ecossistema Node.js. As ferramentas de *software*, suas versões e respectivas finalidades na pesquisa são detalhadas na Tabela 1.

Tabela 1 - Ferramentas e tecnologias utilizadas no ambiente base

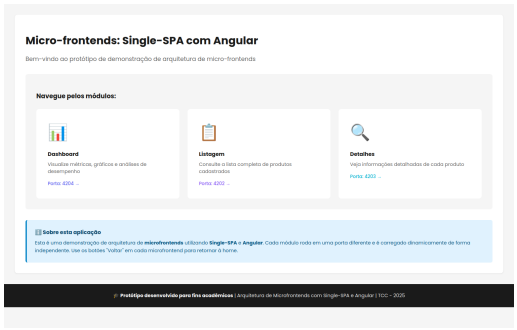
Ferramenta	**Versão**	**Finalidade**
Node.js	18.18.2	*Runtime* JavaScript
NPM	10.7.0	Gerenciador de pacotes
Git	2.42	Controle de versão do código-fonte
Linux Ubuntu	22.04 LTS	Sistema operacional
IntelliJ IDEA	2023.2	Ambiente de desenvolvimento

Fonte: Elaborado pelo autor.

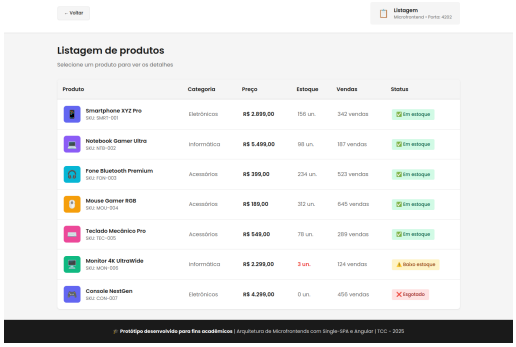
3.2 ESTRUTURA DA APLICAÇÃO E ARQUITETURA DOS PROTÓTIPOS

O sistema desenvolvido para as Provas de Conceito simulou um cenário de aplicação corporativa real, composto por funcionalidades independentes. Para garantir uma comparação justa, a interface visual (UI) e a experiência do usuário (UX) dos protótipos foram mantidas idênticas. A estrutura consistiu em três módulos equivalentes em ambas as abordagens: um *micro-frontend* de *dashboard*, um de listagem e um de detalhes. A Figura 1 ilustra a equivalência visual entre os protótipos, apresentando a tela principal (1a), de listagem (1b), da *dashboard* (1c) e de detalhes (1d).

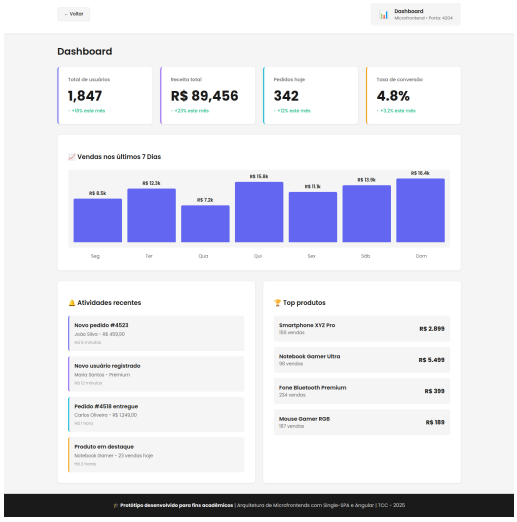
Figura 1 - Equivalência visual dos protótipos demonstrada pelo protótipo da abordagem de Single-SPA



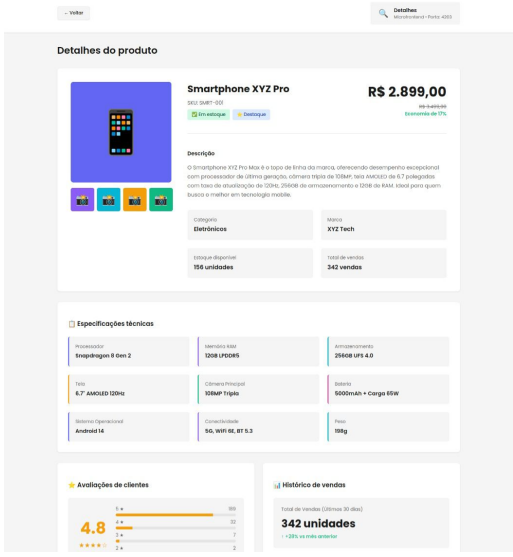
(a) Tela principal.



(b) Tela de listagem.



(c) Tela da *dashboard*.



(d) Tela de detalhes.

Fonte: Elaborado pelo autor.

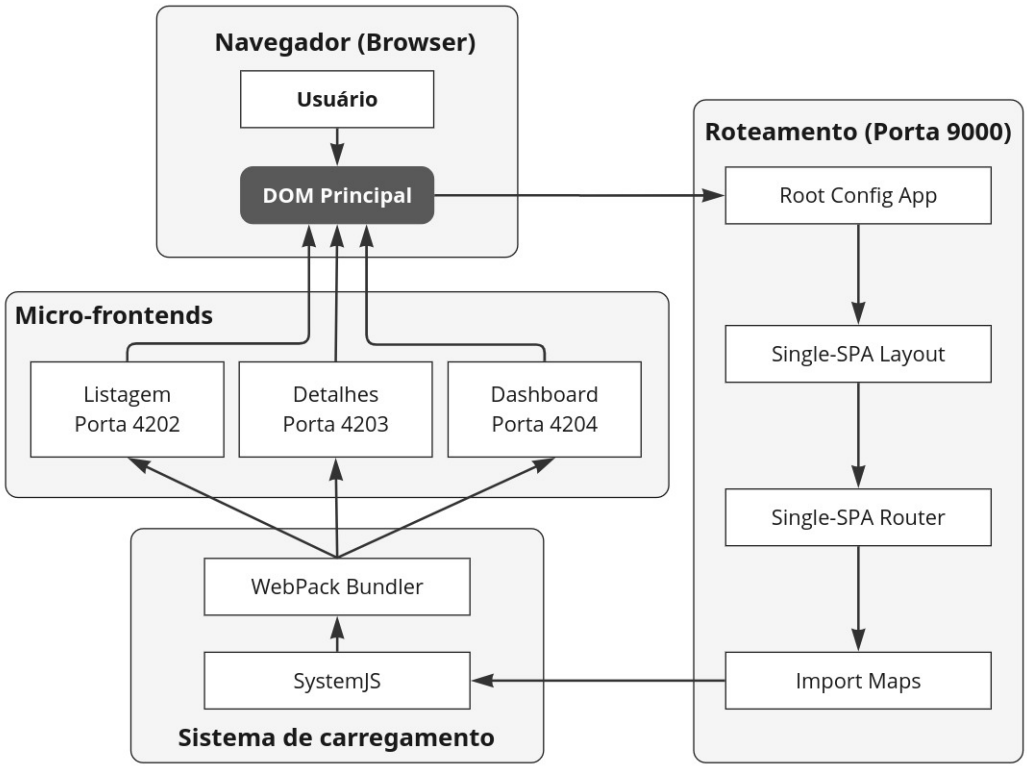
3.2.1 Arquitetura do protótipo com Single-SPA

O primeiro protótipo foi construído com base no *framework* Single-SPA(v6.0.3), que tem como principal característica a orquestração de múltiplos *micro-frontends* dentro de um único ambiente de execução. Nesse modelo, uma aplicação raiz, denominada Root Config, atua como orquestrador central, responsável pela configuração de rotas, carregamento dinâmico dos módulos e gerenciamento do ciclo de vida das aplicações.

A implementação utilizou o Single-SPA Angular (v9.2.0) como adaptador para integrar *micro-frontends* desenvolvidos em Angular (v17.0.0). O sistema de roteamento foi configurado através do Single-SPA Layout (v3.0.0), enquanto TypeScript (v5.0.0) foi empregado como linguagem de programação. O empacotamento dos módulos foi realizado com Webpack (v5.0.0), e a transpilação do código JavaScript ficou a cargo do Babel (v7.28.4).

Cada *micro-frontend* foi implementado como uma aplicação Angular autônoma, com configuração própria de *build* e dependências. A comunicação entre os módulos ocorre por meio de eventos customizados, propriedades e da API de histórico do navegador, permitindo a troca de informações e navegação sincronizada sem acoplamento direto. A Figura 2 apresenta a arquitetura geral do protótipo Single-SPA.

Figura 2 - Arquitetura geral do protótipo Single-SPA



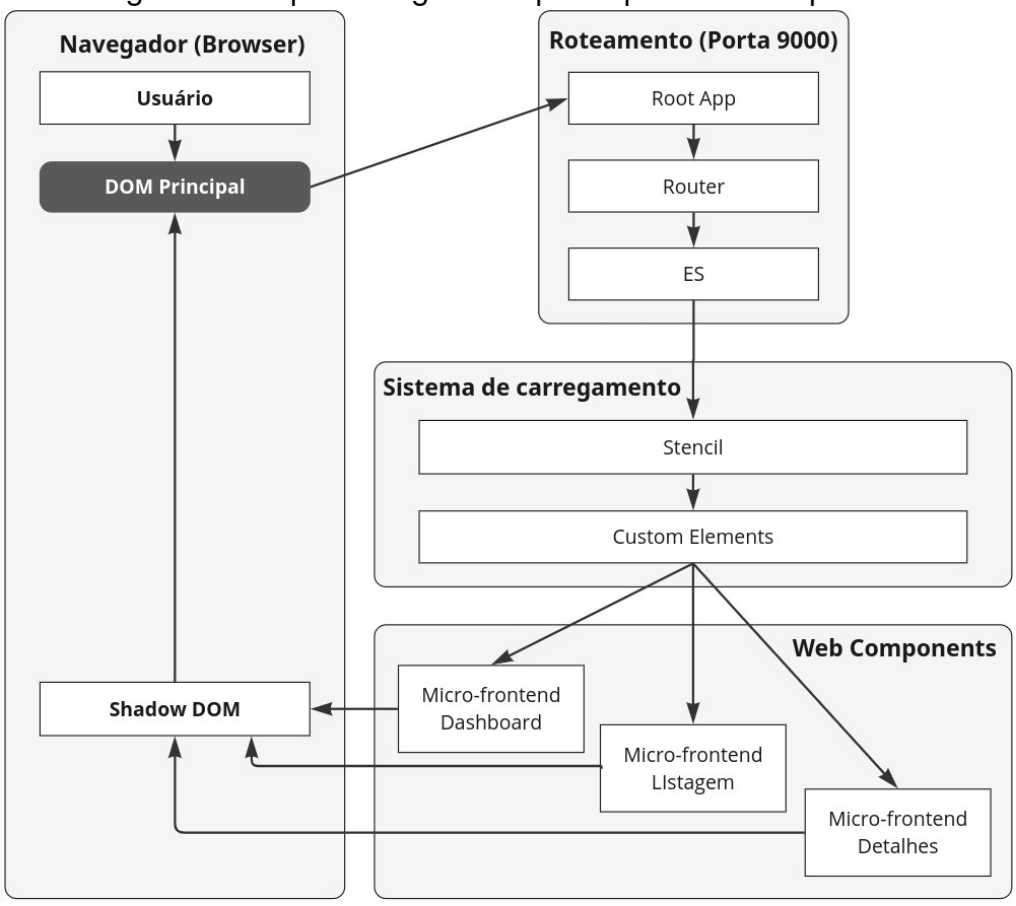
Fonte: Elaborado pelo autor.

3.2.2 Arquitetura do protótipo com Web Components

O segundo protótipo foi implementado com Web Components, utilizando o compilador Stencil (v4.27.1), que gera *Custom Elements* compatíveis com as especificações W3C e suportados nativamente pelos navegadores modernos. Essa abordagem aproveita os recursos de Custom Elements Registry e Shadow DOM, permitindo a criação de componentes encapsulados e reutilizáveis sem dependência de *frameworks* externos de orquestração.

O sistema de roteamento foi implementado através do Stencil Router (v0.6.0), mantendo TypeScript (v5.0.0) como linguagem de programação. A aplicação principal, denominada *Root App*, é responsável por estruturar a interface e coordenar a comunicação entre os componentes. A Figura 3 apresenta a arquitetura geral do protótipo Web Components, demonstrando como o *Root App* coordena os módulos gerados pelo Stencil e registrados automaticamente no navegador.

Figura 3 - Arquitetura geral do protótipo Web Components



Fonte: Elaborado pelo autor.

3.3 METODOLOGIA DAS PROVAS DE CONCEITO

A metodologia de provas de conceito, do inglês *proof of concept* (PoC), foi selecionada devido à sua capacidade de validar a viabilidade de soluções e tecnologias em pequena escala, permitindo testar arquiteturas de forma controlada antes de investimentos mais amplos. Conforme Elliott (2021), uma PoC visa demonstrar que um protótipo, seja um modelo, um artefato ou uma ideia, atinge os resultados e funções esperados, fornecendo evidências concretas para a continuidade do desenvolvimento.

No contexto desta pesquisa, o foco das PoCs consistiu em avaliar a implementação de *micro-frontends* utilizando as abordagens Single-SPA e Web Components, com atenção aos aspectos de modularidade, encapsulamento, isolamento e orquestração. Cada PoC foi estruturada para atingir um objetivo específico previamente definido, possibilitando observar de forma controlada o comportamento do sistema em relação às hipóteses levantadas.

Para tal, cada prova de conceito possuiu sua própria arquitetura e escopo funcional, desenvolvida com os componentes essenciais necessários para validar os objetivos da abordagem estudada. Essa estratégia permitiu isolar variáveis de interesse, reduzir interferências externas e fornecer resultados reprodutíveis e confiáveis sobre a viabilidade técnica e prática das soluções propostas.

3.4 POC 1 - COMPOSIÇÃO E ORQUESTRAÇÃO

Esta primeira Prova de Conceito (PoC) teve como objetivo avaliar a capacidade das abordagens Single-SPA e Web Components de compor e orquestrar múltiplos *micro-frontends* de forma independente e coordenada. O desafio consistiu em construir protótipos com ambas as tecnologias, observando o carregamento dinâmico dos módulos, a navegação entre rotas e a execução dos ciclos de vida dos componentes sem que houvesse interferência entre eles.

3.4.1 Requisitos do desafio

Para atingir os objetivos propostos, foram definidos os seguintes requisitos:

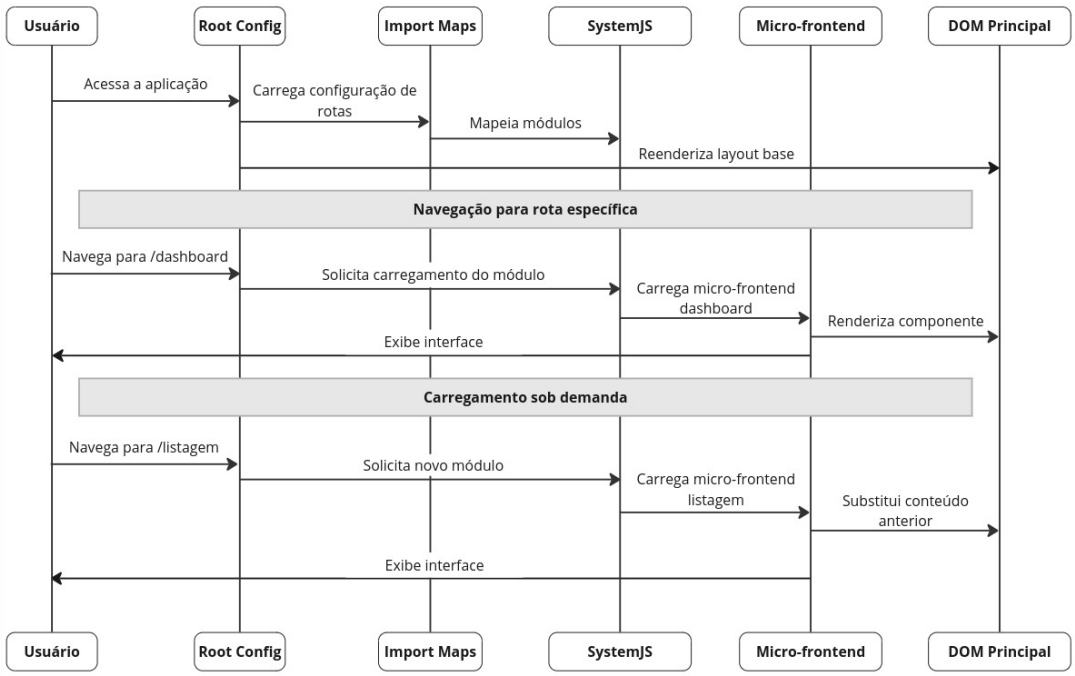
- A aplicação raiz (ou *root*) deve ser capaz de carregar dinamicamente os *micro-frontends*;
- A navegação entre os diferentes *micro-frontends* deve ser fluida e gerenciada pela aplicação principal;

- Cada *micro-frontend* deve operar de forma autônoma, sem interferir no ciclo de vida dos outros;
- O carregamento dos *micro-frontends* deve ser feito sob demanda, ou seja, apenas quando solicitado pelo usuário.

3.4.2 Apresentação da solução

No protótipo com Single-SPA, a aplicação raiz (root config) foi configurada para gerenciar o carregamento dos *micro-frontends* através de Import Maps e SystemJS. Cada *micro-frontend* foi desenvolvido como uma aplicação Angular autônoma. O fluxo de carregamento e navegação pode ser observado na Figura 4.

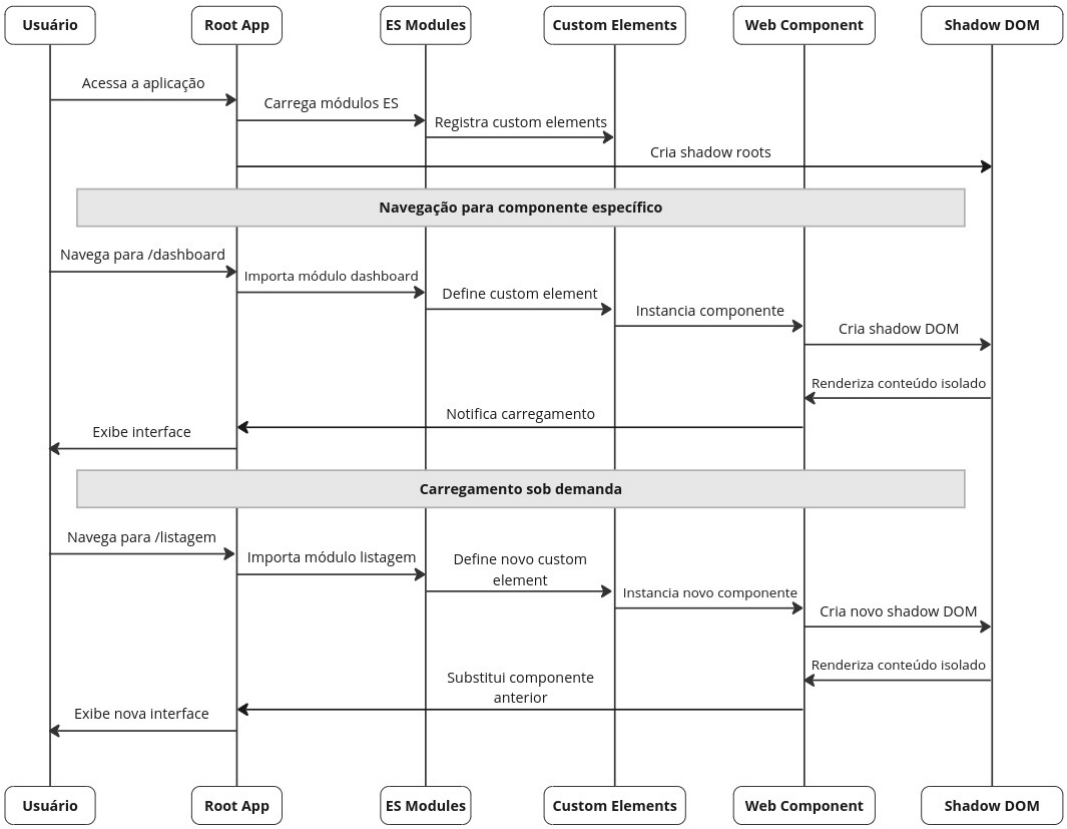
Figura 4 - Fluxo de carregamento com Single-SPA



Fonte: Elaborado pelo autor.

No protótipo com Web Components, o mesmo cenário foi reproduzido utilizando importações dinâmicas de ES Modules e a substituição direta de componentes no DOM. Esta abordagem dispensou a necessidade de um *framework* de orquestração, aproveitando as capacidades nativas do navegador. O fluxo de carregamento pode ser visto na Figura 5.

Figura 5 - Fluxo de carregamento com Web Components



Fonte: Elaborado pelo autor.

3.5 POC 2 – ISOLAMENTO DE ESTILOS E ENCAPSULAMENTO

A segunda Prova de Conceito (PoC) focou na análise do isolamento de estilos, encapsulamento de componentes e prevenção de conflitos entre módulos. O desafio era verificar como cada abordagem lida com a coexistência de diferentes bibliotecas de design e classes de CSS com nomes idênticos em diferentes *micro-frontends*.

3.5.1 Requisitos do desafio

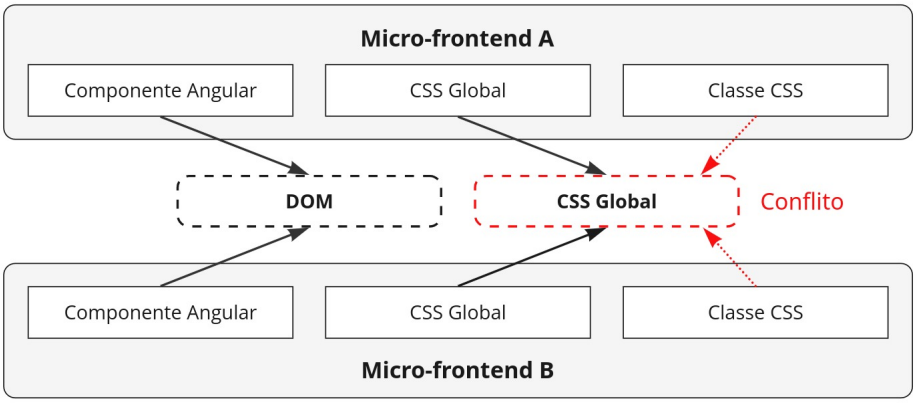
Para atingir os objetivos propostos, foram definidos os seguintes requisitos:

- Os estilos de um *micro-frontend* não devem vazar e afetar outros *micro-frontends*;
- A aplicação deve permitir a utilização de diferentes frameworks de CSS simultaneamente sem conflitos;
- Componentes com nomes de classes CSS idênticos devem coexistir na mesma página sem sobreposição de estilos.

3.5.2 Apresentação da solução

No ambiente Single-SPA, o método consistiu em criar classes CSS com nomes idênticos em dois *micro-frontends* distintos. Ambos os módulos foram configurados para serem carregados simultaneamente na mesma página. O objetivo deste método era verificar como os estilos globais de um *micro-frontend*, aplicados ao DOM compartilhado, se comportariam ao encontrar classes de mesmo nome de outro módulo, conforme o diagrama da Figura 6.

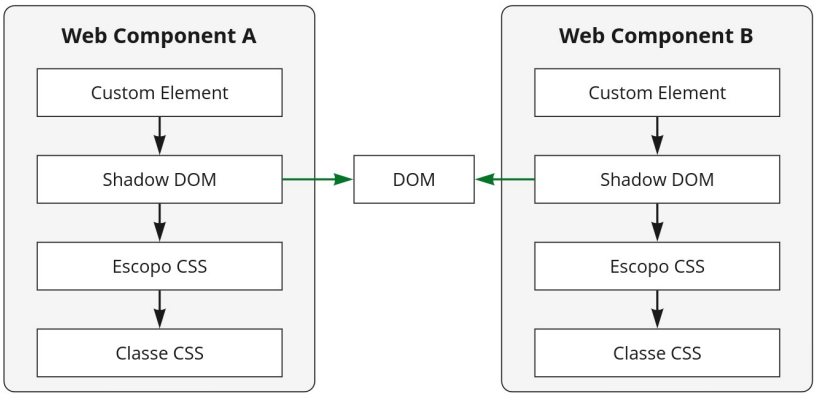
Figura 6 - Gerenciamento de estilos no Single-SPA



Fonte: Elaborado pelo autor.

Na abordagem com Web Components, o mesmo cenário foi replicado. Os *micro-frontends* foram desenvolvidos como componentes encapsulados (Custom Elements) utilizando o Shadow DOM, como demonstra o fluxo da Figura 7.

Figura 7 - Gerenciamento de estilos no Web Components



Fonte: Elaborado pelo autor.

3.6 POC 3 – EVOLUÇÃO E MANUTENÇÃO

A terceira PoC teve como objetivo avaliar a capacidade de evolução e manutenção das arquiteturas, focando na facilidade de adicionar, atu-

alizar e remover componentes de forma independente. O desafio consistiu em simular o ciclo de vida de uma aplicação real, observando o impacto e a complexidade de se realizar alterações em um sistema em produção.

3.6.1 Requisitos do desafio

Para atingir os objetivos propostos, foram definidos os seguintes requisitos:

- A adição de um novo *micro-frontend* deve ser um processo de baixo acoplamento;
- A atualização de um *micro-frontend* existente não deve causar efeitos colaterais nos demais;
- A arquitetura deve permitir o *rollback* de um componente específico com facilidade;
- A complexidade de configuração para integrar novos componentes deve ser mínima.

3.6.2 Apresentação da solução

Na arquitetura Single-SPA, foram desenvolvidos *micro-frontends* Angular independentes, cada um executando em portas distintas com ciclos de vida e dependências próprias. O sistema de registro permitiu a adição de novos módulos de forma declarativa, sem necessidade de alterações nos componentes existentes. A comunicação entre *micro-frontends* foi implementada por meio de eventos do navegador e parâmetros de rota, garantindo baixo acoplamento e possibilitando *deploys* e *rollbacks* individuais.

Na arquitetura com Web Components, foram criados componentes encapsulados utilizando Shadow DOM para isolamento completo de estilos e lógica. A adição do novo componente de detalhes consistiu apenas no registro do elemento customizado e sua inclusão no HTML, sem reconfiguração da aplicação *host*. O encapsulamento garantiu que alterações em um componente não afetassem os demais, facilitando manutenção e reversão de versões específicas.

3.7 CONDUÇÃO DOS TESTES NAS PROVAS DE CONCEITO

A validação comparativa das abordagens Single-SPA e Web Components foi realizada por meio de três Provas de Conceito (PoCs), cada uma focada em um aspecto crítico da arquitetura de *Micro-frontends*: composição e orquestração, isolamento de estilos e encapsulamento, e evolução e manutenção.

Para garantir a relevância e a viabilidade da pesquisa, os testes foram priorizados com base na sua capacidade de gerar dados contrastantes e academicamente significativos, conforme detalhado na Tabela 2.

Tabela 2 - Provas de Conceito e Testes Prioritários

PoC	Testes prioritários	Métrica chave
1	1.1 Teste de carregamento inicial	FCP, LCP, TTI (Lighthouse)
	1.2 Teste de implementação independente	Tempo de integração e complexidade
2	2.1 Análise do Shadow DOM	Encapsulamento de estilos
	2.2 Teste de conflitos de CSS	Número de conflitos (Manual)
3	3.1 Teste de atualização de componente	Impacto nos demais e *deploy* independente
	3.2 Teste de qualidade de código	Manutenibilidade

Fonte: Elaborado pelo autor.

A condução dos testes ocorreu em um ambiente controlado, assegurando condições homogêneas de execução entre as diferentes abordagens. Na PoC 1, o desempenho de carregamento foi avaliado usando a ferramenta Lighthouse para coletar as métricas *First Contentful Paint* (FCP), *Largest Contentful Paint* (LCP) e *Time to Interactive* (TTI). O esforço de implementação foi medido pelo tempo e pelo número de arquivos modificados para adicionar um novo *micro-frontend*.

Na PoC 2, o Chrome DevTools foi utilizado para inspecionar a estrutura do DOM e confirmar a eficácia do Shadow DOM. Em paralelo, um teste de conflito foi realizado injetando-se um estilo propositalmente conflitante na aplicação raiz, seguido da contagem manual de quantos *micro-frontends* foram visualmente afetados.

Por fim, na PoC 3, realizou-se um teste de atualização modificando um *micro-frontend* existente para verificar se a mudança exigia a recompilação de outros módulos. A qualidade do código de cada protótipo foi avaliada com a ferramenta SonarCloud, focando nas métricas de manutenibilidade (classificação A–E) e complexidade ciclomática.

4 RESULTADOS E DISCUSSÃO

Os resultados obtidos nesta pesquisa originaram-se da análise comparativa dos protótipos Single-SPA e Web Components, seguindo-se as métricas e Provas de Conceito (PoCs). O desenvolvimento e a validação dos protótipos permitiram verificar a viabilidade técnica de ambas as abor-

dagens, mas, como será discutido, os experimentos revelaram trade-offs fundamentais em performance, isolamento e manutenibilidade.

A avaliação iniciou-se pela PoC 1 (Composição e orquestração), focada na performance de carregamento e complexidade de integração. O teste de carregamento inicial (Tabela 3) revelou que a abordagem com Web Components obteve um desempenho superior, com um *First Contentful Paint* (FCP) de 1.200 ms, aproximadamente 35% mais rápido que os 1.850 ms do Single-SPA. A interpretação é direta: o pacote principal do Single-SPA, incluindo seu *runtime* e o SystemJS, representou um *overhead* inicial considerável. Estes achados confirmam experimentalmente e, crucialmente, quantificam a tese de Bastos (2020), que identifica o *overhead* de orquestradores como um *trade-off* chave.

Tabela 3 - Resultados do teste de carregamento inicial (Lighthouse)

Métrica	**Single-SPA**	**Web Components**
First Contentful Paint (FCP)	1850 ms	1200 ms
Largest Contentful Paint (LCP)	2900 ms	1950 ms
Time to Interactive (TTI)	3500 ms	2500 ms
Performance Score	78 / 100	92 / 100

Fonte: Elaborado pelo autor.

Complementarmente, o teste de implementação independente (Tabela 4) mediu a complexidade de adicionar um novo *micro-frontend*. O Single-SPA, embora oferecendo uma orquestração robusta, exigiu a modificação de 5 arquivos e apresentou uma complexidade percebida (4 de 5) superior. Em contrapartida, os Web Components, utilizando recursos nativos, simplificaram a integração, exigindo a modificação de apenas 2 arquivos e apresentando complexidade (2 de 5). Esta diferença, refletida nos dados da tabela, indica um ponto de atrito de desenvolvimento significativamente menor para a abordagem nativa na integração de novos módulos.

Tabela 4 - Resultados do teste de implementação independente

Métrica	**Single-SPA**	**Web Components**
Tempo para adicionar novo *micro-frontend*	90 minutos	45 minutos
Arquivos que precisaram ser modificados	5	2
Complexidade (1–5)	4	2

Fonte: Elaborado pelo autor.

A PoC 2 (Isolamento de estilos e encapsulamento) abordou o desafio da coexistência de tecnologias. O protótipo Single-SPA apresentou conflitos de CSS e vazamento de estilos, exigindo convenções adicionais para mitigação. Em contrapartida, os Web Components, através do Shadow DOM, não apresentaram conflitos. Este é um achado significativo, pois valida a tese de Queiroz (2023) sobre a robustez do Shadow DOM como uma fronteira de isolamento nativa e eficaz, reforçando sua garantia de encapsulamento e de preservação da integridade dos componentes.

A PoC 3 (Evolução, escalabilidade e manutenção) demonstrou que ambas as abordagens permitiram o *deploy* independente, validando a premissa fundamental dos *micro-frontends* para a escalabilidade organizacional, apresentada também por Bastos (2020) e Silva (2023). Adicionalmente, o teste de qualidade de código (Tabela 5) forneceu métricas quantitativas de manutenibilidade. Ambas as soluções alcançaram nota ”A” no SonarCloud, alinhando-se aos achados de Silva (2023). No entanto, os Web Components apresentaram uma ligeira vantagem em complexidade ciclomática (8 vs 12) e duplicação de código (1% vs 2%), sugerindo módulos intrinsecamente mais simples.

Tabela 5 - Resultados do teste de qualidade de código (SonarCloud)

Métrica	**Single-SPA**	**Web Components**
Manutenibilidade	95 / 100 (A)	98 / 100 (A)
Complexidade ciclomática	12	8
Duplicação de código	2%	1%

Fonte: Elaborado pelo autor.

Além dos dados quantitativos, a experiência de desenvolvimento revelou nuances: o Single-SPA é facilitado por suas convenções de roteamento, reduzindo a curva de aprendizado inicial para a integração. Em contrapartida, os Web Components exigiram um maior esforço manual na configuração. Um ponto de atenção foi a depuração, que em Web Components com Shadow DOM pode ser mais complexa devido ao isolamento, exigindo ferramentas específicas do navegador.

Os fluxos e testes, portanto, evidenciam as diferenças fundamentais: Web Components são mais performáticos no carregamento e robustos no isolamento, enquanto Single-SPA oferece uma estrutura de orquestração mais madura, mas com maior complexidade de configuração e *overhead* de *runtime*.

A principal limitação deste estudo é a ausência de testes em um

ambiente real de produção, ou seja, em uma aplicação de grande escala com tráfego real e equipes de desenvolvimento distintas. A validação foi restrita a um ambiente controlado de provas de conceito. Isso impossibilitou a análise da aceitação da ferramenta pelos usuários finais, a identificação de barreiras práticas de uso (como a complexidade de *debugging* em um ambiente distribuído) e a coleta de *feedback* sobre a usabilidade da arquitetura em rotinas de trabalho reais.

Outras limitações identificadas incluem a ausência de análise de mecanismos de comunicação entre *micro-frontends* (como Custom Events ou Global State Management), o que ainda exigiria uma implementação ou verificação manual para cenários de alta interdependência. Além disso, os protótipos atuais focaram na comparação de performance e isolamento, mas não implementaram, neste momento, funcionalidades avançadas de *lazy loading* ou Module Federation, que agregariam grande valor na otimização do carregamento.

Ainda assim, os resultados obtidos reforçam o potencial das abordagens de *micro-frontends* como uma solução de apoio à escalabilidade organizacional e técnica. A comparação representa um avanço significativo sobre a mera discussão teórica, oferecendo uma solução centralizada, segura e auditável para a tomada de decisão. Ao prover uma análise quantitativa de *trade-offs* em performance e isolamento, o estudo contribui para a conformidade com as melhores práticas de desenvolvimento *frontend* e, principalmente, fortalece a relação entre a escolha arquitetural e os objetivos de negócio, agregando valor ao processo de desenvolvimento de *software*.

5 CONCLUSÃO

O problema central que motivou esta pesquisa residiu na crescente complexidade do *frontend* de aplicações *web* modernas e na dificuldade de escolher a arquitetura de *software* mais adequada para garantir escalabilidade e manutenibilidade. A solução proposta por este trabalho foi a avaliação comparativa das abordagens de implementação da arquitetura de *micro-frontend* (Single-SPA e Web Components) por meio de provas de conceito. Ao longo do estudo, cumpriu-se integralmente o objetivo geral de avaliar as abordagens, bem como os objetivos específicos de fundamentação teórica, desenvolvimento prático dos protótipos, validação por PoCs e análise dos resultados.

A análise comparativa demonstrou que a escolha entre Single-

SPA e Web Components não se baseou em uma questão de superioridade técnica, mas sim de alinhamento estratégico com o contexto do projeto. O Single-SPA mostrou-se uma solução robusta para a orquestração de múltiplos frameworks e o gerenciamento do ciclo de vida de aplicações heterogêneas.

Em contrapartida, a abordagem com Web Components, potencializada pelo Stencil, destacou-se pela sua aderência aos padrões nativos da *web*, oferecendo um encapsulamento superior e maior interoperabilidade. A discussão crítica revelou que, embora o Single-SPA oferecesse um ecossistema mais maduro para orquestração complexa, ele introduziu uma camada de abstração que pôde aumentar a curva de aprendizado inicial, enquanto os Web Components exigiram maior atenção à comunicação entre componentes e ao gerenciamento de estado global.

Os principais avanços deste trabalho residiram na validação prática e comparativa dessas duas abordagens, gerando um conhecimento que preencheu uma lacuna entre a teoria e a aplicação no mercado. O estudo entregou um material de grande valor para profissionais da área, permitindo-lhes antecipar desafios e escolher a abordagem que melhor se adaptou ao seu contexto técnico e organizacional, contribuindo assim para a construção de sistemas mais escaláveis, manuteníveis e resilientes.

Com base nos conhecimentos adquiridos, bem como nos resultados obtidos, propõe-se para futuros trabalhos a expansão da análise para incluir outras ferramentas de orquestração, como o Webpack Module Federation, a condução de estudos de caso em ambientes de produção de larga escala e a investigação da aplicação de *micro-frontends* em sistemas *mobile*, explorando plataformas nativas ou híbridas.

REFERÊNCIAS

BASTOS, J. P. da S. **Desenvolvimento Web Orientado a Micro Frontends**. Dissertação (Mestrado) — Instituto Superior de Engenharia do Porto, 2020.

ELLIOTT, S. **Proof of concept research**. Philosophy of Science, 2021, v. 88.

GEERS, M. **Micro Frontends in Action**. EUA: Manning Publications Co, 2020.

ISO/IEC/IEEE 42010:2022. **Software, systems and enterprise — Architecture description**. 2022. Disponível em: <https://www.iso.org/standard/79960.html>. Acesso em: 8 jun. 2025.

JACKSON, C. **Micro Frontends**. 2019. Disponível em: <https://martinfowler.com/articles/micro-frontends.html>. Acesso em: 8 mar. 2025.

MEZZALIRA, L. **Building Micro-Frontends: Scaling Teams and Projects, Empowering Developers**. EUA: O'Reilly, 2021.

MONTELIUS, A. **An Exploratory Study of Micro Frontends**. Dissertação (Master's thesis) — Linköping University, Linköping, Sweden, 2021.

NALLAINATHAN, L. **Monólitos para microsserviços usando design controlado por domínio**. EUA: Microsoft Azure Architecture Center, 2024. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/microservices/migrate-monolith>. Acesso em: 03 set. 2024.

PAVLENKO, A. et al. **Micro-frontends: Application of microservices to web front-ends**. Journal of Internet Services and Information Security, 2020.

PELTONEN, S.; MEZZALIRA, L.; TAIBI, D. **Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review**. Information and Software Technology, 2021, v. 136, p. 106571. ISSN 0950-5849.

QUEIROZ, R. C. P. de. **Microfrontends com Web Components e WebPack: Uma abordagem de implementação agnóstica em relação ao framework**. 94 p. Trabalho de Conclusão de Curso — Universidade de Brasília, Brasília, 2023. Il.

RICHARDS, M.; FORD, N. **Fundamentals of Software Architecture: An Engineering Approach**. EUA: O'Reilly, 2020.

SANTOS, I. F. d. **Proposta de uma arquitetura de microfrontend aplicado no Tribunal Superior do Trabalho**. 31 p. Trabalho de Conclusão de Curso — Instituto Metrópole Digital, Universidade Federal do Rio Grande do Norte, Natal, 2024.

SILVA, P. V. d. **Microfrontends e arquitetura limpa: um estudo exploratório orientado a provas de conceito**. 168 p. Trabalho de Conclusão de Curso — Universidade de Brasília, Brasília, 2023. Il.

SOMMERVILLE, I. **Engenharia de Software**. 10ª. ed. Reino Unido: Editora Pearson, 2019. ISBN 9788579361081.