UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

KALMAX DOS SANTOS SOUSA

CONSISTÊNCIA DE UI/UX EM *MICRO FRONTENDS* MULTI-TECNOLOGIA: UM ESTUDO EXPLORATÓRIO COM ÊNFASE EM *DESIGN SYSTEMS* E TÉCNICAS DE INTEGRAÇÃO MODULAR

QUIXADÁ

2026

KALMAX DOS SANTOS SOUSA

CONSISTÊNCIA DE UI/UX EM *MICRO FRONTENDS* MULTI-TECNOLOGIA: UM ESTUDO EXPLORATÓRIO COM ÊNFASE EM *DESIGN SYSTEMS* E TÉCNICAS DE INTEGRAÇÃO MODULAR

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Sidartha Azevedo Lobo de Carvalho.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S697c Sousa, Kalmax dos Santos.
Consistência de UI/UX em micro frontends multi-tecnologia : um estudo exploratório com ênfase em design systems e técnicas de integração modular / Kalmax dos Santos Sousa. – 2026.
79 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2026.
Orientação: Prof. Dr. Sidartha Azevedo Lobo de Carvalho.

1. micro frontends. 2. consistência visual. 3. web components. 4. module federation. I. Título.
CDD 005

KALMAX DOS SANTOS SOUSA

CONSISTÊNCIA DE UI/UX EM *MICRO FRONTENDS* MULTI-TECNOLOGIA: UM ESTUDO EXPLORATÓRIO COM ÊNFASE EM *DESIGN SYSTEMS* E TÉCNICAS DE INTEGRAÇÃO MODULAR

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: 23/01/2026.

BANCA EXAMINADORA

Prof. Dr. Sidartha Azevedo Lobo de Carvalho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Francisco Victor da Silva Pinheiro
Universidade Federal do Ceará (UFC)

Prof. Ma. Leonara de Medeiros Braz
Universidade Federal do Ceará (UFC)

Dedico este trabalho à minha noiva, Isabel. Obrigado por ser meu maior incentivo e a luz que ilumina meus dias, tornando esta conquista possível.

AGRADECIMENTOS

Agradeço primeiramente a Deus, que tem me guiado, protegido e me dado forças diariamente para enfrentar os desafios e não desistir.

À minha noiva, Isabel, minha profunda gratidão. Obrigado por estar comigo em todos os momentos, por ser meu porto seguro e por não medir esforços para me ajudar a concluir este trabalho. Sua presença foi fundamental.

Aos meus pais, pelo amor incondicional e por todo o apoio oferecido durante toda a minha jornada acadêmica. Vocês são a base de tudo o que conquistei.

Ao meu orientador e aos professores da banca, pelo suporte, orientações e pelas valiosas contribuições na avaliação deste trabalho.

À Universidade Federal do Ceará (UFC), especialmente ao Campus de Quixadá, pela excelência no ensino e pelas oportunidades de participação em diversos projetos, que foram essenciais para me tornar um profissional melhor e mais preparado.

Por fim, agradeço aos profissionais da área que aceitaram participar da pesquisa, contribuindo com suas ideias e vivências, enriquecendo grandemente o resultado final deste estudo.

"There is no perfect architecture, only good and bad fits for a context. Everything is a set of trade-offs. Perfection is unavailable, unfortunately." (Luca Mezzalira, 2022, p. 248.)

RESUMO

A adoção da arquitetura de Micro *Frontends* tem se consolidado como uma estratégia para lidar com a crescente complexidade das aplicações web modernas, permitindo maior escalabilidade organizacional e autonomia entre equipes. Entretanto, a fragmentação do *frontend* em múltiplos domínios e tecnologias distintas impõe desafios significativos à manutenção da consistência visual e da experiência do usuário (UX). Nesse contexto, este trabalho apresenta um estudo exploratório que investiga como diferentes decisões arquiteturais de integração em ambientes de Micro *Frontends* multi-tecnologia impactam a consistência da interface do usuário (UI).

A pesquisa analisa, de forma comparativa, duas abordagens amplamente adotadas na indústria: o uso de *Web Components*, com ênfase no encapsulamento nativo de estilos, e o *Module Federation*, focado no compartilhamento dinâmico de código em tempo de execução. Para isso, foram construídos cenários experimentais controlados, nos quais aspectos como isolamento de estilos, ocorrência de conflitos visuais, manutenibilidade da interface e aderência a um *Design System* foram avaliados a partir de métricas técnicas e observações empíricas.

Os resultados indicam que decisões relacionadas ao grau de isolamento ou compartilhamento entre Micro *Frontends* exercem influência direta sobre a resiliência da consistência visual, evidenciando trade-offs entre modularidade, flexibilidade técnica e governança de UI. Como contribuição, este trabalho fornece subsídios para arquitetos de *software* e desenvolvedores *frontend* na escolha de estratégias de integração que favoreçam a sustentabilidade visual e a experiência do usuário em aplicações distribuídas.

Palavras-chave: micro frontends; consistencia visual; web components; module federation.

ABSTRACT

The adoption of the Micro Frontends architecture has emerged as a strategy to address the growing complexity of modern web applications, enabling organizational scalability and greater autonomy among development teams. However, the decomposition of the frontend into multiple domains and heterogeneous technologies introduces significant challenges related to maintaining visual consistency and user experience (UX). In this context, this work presents an exploratory study that investigates how different architectural integration decisions in multi-technology Micro Frontends environments impact user interface (UI) consistency.

This research comparatively analyzes two approaches widely adopted in industry: the use of Web Components, emphasizing native style encapsulation, and Module Federation, focused on dynamic code sharing at runtime. To this end, controlled experimental scenarios were constructed to evaluate aspects such as style isolation, occurrence of visual conflicts, interface maintainability, and adherence to a Design System, based on technical metrics and empirical observations.

The results indicate that decisions regarding the level of isolation or sharing between Micro Frontends directly influence the resilience of visual consistency, revealing trade-offs between modularity, technical flexibility, and UI governance. As a contribution, this study provides insights to software architects and frontend developers in selecting integration strategies that support visual sustainability and user experience in distributed applications.

Keywords: micro frontends; visual consistency; web components; module federation.

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CBSE	Engenharia de Software Baseada em Componentes
CDNs	Redes de Distribuição de Conteúdo, do inglês *Content Delivery Networks*
CSS	*Cascading Style Sheets*
DOM	*Document Object Mode*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer ProtocoL*
IHC	Interação Humano-Computador
JSON	*JavaScript Object Notation*
MFE	*Micro Frontend*
MFEs	*Micro Frontends*
MSL	Mapeamento Sistemático da Literatura
PoC	Prova de Conceito (*Proof of Concept*)
REST	*Representational State Transfer*
SEO	*Search Engine Optimization*
SPA	*Single Page Application*
SSR	*Renderização no Servidor*
UI	Interface de Usuário
UX	Experiência do Usuário
W3C	*World Wide Web Consortium*
XSS	*cross-site scripting*

SUMÁRIO

1 INTRODUÇÃO

O aumento da complexidade das aplicações *web* modernas, impulsionado pela necessidade de rápida evolução, escalabilidade e atuação de múltiplas equipes de desenvolvimento, tem levado à adoção de arquiteturas cada vez mais distribuídas (Taibi; Mezzalira, 2022). Historicamente, no desenvolvimento *backend*, a indústria respondeu aos problemas de escalabilidade dos aplicações monolíticas adotando a arquitetura de microsserviços, que decompõe o sistema em serviços menores, independentes e organizados em torno de domínios de negócio (Lewis; Fowler, 2014).

De forma análoga, no desenvolvimento *frontend*, a crescente complexidade das interfaces de usuário e a necessidade de alinhamento organizacional entre equipes motivaram o surgimento da arquitetura de *Micro Frontends* (MFEs). Essa abordagem propõe a decomposição do *frontend* em unidades menores e semi-independentes, permitindo que diferentes equipes desenvolvam, implantem e mantenham partes específicas da interface de forma autônoma, seguindo princípios semelhantes aos dos microsserviços (Taibi; Mezzalira, 2022; Amorim *et al.*,).

Entretanto, a adoção de MFEs introduz novos desafios, especialmente no que diz respeito à manutenção da consistência visual e da Experiência do Usuário (UX), principalmente quando múltiplas equipes operam em um ambiente multi-tecnologia (Marco; Farias, 2024). A literatura aponta que a heterogeneidade tecnológica é um dos principais fatores que dificultam a manutenção de uma interface coesa, devido a fragmentação da responsabilidade visual entre as diferentes bases de código (Antunes *et al.*,).

A Interface de Usuário (UI) e a UX são atributos percebidos de forma integrada, e qualquer inconsistência visual ou comportamental pode comprometer a percepção de qualidade da aplicação. Estudos sobre estética visual indicam que a consistência de elementos gráficos, cores, tipografia e layout exerce influência direta na avaliação subjetiva de qualidade e usabilidade por parte dos usuários (Lavie; Tractinsky, 2004). Nesse contexto, a ausência de mecanismos arquiteturais que favoreçam a uniformidade visual pode impactar negativamente a UX, mesmo quando os requisitos funcionais são atendidos.

Para mitigar a inconsistência visual, a indústria recorre amplamente ao uso de *Design Systems*, que atuam na centralização de padrões e diretrizes de design (Suarez *et al.*, 2019). No entanto, a aplicação eficaz de um *Design System* em arquiteturas distribuídas depende das decisões de engenharia sobre como os MFEs são integrados. Taibi e Mezzalira (2022) destacam

que as decisões de composição e comunicação definem a robustez da arquitetura. Diante disso, surgem estratégias que promovem o isolamento estrito de estilos, frequentemente implementado via *Web Components* para garantir encapsulamento (Shah, 2023) e o compartilhamento de recursos em tempo de execução, facilitado por tecnologias como *Module Federation* (Mane *et al.*, 2024).

Apesar da existência dessas abordagens, a literatura carece de análises comparativas que avaliem como essas decisões arquiteturais específicas impactam, na prática, a resiliência da consistência visual. Este estudo é motivado pela necessidade de preencher essa lacuna, investigando os *trade-offs* entre isolamento e compartilhamento em cenários controlados de desenvolvimento, buscando beneficiar diretamente arquitetos de *software*, desenvolvedores *frontend* e pesquisadores da área de engenharia de *software* com interesses em estratégias de integração modular.;

1.1 Objetivos

1.1.1 Objetivo geral

O objetivo geral deste trabalho é analisar o impacto de diferentes decisões arquiteturais de integração em arquiteturas de Micro Frontends (MFEs) heterogêneos sobre a consistência visual da Interface do Usuário (UI), considerando estratégias de isolamento de estilos, compartilhamento de recursos e governança de UI.

1.1.2 Objetivos específicos

- Investigar como diferentes estratégias arquiteturais de integração em MFEs influenciam a ocorrência de conflitos de estilo e a manutenção de uma identidade visual coerente ao longo da aplicação.
- Analisar como o grau de isolamento e de compartilhamento entre MFEs afeta a manutenibilidade da interface e a propagação de mudanças visuais em ambientes *frontend* heterogêneos.
- Avaliar os *trade-offs* dessas decisões arquiteturais sob a perspectiva da integridade da interface e da segurança estrutural, identificando qual abordagem favorece a sustentabilidade visual em ambientes distribuídos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos que fundamentam esta pesquisa, abordando a arquitetura de MFEs, os princípios da consistência visual de interfaces, a importância dos *design systems* em ambientes distribuídos, e as técnicas de integração modular, com ênfase em *Web Components* e *Module Federation*.

2.1 *Micro Frontends*: definição, motivação e desafios

A arquitetura de MFEs vem ganhando destaque como uma evolução natural dos microsserviços para o desenvolvimento de interfaces. Nessa abordagem, a aplicação *frontend* é fragmentada em unidades menores, autônomas e coesas que podem ser desenvolvidas e implantadas de forma independente por equipes distintas, possibilitando maior escalabilidade organizacional e flexibilidade tecnológica (Taibi; Mezzalira, 2022; Mezzalira, 2021).

De acordo com Amorim *et al.* (), MFEs permitem acelerar o desenvolvimento e a entrega de funcionalidades ao descentralizar responsabilidades e promover a especialização técnica. Essa fragmentação também permite que diferentes domínios da aplicação evoluam em ritmos distintos, sem causar bloqueios entre as equipes, aumentando a produtividade e diminuindo o tempo de *deploy*.

Contudo, essa flexibilidade introduz novos desafios. A utilização de diferentes bibliotecas, *frameworks* e práticas de codificação pode acarretar uma experiência de usuário fragmentada e inconsistências visuais entre os domínios. Marco e Farias (2024), ao realizarem um mapeamento sistemático sobre MFEs, destacam que a heterogeneidade tecnológica é um dos principais fatores que dificultam a manutenção de uma interface coesa, principalmente quando não há diretrizes centralizadas, como um *design system* bem definido.

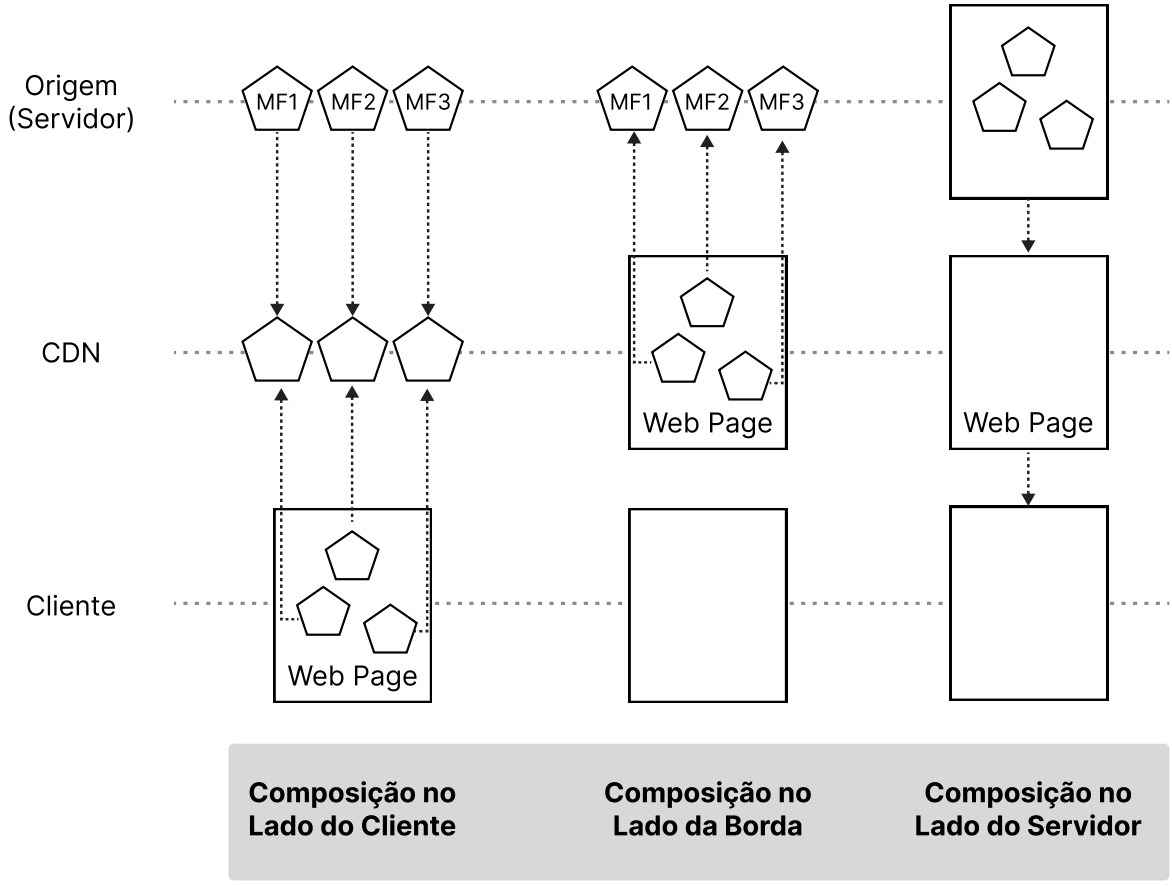
Taibi e Mezzalira (2022) propuseram um *framework* de decisão que deve ser utilizado nas fases iniciais da adoção de MFEs, baseado em quatro eixos: divisão, composição, roteamento e comunicação. Esses eixos guiam as escolhas arquiteturais fundamentais da aplicação e afetam diretamente sua coesão visual e desempenho. As principais formas de composição dos MFEs são apresentadas na Figura 1 e podem ser resumidas como:

– **Client-side composition:** Os MFEs são carregados diretamente no navegador do usuário, geralmente via *Web Components*, *iframes* ou *Module Federation*. Essa abordagem oferece alta independência e escalabilidade, mas exige estratégias rigorosas para manter a

padronização visual (Mezzalira, 2021).

– **Server-side composition:** A agregação das interfaces ocorre no *backend*, permitindo uma melhor integração com sistemas de *cache* e *Search Engine Optimization* (SEO). No entanto, pode reduzir a autonomia técnica entre as equipes (Taibi; Mezzalira, 2022).
– **Edge-side composition:** Os MFEs são compostos em servidores de borda, como Redes de Distribuição de Conteúdo, do inglês *Content Delivery Networks* (CDNs), otimizando a latência e o desempenho, mas requerendo infraestrutura mais complexa (Marco; Farias, 2024).

Figura 1 – Composição de *Micro Frontends* no lado do cliente, da borda e do servidor.



Fonte: Adaptado de Taibi e Mezzalira (2022)

É importante ressaltar que a governança visual se torna crítica nesse cenário. Conforme argumentado por Taibi e Mezzalira (2022), a ausência de um sistema de *design* unificado pode resultar em interfaces fragmentadas, experiências divergentes entre seções da aplicação e dificuldades na manutenção de componentes compartilhados.

Amorim *et al.* () reforçam que o sucesso da adoção de MFEs depende de fatores não

apenas técnicos, mas também organizacionais, como a cultura de colaboração, a definição clara de contratos entre domínios e a aplicação rigorosa de práticas de engenharia de *software* voltadas à reutilização e padronização.

A presente pesquisa parte da constatação de que, embora os benefícios da arquitetura sejam amplamente reconhecidos, ainda há uma lacuna na investigação sobre como manter a consistência visual entre MFEs heterogêneos, especialmente em ambientes que utilizam tecnologias distintas e equipes separadas. A avaliação comparativa entre abordagens técnicas, como W*eb Components* e *Module Federation*, é uma contribuição relevante para preencher essa lacuna, permitindo entender qual delas favorece a manutenção da consistência visual de forma mais efetiva.

2.2 Consistência visual e métricas de avaliação

A consistência visual é um dos princípios fundamentais do *design* de interfaces e da UX. Trata-se da capacidade de manter padrões visuais e interativos uniformes ao longo de diferentes partes de uma aplicação, contribuindo para a previsibilidade, legibilidade e compreensão da interface. A ausência de consistência pode gerar confusão, aumentar a carga cognitiva do usuário e prejudicar a credibilidade do sistema (Moshagen; Thielsch, 2010).

2.2.1 Princípios de consistência e complexidade visual

De acordo com Nielsen (1994), a consistência é uma das dez heurísticas de usabilidade e pode manifestar-se em diferentes níveis, como:

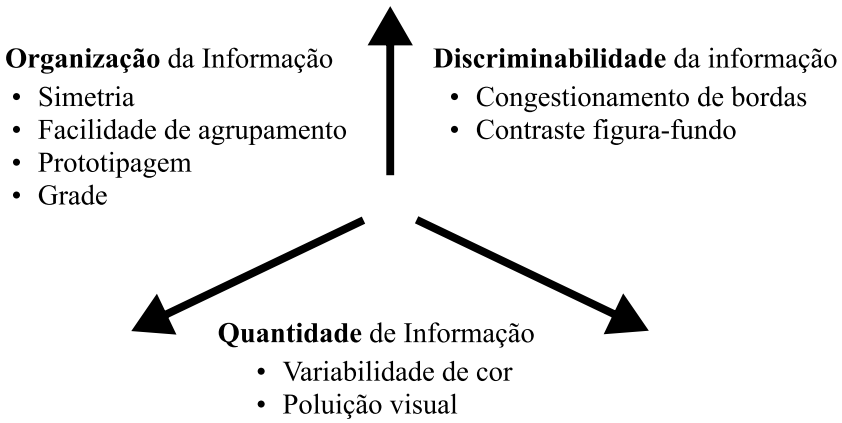
- **Consistência interna:** uniformidade dentro da própria aplicação.
- **Consistência externa:** conformidade com convenções de mercado e padrões da plataforma.
- **Consistência visual:** padronização de cores, tipografias, ícones e espaçamentos.
- **Consistência funcional:** previsibilidade de comportamentos e respostas do sistema.

Estudos em psicologia e Interação Humano-Computador (IHC) estabeleceram uma forte relação entre a complexidade visual e a estética de uma interface, onde estímulos visualmente mais simples tendem a ser percebidos como mais belos, devido a uma maior fluidez no processamento mental. Nesse sentido, a busca pela consistência é também uma busca pela simplicidade e pela redução da complexidade (Miniukovich; Angeli, 2014).

No contexto de aplicações monolíticas, a manutenção desses padrões é facilitada pelo controle centralizado do *frontend*. Contudo, em arquiteturas distribuídas como os MFEs, a responsabilidade pela construção visual é fragmentada entre diferentes equipes e tecnologias, o que tende a gerar desvios visuais e interativos se não houver governança adequada (Antunes *et al.*, ; Marco; Farias, 2024).

Miniukovich e Angeli (2014) propõem um modelo que classifica os determinantes da complexidade visual em três dimensões principais, conforme ilustrado na Figura 2:

Figura 2 – Classificação dos determinantes da complexidade visual



Fonte: Adaptado de Miniukovich e Angeli (2014)

- **Quantidade de Informação:** Refere-se à variação de elementos na cena, como o excesso de itens e a variabilidade de cores. Uma interface consistente gerencia essa dimensão ao limitar o número de elementos visuais e padronizar a paleta de cores, geralmente por meio de *tokens* de *design*.
- **Organização da Informação:** Diz respeito a como os elementos são estruturados. Fatores como simetria, alinhamento a um *grid*, facilidade de agrupamento e prototipicidade são cruciais. Interfaces consistentes se destacam nesta dimensão ao empregar *layouts* regulares e componentes reutilizáveis.
- **Discriminabilidade da Informação:** Relaciona-se à facilidade de distinguir os elementos. Fatores como o contraste entre figura e fundo e o congestionamento de bordas afetam diretamente a capacidade do usuário de processar a informação visualmente.

Além disso, a literatura sugere que inconsistências não se limitam à aparência, mas também afetam a semântica da interação. Por exemplo, um botão com o mesmo propósito pode apresentar variações de cor, ícone ou posição em domínios distintos, dificultando o reconheci-

mento por parte do usuário. Segundo Law *et al.* (2009), a inconsistência é uma das causas mais frequentes de frustração e abandono de tarefas em sistemas digitais.

2.2.2 Métricas para avaliação da integridade visual

Em arquiteturas distribuídas, a avaliação da consistência transcende a análise subjetiva e exige métricas objetivas de conformidade técnica. A literatura aponta que a validação de interfaces em sistemas de componentes deve focar na paridade de renderização e no isolamento de estilos.

A abordagem quantitativa foca na fidelidade da implementação em relação às especificações de design *(Pixel Perfect)*. Segundo Miniukovich e Angeli (2014), métricas automáticas baseadas no produto final (renderização) são eficazes para detectar desvios de complexidade e simetria. No contexto de engenharia de *software frontend*, isso se traduz na verificação de propriedades computadas (*Computed Styles*) pelo navegador.

Para garantir a robustez da arquitetura, utilizam-se duas categorias principais de verificação:

- **Conformidade de estilo:** Verificação se os valores de tokens de design (cores, tipografia, espaçamentos) aplicados no DOM correspondem exatamente aos definidos no *Design System*. Ferramentas como *Stylelint* e inspeção via *DevTools* são comuns para essa validação.
- **Resiliência ao vazamento:** Capacidade do componente de manter sua integridade visual mesmo quando submetido a conflitos de CSS global, testando a eficácia do encapsulamento.

2.2.3 Design System como solução para a inconsistência

Para resolver os desafios de inconsistência em escala, as equipes utilizam um *design system*. Segundo Kholmatova (2017), um *design system* é um conjunto de padrões conectados e práticas compartilhadas, organizados coerentemente para servir aos propósitos de um produto digital. Ele geralmente contém uma coleção de componentes reutilizáveis, guiados por padrões claros, que une as equipes de produto em torno de uma linguagem visual comum. Essa coleção geralmente é composta por:

- **Padrões:** Representam a linguagem visual, como cores e tipografia, muitas vezes gerenciada através de *tokens* de *design*.

– **Componentes:** Blocos de construção da UI, como botões e formulários, que consomem esses padrões.

Ao tornar o *design* reutilizável, essa abordagem permite que as *equipes* construam produtos melhores e mais rapidamente, reduzindo a dívida de design e garantindo uma UX mais coesa e previsível (Suarez *et al.*, 2019).

2.3 Infraestrutura de interface e padronização

A transição para arquiteturas distribuídas exige uma "infraestrutura visual"compartilhada para evitar a fragmentação da interface. Em ambientes multi-tecnologia, a eficácia desse sistema depende de sua capacidade de abstração e organização, garantindo que as equipes consumam os mesmos padrões visuais sem duplicar esforços.

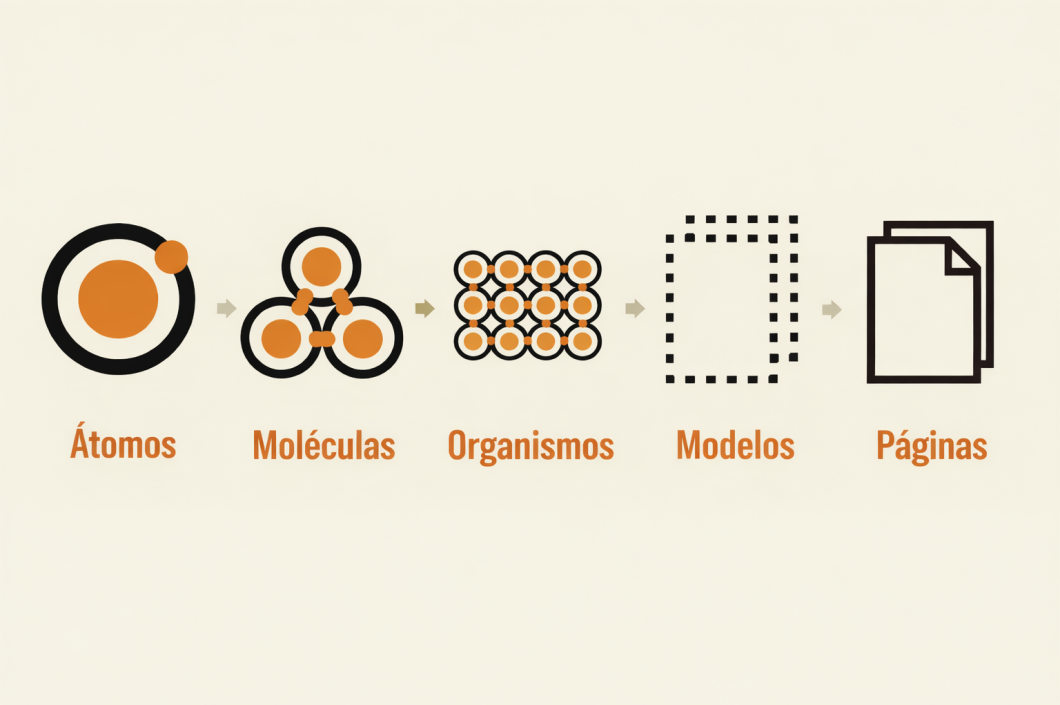
2.3.1 Metodologia Atomic Design

Para operacionalizar a construção de interfaces modulares, a indústria adota metodologias que favorecem a composição e a reusabilidade. Uma das abordagens mais consolidada nesse contexto é o *Atomic Design*, proposta por Frost (2016). O autor argumenta que as páginas *web* modernas não devem ser projetadas como telas estáticas isoladas, mas sim como sistemas hierárquicos de componentes vivos.

Frost (2016) estabelece uma analogia direta com a química básica para criar um modelo mental que permite aos desenvolvedores e *designers* alternarem entre a visão abstrata (o componente isolado) e a visão concreta (o componente no contexto). A metodologia organiza a interface em cinco estágios distintos de complexidade crescente, conforme demonstrado visualmente na Figura 3 e listados a seguir.

– **Átomos:** São os blocos de construção fundamentais e indivisíveis da interface. No contexto de *software*, correspondem às tags *HyperText Markup Language* (HTML) nativas (como `<label>`, `<input>` ou `<button>`), bem como elementos de estilo abstratos, como paletas de cores, tipografia e animações. Segundo Frost (2016), os átomos, isoladamente, possuem pouca utilidade funcional, mas servem como a referência global de estilo da aplicação.
– **Moléculas:** Constituem grupos de átomos unidos para executar uma função simples e indivisível. A combinação de um átomo de rótulo, um campo de entrada e um botão, por exemplo, forma uma Molécula de Busca. Este estágio representa a menor unidade de

Figura 3 – Os cinco estágios da metodologia Atomic Design.



Fonte: Adaptado de Frost (2016).

"funcionalidade"que pode ser testada e reutilizada no contexto de engenharia de *software*.

- **Organismos:** São componentes de interface complexos, formados pela combinação de moléculas, átomos e até outros organismos. Um cabeçalho de uma aplicação de *e-commerce*, por exemplo, é um organismo que pode conter uma molécula de busca, uma molécula de navegação e um átomo de logotipo. Diferente das moléculas, os organismos já definem seções distintas e reconhecíveis da interface.
- **Modelos:** Neste estágio, a analogia química é abandonada em favor de uma terminologia mais adequada para a arquitetura de informação. Os modelos (*templates*) articulam os organismos em um *layout* de página, focando na estrutura de conteúdo subjacente e não no conteúdo final. Eles definem o esqueleto da página, garantindo que os componentes funcionem em harmonia.
- **Páginas:** Representam instâncias específicas dos modelos preenchidas com conteúdo real. É neste estágio que a eficácia do sistema é validada, permitindo testar a resiliência dos padrões de design frente a variações de dados reais.

A adoção do *Atomic Design* em arquiteturas distribuídas, como Micro Frontends (MFEs), oferece uma vantagem arquitetural significativa com a granularidade de governança. Ao definir claramente o que é um átomo ou uma molécula, as equipes podem estabelecer contratos de interface mais claros entre os MFEs, garantindo que alterações em um átomo básico se

propaguem previsivelmente por toda a cadeia de dependência, até as páginas finais (Frost, 2016).

2.3.2 Design Tokens

Enquanto a taxonomia do *Atomic Design* organiza a estrutura dos componentes, a consistência visual em ambientes multi-plataforma exige uma estratégia robusta para a distribuição de valores de estilo. Vesselov e Davis (2019) definem *Design Tokens* como entidades nomeadas que armazenam atributos de interface do usuário. Na prática, isso significa definir cores, fontes e espaçamentos em um único ponto, garantindo sua aplicação consistente em diferentes contextos.

Diferente de variáveis tradicionais de CSS ou pré-processadores que vivem apenas no código *frontend*, os *tokens* atuam como uma camada de abstração de dados. Segundo Vesselov e Davis (2019), eles são frequentemente armazenados em formatos de dados agnósticos, como *JavaScript Object Notation* (JSON) ou YAML. Essa característica permite que os valores visuais sejam compartilhados e transformados para "quase qualquer formato", seja para objetos *JavaScript*, variáveis para a *Web*, ou arquivos nativos.

Ao centralizar as decisões de *design* em *tokens*, elimina-se a necessidade dE inserção manual de valores em múltiplos repositórios. Isso garante que uma atualização em um valor de cor na fonte se propague para todas as tecnologias consumidoras, facilitando a implementação de temas e a manutenção da consistência em escala (Vesselov; Davis, 2019).

2.4 Técnicas de integração para *Micro Frontends*

A escolha da estratégia de integração em MFEs define não apenas o modelo de *deploy*, mas também o comportamento do CSS e a robustez da consistência visual. A literatura descreve diferentes estratégias de isolamento, dentre elas o encapsulamento nativo via padrões *web* e a composição em tempo de execução via *bundlers*. Nesse contexto, os *Web Components*, enquanto *Application Programming Interface* (API) nativa do navegador, e o *Module Federation*, recurso do *bundler Webpack*, surgem como abordagens distintas para a composição e o isolamento de MFEs.

2.4.1 Web Components

Web Components são um conjunto de tecnologias padronizadas pelo *World Wide Web Consortium* (W3C) que permitem a criação de elementos HTML personalizados com encapsulamento de estilo e comportamento, por meio de *Custom Elements*, *Shadow* DOM e HTML *Templates* (MDN Contributors, 2024). Essa abordagem permite que componentes visuais sejam definidos de forma independente do restante da aplicação, mantendo-se agnósticos a *frameworks* e a outras tecnologias, uma vez que seus estilos e comportamentos permanecem isolados do escopo global do documento. Em particular, o encapsulamento promovido pelo *Shadow DOM* reduz a ocorrência de conflitos de estilo entre módulos distintos, favorecendo o isolamento visual e contribuindo para a manutenção da consistência da interface em arquiteturas baseadas em MFEs.

Um estudo de Shah (2023) baseia-se na capacidade dos *Web Components* de promover modularidade, reusabilidade e escalabilidade, investigando o uso de *customized built-in elements*: uma variação que estende elementos HTML nativos. Ele afirma que esses componentes são essenciais para a implementação de *design systems* eficazes. Isso acontece porque ao criar componentes de UI que encapsulam a identidade visual e o comportamento definidos pelo *design system*, as equipes podem garantir uma experiência de usuário consistente e coesa em diversas plataformas digitais.

Web Components oferecem vários benefícios para MFEs: promovem isolamento visual, evitam poluição de estilos globais e facilitam a reutilização de componentes heterogêneos. Porém, desafios permanecem na orientação de comunicação e coordenação entre domínios e na integração com *frameworks* modernos exigindo *layouts* e conceitos de *design system* bem definidos (Shah, 2023).

2.4.2 Module Federation

O *Module Federation*, introduzido no *Webpack 5*, é uma tecnologia que permite que diferentes aplicações (ou *builds*) compartilhem módulos *JavaScript* dinamicamente em tempo de execução, sem a necessidade de *rebuilds* completos (Webpack Contributors, 2024). Esta tecnologia permite que uma aplicação "hospedeira" importe código JavaScript de uma aplicação "remota" dinamicamente, como se fosse um módulo local.

A principal vantagem, conforme destacado por Mane *et al.* (2024), é eficiência

de desempenho proporcionada pelo compartilhamento de instâncias de bibliotecas e pelo *lazy loading*, que evitam *downloads* duplicados e reduzem tanto o tamanho do *bundle* quanto o tempo de carregamento, demonstrando que o desacoplamento das equipas resulta em ciclos de entrega significativamente mais ágeis e frequentes.

Contudo, os autores também destacam desafios, como a necessidade de estabelecer canais de comunicação claros entre as equipes, gerenciar a complexidade de múltiplas tecnologias e garantir que os diferentes componentes se integrem de forma a oferecer uma experiência de usuário coesa. Antunes *et al.* () alertam que, embora o *Module Federation* resolva o problema da distribuição de código *JavaScript*, ele exacerba a complexidade da governança de estilos em equipas distribuídas.

2.4.3 Análise comparativa das abordagens

Considerando os desafios de integração em arquiteturas baseadas em MFEs, é essencial compreender as principais diferenças entre as abordagens *Web Components* e *Module Federation*. Ambas oferecem soluções para a modularização da interface, porém com impactos distintos sobre aspectos como isolamento visual, compartilhamento de dependências e governança de estilos. A seguir, apresenta-se uma síntese comparativa entre essas duas técnicas, destacando suas principais características técnicas e implicações para a consistência visual da interface.

Quadro 1 – Comparação entre *Web Components* e *Module Federation* em contextos de *Micro Frontends*

Critérios	***Web Components***	*Module Federation*
Isolamento de estilo	Nativo via *Shadow* DOM	Requer estratégias manuais, como CSS *Modules*
Independência tecnológica	Alta (agnóstico a *frameworks*)	Limitada ao ecossistema *JavaScript* com *Webpack*
Compartilhamento de dependências	Baixo: componentes encapsulados	Alto: permite compartilhamento dinâmico entre domínios
Facilidade de configuração	Relativamente simples; usa API do navegador	Complexa: exige configuração detalhada do *Webpack*
Reuso e interoperabilidade	Alto: componentes reutilizáveis em múltiplos projetos	Alto: módulos podem ser consumidos em tempo de execução
Escalabilidade organizacional	Adequado para UI isolada entre times	Adequado para aplicações com código compartilhado e *deploy* independente
Riscos para a consistência visual	Necessita padronização via *tokens*	Requer isolamento de estilos entre domínios federados

Fonte: Elaborado pelo autor

Como se observa no Quadro 1, os *Web Components* se destacam pelo encapsu-

lamento nativo de estilos e pela independência de *frameworks*, tornando-se particularmente adequados em cenários com forte foco na consistência visual e modularidade da interface. Já o *Module Federation* oferece vantagens em termos de compartilhamento dinâmico de código e escalabilidade técnica, sendo mais apropriado para aplicações que requerem integração funcional entre domínios e equipes. A escolha entre essas abordagens, portanto, deve levar em conta não apenas aspectos de arquitetura e manutenção, mas também os objetivos de experiência do usuário e padronização visual em sistemas compostos.

2.5 O problema do escopo global e colisões de estilo

A principal causa técnica de inconsistências visuais em arquiteturas compostas por múltiplos MFEs reside na natureza da linguagem CSS. Conforme definido pela especificação de Etemad e Suzanne (2025), o modelo padrão de estilização na *web* opera em um escopo global. Isso significa que, em um documento HTML tradicional, qualquer regra de estilo declarada é, por padrão, acessível e aplicável a qualquer elemento da árvore DOM que corresponda ao seu seletor.

Em sistemas monolíticos, esse comportamento é gerenciável através de convenções de nomenclatura. Porém, em arquiteturas distribuídas onde múltiplas equipes injetam código na mesma página, o escopo global cria o risco de colisão de estilos. Meyer e Weyl (2017) explicam que essas colisões ocorrem quando duas regras distintas competem para estilizar o mesmo elemento. O navegador resolve esse conflito através do algoritmo de cascata e especificidade, onde a regra com seletor mais específico ou declarada por último sobrescreve a anterior.

Esse mecanismo dá origem ao fenômeno de *CSS Bleeding* (Vazamento de Estilo), uma das falhas mais críticas em integrações via *Module Federation*. O vazamento ocorre em duas direções:

- **Vazamento externo (*outbound*):** Quando o CSS genérico de um micro *frontend* afeta indevidamente componentes de outros MFEs ou da aplicação hospedeira.
- **Vazamento interno (*inbound*):** Quando estilos globais da aplicação hospedeira penetram no micro frontend, alterando sua aparência original e desrespeitando o *Design System*.

A validação da robustez arquitetural de um micro *frontend*, portanto, depende da capacidade da técnica de integração de criar um novo contexto de empilhamento ou isolar o escopo do DOM, impedindo que a cascata global interfira na integridade visual do componente (Etemad; Suzanne, 2025).

3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados trabalhos relevantes que fundamentam a presente pesquisa. Foram selecionados estudos que abordam a arquitetura de micro-frontends sob diferentes perspectivas: seus princípios e implementações, os benefícios e limitações de sua adoção em um contexto industrial, e os resultados de sua aplicação com foco em modularidade.

3.1 *Micro-Frontends: Principles, Implementations, and Pitfalls*

O trabalho de Taibi e Mezzalira (2022) apresenta uma visão geral da arquitetura de MFEs, descrevendo seus princípios fundamentais, diferentes abordagens de implementação e os desafios (ou "armadilhas") associados. Os autores argumentam que, embora a abordagem MFEs ofereça vantagens ao decompor o frontend em microaplicações semi-independentes, ela também introduz desvantagens, como o risco de sobrecarga de comunicação e uma experiência de usuário inconsistente se a governança por trás de um *design system* não for bem planejada.

O artigo propõe um *framework* de decisão baseado em quatro escolhas técnicas essenciais que devem ser feitas no início de um projeto de MFEs:

- **Divisão (*Split*):** Horizontal (múltiplos MFEs na mesma visão) ou Vertical (um MFEs por vez).
- **Composição:** Onde a aplicação será montada — no lado do cliente (*client-side*), no lado do servidor (*server-side*) ou na borda (*edge-side*);
- **Roteamento:** Como a navegação entre as visões será gerenciada, dependendo da técnica de composição escolhida.
- **Comunicação:** Como os MFEs trocarão dados entre si, utilizando desde *Web Storage* e *query strings* até padrões de *event emitter*.

Os autores descrevem várias tecnologias de implementação para composição no lado do cliente, incluindo *iFrames*, *Web Components* e *Module Federation*. Eles destacam *Web Components* como um padrão consolidado para criar componentes agnósticos a *frameworks*, permitindo ocultar detalhes de implementação por meio de padrões *web*. Já o *Module Federation*, disponível no *Webpack 5*, é apresentado como uma solução que gerencia dependências de bibliotecas e permite o compartilhamento bidirecional entre múltiplos MFEs, simplificando a composição e focando em performance através de *lazy loading*.

Embora o trabalho de Taibi e Mezzalira (2022) seja essencial para a catalogação

teórica desses riscos, ele possui um caráter classificatório e de relato de experiência. O estudo não apresenta experimentos controlados que comparem a eficácia de diferentes técnicas de integração para mitigar esses problemas específicos. Esta é a lacuna que a presente pesquisa preenche: partindo do risco identificado pelos autores, este trabalho executa testes de estresse práticos para entender como diferentes decisões arquiteturais reagem às falhas de consistência listadas na literatura.

3.2 *Investigating Benefits and Limitations of Migrating to a Micro-Frontends Architecture*

Antunes *et al.* () investigam os benefícios e as limitações da migração de uma aplicação monolítica do mundo real para uma arquitetura de MFEs, sob a perspectiva dos desenvolvedores. A pesquisa foi conduzida utilizando a abordagem de *Action Research*, que incluiu o diagnóstico de problemas de colaboração entre equipes, o planejamento e a execução da migração, e uma avaliação por meio de um *workshop* com os desenvolvedores.

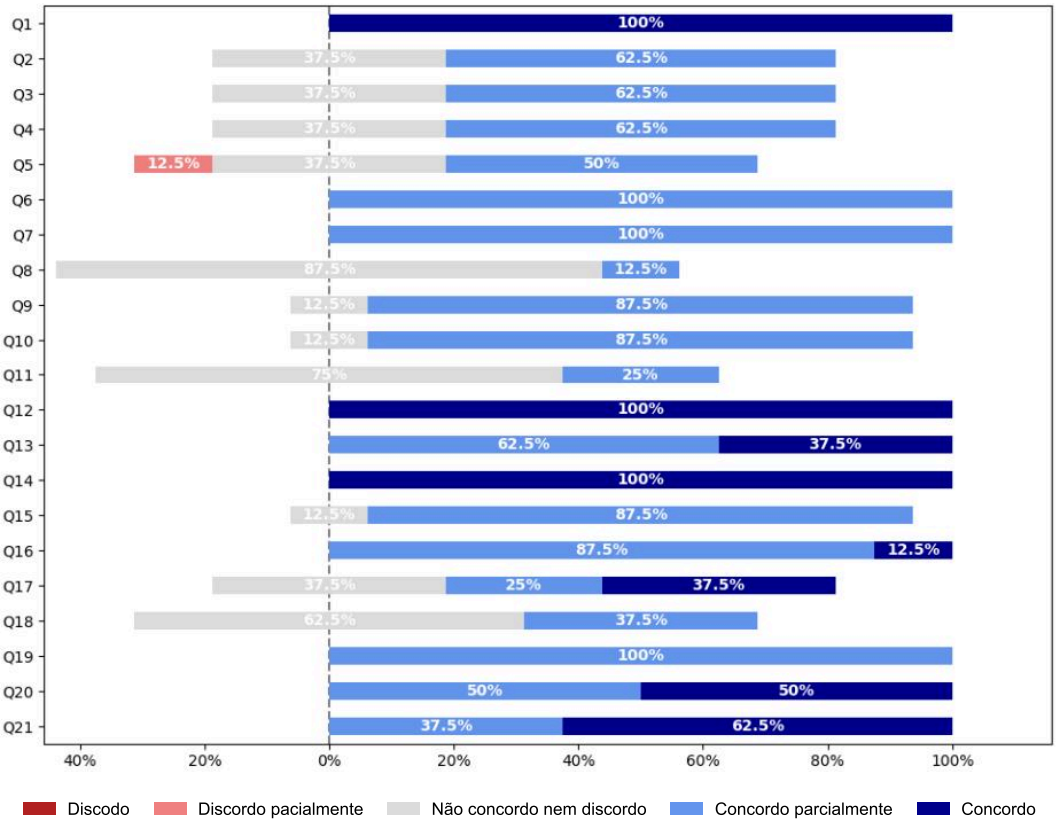
Durante a fase de intervenção, a aplicação monolítica existente foi reestruturada em componentes menores e autônomos. A composição foi realizada no lado do cliente (*client-side*) utilizando *Module Federation*, uma tecnologia integrada ao *bundler* da aplicação (*Webpack*). A escolha foi justificada pela facilidade em definir bibliotecas compartilhadas, carregando-as apenas uma vez, e pelo aproveitamento de uma biblioteca *Angular* existente que já utiliza *Module Federation*.

O estudo focou na perspectiva da Engenharia de Software, avaliando não apenas métricas técnicas, mas a percepção da equipe de desenvolvimento. Após a intervenção técnica, os autores realizaram uma validação qualitativa por meio de grupos focais (*focus groups*), onde os desenvolvedores debateram os ganhos e as dificuldades encontradas no novo ambiente distribuído. Os resultados dessa avaliação, consolidados na Figura 4, demonstram que, embora a autonomia das equipes (Q2) tenha sido bem avaliada, surgiram barreiras operacionais significativas.

Especificamente, os participantes relataram dificuldades críticas relacionadas à manutenção da consistência de UI/UX (referenciada como Q15 nos resultados) e à necessidade de maior coordenação entre as equipes (Q18). O artigo conclui que a fragmentação do código, sem ferramentas de suporte adequadas, exige um esforço cognitivo elevado para evitar divergências visuais entre os micro frontends, sugerindo a adoção de processos de alinhamento e documentação como medidas mitigatórias.

Esse artigo reconhece explicitamente a dificuldade em manter uma experiência de

Figura 4 – Percepção dos desenvolvedores sobre os impactos da migração.



Fonte: Adaptado de Antunes *et al.* ().
Nota: As barras inferiores (em tons mais escuros/vermelhos) indicam os pontos onde os desenvolvedores perceberam maiores limitações, com destaque para a Consistência de UI (Q15) e Coordenação (Q18).

usuário uniforme em contextos multi-tecnologia. Além disso, ele propõe como solução o uso de bibliotecas compartilhadas de UI, embora também discuta os riscos relacionados a conflitos e acoplamento entre equipes.

3.3 *Micro Frontend Architecture With Webpack Module Federation: Enhancing Modularity Focusing On Results And Their Implications*

O trabalho de Mane *et al.* (2024) explora a implementação da arquitetura de MFEs utilizando especificamente *Module Federation* para melhorar a modularidade, escalabilidade e colaboração entre equipes. A pesquisa baseia-se em uma análise de múltiplos estudos de caso para destacar melhorias na velocidade de desenvolvimento e na performance da aplicação.

Os autores reportam resultados quantitativos significativos, como uma redução de 30% nos tempos de implantação e uma diminuição de aproximadamente 25% nos tempos de carregamento inicial em aplicações que utilizam *Module Federation* em comparação com

arquiteturas monolíticas. Esses ganhos são atribuídos ao carregamento dinâmico dos MFEs, que otimiza a utilização de recursos. A Tabela 1 resume a análise de métricas de performance antes e depois da implementação.

Tabela 1 – Resultados de performance e agilidade com *Module Federation*

Métrica Analisada	**Monólito (Antes)**	**MFE (Depois)**	**Melhoria**
Tempo Médio de Carregamento	3.5s	2.5s	**28.6%**
Tamanho Médio do *Bundle*	5.0 MB	3.5 MB	**30.0%**
Tempo de Resposta da API	500 ms	350 ms	**30.0%**
Tempo de Implantação (*Deploy*)	-	-	**30.0% (Redução)**
Frequência de *Deploy*	-	-	**60.0% (Aumento)**

Fonte: Adaptado de Mane *et al.* (2024).

Este estudo enfatiza que, apesar dos benefícios, desafios persistem, como a necessidade de estabelecer canais de comunicação claros entre as equipes, gerenciar a complexidade de múltiplas tecnologias e garantir que os componentes se integrem para oferecer uma experiência de usuário coesa. O artigo argumenta que a estrutura organizacional e a cultura de colaboração desempenham um papel crucial no sucesso da implementação de MFEs. A principal contribuição deste trabalho é a quantificação dos benefícios de performance e agilidade obtidos com o uso de *Module Federation*, reforçando seu papel como uma ferramenta facilitadora para a arquitetura de MFEs.

3.4 *Harnessing Customized Built-In Elements for Component-based Software Engineering* (Shah, 2023)

Em uma abordagem focada na padronização de interfaces, Shah (2023) investiga o papel dos *Web Components* como tecnologia habilitadora para a (Engenharia de Software Baseada em Componentes (CBSE)). O autor explora especificamente o conceito de "Elementos Nativos Customizados", defendendo que a extensão de tags HTML nativas (como estender a funcionalidade de um `<button>` ou `<table>`) oferece o equilíbrio ideal entre acessibilidade herdada e modularidade moderna.

A análise de Shah (2023) destaca que a principal vantagem desta tecnologia para grandes sistemas corporativos é o encapsulamento rigoroso. Ao utilizar as APIs nativas do navegador (*Custom Elements* e *Shadow DOM*), os desenvolvedores conseguem criar bibliotecas de componentes agnósticas ao framework. O estudo argumenta que essa característica é fun-

damental para a implementação de *Design Systems* resilientes, pois garante que as regras de negócio e de estilo (identidade visual) permaneçam isoladas dentro do componente, protegidas contra interferências externas do restante da aplicação.

Além dos benefícios de encapsulamento, o autor aborda os desafios de segurança e compatibilidade. Shah (2023) alerta para a necessidade de estratégias defensivas contra vulnerabilidades como *cross-site scripting* (XSS) (*Cross-Site Scripting*) dentro dos componentes e discute a importância de *polyfills* para garantir o funcionamento em navegadores legados. A conclusão do trabalho posiciona os *Web Components* não apenas como uma ferramenta de UI, mas como um padrão arquitetural para garantir a longevidade e a manutenibilidade de aplicações complexas.

3.5 Comparativo

Os trabalhos analisados fornecem uma base sólida sobre os princípios, desafios e benefícios da arquitetura de MFEs. Taibi e Mezzalira (2022) oferecem um guia conceitual sobre as decisões arquiteturais, descrevendo diversas formas de integração. Antunes *et al.* () validam empiricamente, através de um estudo de caso industrial, muitos desses desafios, apontando a consistência de UI/UX como uma preocupação relevante e o uso de *Module Federation* como uma solução de integração viável. Enquanto isso, Mane *et al.* (2024) focam nos ganhos de performance e modularidade proporcionados especificamente pelo *Module Federation*, consolidando seus benefícios práticos. Por fim, Shah (2023) explora o potencial dos *Web Components* para a criação de componentes reutilizáveis e visualmente consistentes dentro de *design systems*.

No entanto, nenhum dos trabalhos apresentados realiza uma comparação empírica direta entre *Web Components* e *Module Federation* sob a ótica específica da integridade visual em cenários de estresse. Enquanto Antunes *et al.* () relatam a dificuldade de manter a consistência via percepção dos desenvolvedores, e Shah (2023) defende conceitualmente o encapsulamento, a literatura carece de um estudo que coloque as duas abordagens em confronto técnico direto para avaliar a resiliência contra vazamento de estilos (*CSS Bleeding*) e a fidelidade de renderização em um ambiente heterogêneo.

É nesta lacuna que o presente trabalho se insere. A proposta desta pesquisa avança o estado da arte ao realizar um experimento técnico controlado para avaliar e comparar as duas estratégias. Diferente de abordagens focadas em performance de rede ou percepção subjetiva de usabilidade, este estudo foca na consistência visual técnica, avaliada por meio de métricas de

conformidade de estilo (*Pixel Perfect*), testes de robustez contra conflitos de CSS global e análise de manutenibilidade. A Tabela **??** resume as principais características dos trabalhos relacionados e destaca as particularidades da presente proposta.

Quadro 2 – Quadro comparativo dos trabalhos relacionados e da proposta do TCC

Critério	**(Taibi; Mezzalira, 2022)**	**(Antunes *et al.*,)**	**(Mane *et al.*, 2024)**	**(Shah, 2023)**	**Proposta do TCC**
Tipo de Estudo	Revisão de princípios e experiência dos autores.	*Action Research* em ambiente industrial.	Estudo de caso múltiplo com análise quantitativa.	Revisão conceitual da tecnologia.	**Sondagem Inicial (Survey) e Experimento Controlado.**
Contexto	Genérico / Conceitual.	Migração de sistema real (indústria de óleo e gás).	Estudos de caso (e-commerce, saúde, etc).	Aplicação em CBSE e *Design Systems*.	**Ambiente multi-tecnologia (Protótipo E-commerce).**
Técnica Analisada	*Web Components, Module Federation, iFrames*, etc.	*Module Federation*.	*Module Federation*.	*Web Components*.	***Web Components* vs. *Module Federation*.**
Desafios	Inconsistência visual, sobrecarga de comunicação.	Consistência de UI/UX, gestão de dependências.	Integração, comunicação, UX coesa.	Compatibilidade, performance, acessibilidade.	**Resiliência a conflitos de estilo.**
Métricas	N/A (Conceitual).	Qualitativas (grupo focal, TAM).	Quantitativas (performance, *bundle size*).	N/A (Conceitual).	**Conformidade de Estilo, Robustez a Falhas e Manutenibilidade.**
Foco na Consistência Visual	Recomenda governança com *design system*.	Apontada como desafio crítico pelos times.	Ponto essencial para uma experiência coesa.	Apresentada como benefício do encapsulamento.	**Foco central, com validação via Testes de Estresse.**

Fonte: Elaborado pelo autor

4 METODOLOGIA

Este capítulo detalha os procedimentos metodológicos empregados para atingir o objetivo geral deste trabalho. Trata-se de um estudo de natureza aplicada e abordagem qualitativa, conduzido através do método de estudo de caso baseado em artefatos. A investigação estrutura-se na construção de protótipos de software em cenários controlados, com a intenção analisar tecnicamente o impacto de decisões arquiteturais na consistência visual.

A execução do método foi dividida em três etapas: (i) sondagem inicial para validação do problema; (ii) construção dos cenários arquiteturais controlados; e (iii) avaliação comparativa baseada em testes de estresse.

4.1 Sondagem Inicial

A primeira etapa da pesquisa consistiu no planejamento e execução de um levantamento de dados primários (*survey*) junto a profissionais da indústria de *software*. A decisão por iniciar o estudo com uma sondagem empírica baseou-se na necessidade de validar as premissas teóricas, garantindo que os desafios de consistência visual investigados refletissem problemas reais. O objetivo principal foi mapear a prevalência de ambientes heterogêneos e qualificar as dificuldades de governança de UI/UX.

O público-alvo foi definido como profissionais de Tecnologia da Informação (desenvolvedores *frontend*, *fullstack* e arquitetos) com experiência prática em Micro *Frontends* (MFEs). Utilizou-se uma amostragem não-probabilística por conveniência, com divulgação realizada nos meses de outubro e novembro de 2025 através de abordagens ativas no *LinkedIn* e passivas em grupos técnicos no *WhatsApp*.

A coleta de dados ocorreu via *Google Forms*, configurado para garantir o anonimato dos respondentes. Para assegurar a qualidade do instrumento, foi conduzido previamente um estudo piloto com 6 profissionais, com o intuito de melhorar a clareza dos enunciados eliminando ambiguidades nos termos técnicos e estimar o tempo de resposta.

O questionário final foi estruturado em cinco blocos temáticos que fundamentaram os requisitos da fase seguinte (construção dos cenários):

1. **Perfil do Participante:** Caracterização da senioridade e tempo de experiência.
2. **Contexto de Uso de Micro *Frontends*:** Mapeamento da heterogeneidade tecnológica para justificar a escolha da *stack* (React, Vue e Angular).

3. **Experiência Pessoal:** Investigação sobre a curva de aprendizado e barreiras de entrada.
4. **Desafios Técnicos:** Identificação dos gargalos de comunicação entre *design* e desenvolvimento.
5. **Consistência Visual e *Design System*:** Análise da adoção de *Design Systems* e tipos de inconsistências visuais frequentes.

O roteiro completo, contendo todas as questões aplicadas, encontra-se detalhado no **Apêndice A**. Os dados obtidos foram tratados de forma agregada para guiar a definição dos artefatos descritos a seguir.

4.2 Construção dos Cenários Experimentais

A segunda etapa do trabalho envolveu o desenvolvimento de artefatos de *software* para fins de comparação técnica. Foi desenvolvida uma Prova de Conceito (*Proof of Concept*) (PoC) simulando uma aplicação de comércio eletrônico (*E-commerce*), caracterizada pela divisão vertical de domínios funcionais.

Para isolar a variável arquitetura e garantir que o foco da investigação permanecesse exclusivamente na camada de apresentação (*Frontend*), a camada de serviços (*Backend*) foi abstraída. O fornecimento de dados (catálogo de produtos, preços e detalhes) foi implementado através de arquivos JSON estáticos (*Mock Data*), simulando o contrato de uma API REST. Essa decisão elimina a latência de rede e dependências externas que poderiam interferir nos testes de carregamento dos módulos.

Previamente à etapa de codificação, foram elaborados protótipos de alta fidelidade de todas as telas da aplicação (Listagem, Detalhes e Carrinho) utilizando a ferramenta Figma. Estes protótipos serviram como especificação funcional e visual para o desenvolvimento, garantindo que ambos os cenários arquiteturais (A e B) perseguissem exatamente o mesmo resultado visual.

Para reproduzir os desafios de interoperabilidade identificados na sondagem inicial (Seção 4.1), optou-se pela utilização de uma arquitetura multi-tecnologia composta por quatro MFEs distintos:

- **Aplicação Shell (React):** Responsável pelo *layout* estrutural, menu de navegação global e orquestração das rotas.
- **MFE Lista de Produtos (Vue.js):** Responsável pela renderização do catálogo de itens e filtros de busca.
- **MFE Detalhes do Produto (Angular):** Responsável pela exibição das informações

detalhadas e especificações técnicas de um item selecionado.

– **MFE Carrinho de Compras (React):** Responsável pelo gerenciamento do estado da compra e persistência local.

Para isolar a variável arquitetura e garantir a validade interna do experimento, utilizou-se o *Design System* como variável de controle. Para operacionalizar essa escolha, adotou-se o ***Material Design 3*** (Google, 2025), em função de sua ampla adoção na indústria, documentação consolidada e definição explícita de *design tokens*. Todos os MFEs, independentemente da tecnologia base, foram obrigados a consumir os mesmos *tokens* de *design* (cores, tipografia, espaçamentos) definidos por este padrão, assegurando que quaisquer divergências observadas fossem decorrentes da estratégia de integração e não de decisões subjetivas de *design*.

Os artefatos foram implementados em dois cenários distintos, diferindo apenas na estratégia técnica de composição:

4.2.1 Cenário A: Estratégia de Padronização (Web Components)

Neste cenário, a arquitetura foi definida com foco na interoperabilidade agnóstica. A estratégia adotada consiste no desacoplamento total entre a camada de apresentação e a lógica dos *frameworks*. Para isso, determinou-se o uso da especificação de *Web Components*, padronizada pelo W3C, permitindo que os elementos do *Design System* sejam instanciados como componentes universais.

Além da reutilização, esta abordagem foi selecionada por oferecer nativamente o recurso de *Shadow DOM*. Metodologicamente, o uso desta tecnologia tem o objetivo de isolar a árvore de elementos do componente, criando uma barreira de encapsulamento que deve impedir, teoricamente, a interferência de estilos globais da aplicação hospedeira (*Shell*) sobre os MFEs.

4.2.2 Cenário B: Estratégia de Federação (Module Federation)

Em contrapartida, o segundo cenário adota uma estratégia de integração em tempo de execução. A arquitetura foi estruturada utilizando a ferramenta *Webpack Module Federation* para orquestrar o carregamento dinâmico dos módulos remotos. Diferente da abordagem anterior, esta estratégia privilegia o compartilhamento de dependências e recursos comuns entre a aplicação *Shell* e os MFEs, evitando a duplicidade de bibliotecas no navegador.

A escolha deste cenário tem como objetivo metodológico simular um ambiente propenso ao acoplamento visual. Como essa tecnologia não impõe um isolamento nativo de

DOM, os MFEs coexistem no mesmo escopo global de estilos. Isso permite testar a resiliência da arquitetura diante do "vazamento de estilos" (*CSS Bleeding*) e verificar se a mistura de tecnologias (*frameworks* distintos) no mesmo documento HTML gera conflitos de renderização.

4.3 Protocolo de Avaliação Experimental

A terceira etapa da metodologia consiste na execução de um protocolo experimental controlado, projetado para mensurar e comparar a resiliência das estratégias de integração dos Cenários A *(Web Components)* e B *(Module Federation)*. O foco reside em avaliar como cada arquitetura mantém a consistência visual quando submetida a condições de estresse comuns em ambientes de MFEs multi-tecnologia e distribuídos. O protocolo é composto por três experimentos sequenciais, cada um investigando um vetor de falha potencial distinto.

4.3.1 Teste de Vazamento de Estilos (CSS Bleeding)

Este experimento avalia a capacidade fundamental de isolamento visual de cada abordagem, testando suas defesas contra a interferência indevida de estilos CSS globais.

4.3.1.1 Procedimento

1. **Linha de Base:** A aplicação composta é carregada e uma captura de tela (screenshot) inicial é registrada.
2. **Injeção de Estresse:** Através do console do navegador, executa-se um *script* que injeta de forma programática folhas de estilo CSS agressivas no documento principal. Estas regras são intencionalmente genéricas e utilizam `!important` para maximizar o conflito.
3. *Coleta de Evidências:* Novas capturas de tela são tiradas após a injeção. Paralelamente, utiliza-se a aba *Elements* e *Computed Styles* das Ferramentas do Desenvolvedor (DevTools) do navegador para inspecionar se os valores computados dos *design tokens* nos componentes-alvo foram alterados em relação à linha de base.

4.3.1.2 Critérios de Avaliação

– Se os componentes em todos os MFEs mantiveram inalterados seus estilos visuais definidos pelo *Design System*, ignorando completamente as regras injetadas, são considerados **resistente**.

– Se um ou mais componentes em qualquer micro *frontend* apresentaram alterações visuais, herdando total ou parcialmente os estilos injetados, serão considerados **vulnerável**.

4.3.2 Teste de Isolamento Estrutural do DOM

Este experimento avalia a capacidade da arquitetura de integração em proteger a estrutura interna da árvore DOM dos componentes contra acesso e manipulação direta a partir do escopo global do documento. Enquanto o teste anterior (4.3.1) avalia o vazamento de estilos, este foca no isolamento estrutural dos elementos HTML que compõem a interface.

4.3.2.1 Procedimento

O teste será conduzido a partir do console do navegador, no contexto global da página da aplicação `Shell`, após a carga completa de um MFE alvo. Serão executados três comandos sequenciais, cada um testando um nível de acesso diferente à árvore DOM interna do componente:

1. **Teste de Acesso de Leitura (*Read*):** Tentativa de seleção de um elemento interno do componente usando `document.querySelector` com um seletor específico.
2. **Teste de Acesso de Modificação (*Write*):** Caso o elemento seja localizado, tentativa de alteração de seu conteúdo textual e aplicação de estilos *inline* diretamente.
3. **Teste de Acesso Estrutural (*Structure*):** Tentativa de injeção de um novo nó HTML dentro de um *container* interno do componente.

4.3.2.2 Critérios de Avaliação

Para cada cenário (A e B), o resultado será classificado com base na observação da interface e nos logs do console após a execução dos três comandos:

– Se nenhuma das três operações for bem-sucedida, ou seja, O `querySelector` retorna `null`, o conteúdo e a aparência do componente permanecem inalterados, e nenhum novo elemento é visualmente injetado. Nesse caso, classifica-se como **encapsulado**.

– Ae qualquer uma das operações for bem-sucedida, resultando em: Localização do elemento interno no console; Alteração visível do conteúdo ou estilo do componente; Aparição de um novo elemento dentro do componente. A partir disso, classifica-se como **exposto**.

4.3.3 Consolidação e Análise dos Dados

Os resultados binários e as evidências técnicas coletados em cada experimento serão sistematizados em uma matriz comparativa. Esta matriz contrastará diretamente o desempenho dos Cenários A e B frente a cada vetor de estresse, servindo como base objetiva para a análise discursiva no Capítulo 5. O objetivo final é transcender a simples identificação de um "vencedor", elucidando os mecanismos específicos e os *trade-offs* intrínsecos que cada decisão arquitetural impõe à consistência visual do sistema.

5 RESULTADOS

Este capítulo apresenta os dados coletados durante a sondagem inicial, detalha a implementação técnica dos cenários experimentais e discute os resultados obtidos nos testes de consistência visual.

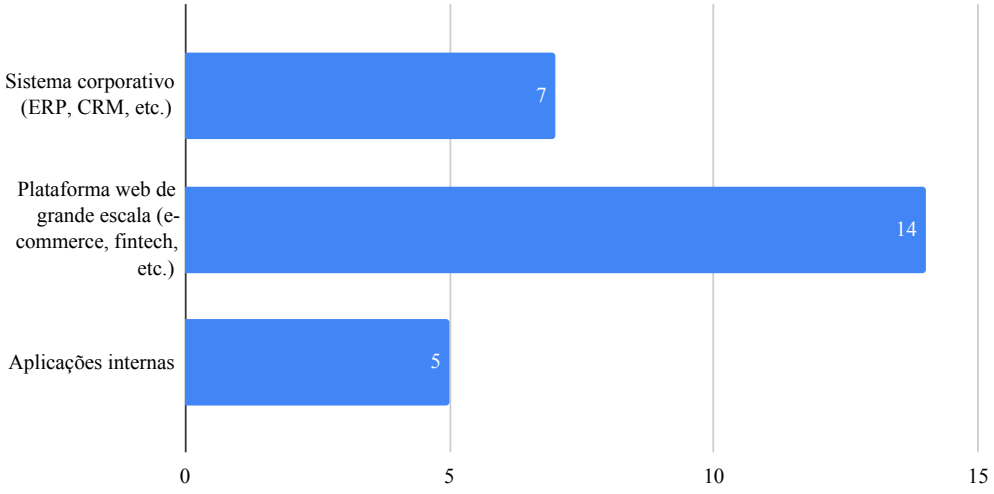
5.1 Resultados da Sondagem Inicial (*Survey*)

A sondagem inicial, realizada com 18 profissionais da indústria, cumpriu o papel fundamental de validar as premissas de engenharia adotadas neste trabalho. A análise dos dados permite traçar o perfil do "Projeto Padrão" de Micro Frontends MFEs no mercado, justificando as escolhas de escopo da PoC.

5.1.1 Contexto de Aplicação e Divisão Arquitetural

Para garantir que os artefatos desenvolvidos refletissem desafios reais, investigou-se onde e como a arquitetura é utilizada. Os resultados demonstraram uma predominância absoluta de aplicações de alta complexidade, conforme ilustra a Figura 5.

Figura 5 – Tipos de Projetos que utilizam MFEs (Múltipla escolha)



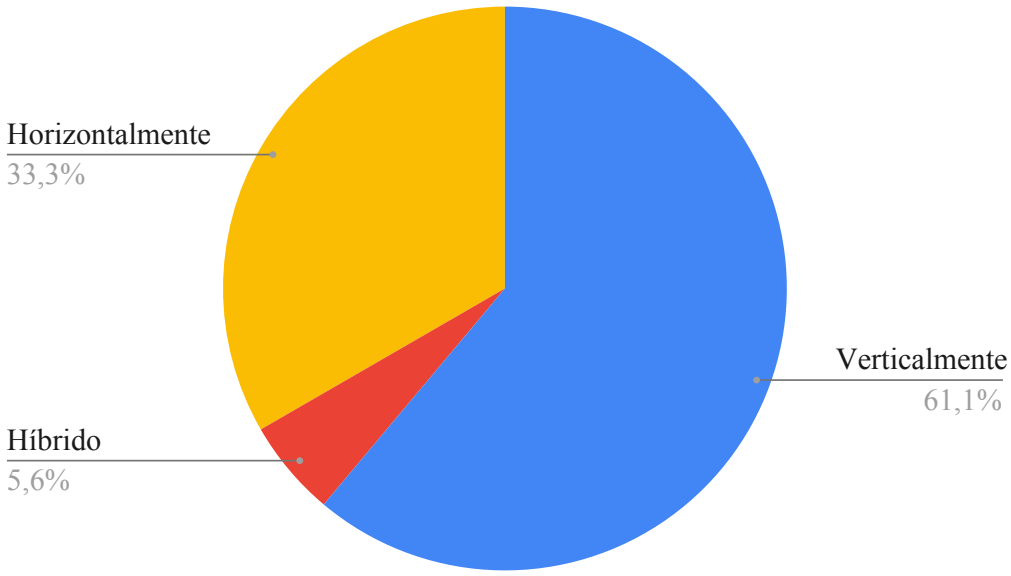
Fonte: Elaborado pelo autor.

A grande maioria dos respondentes indicou atuar em "Plataformas web de grande escala", seguido por "Sistemas Corporativos". Esse dado valida a decisão de construir a PoC simulando um *E-commerce*, alinhando o experimento ao cenário mais comum do mercado.

Quanto à estratégia de decomposição (Figura 6), houve uma preferência clara pela

Divisão Vertical (um MFE por funcionalidade ou página) em detrimento da divisão Horizontal ou de modelos híbridos.

Figura 6 – Estratégia de Divisão dos Micro *Frontends*)



Fonte: Elaborado pelo autor.

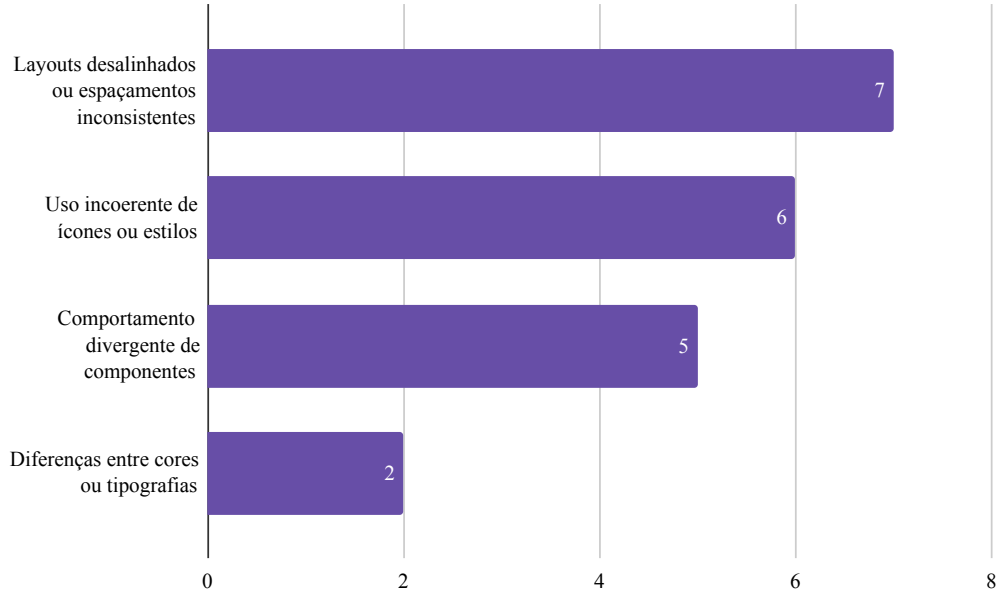
Estes dados conferem respaldo à arquitetura escolhida para os cenários experimentais, onde cada rota da aplicação (Lista, Detalhes, Carrinho) é servida por um micro *frontend* distinto, refletindo a prática majoritária da indústria.

5.1.2 Principais Inconsistências Observadas

Para qualificar o problema, os participantes identificaram os tipos de divergências visuais mais frequentes em seus projetos. A **Figura 7** destaca os quatro problemas mais recorrentes.

A predominância de falhas estruturais, como "*Layouts* desalinhados ou espaçamentos inconsistentes" (citado por 38,9% da amostra) e "Uso incoerente de estilos"(33,3%), fornece o embasamento empírico para os testes de estresse realizados na fase experimental. Esses dados indicam que as falhas de CSS (cascata e especificidade) são os principais vetores de degradação visual, justificando o foco da avaliação técnica na resiliência contra o vazamento de estilos.

Figura 7 – Principais Inconsistências Visuais em Ambientes Distribuídos



Fonte: Elaborado pelo autor.

1—-

5.2 Implementação dos Cenários Experimentais

A materialização deste estudo de caso seguiu um fluxo de trabalho iterativo, partindo da definição visual, passando pela estruturação da infraestrutura de suporte e culminando na codificação dos dois cenários arquiteturais distintos. Para reproduzir a complexidade de um ambiente heterogêneo real, a arquitetura lógica da aplicação foi definida com tecnologias distintas para cada domínio de negócio. A Figura 8 ilustra a organização dos quatro MFEs e sua relação com a infraestrutura de dados.

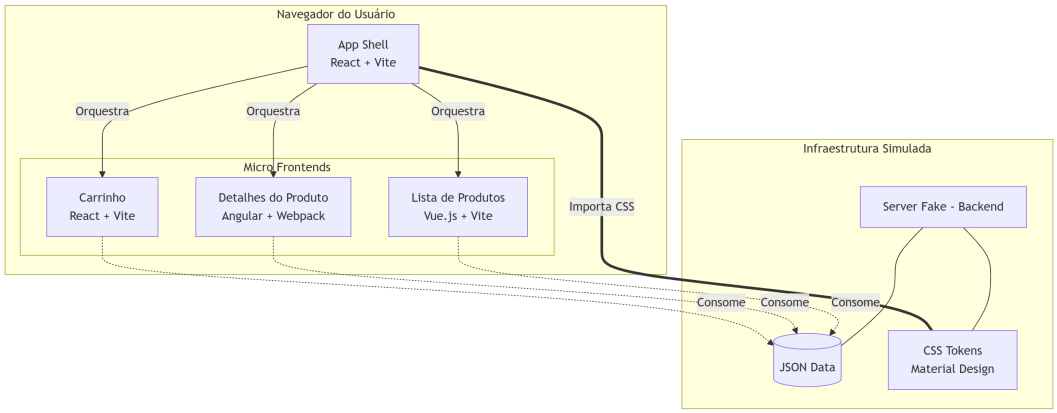
Conforme observado na figura, o `Shell` (desenvolvido em React) atua como o orquestrador central, responsável por carregar os módulos de Lista (Vue.js), Detalhes (Angular) e Carrinho (React), independentemente da estratégia de integração utilizada (Cenário A ou B).

5.2.1 Etapa 1: Definição do Baseline e Infraestrutura

O desenvolvimento dos artefatos iniciou-se pela definição rigorosa da camada visual, e não pela codificação. Para assegurar a comparabilidade objetiva entre os cenários, foram elaborados protótipos de alta fidelidade no Figma correspondentes às três interfaces funcionais da aplicação: Listagem de Produtos, Detalhes e Carrinho de Compras.

Utilizou-se estritamente o *UI Kit* oficial do *Material Design 3*, estabelecendo um

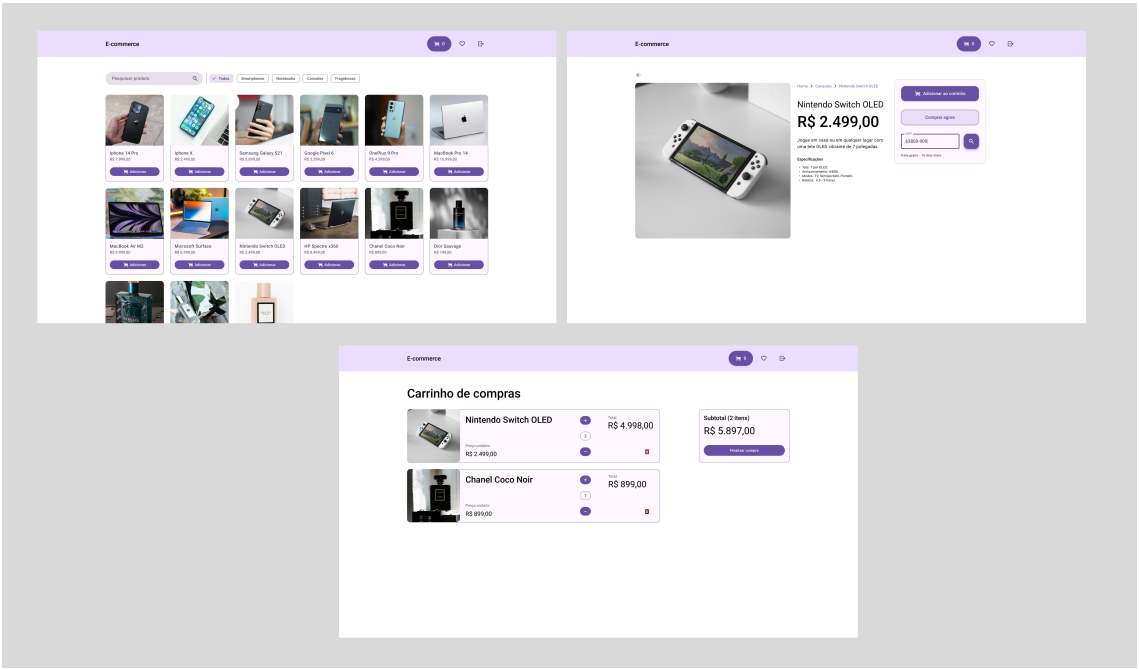
Figura 8 – Arquitetura Lógica e Distribuição Tecnológica da Aplicação



Fonte: Elaborado pelo autor.

"gabarito visual"(*Baseline*) imutável. Desta forma, qualquer desvio de renderização observado posteriormente no código poderia ser atribuído à arquitetura, e não a decisões subjetivas de *design*, conforme ilustra a Figura 9.

Figura 9 – Protótipos de alta fidelidade (Baseline Visual)



Fonte: Elaborado pelo autor.

Paralelamente, foi implementada uma infraestrutura de suporte centralizada ("Servidor Fake") para atuar como fonte única de distribuição de recursos. Este servidor disponibilizou

os ativos essenciais via protocolo *HyperText Transfer ProtocoL* (HTTP):

1. **Dados (JSON):** Uma *Mock API* estática fornecendo uma lista de 15 produtos com estrutura de dados complexa (incluindo preços, descrições longas e especificações aninhadas) e imagens hospedadas localmente (origem: Unsplash[1]), simulando o contrato real de uma API *Representational State Transfer* (REST).
2. **Estilos (*Design Tokens*):** O servidor hospedou os arquivos CSS oficiais do repositório *Material Design Tokens* (Google, 2024). Estes arquivos expõem os tokens (cores e tipografia) através de Variáveis CSS, permitindo que o tema seja consumido nativamente pelos navegadores.

Como decisão arquitetural crítica, determinou-se que o arquivo de tokens (`theme.css`) fosse importado exclusivamente no documento `index.html` da aplicação `Shell`, em ambos os cenários. Nenhum micro *frontend* importou o CSS global individualmente. Essa restrição foi imposital para testar a capacidade da arquitetura de propagar (herdar) estilos globais para os fragmentos encapsulados ou federados.

5.2.2 Etapa 2: Construção do Cenário A (Padronização via Web Components)

A implementação do primeiro cenário priorizou o isolamento e o reuso. A estratégia adotada foi o desenvolvimento *Bottom-Up* (de baixo para cima), iniciando pelos elementos menores.

5.2.2.1 Biblioteca de Componentes Compartilhados

Antes de construir as telas, criou-se uma pasta isolada denominada `shared-components`. Nela, foram implementados os "átomos"da interface (botões, cards vazios e *tags*) seguindo os *tokens* do *Material Design*. Estes componentes agnósticos serviram de base para reduzir a duplicação de código visual e promover a reutilização de componentes neste cenário.

5.2.2.2 Orquestração e Isolamento

Na sequência, desenvolveu-se o `Shell` (React) e os MFEs funcionais: `mfe-product-list`, `mfe-product-details` e `mfe-cart`.

A integração ocorreu através da conversão de cada MFE em um *Web Component*

[1] https://unsplash.com/pt-br

nativo. No caso do Angular, utilizou-se a biblioteca `@angular/elements`. Para React e Vue, foram criados *wrappers* manuais para instanciar a aplicação dentro de uma classe `HTMLElement`. Para o roteamente implementada uma navegação do tipo *Soft Navigation*.

Esta abordagem permitiu uma experiência fluida de *Single Page Application* (SPA), mesmo com tecnologias distintas coexistindo.

5.2.3 Etapa 3: Construção do Cenário B (Integração via Module Federation)

O segundo cenário foi construído com foco na integração em tempo de execução, sem o uso de *Web Components*. A ordem de desenvolvimento partiu do `Shell` para os MFEs, focando na configuração dos empacotadores (*bundlers*).

5.2.3.1 Desafios de Configuração e Duplicação

Diferente do cenário anterior, aqui não foi utilizada a biblioteca de componentes compartilhados. Cada micro *frontend* implementou seus próprios componentes visuais internamente. Essa decisão reflete um cenário comum em times verticais que possuem autonomia total, mas gera o efeito colateral de duplicação de código.

A infraestrutura exigiu uma configuração híbrida complexa para lidar com a heterogeneidade das ferramentas de *build*. Nos projetos inicializados com o Vite, os que utilizavam React e Vue, foi através do `@originjs/vite-plugin-federation`. Já na página de detalhes (Angular) aconteceu via `@angular-architects/module-federation`, uma vez que o framework depende fortemente do Webpack.

5.2.3.2 O Padrão de Montagem

Para renderizar o Vue e o Angular dentro da árvore de componentes do React (`Shell`), adotou-se o padrão de injetar uma função de montagem. O `Shell` não importa um componente, mas executa uma função remota que recebe uma referência do DOM (`ref`).

Quanto à navegação, devido à complexidade de compartilhar o contexto de rotas entre roteadores de frameworks diferentes (Vue Router, Angular Router), optou-se pela navegação nativa via `window.location.href` ou links simples (`<a href>`). Embora funcional, isso resultou em *Hard Reloads* a cada troca de micro frontend.

5.3 Análise experimental

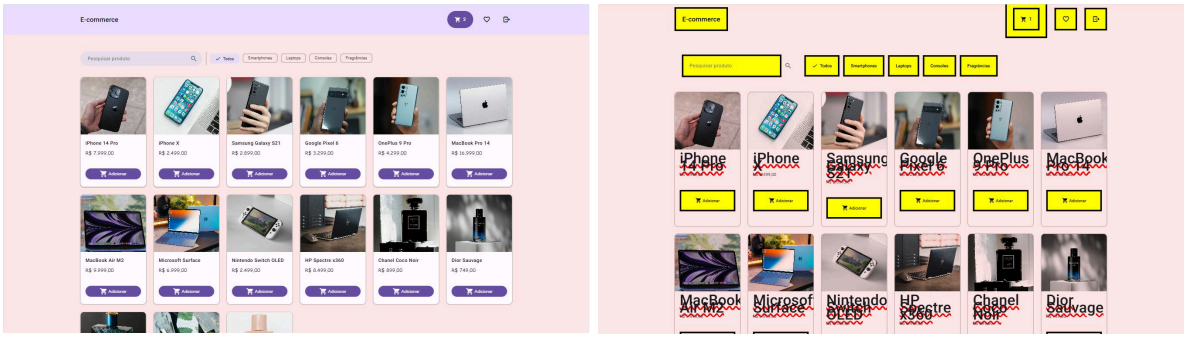
Esta seção apresenta os resultados obtidos a partir da execução dos protocolos de teste definidos na metodologia. A análise concentra-se em evidenciar como cada decisão arquitetural reagiu às tentativas de violação de consistência, cruzando dados qualitativos (observação visual) e quantitativos (exposição do DOM).

5.3.1 Teste de vazamento de estilos

O teste de estresse consistiu na execução de um *script* de injeção de falhas diretamente no console do navegador. O algoritmo foi desenhado para atacar três camadas críticas da interface: o *layout* global, os elementos de interação (botões/*inputs*) e a hierarquia tipográfica.

O código 1, descrito no apêndice B, apresenta a rotina utilizada, que força a sobrescrita de estilos via `!important`. A Figura 10 apresenta o resultado visual imediato da aplicação deste *script* nos dois cenários estudados.

Figura 10 – Comparativo visual pós-injeção de falhas



Fonte: Elaborado pelo autor.

Embora a Figura 10 apresente o impacto visual macroscópico, foi realizada uma auditoria granular para classificar a extensão dos danos. A Tabela 2, disponível no Apêndice **??**, detalha o status de cada elemento inspecionado nos dois cenários.

Tabela 2 – Log de Auditoria de Consistência Visual

Cenário	Rota (MFE)	Componente	Observação Visual	Status
Cenário A	–	body (Shell)	Fundo ficou rosado e a tipografia foi alterada para Comic Sans.	**Vulnerável**
Cenário A	–	header (Shell)	Botões e tipografia do título permaneceram inalterados.	**Resistente**
Cenário A	Lista (Vue)	app-card (Produto)	Fundo branco e bordas originais mantidas, ignorando o estilo global.	**Resistente**
Cenário A	Lista (Vue)	chips (Produto)	Fundo, bordas e tipografia preservados.	**Resistente**
Cenário A	Lista (Vue)	textos (Produto)	Textos mantidos, com exceção do preço, que passou a usar Comic Sans.	**Resistente**
Cenário A	Detalhe (Angular)	breadcrumbs	Cores preservadas, porém tipografia alterada.	**Vulnerável**
Cenário A	Detalhe (Angular)	card (Detalhe)	Fundo e bordas mantidos; botões preservaram formato e cores.	**Resistente**
Cenário A	Detalhe (Angular)	Info. Produto	Tipografia preservada, mas a cor da fonte tornou-se vermelha.	**Vulnerável**
Cenário A	Carrinho (React)	textos	Fonte preservada; alteração apenas na cor do texto.	**Vulnerável**
Cenário A	Carrinho (React)	Lista de Itens	Componentes mantiveram o estilo definido no Design System.	**Resistente**
Cenário A	Carrinho (React)	card (Carrinho)	Fundo branco e bordas preservadas; botões inalterados.	**Resistente**
Cenário B	–	body (Shell)	Fundo rosado e tipografia alterada globalmente.	**Vulnerável**
Cenário B	–	header (Shell)	Botões perderam padding e border-radius.	**Vulnerável**
Cenário B	Lista (Vue)	app-card	Layout comprometido (padding e margin); fundo rosado.	**Vulnerável**
Cenário B	Lista (Vue)	chips	Tipografia alterada para Comic Sans e cor vermelha.	**Vulnerável**

Continua na próxima página...

Cenário	Rota (MFE)	Componente	Observação Visual	Status
Cenário B	Lista (Vue)	textos	Todos os textos com Comic Sans e cor vermelha.	**Vulnerável**
Cenário B	Detalhe (Angular)	breadcrumbs	Alteração simultânea de cor e tipografia.	**Vulnerável**
Cenário B	Detalhe (Angular)	card (Detalhe)	Fundo rosado e botões perderam formatação original.	**Vulnerável**
Cenário B	Detalhe (Angular)	Info. Produto	Alterações na cor e no tamanho dos títulos.	**Vulnerável**
Cenário B	Carrinho (React)	textos	Tipografia e cores alteradas globalmente.	**Vulnerável**
Cenário B	Carrinho (React)	Lista de Itens	Botões de remoção ficaram quadrados e amarelos.	**Vulnerável**
Cenário B	Carrinho (React)	card (Carrinho)	Fundo rosado e bordas quadradas.	**Vulnerável**

A análise detalhada da Tabela 2 revela nuances importantes sobre o comportamento de cada arquitetura. No Cenário A, a predominância do status "resistente" confirma a eficácia do *Shadow* DOM como barreira de contenção para propriedades estruturais. Observe-se que componentes complexos, como o `app-card` e os botões, mantiveram suas dimensões, bordas e espaçamentos (*padding*) inalterados, ignorando as regras agressivas injetadas com `!important`.

No entanto, a tabela aponta vulnerabilidades específicas neste cenário, concentradas em elementos textuais (`breadcrumbs` e textos informativos). Isso ocorre porque, na especificação do DOM, propriedades como `color` e `font-family` são herdáveis por padrão e conseguem atravessar a fronteira do *Shadow DOM* se não forem explicitamente redefinidas dentro do componente. Na realidade prática, isso significa que o isolamento via *Web Components* protege o *layout*, evitando quebras de quebra-cabeça na tela, mas ainda exige cuidados manuais para garantir a imutabilidade tipográfica.

Por outro lado, o Cenário B apresentou uma contaminação sistêmica, classificada quase integralmente como "vulnerável". A ausência de encapsulamento permitiu que o *script* de estresse sobrescrevesse regras de todos os MFEs simultaneamente. Na prática industrial, esse comportamento representa um risco crítico: uma alteração acidental no CSS global ou a importação de uma biblioteca de terceiros mal formatada pode degradar a experiência do usuário

em toda a plataforma, desfigurando botões e fundos, como evidenciado pela alteração dos *chips* e *cards* na tabela.

Conclui-se, portanto, que existe um *trade-off* operacional claro. O Cenário A oferece segurança estrutural, ideal para ambientes onde equipes distintas não confiam plenamente no código umas das outras, mas cobra o preço na dificuldade de compartilhar temas globais. Já o Cenário B, apesar da fragilidade demonstrada, favorece a governança de marca, pois permite que uma atualização na identidade visual, como a cor primária ou a fonte institucional, seja propagada instantaneamente para todos os MFEs, desde que haja disciplina rigorosa para evitar conflitos de nomenclatura.

5.3.2 Teste de análise de estrutura do DOM

Este experimento teve como objetivo verificar a segurança estrutural e o nível de isolamento dos MFEs). Conforme o protocolo definido, foram executados scripts de auditoria via console do navegador (detalhados no Apêndice B), projetados especificamente para tentar realizar três operações invasivas em componentes internos de cada *Micro Frontend* (MFE): (i) Rastreamento (leitura de nós do DOM), (ii) Modificação (alteração de propriedades de texto ou estilo) e (iii) Injeção (inserção forçada de elementos HTML externos).

O Quadro 3 sumariza os resultados obtidos após a execução dos *scripts* nos dois cenários arquiteturais.

A análise detalhada do Quadro 3 permite quantificar o conceito de "isolamento total" observado no Cenário A. O dado fundamental encontra-se na linha de leitura, onde o *script* de auditoria não conseguiu acessar nenhum dos elementos internos para todos os componentes baseados em *Web Components*. Esse resultado evidencia que o *Shadow* DOM atuou como uma barreira intransponível para o escopo global. O isolamento, portanto, define-se tecnicamente pela invisibilidade seletiva, uma vez que foi incapaz de rastrear ou obter uma referência aos nós internos, as tentativas subsequentes de ataque tornaram-se inviáveis. Consequentemente, as operações de modificação e injeção apresentam *status* de falha não por erro de execução, mas pela inexistência de um alvo acessível, garantindo a integridade do componente.

Em contrapartida, o Cenário B apresentou uma exposição completa da árvore de elementos. Como os MFEs são renderizados diretamente no *Light* DOM da aplicação hospedeira, não houve qualquer barreira técnica para os *scripts* de auditoria.

Os resultados validam que o Cenário B possui uma fragilidade arquitetural inerente

Quadro 3 – Resultados da auditoria de encapsulamento de DOM

MFE (Tecnologia)	**Operação**	**Cenário A (*Web Components*)**	**Cenário B (*Module Federation*)**
Lista (Vue.js)	Leitura	**Bloqueada** (0 elementos acessíveis)	**Exposta** (148 elementos acessíveis)
Lista (Vue.js)	Modificação	Falha (Seletor não encontrado)	Sucesso (Alterado `H3.body-large`)
Lista (Vue.js)	Injeção	Falha (Alvo inexistente)	Sucesso (Div inserida em `.controls`)
Carrinho (React)	Leitura	**Bloqueada** (0 elementos acessíveis)	**Exposta** (52 elementos acessíveis)
Carrinho (React)	Modificação	Falha (Seletor não encontrado)	Sucesso (Alterado `H1.display-medium`)
Carrinho (React)	Injeção	Falha (Alvo inexistente)	Sucesso (Div inserida em `.cart-container`)
Detalhes (Angular)	Leitura	**Bloqueada** (0 elementos acessíveis)	**Exposta** (43 elementos acessíveis)
Detalhes (Angular)	Modificação	Falha (Seletor não encontrado)	Sucesso (Alterado `H1.headline-large`)
Detalhes (Angular)	Injeção	Falha (Alvo inexistente)	Sucesso (Div inserida em `.page-container`)

Fonte: Elaborado pelo autor

quanto ao encapsulamento. A exposição de centenas de nós internos ao escopo global aumenta exponencialmente o risco de conflitos de *namespace* e "efeitos colaterais", onde *scripts* de terceiros podem acidentalmente quebrar a interface. O Cenário A, por sua vez, provou ser arquiteturalmente seguro, garantindo que o estado interno da interface permaneça imutável por agentes externos.

5.3.3 Síntese dos resultados

A análise cruzada dos experimentos de vazamento de estilos e isolamento estrutural revela que não existe uma estratégia de integração isenta de custos arquiteturais. Observou-se uma relação de compromisso inversa entre segurança e coesão sistêmica.

Enquanto o Cenário A priorizou a blindagem do componente, resultando em uma arquitetura resiliente a falhas externas mas mais rígida para a propagação de temas, o Cenário B privilegiou a fluidez na integração de dependências, ao custo de expor a estrutura interna dos MFEs a manipulações indevidas no escopo global.

O Quadro 4 apresenta a Matriz de *trade-offs* elaborada a partir dos dados coletados, contrastando as características técnicas observadas em cada abordagem frente aos vetores de inconsistência analisados.

Quadro 4 – Matriz de *trade-offs*: isolamento vs. federação

Critério de Avaliação	**Cenário A: Padronização (Web Components)**	**Cenário B: Federação (Module Federation)**
Resiliência Visual	**Alta.** A barreira do Shadow DOM impediu efetivamente a contaminação por estilos globais (CSS Bleeding).	**Baixa.** A interface mostrou-se permeável a conflitos, exigindo governança manual de nomes de classes.
Segurança do DOM	**Total.** A árvore interna dos componentes permaneceu inacessível a scripts de leitura e injeção do escopo global.	**Nula.** Elementos internos ficaram expostos, permitindo manipulação externa e injeção de nós arbitrários.
Propagação de Temas	**Complexa.** Exige repasse explícito de variáveis CSS ou o uso de seletores como `::part()`.	**Fluida.** Herda automaticamente os estilos e tokens definidos no *Shell*, facilitando a coesão.
Governança	Focada em contratos rígidos e estabilidade individual do componente.	Focada em velocidade de desenvolvimento e compartilhamento de recursos.
Indicação de Uso	Ambientes heterogêneos, legados ou com times distintos sem confiança mútua.	Ambientes tecnologicamente homogêneos com Design System maduro e centralizado.

Fonte: Elaborado pelo autor

6 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho dedicou-se a analisar o impacto das decisões arquiteturais na consistência de interface em aplicações distribuídas baseadas em MFEs. Motivada pela crescente adoção desta arquitetura em ambientes corporativos e pela escassez de literatura técnica sobre estratégias de governança visual em ecossistemas heterogêneos, a pesquisa comparou empiricamente duas abordagens de integração: o isolamento via *Web Components* e o compartilhamento via *Module Federation*.

Os resultados obtidos confirmam que o objetivo geral foi plenamente atingido, demonstrando que a consistência visual não é uma consequência natural da adoção de um *design system*, mas um atributo diretamente dependente da estratégia de encapsulamento do DOM e da gestão de dependências. Essa percepção de mercado foi corroborada pelos experimentos técnicos deste trabalho, que evidenciaram como a ausência de fronteiras arquiteturais expõe a aplicação a falhas de degradação visual.

A análise comparativa sintetizada na matriz de *trade-offs* revelou uma dicotomia fundamental entre robustez estrutural e agilidade de propagação. O Cenário que utiliuzou a técnica de isolamento (*Web Components*) provou ser a estratégia definitiva para garantir a integridade visual em ambientes tecnologicamente mistos. Os testes de estresse demonstraram que essa abordagem cria uma "blindagem" efetiva, onde os componentes permaneceram visualmente intactos mesmo sob a injeção agressiva de falhas globais. A contagem nula de elementos expostos no DOM confirma que o isolamento nativo é a barreira mais eficaz para conter o vazamento de estilos de forma determinística. No entanto, essa segurança impõe um custo operacional: a propagação de temas globais tornou-se mais rígida, exigindo o uso estrito de Variáveis CSS ou seletores específicos para perfurar o encapsulamento.

Em contrapartida, o Cenário que utilizou a técnica de compartilhamento (*Module Federation*) destacou-se pela eficiência na coesão sistêmica. A capacidade de compartilhar o escopo global permitiu que atualizações nos *design tokens* fossem refletidas instantaneamente em todos os MFEs, sem a necessidade de intervenções individuais. Contudo, essa permeabilidade resultou em uma fragilidade estrutural severa. A exposição de centenas de elementos internos ao escopo global tornou a interface suscetível a manipulações e efeitos colaterais imprevisíveis, validando a hipótese de que a federação, por padrão, exige uma disciplina de governança de CSS muitas vezes inviável em equipes descentralizadas.

Conclui-se, portanto, que não existe uma arquitetura superior em termos absolutos,

mas sim estratégias adequadas a requisitos de engenharia distintos. Para ecossistemas caracterizados pela alta heterogeneidade tecnológica, onde coexistem diferentes *frameworks* ou legados e a confiança entre módulos é baixa, o isolamento via *Web Components* apresenta-se como a escolha técnica mais segura para garantir a integridade da interface. Por outro lado, em ambientes homogêneos onde a governança é centralizada e a velocidade de evolução é prioritária, a estratégia de federação oferece maior agilidade, desde que acompanhada de ferramentas rigorosas de *linting* e escopo de estilos.

As contribuições deste estudo estendem-se à disponibilização de evidências empíricas sobre a vulnerabilidade do DOM em arquiteturas federadas e à validação das *CSS Custom Properties* como mecanismo essencial para a interoperabilidade em sistemas agnósticos. Embora o trabalho tenha se limitado a simulações renderizadas ao lado do cliente, os artefatos e a matriz de decisão apresentados servem como um guia prático para arquitetos de *software* mitigarem riscos visuais em projetos de larga escala.

Como trabalhos futuros, sugere-se a expansão da análise para cenários de *Renderização no Servidor* (SSR) e a investigação de pipelines de CI/CD integrados a testes de regressão visual automatizados, visando reduzir a fragilidade estrutural observada. Recomenda-se também a exploração de ferramentas de *linting* e escopo de CSS para a mitigação proativa de vazamentos de estilo e, dada a escassez de literatura técnica específica identificada durante este estudo, propõe-se a realização de um Mapeamento Sistemático da Literatura (MSL) focado na consistência visual em sistemas distribuídos, a fim de consolidar o corpo de conhecimento acadêmico sobre governança de UI em MFEs

A interpretação dos resultados deste estudo deve considerar certas limitações e ameaças à validade, iniciando pela sondagem preliminar cuja amostra reduzida e por conveniência impede generalizações estatísticas sobre a indústria. Adicionalmente, destaca-se o risco de viés de construção, uma vez que a implementação dos cenários e a execução dos testes foram conduzidas por um único desenvolvedor, o que pode refletir níveis desiguais de proficiência entre as tecnologias abordadas, somado ao fato de os experimentos terem ocorrido em ambiente controlado, isolando variáveis críticas de produção real, como latência de rede e interferência de *scripts* de terceiros.

REFERÊNCIAS

AMORIM, G.; ROCHA, L.; MENDES, F.; SANTOS, R.; CANEDO, E. Guidelines for adoption micro-frontend architecture. In: SIMPóSIO BRASILEIRO DE SISTEMAS DE INFORMAçãO, 21., 2025, Recife. **Anais [...]**. Poto Alegre: SBC. p. 713–722. Disponível em: https://sol.sbc.org.br/index.php/sbsi/article/view/34388. Acesso em: 25 out. 2025.

ANTUNES, F.; LIMA, M.; ARAúJO, M.; TAIBI, D.; KALINOWSKI, M. Investigating benefits and limitations of migrating to a micro-frontends architecture. In: SIMPóSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 38., 2024, Curitiba. **Anais [...]**. Curitiba: SBC. p. 103–113. Disponível em: https://sol.sbc.org.br/index.php/sbes/article/view/30353. Acesso em: 25 out. 2025.

ETEMAD, E. J.; SUZANNE, M. E. **CSS Cascading and Inheritance Level 5**. Cambridge, MA, USA, 2025. Disponível em: https://www.w3.org/TR/css-cascade-5/. Acesso em: 18 nov. 2025.

FROST, B. **Atomic Design**. Pittsburgh, Pennsylvania, Estados Unidos: Brad Frost, 2016. ISBN 9780998296609. Disponível em: https://atomicdesign.bradfrost.com/. Acesso em: 12 nov. 2025.

GOOGLE. **Material Tokens**. 2024. Disponível em: https://github.com/material-foundation/material-tokens. Acesso em: 16 dez. 2025.

GOOGLE. **Material Design 3**. 2025. Disponível em: https://m3.material.io/. Acesso em: 13 dez. 2025.

KHOLMATOVA, A. **Design Systems**: A practical guide to creating design languages for digital products. Freiburg, Alemanha: Smashing Magazine AG, 2017. ISBN 9783945749586.

LAVIE, T.; TRACTINSKY, N. Assessing dimensions of perceived visual aesthetics of web sites. **International Journal of Human-Computer Studies**, v. 60, n. 3, p. 269–298, 2004. Disponível em: https://www.sciencedirect.com/science/article/pii/S1071581903001642. Acesso em: 10 jun. 2025.

LAW, E. L.-C.; ROTO, V.; HASSENZAHL, M.; VERMEEREN, A. P. O. S.; KORT, J. Understanding, scoping and defining user experience: A survey approach. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 27. **Proceedings [...]**. Boston, USA: Association for Computing Machinery, 2009. p. 719–728.

LEWIS, J.; FOWLER, M. **Microservices**: a definition of this new architectural term. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 30 out. 2025.

MANE, H. R.; KUMAR, A.; DANDU, M. M. K.; GOEL, P. D. P.; JAIN, P. A.; SHRIVASTAV, E. A. Micro frontend architecture with webpack module federation: Enhancing modularity focusing on results and their implications. **Journal of Quantum Science and Technology (JQST)**, v. 1, n. 4, p. 25—57, 11 2024. Disponível em: https://jqst.org/index.php/j/article/view/95. Acesso em: 19 jun. 2025.

MARCO, V.; FARIAS, K. **Exploring the Technologies and Architectures Used to Develop Micro-frontend Applications**: A systematic mapping and emerging perspectives. 2024. Disponível em: https://ssrn.com/abstract=4750661. Acesso em: 15 jun. 2025.

MDN Contributors. **Web Components - MDN Web Docs**. 2024. Disponível em: https://developer.mozilla.org/en-US/docs/Web/Web_Components. Acesso em: 20 nov. 2025.

MEYER, E. A.; WEYL, E. **CSS: The Definitive Guide**: Visual presentation for the web. 4. ed. Sebastopol, Califórnia, Estados Unidos: O'Reilly Media, 2017. ISBN 9781449393199.

MEZZALIRA, L. **Building Micro-Frontends**: Scaling teams and projects, empowering developers. 1. ed. Sebastopol, Califórnia, Estados Unidos: O'Reilly Media, 2021. ISBN 9781492082965.

MINIUKOVICH, A.; ANGELI, A. D. Quantification of interface visual complexity. In: INTERNATIONAL WORKING CONFERENCE ON ADVANCED VISUAL INTERFACES, 2014. **Proceedings [...]**. Como, Itália: ACM, 2014. p. 153–160.

MOSHAGEN, M.; THIELSCH, M. T. Facets of visual aesthetics. **International Journal of Human-Computer Studies**, v. 68, n. 10, p. 689–709, 2010. Disponível em: https://www.sciencedirect.com/science/article/pii/S1071581910000777. Acesso em: 7 abr. 2025.

NIELSEN, J. **10 Usability Heuristics for User Interface Design**. 1994. Disponível em: https://www.nngroup.com/articles/ten-usability-heuristics/. Acesso em: 18 jun. 2025.

SHAH, H. Harnessing customized built-in elements: Empowering component-based software engineering and design systems with html5 web components. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, SOFT COMPUTING AND APPLICATIONS, 13. **Proceedings [...]**. [S. l.]: AIRCC Publishing Corporation, 2023. p. 247–259.

SUAREZ, M.; ANNE, J.; SYLOR-MILLER, K.; MOUNTER, D.; STANFIELD, R. **Design Systems Handbook**. Estados Unidos da América: DesignBetter.Co by InVision, 2019.

TAIBI, D.; MEZZALIRA, L. Micro-frontends: Principles, implementations, and pitfalls. **SIGSOFT Softw. Eng. Notes**, v. 47, n. 4, p. 25—-29, 9 2022. Disponível em: https://doi.org/10.1145/3561846.3561853. Acesso em: 23 abr. 2025.

VESSELOV, S.; DAVIS, T. **Building Design Systems**: Unify user experiences through a shared design language. 1. ed. Nova York, Estados Unidos: Apress, 2019. ISBN 9781484245149.

Webpack Contributors. **Module Federation - Webpack Documentation**. 2024. Disponível em: https://webpack.js.org/concepts/module-federation/. Acesso em: 23 nov. 2025.

APÊNDICE A – QUESTIONÁRIO DE SONDAGEM INICIAL

Bloco 1: Perfil do Participante

Questão 1. Idade:

(a) Menos de 20 anos

(b) 20 a 29 anos

(c) 30 a 39 anos

(d) 40 a 49 anos

(e) 50 anos ou mais

Questão 2. Gênero:

(a) Feminino

(b) Masculino

(c) Prefiro não informar

(d) Outro:

Questão 3. Tempo de experiência em desenvolvimento *frontend*:

(a) Menos de 1 ano

(b) 1 a 3 anos

(c) 3 a 5 anos

(d) Mais de 5 anos

Questão 4. Nível de atuação profissional

(a) Estagiário/*Trainee*

(b) Júnior

(c) Pleno

(d) Sênior

(e) Líder técnico/Arquiteto

Questão 5. Região ou pais que trabalha.

Bloco 2: Contexto de uso de Micro *Frontends*

Questão 1. Em que tipo de projeto você já utilizou MFEs?:

☐ Sistema corporativo (ERP, CRM, etc.).

☐ Plataforma *web* de grande escala (e-*commerce*, *fintech*, etc.)

- ☐ Aplicações internas
- ☐ Outro:

Questão 2. Como os micro *frontends* eram divididos no projeto?

(a) Verticalmente (um MFE por funcionalidade ou páginas)

(b) Horizontalmente (múltiplos MFEs compondo a mesma tela)

(c) Outro:

Questão 3. Descreva como as equipes de desenvolvimento eram organizadas em relação a esses MFEs.

Questão 4. Em sua opinião, como essa estrutura de equipes impactou a manutenção da consistência visual (UI/UX)?

(a) Facilitou a consistência (impacto positivo)

(b) Dificultou a consistência (impacto negativo)

(c) Não teve impacto significativo

(d) Não sei avaliar

Questão 5. Quais tecnologias ou *frameworks* eram utilizados nos micro *frontends*?

Questão 6. Sobre essas tecnologias, os MFEs eram:

(a) Homogêneos (Todos usavam as mesmas tecnologias)

(b) Parcialmente Heterogêneos (Os MFEs tinham uma tecnologia principal em comum, mas outras eram diferentes)

(c) Totalmente Heterogêneos (Cada MFE podia usar qualquer *stack* ou tecnologia)

(d) Não sei iinformar

(e) Outro:

Bloco 3: Experiência Pessoal com MFEs

Questão 1. Como ocorreu seu primeiro contato profissional com a arquitetura de MFEs?

(a) Projeto novo iniciado com MFEs

(b) Migração de um sistema monolítico

(c) Entrei em um projeto que já utilizava MFEs

(d) Outro:

Questão 2. Quais foram as suas principais dificuldades iniciais ao começar a trabalhar com essa arquitetura?

Questão 3. Como foi o seu processo de aprendizado/adaptação às ferramentas e padrões utilizados?

Bloco 4: Desafios Técnicos e Organizacionais

Questão 1. Quais desses desafios você considera mais relevantes em projetos com MFEs?

- ☐ Comunicação entre micro *frontends*
- ☐ Gestão de dependências compartilhadas
- ☐ Performance e tempo de carregamento
- ☐ Manutenção da consistência visual
- ☐ Padronização do código
- ☐ Integração entre equipes
- ☐ Governança de *design system*
- ☐ Outro:

Questão 2. Como as equipes de desenvolvimento e design se comunicavam para manter uma experiência coesa de UI/UX?

- ☐ Reuniões regulares
- ☐ Documentação centralizada
- ☐ Ferramentas de design compartilhado (ex.: Figma, Storybook)
- ☐ Uso de *tokens* de design
- ☐ Outro:

Bloco 5: Consistência Visual e *Design System*

Questão 1. O projeto possui/possuia um *Desgin System* centralizado?

(a) Sim, completamente adotado

(b) Parcialmente adotado

(c) Não

Questão 2. Como o *Design System* era distribuído e mantido entre os micro *frontends*?

- ☐ Repositório centralizado
- ☐ Biblioteca compartilhada (*npm, private registry*)
- ☐ *Web Components* reutilizáveis
- ☐ Não se aplica

☐ Outro:

Questão 3. Que estratégias ou práticas eram usadas para garantir a consistência visual entre MFEs?

☐ Revisões de código e design

☐ *Linters* e ferramentas de padronização

☐ *Storybook* ou catálago de componentes

☐ Testes de interface automatizados

☐ Outro:

Questão 4. Quais os principais problemas de incosistência você observou?

☐ Diferenças entre cores ou tipografias

☐ Comportamento divergente de componentes

☐ *Layouts* desalinhados ou espaçamentos inconsistentes

☐ Uso incoerente de ícones ou estilos

☐ Outro:

Questão 5. Vocês utilizam alguma métrica ou processo para avaliar a consistência visual ou a usabilidade dos MFEs?

(a) Sim

(b) Não

Questão 6. Se sim, qual?

APÊNDICE B – CÓDIGOS-FONTES UTILIZADOS PARA EXPERIMENTOS

Código-fonte 1 – Script de Injeção de Falhas

```
(function() {;
    const style = document.createElement('style');
    style.id = 'chaos-monkey-style';
    style.innerHTML = `
      body, main, div, section, article, header, footer {
          background-color: #ffe6e6 !important;
          font-family: 'Comic Sans MS', 'Chalkboard SE',
             sans-serif !important;
          color: #ff0000 !important;
      }

      button, a, input {
          border: 5px solid black !important;
          background-color: yellow !important;
          color: black !important;
          border-radius: 0px !important;
          padding: 20px !important;
          box-shadow: none !important;
      }

      h1, h2, h3, h4, h5 {
          text-decoration: underline wavy red !important;
          font-size: 3rem !important;
      }
    `;
    document.head.appendChild(style);
})();
```

Código-fonte 2 – Script de Exposição de Elementos na Listagem de Produtos (Vue)

```
(async function() {
    console.log('     INICIANDO TESTE DE ISOLAMENTO
       ESTRUTURAL DO DOM - CEN RIO A (Web Components)');
    console.log('MFE alvo: Lista de Produtos (Vue)');
    console.log('---');

    await new Promise(resolve => setTimeout(resolve, 1500))
       ;

    console.log('1. TESTE DE LEITURA (Read)');

    const seletoresParaTestar = [
        '.product-grid',
        '.card',
        '.image-area',
        '.info',
        '.name',
        '.price',
        'app-button',
        'app-card',
        'app-tag',
        '.search-box',
        '.filters',
        '.search-input',
        '.material-icons',
        '[class*="product"]',
        '[class*="card"]',
        'button',
        'img'
    ];

```

```
30      let elementosEncontrados = [];
31      seletoresParaTestar.forEach(seletor => {
32          const elementos = document.querySelectorAll(seletor
               );
33          if (elementos.length > 0) {
34              console.log(`       Encontrado: ${seletor} (${
                   elementos.length} elementos)`);
35              elementosEncontrados.push(...Array.from(
                   elementos));
36          } else {
37              console.log(`       N o encontrado: ${seletor
                   }`);
38          }
39      });
40
41      console.log(`   Total de elementos acess veis no DOM
           global: ${elementosEncontrados.length}`);
42
43      console.log('\n2. TESTE DE MODIFICA  O (Write)');
44      if (elementosEncontrados.length > 0) {
45          const nomeProduto = document.querySelector('.name')
                || elementosEncontrados[0];
46          const textoOriginal = nomeProduto.textContent || '
               Sem texto';
47
48          console.log(`   Elemento selecionado: ${nomeProduto
               .tagName}.${nomeProduto.className}`);
49          console.log(`   Texto original (in cio): "${
               textoOriginal.substring(0, 30)}..."`);
50
51          try {
52
```

```
53              nomeProduto.style.border = '3px solid red';
54              nomeProduto.style.padding = '10px';
55              nomeProduto.style.backgroundColor = 'rgba(255,
                    0, 0, 0.1)';
56
57              nomeProduto.textContent = '     PRODUTO
                    MODIFICADO VIA CONSOLE GLOBAL';
58
59              console.log('       MODIFICA  O APLICADA:');
60
61          } catch (error) {
62              console.log('       Erro na modifica  o:',
                    error.message);
63          }
64      } else {
65          console.log('          Nenhum elemento encontrado
                para modificar');
66      }
67
68      console.log('\n3. TESTE ESTRUTURAL (Structure)');
69
70      const containersPossiveis = [
71          '.mfe-container',
72          '.product-grid',
73          '.controls',
74          'app-card',
75      ];
76
77      let containerEncontrado = null;
78      for (const seletor of containersPossiveis) {
79          const container = document.querySelector(seletor);
80          if (container) {
```

```
81              containerEncontrado = container;
82              console.log(`   Container encontrado: ${seletor
                    }`);
83              break;
84          }
85      }
86
87      if (containerEncontrado) {
88          try {
89              const elementoInjetado = document.createElement
                    ('div');
90              elementoInjetado.innerHTML = `
91                  <div style="
92                      border: 4px dashed #ff0000;
93                      background: #fffacd;
94                      padding: 20px;
95                      margin: 15px 0;
96                      font-weight: bold;
97                      text-align: center;
98                      font-family: sans-serif;
99                  ">
100                         ELEMENTO INJETADO VIA CONSOLE
                        GLOBAL<br>
101                     <small>Se voc  v  isso, o DOM do MFE
                        n o est  encapsulado pelo Shadow
                        DOM</small>
102                 </div>
103             `;
104
105
106             containerEncontrado.insertBefore(
                    elementoInjetado, containerEncontrado.
```

```
                firstChild);
107
108             console.log('       ELEMENTO INJETADO COM
                    SUCESSO');
109
110         } catch (error) {
111             console.log('       Erro na injeção:', error.
                    message);
112         }
113     } else {
114         console.log('          Nenhum container adequado
                encontrado');
115     }
116
117     console.log('\n---');
118     console.log('     RESUMO E CLASSIFICAÇÃO');
119     console.log('---');
120
121     const leituraPossivel = elementosEncontrados.length >
            0;
122     const modificacaoPossivel = elementosEncontrados.length
            > 0;
123     const injecaoPossivel = containerEncontrado !== null;
124
125     console.log(`Leitura de elementos: ${leituraPossivel ?
            '    POSSÍVEL' : '    IMPOSSÍVEL'}`);
126     console.log(`Modificação de conteúdo: ${
            modificacaoPossivel ? '    POSSÍVEL' : '
            IMPOSSÍVEL'}`);
127     console.log(`Injeção estrutural: ${injecaoPossivel ?
            '    POSSÍVEL' : '    IMPOSSÍVEL'}`);
128 })();
```

Código-fonte 3 – Script de Exposição de Elementos nos Detalhes do Produto (Angular)

```
(async function() {
    console.log('     INICIANDO TESTE DE ISOLAMENTO
       ESTRUTURAL DO DOM');
    console.log('MFE alvo: Detalhes do Produto (Angular)');
    console.log('---');

    await new Promise(resolve => setTimeout(resolve, 2000))
       ;

    console.log('1. TESTE DE LEITURA (Read)');

    const seletoresParaTestar = [
        '.page-container',
        '.product-layout',
        '.image-box',
        '.info-box',
        '.buy-card',
        '.main-img',
        '.breadcrumbs',
        '.headline-large',
        '.display-large',
        '.desc',
        '.specs',
        '.label-large',
        '.body-small',
        '.btn-add',
        '.btn-buy',
        '.cep-area',
        '.cep-input-group',
        '.cep-input',
        '.btn-icon',
```

```
30          '.frete-txt',
31          '.back-link',
32          '.material-icons',
33          'ul',
34          'li',
35          'strong',
36          'button',
37          'img'
38      ];
39
40      let elementosEncontrados = [];
41      seletoresParaTestar.forEach(seletor => {
42          const elementos = document.querySelectorAll(seletor
                );
43          if (elementos.length > 0) {
44              console.log(`        Encontrado: ${seletor} (${
                    elementos.length} elementos)`);
45              elementosEncontrados.push(...Array.from(
                    elementos));
46          } else {
47              console.log(`        N o encontrado: ${seletor
                    }`);
48          }
49      });
50
51      console.log(`   Total de elementos acess veis no DOM
            global: ${elementosEncontrados.length}`);
52
53      console.log('\n2. TESTE DE MODIFICA  O (Write)');
54
55      if (elementosEncontrados.length > 0) {
56          const nomeProduto = document.querySelector('.
```

```
        headline-large') || elementosEncontrados[0];
57      const textoOriginal = nomeProduto.textContent || '
        Sem texto';
58
59      console.log(`   Elemento selecionado: ${nomeProduto
        .tagName}.${nomeProduto.className}`);
60      console.log(`   Texto original (in cio): "${
        textoOriginal.substring(0, 30)}..."`);
61
62      try {
63          nomeProduto.style.border = '3px solid purple';
64          nomeProduto.style.padding = '10px';
65          nomeProduto.style.backgroundColor = 'rgba(128,
            0, 128, 0.1)';
66
67          nomeProduto.textContent = '     PRODUTO
            MODIFICADO VIA CONSOLE GLOBAL';
68
69          console.log('       MODIFICA  O APLICADA:');
70
71      } catch (error) {
72          console.log('       Erro na modifica  o:',
            error.message);
73      }
74  } else {
75      console.log('          Nenhum elemento encontrado
        para modificar');
76  }
77
78  console.log('\n3. TESTE ESTRUTURAL (Structure)');
79
80  const containersPossiveis = [
```

```
81          '.page-container',
82          '.product-layout',
83          '.info-box',
84          '.buy-card',
85      ];
86
87      let containerEncontrado = null;
88      for (const seletor of containersPossiveis) {
89          const container = document.querySelector(seletor);
90          if (container) {
91              containerEncontrado = container;
92              console.log(`   Container encontrado: ${seletor
                    }`);
93              break;
94          }
95      }
96
97      if (containerEncontrado) {
98          try {
99              const elementoInjetado = document.createElement
                    ('div');
100             elementoInjetado.innerHTML = `
101                 <div style="
102                     border: 4px dashed #800080;
103                     background: #f0e6f5;
104                     padding: 20px;
105                     margin: 15px 0;
106                     font-weight: bold;
107                     text-align: center;
108                     font-family: sans-serif;
109                 ">
110                         ELEMENTO INJETADO VIA CONSOLE
```

```
                    GLOBAL (ANGULAR)<br>
111                 <small>Se voc  v  isso, o DOM do MFE
                    Angular n o est  encapsulado pelo
                    Shadow DOM</small>
112             </div>
113         `;
114
115         containerEncontrado.insertBefore(
                elementoInjetado, containerEncontrado.
                firstChild);
116
117         console.log('       ELEMENTO INJETADO COM
                SUCESSO');
118
119     } catch (error) {
120         console.log('       Erro na inje  o:', error.
                message);
121     }
122 } else {
123     console.log('          Nenhum container adequado
            encontrado');
124 }
125
126 console.log('\n4. TESTE ESPEC FICO PARA ESTRUTURA
        ANGULAR');
127
128 const angularSelectors = [
129     '*[ng-version]',
130     '*[ng-reflect]',
131     '*[_ngcontent]',
132     '*[ng-component]',
133     'app-root',
```

```
134         'app-product-detail'
135     ];
136 
137     angularSelectors.forEach(selector => {
138         const elements = document.querySelectorAll(selector
                );
139         if (elements.length > 0) {
140             console.log(`       Elementos Angular
                    encontrados: ${selector} (${elements.length
                    })`);
141 
142             elements.forEach((el, idx) => {
143                 console.log(`      Elemento ${idx + 1}:`);
144                 console.log(`        Tag: ${el.tagName}`);
145 
146                 if (el.hasAttribute('ng-version')) {
147                     console.log(`        Vers o Angular: $
                            {el.getAttribute('ng-version')}`);
148                 }
149 
150                 const ngContentAttrs = Array.from(el.
                    attributes)
151                     .filter(attr => attr.name.startsWith('
                            _ngcontent'))
152                     .map(attr => attr.name);
153 
154                 if (ngContentAttrs.length > 0) {
155                     console.log(`        Atributos de
                            encapsulamento: ${ngContentAttrs.
                            join(', ')}`);
156                 }
157             });
```

```
158         }
159     });
160 
161     console.log('\n---');
162     console.log('     RESUMO E CLASSIFICA  O ');
163     console.log('---');
164 
165     const leituraPossivel = elementosEncontrados.length >
          0;
166     const modificacaoPossivel = elementosEncontrados.length
          > 0;
167     const injecaoPossivel = containerEncontrado !== null;
168 
169     console.log(`Leitura de elementos: ${leituraPossivel ?
          '    POSS VEL' : '    IMPOSS VEL'}`);
170     console.log(`Modifica  o de conte do: ${
          modificacaoPossivel ? '    POSS VEL' : '
          IMPOSS VEL'}`);
171     console.log(`Inje  o estrutural: ${injecaoPossivel ?
          '    POSS VEL' : '    IMPOSS VEL'}`);
172 })();
```

Código-fonte 4 – Script de Exposição de Elementos no Carrinho de Compras (React)

```
1 (async function() {
2     console.log('     INICIANDO TESTE DE ISOLAMENTO
          ESTRUTURAL DO DOM=');
3     console.log('MFE alvo: Carrinho de Compras (React)');
4     console.log('---');
5 
6     await new Promise(resolve => setTimeout(resolve, 1500))
          ;
```

```
7
8       console.log('1. TESTE DE LEITURA (Read)');
9
10      const seletoresParaTestar = [
11          '.cart-container',
12          '.cart-items',
13          '.cart-item',
14          '.card-main',
15          '.card-image',
16          '.card-details',
17          '.card-actions',
18          '.card-total',
19          '.summary-card',
20          '.item-total-info',
21          '.quantity-control',
22          '.delete-btn',
23          'app-button',
24          '.material-icons',
25          '.headline-large',
26          '.title-large',
27          '.title-small',
28          '.display-medium',
29          '.display-small',
30          'button',
31          'img'
32      ];
33
34      let elementosEncontrados = [];
35      seletoresParaTestar.forEach(seletor => {
36          const elementos = document.querySelectorAll(seletor
                );
37          if (elementos.length > 0) {
```

```
38              console.log(`        Encontrado: ${seletor} (${
                    elementos.length} elementos)`);
39              elementosEncontrados.push(...Array.from(
                    elementos));
40          } else {
41              console.log(`        N o encontrado: ${seletor
                    }`);
42          }
43      });
44
45      console.log(`   Total de elementos acess veis no DOM
            global: ${elementosEncontrados.length}`);
46
47      console.log('\n2. TESTE DE MODIFICA  O (Write)');
48
49      if (elementosEncontrados.length > 0) {
50
51          const tituloCarrinho = document.querySelector('.
                display-medium') || document.querySelector('.
                headline-large') || elementosEncontrados[0];
52          const textoOriginal = tituloCarrinho.textContent ||
                'Sem texto';
53
54          console.log(`   Elemento selecionado: ${
                tituloCarrinho.tagName}.${tituloCarrinho.
                className}`);
55          console.log(`   Texto original: "${textoOriginal.
                substring(0, 40)}..."`);
56
57
58          try {
59
```

```
60              tituloCarrinho.style.border = '3px solid blue';
61              tituloCarrinho.style.padding = '15px';
62              tituloCarrinho.style.backgroundColor = 'rgba(0,
                    0, 255, 0.1)';
63
64
65              tituloCarrinho.textContent = '     CARRINHO
                   MODIFICADO VIA CONSOLE GLOBAL';
66
67              console.log('       MODIFICA  O APLICADA:');
68
69          } catch (error) {
70              console.log('       Erro na modifica  o:',
                   error.message);
71          }
72      } else {
73          console.log('          Nenhum elemento encontrado
               para modificar');
74      }
75
76      console.log('\n3. TESTE ESTRUTURAL (Structure)');
77
78      const containersPossiveis = [
79          '.cart-container',
80          '.content-wrapper',
81          '.cart-items',
82          '.summary-card',
83      ];
84
85      let containerEncontrado = null;
86      for (const seletor of containersPossiveis) {
87          const container = document.querySelector(seletor);
```

```
88          if (container) {
89              containerEncontrado = container;
90              console.log(`   Container encontrado: ${seletor
                    }`);
91              break;
92          }
93      }
94
95      if (containerEncontrado) {
96          try {
97              const elementoInjetado = document.createElement
                    ('div');
98              elementoInjetado.innerHTML = `
99                  <div style="
100                     border: 4px dashed #0000ff;
101                     background: #e6f7ff;
102                     padding: 20px;
103                     margin: 15px 0;
104                     font-weight: bold;
105                     text-align: center;
106                     font-family: sans-serif;
107                 ">
108                         ELEMENTO INJETADO VIA CONSOLE
                            GLOBAL (CARRINHO)<br>
109                     <small>Se voc  v  isso, o DOM do MFE
                            n o est  encapsulado pelo Shadow
                            DOM</small>
110                 </div>
111             `;
112
113
114             containerEncontrado.insertBefore(
```

```
            elementoInjetado, containerEncontrado.
                firstChild);
115
116             console.log('       ELEMENTO INJETADO COM
                    SUCESSO');
117
118         } catch (error) {
119             console.log('       Erro na inje  o:', error.
                    message);
120         }
121     } else {
122         console.log('          Nenhum container adequado
                encontrado');
123     }
124
125     console.log('\n---');
126     console.log('     RESUMO E CLASSIFICA  O ');
127     console.log('---');
128
129     const leituraPossivel = elementosEncontrados.length >
            0;
130     const modificacaoPossivel = elementosEncontrados.length
             > 0;
131     const injecaoPossivel = containerEncontrado !== null;
132
133     console.log(`Leitura de elementos: ${leituraPossivel ?
            '    POSS VEL' : '    IMPOSS VEL'}`);
134     console.log(`Modifica  o de conte do: ${
            modificacaoPossivel ? '    POSS VEL' : '
            IMPOSS VEL'}`);
135     console.log(`Inje  o estrutural: ${injecaoPossivel ?
            '    POSS VEL' : '    IMPOSS VEL'}`);
```

```
136 })();
```