Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Micro-Frontends: Estratégias e Diretrizes para Lidar com os Impactos e Desafios da sua Adoção

Giovanni Cunha de Amorim

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientadora
Prof.a Dr.a Edna Dias Canedo

Brasília
2025

Ficha Catalográfica

AA524m	Amorim, Giovanni Micro-frontends: Estratégias e Diretrizes para lidar com os Impactos e Desafios da sua Adoção / Giovanni Amorim; Orientadora: Edna Dias Canedo. Brasília 2025. 138 p. Dissertação (Mestrado Profissional em Computação Aplicada) Universidade de Brasília, 2025. 1. micro-frontend. 2. microsservicos. 3. monolito. 4. arquitetura de software. 5. desenvolvimento web. I. Canedo, Edna

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Micro-Frontends: Estratégias e Diretrizes para Lidar com os Impactos e Desafios da sua Adoção

Giovanni Cunha de Amorim

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof.a Dr.a Edna Dias Canedo (Orientadora)
Universidade de Brasília (UnB)

Prof. Dr. Sérgio Antônio A. de Freitas
Universidade de Brasília (UnB)

Prof. Dr. Gilmar dos Santos Marques
União Pioneira de Integração Social(UPIS)

Prof.a Dr.a Edna Dias Canedo
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 20 de Fevereiro de 2025

Dedicatória

A Deus e à minha família, minha esposa e minha filha, que são a base da minha força e perseverança, dedico o resultado deste trabalho.

Agradecimentos

A Deus em primeiro lugar, agradeço pela força, sabedoria e a oportunidade de realizar este trabalho. Gostaria de expressar meu mais profundo amor e gratidão à minha esposa Mônica Porto e à minha filha Giovanna Porto de Amorim pelo constante amor, apoio e compreensão. Vocês foram fundamentais para que eu pudesse concluir esta fase das nossas vidas.

À Prof[a]. Edna Dias Canedo pela confiança depositada em mim e no meu trabalho e pela paciência. Sua orientação excepcional foi crucial para o desenvolvimento deste estudo. Sou grato pelas oportunidades que você me proporcionou e espero continuar colaborando com você em outras oportunidades.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Contexto: A complexidade associada à incorporação de novas tecnologias em projetos monolíticos da camada frontend tem representado um desafio para às equipes de desenvolvimento de software. Enquanto o backend avançou com microsserviços, o frontend enfrentou obstáculos relacionados à redundância de código, consistência, escalabilidade e ausência de uma abordagem modular. Em resposta a esses desafios, surgiu a arquitetura de micro-frontends, propondo uma arquitetura modular e mais independente. **Objetivo:** Investigar os impactos e desafios associados à adoção da arquitetura de micro-frontends e fornecer diretrizes práticas para as organizações implementá-las. **Método:** Conduzimos uma Revisão Sistemática da Literatura (SLR) para identificar os padrões e visões arquiteturais utilizados nos micro-frontends. Além disso, conduzimos dois surveys para validar estratégias e diretrizes na adoção da arquitetura. **Resultados:** Na percepção dos participantes dos surveys o guia proposto pode facilitar o processo de adoção da arquitetura de micro-frontends. Quanto ao estudo de caso, identificamos aspectos relacionados à implementação prática, abrangendo desempenho e escalabilidade, manutenção, colaboração e desenvolvimento paralelo. Os participantes também relataram complexidade associada à gestão de múltiplos módulos menores, bem como a falta de experiência por parte dos desenvolvedores com as tecnologias da arquitetura de micro-frontends. O estudo de caso também ratificou os achados dos surveys. **Conclusão:** As descobertas destacaram a relevância da arquitetura Micro-frontends na modernização da camada frontend. No entanto, é importante ressaltar que a necessidade de sua adoção está fortemente relacionada à complexidade e tamanho do projeto e do time. Assim, a adoção da arquitetura de micro-frontends com o apoio de um conjunto de decisões metodológicas e tecnológicas possui uma chance maior de sucesso na sua implementação.

Palavras-chave: micro-frontend, microsserviços, monolito, arquitetura de software , desenvolvimento web

Abstract

Context: The complexity associated with incorporating new technologies into monolithic frontend projects has posed a challenge for software development teams. While the back-end has advanced with microservices, the front-end has faced challenges related to code redundancy, consistency, scalability, and the lack of a modular approach. In response to these challenges, the micro-frontend architecture emerged, proposing a more modular and independent architecture. **Objective:** To investigate the impacts and challenges associated with adopting the micro-frontend architecture and to provide practical guidelines for organizations to implement it. **Method:** We conducted a Systematic Literature Review (SLR) to identify the architectural patterns and visions used in micro-frontends. In addition, we conducted two surveys to validate strategies and guidelines for adopting the architecture. **Results:** In the perception of the survey participants, the proposed guide can facilitate the process of adopting the micro-frontend architecture. In terms of the case study, we identified aspects related to practical implementation, covering performance and scalability, maintenance, collaboration, and parallel development. Participants also reported complexity associated with managing multiple smaller modules, as well as the lack of developer experience with micro-frontend architecture technologies. The case study also corroborated the findings of the surveys. **Conclusion:** The findings highlighted the relevance of the Micro-frontend architecture in modernizing the frontend layer. However, it is important to note that the need for its adoption is strongly related to the complexity and size of the project and team. Thus, adopting the micro-frontend architecture with the support of a set of methodological and technological decisions has a greater chance of success in its implementation.

Keywords: micro-frontend, microservices, monolith, software architecture, web development

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A busca contínua pela eficiência no desenvolvimento de software é uma prioridade para as organizações, independentemente de estarem criando novos aplicativos ou mantendo sistemas legados através de incrementos [3]. No entanto, esse esforço é ainda mais complexo em um ambiente onde a pressão por entregas rápidas está constantemente aumentando [4]. Entre os desafios enfrentados, destacam-se a necessidade de tornar as implantações mais independentes, permitindo que diferentes partes do sistema evoluam autonomamente. Problemas organizacionais, como estruturas rígidas e falhas na comunicação, dificultam a colaboração entre equipes, ampliando a crescente complexidade do processo. Além disso, o aumento do tamanho do código, especialmente à medida que as aplicações se tornam mais intrincadas, contribui para a dificuldade em alcançar uma maior frequência e eficiência nos lançamentos de versões de software, agravando o desafio do desenvolvimento de software e a necessidade de soluções mais eficientes [5].

Por volta de 2009, surgiu a abordagem de microsserviços como uma evolução da arquitetura *Services Oriented Arquitecture* (SOA) [6]. Esta abordagem prometeu a decomposição de aplicações monolíticas complexas em subdomínios menores, cada um fornecendo serviços, testes e implantações de forma isolada [7], o que poderia ser uma solução para os desafios mencionados. Conforme destacou Yang et al. [8], os microsserviços são um conjunto de serviços interconectados de maneira flexível e com baixo grau de dependência mútua. São projetados de maneira modular, consistindo em pequenos blocos funcionais que se comunicam através de *Application Programming Interface* (API). Cada bloco funcional possui uma responsabilidade clara e função específica, permitindo desenvolvimento, testes e implantações independentes. O resultado é a criação de sistemas de software mais flexíveis, robustos e ágeis, oferecendo uma experiência aprimorada aos usuários finais [8].

Em 2021, a IBM publicou os resultados de uma pesquisa com clientes, evidenciando como as organizações ganharam velocidade, escalabilidade e resiliência ao adotar a abordagem de microsserviços em seus projetos [9]. Além disso, a pesquisa destacou que a

capacidade de reutilizar componentes, adotar a integração e entrega contínua e gerenciar eficazmente equipes distribuídas são aspectos adicionais que amplificam o valor dos microsserviços.

Contudo, essa abordagem não foi inicialmente aplicada ao *frontend* [1]. Somente em novembro de 2016 o conceito de micro-frontend foi introduzido por Geers[2] no periódico ThoughtWorks Technology Radar [10], e agora é adotado por grandes companhias como SAP, Springer, Zalando, New Relic, Ikea, Starbucks, DAZN e outras [11]. Geers[2] observou que a camada de frontend, normalmente desenvolvida por uma única equipe, enfrentava desafios significativos à medida que crescia em complexidade. A manutenção ficava cada vez mais onerosa, e tornou-se evidente que uma solução semelhante a dos microsserviços era necessária para tratar estes problemas e replicar seu impacto transformador na indústria de software [12].

Davidi e Mezzalira [1] abordaram os princípios dos microsserviços aplicados ao desenvolvimento de aplicações frontend que validam as observações de Geers [1, 2]. Esses princípios incluem o *Domain-Driven Design* (DDD), automação, ocultação de detalhes de implementação, descentralização da governança, deploy independente, isolamento de falhas e alta observabilidade. Os princípios promovem a criação de sistemas de software que atendem melhor às necessidades do domínio de negócios, facilitam a automação de tarefas, escondem detalhes técnicos desnecessários, permitem decisões descentralizadas, possibilitam atualizações independentes, reduzem o impacto de falhas e permitem o monitoramento em tempo real do sistema.

Yang et al.[8], exploraram outras estratégias do micro-frontend, que se mostraram aplicáveis ao contexto do Content Management System (CMS), uma ferramenta que oferece forte suporte à informatização e se torna uma parte importante da construção de informações empresariais e do governo eletrônico. Para os autores, o design do CMS baseado em micro-frontend permite que as equipes desenvolvam de forma independente, implementem rapidamente e testem individualmente, ajudando na integração contínua, implantação contínua e entrega contínua. Davide e Mezzalira [1] apontaram estratégias semelhantes. Os autores destacaram que, ao contrário dos monolitos, a arquitetura dos micro-frontends pressupõe que cada equipe seja responsável por uma área específica do negócio ou por funções bem definidas, concentrando seus esforços na implementação abrangente, desde o desenvolvimento do *backend* [2] até o frontend. Essas estratégias estão alinhadas com os domínios de negócios, o que facilita a ocultação dos detalhes de implementação entre as equipes e suas respectivas aplicações. Assim, cada micro-frontend é projetado de ma-

[1] Interface de usuário visível e interativa, que inclui elementos como botões, formulários, layouts e tudo o que os usuários veem e utilizam ao interagir com uma aplicação web.

[2] O backend é a parte de um sistema que lida com o processamento e armazenamento de dados, funcionando nos bastidores, sem interação direta com o usuário.

neira que não seja necessário conhecer os detalhes internos de implementação de outros micro-frontends para interagirem entre si ou utilizá-los [13].

De forma geral, esses trabalhos apontam que a adoção do micro-frontend visa, sobretudo, a criação de uma arquitetura de software com menor dependência de tecnologias específicas. Isso significa uma maior flexibilidade na escolha e na incorporação de diferentes linguagens de programação, como JavaScript e TypeScript, assim como a integração de diversos frameworks, a exemplo de Angular, ReactJS e Vue, e outras tecnologias e metodologias de acordo com as necessidades específicas de cada projeto [14]. Essa capacidade de escolha da *stack* tecnológica, anteriormente se encontrava limitada no contexto dos monolitos frontend [15]. No entanto, embora traga consigo inúmeras vantagens, o micro-frontend não pode ser considerado como uma solução mágica que resolverá todos os problemas na camada de frontend [16]. Esta arquitetura também apresenta desvantagens significativas. Em determinados contextos de projetos, como o de migração por exemplo, as desvantagens podem superar as vantagens, tornando os micro-frontends uma escolha menos apropriada [1, 14]. Portanto, deve-se avaliar cuidadosamente as circunstâncias do projeto para determinar sua adequação e maximizar seus benefícios.

1.1 Problema de Pesquisa

A crescente adoção de arquiteturas baseadas em micro-frontends por organizações de grande porte representa uma tendência no desenvolvimento de software [17]. Isso reflete a busca constante por abordagens mais flexíveis e eficazes na construção de aplicações frontends [12]. Pölöskei e Bub [15] afirmaram que algumas organizações desenvolveram seus portfólios de aplicativos fragmentando os monolitos como se fossem ilhas, com endpoints web separados para cada serviço. Entretanto, lidar com um monolito de grande porte utilizando essa estratégia pode resultar em menor eficiência operacional. Com micro-frontends, é concebível mesclar todo o portfólio de aplicativos distribuídos através da implementação de um único portal. À medida que as organizações enfrentam a necessidade de evoluir suas aplicações em um ritmo acelerado, o conceito de micro-frontend tem se destacado como uma resposta eficaz em contexto como o mencionado por Pölöskei e Bub [15], permitindo que diferentes partes de uma aplicação sejam desenvolvidas e implantadas de forma independente, promovendo escalabilidade e agilidade no desenvolvimento [6].

No entanto, ao decidir implementar arquiteturas baseadas em micro-frontends, as organizações se deparam com algumas desvantagens. Para Geers [2], essas desvantagens podem ser agrupadas em quatro categorias: 1) redundância de códigos, 2) consistência, 3) heterogeneidade, e 4) mais código no frontend.

A “redundância de códigos” ocorre devido ao fato de que várias equipes frequentemente trabalham em diferentes partes da aplicação, o que pode levar à duplicação de esforços e código [15]. A “consistência” ocorre quando várias equipes estão envolvidas no desenvolvimento de diferentes módulos ou funcionalidades, e o conjunto de elementos e fatores relativos à interação do usuário com a aplicação, também conhecida como User Experience (UX)[17], não resulta em uma percepção positiva, coerente e consistente, levando à perda de eficiência no desenvolvimento [15].

A categoria da “heterogeneidade” aborda a flexibilidade de escolher diferentes stacks tecnológicas para diferentes partes da aplicação[1]. Geers [2] afirma-se que a capacidade de usar várias stacks tecnológicas não implica necessariamente que se deva empregar várias delas indiscriminadamente. Quando todas as equipes optam por usar as mesmas tecnologias, os principais benefícios das atualizações autônomas de versões e da menor sobrecarga de comunicação ainda permanecem. Porém, usar várias tecnologias causa heterogeneidade, e consequentemente aumenta a complexidade na integração e na manutenção, criando desafios adicionais para a equipe de desenvolvimento.

Por fim, a categoria de “mais código no frontend” que trata da necessidade de se escrever mais códigos javascript e css, o que também pode causar redundância e prejudicar a performance da aplicação. Davide e Mezzalira [11] também propuseram quatro decisões chaves que devem ser consideradas antes de iniciar um projeto com micro-frontends: a primeira é a decisão de “Divisão Horizontal ou Vertical”, que determina se diferentes equipes trabalharão em diferentes frontends na mesma visualização (divisão horizontal) ou se cada equipe será responsável por uma visualização específica (divisão vertical); a segunda é a decisão de “Lado de Composição”, que envolve escolher entre a composição no lado do cliente, onde cada micro-frontend é carregado dinamicamente no navegador, ou no lado do servidor; a terceira decisão é a de “Roteamento”, que se refere à maneira como as solicitações do usuário são direcionadas para o micro-frontend apropriado; e, por fim, a quarta decisão de “Comunicação entre os Micro-frontends”, que aborda como facilitar a comunicação entre as aplicações, seja através do uso de eventos personalizados ou do compartilhamento de serviços.

Diante dos problemas apresentados de ineficiência dos monolitos de grande porte [14]; da redundância de código, consistência, heterogeneidade e excesso de códigos [2] e os problemas de decisões arquiteturais que tratam sobre composição, roteamento, comunicação entre as aplicações e a decisão de adotar um design arquitetural vertical ou horizontal [11], buscaremos validar os desafios mencionados pela literatura e identificar seus impactos na construção de aplicações web complexas contextualizando-os a um estudo de caso aplicado na companhia Serasa Experian. A companhia, é uma das principais organizações no setor de serviços financeiros e de crédito, presente em mais de 30 países [18], e vem adotando

a abordagem de micro-frontends como parte de seus esforços contínuos para aprimorar a eficiência no desenvolvimento de frontend. Este estudo de caso busca explorar como a empresa aplicou a arquitetura de micro-frontends, obtendo benefícios significativos em termos de flexibilidade, colaboração e eficiência no desenvolvimento.

Neste estudo, pretendemos não apenas explorar os benefícios potenciais da abordagem de micro-frontends, mas também lançar luz sobre possíveis problemas e impactos adicionais identificados no estudo de caso da empresa Serasa Experian e que ainda não foram completamente identificados pela literatura existente. Estes desafios incluem, mas não se limitam, à gestão de dependências entre os diferentes módulos do frontend[19], garantia de consistência na interface do usuário[1], flexibilidade da modularização, manutenção e desenvolvimento contínuo de aplicações, escalabilidade do processo de construção do software[2] e separação de responsabilidades entre as equipes de desenvolvimento, que pode levar a uma maior eficiência e escalabilidade no processo de construção de software.

Outro aspecto que merece atenção especial e que foi considerado e amplamente discutido pelo time de desenvolvimento frontend na Serasa Experian, é a decisão pela escolha da stack tecnológica ideal. Por exemplo, embora existam diversos frameworks de micro-frontends disponíveis, nem todos podem ser aplicados de forma universal, especialmente quando a arquitetura de software precisa se adaptar à evolução do negócio [11]. Atualmente, os frameworks mais populares para micro-frontends incluem Single-Spa[3], Module Federation (parte da família Webpack)[4], e Luigi da Zalando[5]. A escolha da *stack* tecnológica mais apropriada dependerá das necessidades específicas de cada projeto e da compatibilidade com a evolução da arquitetura de software, no caso da Serasa Experian a escolha foi pelo framework Single-SPA depois de uma minuciosa análise de outras opções que incluiu Provas de Conceitos (POC). Outro exemplo importante na indústria é o da Spotify, que usou a arquitetura de iFrames para seus aplicativos de desktop e web, no entanto, a empresa decidiu abandonar essa abordagem no aplicativo web devido ao baixo desempenho e passaram a adotar uma arquitetura baseada no framework React aliado à biblioteca Redux, o que tornou mais fácil o compartilhamento entre componentes e a visualizações, deixando o fluxo de dados claro e melhorando a capacidade de depuração e testabilidade do sistema[20].

Portanto, a justificativa deste estudo não se limita apenas a identificar essas questões, mas também fornecer orientações práticas e insights para as organizações que estão considerando a adoção desta arquitetura. Ao fazer isso, esperamos contribuir para a compreensão das complexidades inerentes à adoção de micro-frontends e, assim, auxiliar a

[3]https://single-spa.js.org/
[4]https://webpack.js.org/concepts/module-federation/
[5]https://luigi-project.io/

indústria na escolha das melhores tecnologias e metodologias para uma implementação mais eficaz e eficiente[15].

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é investigar os impactos e desafios associados à adoção de micro-frontends no desenvolvimento de aplicações web complexas, explorando possíveis soluções para esses desafios, com o intuito de fornecer diretrizes práticas às organizações que desejam implementar essa abordagem arquitetural. Um estudo de caso foi realizado nos sistemas da companhia Serasa Experian a fim de avaliar essas soluções.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Investigar os motivos que levam as organizações a adotarem arquiteturas baseadas em micro-frontends e avaliar o impacto dessa adoção em seus processos de desenvolvimento de software.
- Analisar os desafios enfrentados pelas organizações ao implementar micro-frontends.
- Avaliar os benefícios percebidos pelas organizações que adotaram micro-frontends, como a capacidade de atualizar partes independentes da aplicação e melhorar a escalabilidade.
- Identificar as práticas recomendadas e os padrões que podem ser aplicados para enfrentar os desafios associados à adoção de micro-frontends.
- Realizar um estudo de caso prático, com o objetivo de entender as experiências, lições aprendidas e soluções específicas desenvolvidas para superar desafios relacionados à adoção de micro-frontends.
- Propor recomendações práticas para organizações que desejam adotar ou otimizar arquiteturas baseadas em micro-frontends, com base nas descobertas e análises desse estudo.

1.2.3 Resultados Esperados e Contribuições

- Identificar as melhores práticas adotadas pela indústria ao implementar a arquitetura micro-frontend.
- Apresentar um guia das soluções tecnológicas utilizadas pelas organizações.
- Apresentar um modelo prático de priorização e definição de indicadores específicos para a viabilidade da adoção da arquitetura baseadas em micro-frontends.

1.2.4 Metodologia

Esta pesquisa adota uma abordagem em fases, como demonstrado na Figura 1.1. Inicialmente, conduziu-se uma Revisão Sistemática da Literatura (RSL) para estabelecer uma base de conhecimento que fundamentou na criação do Guia para Adoção de Micro-Frontends (GAM) na fase subsequente. As fases subsequentes compreendem iterações dedicadas à avaliação e ao aprimoramento contínuo do GAM, incorporando processos de validação e verificação para garantir sua eficácia e aplicabilidade prática. Com relação as versões do GAM, foi seguido as orientações propostas pelo Versionamento Semântico[6], a primeira versão (v1.0.0-alpha) desenvolvida na fase 2, foi a primeira versão estável de pré-lançamento, em seguida esta versão foi atualizada para a versão v1.0.0-beta com as validações do survey A na fase 3. Durante a fase 4 foi aplicado as verificações do estudo de caso e a versão subiu para v1.0.0-RC. Finalmente, após a validação final aplicando o survey B, a versão final do GAM foi lançada como v1.0.0.

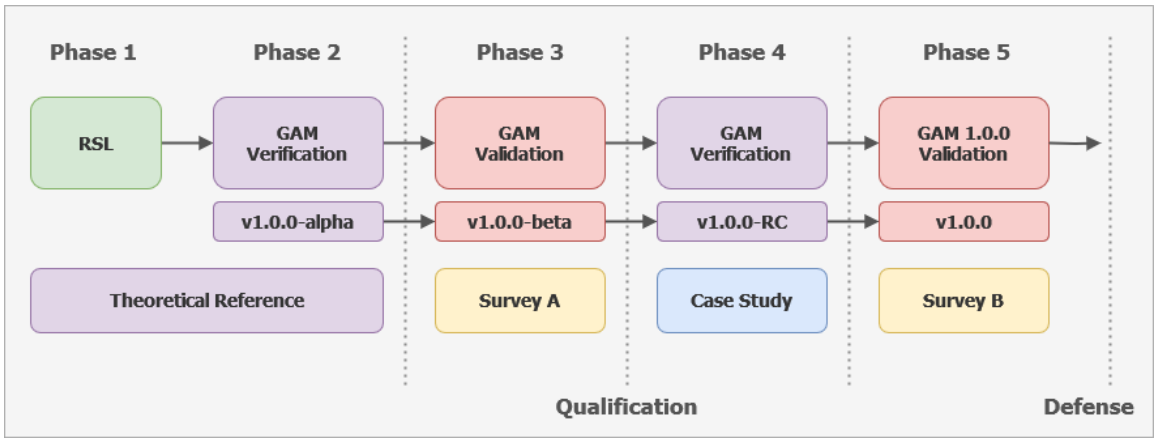


Figura 1.1: Fases da Execução da Pesquisa

A Tabela 1.1 retrata os métodos empregados em cada fase da metodologia, juntamente com o propósito de utilizar cada método. A Revisão Sistemática da Literatura (SLR) é

[6]https://semver.org/lang/pt-BR/

complementada por uma análise utilizando o Referencial Teórico, a qual auxiliará na extração de dados e na transformação desses dados em diretrizes que integrarão o GAM. Por outro lado, os demais métodos fazem parte do processo de verificação e validação. Validaremos o guia tanto de forma quantitativa, por meio de uma pesquisa, quanto de forma qualitativa, por meio de um estudo de caso. Primeiramente, conduzimos um Survey Online com usuários especialistas em frontend para investigar suas percepções sobre a arquitetura Micro-frontend e os fluxos propostos pelo GAM. Em seguida, iniciamos o Estudo de Caso (5) com o propósito de validar o método, aplicando o GAM em um Projeto Piloto. Neste contexto, os desafios enfrentados e as técnicas aplicadas durante a execução do Projeto Piloto são incorporados e validados no GAM. Posteriormente, realizamos a validação do GAM por meio de um Survey online específico para esta fase.

Método	**Fase**	**Propósito**	**Descrição**
RSL	3	1	Investigar as estratégias adotadas para a implementação de micro-frontends e compreender seu impacto na aplicação da arquitetura.
Referencial Teórico	2	1 e 2	Elaborar o GAM, a partir dos resultados da SLR.
Surveys	3 e 5		Validar o guia com desenvolvedores frontend na Fase 3 e verificar a eficácia do guia com desenvolvedores na Fase 5.
Estudo de Caso	5	4 a 6	Desenvolver o projeto Piloto e verificar a eficácia do guia após ser avaliado por membros das equipes de desenvolvimento de frontend.

Tabela 1.1: Composição dos métodos no contexto do GAM

A Tabela 1.1 descreve a composição dos métodos de design da pesquisa no contexto do GAM, destacando suas fases correspondentes e propósitos específicos.

Metodologia para o Desenvolvimento do GAM

Os passos da metodologia para o desenvolvimento do GAM incorporam os princípios fundamentais presentes no Guia SWEBOK desenvolvido e mantido pela IEEE Computer Society [21]. Para garantir um alinhamento com o guia, foram realizadas algumas adaptações. A Tabela 1.2 apresenta uma comparação entre as fases do GAM **??** e as áreas de conhecimentos (Knowledge Area - KAs) do Swebok, destacando as nuances e correspondências entre elas.

Verificação e Validação do GAM

A verificação e validação desempenham papéis importantes em qualquer metodologia de desenvolvimento de software, incluindo no GAM. Neste contexto, a verificação consistiu

Fase do GAM	Swebok KAs	Correlação
Estudo da Viabilidade	Gerenciamento de Engenharia de Software, Economia da Engenharia de Software	Durante o estudo da viabilidade, a gestão de engenharia de software e aspectos gerenciais devem ser considerados.
Implementação Prática	Design de Software, Construção de Software, Gerenciamento de Configuração de Software, Modelos e Métodos de Engenharia de Software	Na Implementação Prática, o design, construção, gestão de configuração e métodos de engenharia de software são áreas críticas.
Expansão	Manutenção de Software, Qualidade de Software, Modelos e Métodos de Engenharia de Software	A expansão requer um foco em manutenção, qualidade de software e aplicação de modelos e métodos de engenharia.
Projeto Piloto	Processo de Engenharia de Software, Prática Profissional em Engenharia de Software	O processo de engenharia de software e práticas profissionais são fundamentais em todas as fases, incluindo projetos piloto.

Tabela 1.2: Correlação entre Fases do GAM e Swebok KAs

em avaliar se o guia foi construído de maneira adequada, considerando os resultados da Revisão Sistemática da Literatura (RSL) e a incorporação das etapas do processo de adoção de micro-frontends. Por outro lado, com base no projeto piloto, no survey B e em documentações de software, a validação buscou confirmar se o guia pode ser eficaz em sua função principal, ou seja, se é capaz de auxiliar na adoção bem-sucedida de micro-frontends, atendendo às expectativas dos *stakeholders* e produzindo os resultados desejados.

A Tabela 1.3 apresenta a estrutura para a integração de práticas do Guia de Conhecimento de Engenharia de Software (SWEBOK) no Guia de Adoção de Micro-frontends (GAM). Esta correlação fornece uma visão articulada e integrada de como as diversas disciplinas da engenharia de software se aplicam em cada etapa específica do processo de adoção de Micro-frontends. Nas fases do GAM, cada correlação entre as Áreas de Conhecimento (KAs) do SWEBOK destaca aspectos importantes para o sucesso do projeto. No entanto, vale ressaltar que essas correlações não apenas delineiam a relevância teórica, mas também estabelecem diretrizes práticas para verificações e validações eficazes, as quais serão conduzidas por meio de dois Survey direcionados a públicos específicos: 1º Survey, para os desenvolvedores Especialistas em Frontend e o 2º Survey, para Desenvolvedores Especialistas Convidados, este último grupo será encarregado de aplicar o

Fase do GAM	Verificação	Validação
Estudo da Viabilidade	Revisão pelos stakeholders para garantir que as análises de viabilidade técnica e gerencial estejam corretas.	Confirmação de que as análises de viabilidade são precisas e atendem às expectativas dos stakeholders.
Implementação Prática	Revisão do design, construção e gerenciamento de configuração por partes interessadas.	Garantia de que o design, a construção e a configuração do sistema estão em conformidade com os requisitos.
Expansão	Revisão dos planos de manutenção, qualidade e aplicação de modelos e métodos.	Certificação de que as estratégias de expansão consideram adequadamente a manutenção, a qualidade e a aplicação de modelos.
Projeto Piloto	Avaliação do processo de engenharia de software e práticas profissionais em todas as fases, incluindo o projeto piloto.	Confirmação de que o processo e as práticas profissionais estão alinhados com os objetivos do projeto piloto.

Tabela 1.3: Verificação e Validação nas Fases do GAM

GAM no projeto piloto. Na fase Estudo da Viabilidade, onde a gestão de engenharia de software e aspectos econômicos são fundamentais, questões específicas do 1º Survey serão elaborados para revisão dos *stakeholders*. Essas questões visam garantir que as análises de viabilidade técnica e gerencial estejam corretas e atendam às expectativas dos envolvidos. Durante a Implementação Prática, quando o design, construção e gerenciamento de configuração assumem foco, questões baseadas nestes contextos serão incluídas no 1º survey. Essas questões buscarão revisar aspectos específicos, garantindo que o design, construção e configuração do sistema estejam alinhados com os requisitos estabelecidos. Na fase de Expansão, os planos de manutenção, qualidade e aplicação de modelos e métodos serão submetidos à revisão através de questões direcionadas a estes temas, ainda para o público do 1º Survey. A certificação de que as estratégias de expansão consideram adequadamente a manutenção, qualidade e a aplicação de modelos, será validada por meio das respostas desses desenvolvedores e contribuirá com versão do GAM para v2.0-beta. Finalmente, na fase de Projeto Piloto, onde o processo de engenharia de software e práticas profissionais são fundamentais, incluiremos 2º Survey com questões abrangentes relacionadas ao uso do GAM no projeto piloto. Essas questões servirão para avaliação do processo das etapas e serão a base para a construção da versão do GAM v2.0. Essa abordagem de questionários não apenas fortalece as verificações e validações, mas também integra efetivamente a par-

ticipação e feedback contínuo dos stakeholders, contribuindo para a eficácia e a relevância prática do GAM ao longo do ciclo de adoção da arquitetura Micro-frontend.

1.3 Publicações

A referência abaixo aponta para um artigo aceito para publicação em uma conferência internacional. Este artigo apresenta diretrizes e insights para a implementação da arquitetura de micro-frontend no desenvolvimento de software, tema central desta dissertação. Aceito para apresentação na 27th International Conference on Enterprise Information Systems (ICEIS).

As referências abaixo apontam para dois artigo aceitos para publicação em dois eventos científicos, destacando sua relevância no tema da arquitetura de micro-frontend no desenvolvimento de software. Os eventos são a 27th International Conference on Enterprise Information Systems (ICEIS) e o Simpósio Brasileiro de Sistemas de Informação (SBSI 2025) – Inovação com Equidade, Diversidade e Inclusão em Sistemas de Informação. Nestes trabalhos, são apresentadas diretrizes e insights para a implementação da arquitetura de micro-frontend, tema central desta dissertação.

Amorim, G. C., & Canedo, E. D. (2025). Micro-Frontend Architecture in Software Development: A Systematic Mapping Study. In Proceedings of the *27th International Conference on Enterprise Information Systems (ICEIS)*.

Amorim, G. C., Mendes, F. F., Bastos, L. R. S., Santos, R. P., & Canedo, E. D. (2025). Guidelines for Adoption of Micro-frontend Architecture. Simpósio Brasileiro de Sistemas de Informação (SBSI 2025) – Inovação com Equidade, Diversidade e Inclusão em Sistemas de Informação.

1.4 Disponibilidade dos Dados

Todos os dados analisados neste estudo estão disponíveis no Zenodo em:

https://doi.org/10.5281/zenodo.13160674

1.5 Organização da Dissertação

Esta dissertação está organizada em 7 Capítulos, além da Introdução, conforme descritos abaixo.

- Capítulo 2 — Referencial Teórico (Fases 1 e 2): Apresenta teorias e conceitos sobre a arquitetura de Micro-frontend estabelecidas por pesquisadores e pela indústria de software;
- Capítulo 3 — Revisão Sistemática da Literatura (Fase 1): apresenta o protocolo e os resultados da RSL;
- Capítulo 4 — GAM (Fases 2): apresenta como os resultados da RSL foram transformados na primeira versão do GAM e validados para fornecer a segunda versão;
- Capítulo 5 — Estudo de Caso (Fase 3 a 6): apresenta o estudo de caso aplicado ao guia e valida seus métodos por meio da análise do estudo e o resultado do survey direcionado aos membros participantes para gerar a versão final do GAM;
- Capítulo 6 — Discussão: discute as implicações, contribuições e ameaças à validade em relação aos resultados alcançados por este trabalho;
- Capítulo 7 — Conclusão: resume o trabalho realizado e os resultados alcançados

Capítulo 2

Referencial Teórico

Este capítulo tem como objetivo explorar as nuances das arquiteturas contemporâneas, concentrando-se especialmente na evolução da Arquitetura Micro-Frontend. Inicialmente, é feita uma breve referência sobre a tradicional Arquitetura Monolítica, caracterizada por sua abordagem integrada e centralizada. Em seguida, descreve-se sobre a flexibilidade e escalabilidade proporcionadas pela Arquitetura de Microsserviços. Por fim, discute-se as características da Arquitetura Micro-Frontend, uma abordagem que redefine a forma como os componentes frontend são desenvolvidos e implantados.

2.1 Arquitetura Monolítica

A arquitetura monolítica é uma abordagem tradicional no desenvolvimento de software, na qual uma aplicação é concebida como uma entidade única e coesa. Nesse modelo, todas as funcionalidades e componentes são integrados em um único código-fonte e processo de execução. Geralmente, uma aplicação monolítica compartilha um banco de dados central e é desenvolvida como uma unidade indivisível [22]. Apresentaremos duas abordagens para a implementação do monolito, a primeira chamamos de "*Single Schema*" onde todas as funcionalidades, componentes e até mesmo o banco de dados são integrados em um bloco. Essa estratégia simplifica o desenvolvimento e a manutenção, proporcionando uma coesão completa entre os elementos do sistema.

No modelo que separa frontend e backend chamamos de "*Splited Schema*", infere-se a divisão clara de responsabilidades entre a interface do usuário e a lógica de negócios. Essa abordagem promove uma maior modularidade e facilita o desenvolvimento paralelo, já que as camadas podem evoluir de forma independente. Embora essa separação ofereça benefícios em termos de manutenção e escalabilidade, a sobrecarga de comunicação entre as partes do sistema deve ser cuidadosamente considerada ao tomar decisões de arquitetura [22].

A abordagem monolítica oferece diversas vantagens. Em primeiro lugar, destaca-se a simplicidade de desenvolvimento e implantação, característica reconhecida por sua eficiência em projetos de menor complexidade [7]. Além disso, a facilidade de compartilhamento de estado entre os módulos da aplicação simplifica a comunicação e o intercâmbio de dados entre diferentes partes do sistema. Outra vantagem significativa é a manutenção centralizada, pois a centralização de todos os componentes em um único código-base facilita as atualizações e correções, proporcionando uma gestão mais integrada [5].

No entanto, a abordagem monolítica também apresenta desafios. Em primeiro lugar, a escala vertical limitada representa uma dificuldade, pois a capacidade de aumentar o desempenho de uma única instância da aplicação pode ser limitada, tornando-se um obstáculo quando há um aumento significativo na carga de trabalho [14]. Além disso, a dificuldade na adoção de novas tecnologias é outro desafio, pois introduzir inovações ou realizar atualizações expressivas pode ser mais complexo devido à interdependência dos diversos componentes na arquitetura monolítica [7]. Por fim, o acoplamento destaca-se como um desafio adicional, uma vez que alterações em uma parte da aplicação podem impactar outras áreas, e, consequentemente, dificulta a manutenção e a evolução da aplicação ao longo do tempo [5].

Embora a arquitetura monolítica tenha suas vantagens em termos de simplicidade e manutenção centralizada, muitas organizações exploram abordagens mais distribuídas, como os microsserviços, para enfrentar os desafios associados ao crescimento e à complexidade das aplicações modernas.

2.2 Arquitetura Microsserviços

Conforme Lewis e Fowler [14], o estilo arquitetural de microsserviços consiste em desenvolver uma única aplicação como um conjunto de pequenos serviços. Cada serviço opera em seu próprio processo e se comunica por meio de APIs. Esses serviços são construídos de acordo com as regras de negócio, permitindo deploys independentes por meio de automação. A característica fundamental da arquitetura de microsserviços é o desacoplamento, o qual viabiliza a escalabilidade. Isso possibilita que múltiplas equipes trabalhem de forma independente, cada uma responsável por seu próprio serviço.

A arquitetura de microsserviços oferece diversas vantagens. Primeiramente, destaca-se a escalabilidade e flexibilidade, permitindo a escalabilidade independente de partes específicas da aplicação para lidar com demandas variáveis em diferentes componentes de forma eficaz [7]. Além disso, a desenvolvimento independente proporciona a flexibilidade para equipes trabalharem em paralelo, utilizando linguagens de programação e tecnologias mais adequadas para cada serviço [14]. A resiliência e tolerância a falhas são outras

vantagens significativas, uma vez que a independência dos microsserviços impede que a falha em um serviço afete o funcionamento dos outros, contribuindo para uma arquitetura mais robusta [7]. Por fim, a possibilidade de evolução gradual e atualizações contínuas destaca-se, permitindo a introdução de novos serviços ou atualizações de forma contínua, sem a necessidade de reescrever toda a aplicação [14].

Entretanto, a adoção da arquitetura de microsserviços apresenta maior complexidade na comunicação demandando uma gestão cuidadosa para garantir eficiência e integridade nas interações [5]. A consistência de dados distribuídos representa outro obstáculo significativo, exigindo estratégias cuidadosas para manter a coesão dos dados entre os serviços [7]. Além disso, a realização de testes e monitoramento requer a adoção de ferramentas e práticas específicas para garantir a qualidade e o desempenho contínuos [5]. Por fim, o *overhead* operacional é um desafio adicional, pois a gestão de vários microsserviços implica um aumento no custo operacional, exigindo investimentos em ferramentas e equipes especializadas para garantir o gerenciamento eficiente de todos os serviços [14].

A escolha entre arquitetura monolítica e microsserviços dependerá das necessidades específicas do projeto, considerando aspectos como tamanho do projeto, necessidade de escalabilidade, flexibilidade, complexidade operacional e evolução contínua.

2.3 Arquitetura Micro-frontend

O termo "Micro-Frontend" refere-se a uma abordagem arquitetural que se inspira nos princípios dos Microsserviços, aplicados ao desenvolvimento da camada de frontend de uma aplicação. Essa abordagem visa superar os desafios enfrentados por aplicações monolíticas, permitindo a construção, implantação e manutenção independentes de diferentes partes do frontend [1].

Os Micro-Frontends são unidades independentes e autocontidas de funcionalidades na camada de frontend, cada uma com seu próprio ciclo de vida de desenvolvimento e deploy. Essas unidades podem ser desenvolvidas, testadas e implantadas separadamente, permitindo que equipes trabalhem de forma independente em diferentes partes da interface do usuário. Isso não só agiliza o processo de desenvolvimento, mas também facilita a escalabilidade e a manutenção, uma vez que as alterações em uma parte do frontend não impactam necessariamente outras partes[1].

Essa arquitetura é particularmente valiosa em contextos de grandes aplicações e equipes distribuídas, onde a modularidade e a independência de implementação são cruciais. Ao adotar os princípios dos Micro-Frontends, as organizações podem alcançar maior flexibilidade, agilidade e facilidade de evolução em seus projetos de desenvolvimento frontend[1].

Para compreendermos melhor o que envolve a abordagem da arquitetura micro-frontend dividimos este capítulo em duas seções principais: a seção das Decisões de Integração, que abordam aspectos exclusivos da abordagem micro-frontend, e a seção de Estratégias de implementação, que envolve métodos e tecnologias para desenvolver e implantar uma aplicação[1].

2.3.1 Decisões de Integração

A integração da arquitetura Micro-Frontend compreende uma série de deliberações estratégicas ao longo do processo de adoção e implementação. Luca Mezzalira [1] desenvolveu um framework baseado nas decisões críticas para a adoção desta arquitetura, as chamadas "Decisões de Integração" que desempenham um papel central, moldando a forma como os Micro-Frontends se comunicam, são roteados, compostos e, por fim, integrados no ecossistema da aplicação. Estas decisões são classificadas em quatro categorias principais: Definição, Roteamento, Composição e Comunicação.

Definição

No início do processo de adoção da arquitetura Micro-Frontend, a primeira decisão a ser tomada refere-se à definição de como considerar um Micro-Frontend do ponto de vista de sua estratégia de divisão. A escolha central gira em torno de ter múltiplos Micro-Frontends na mesma visualização (horizontal split) ou ter apenas um Micro-Frontend por visualização (vertical split).

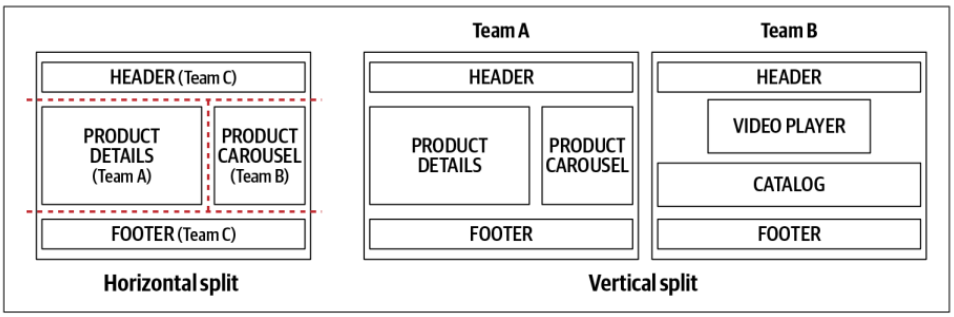


Figura 2.1: Horizontal versus Vertical Split[1]

Essas decisões estratégicas moldam o modo como os Micro-Frontends serão organizados, refletindo nas dinâmicas de trabalho das equipes e na estrutura da aplicação como um todo, ressaltando a importância de escolher uma abordagem alinhada com as necessidades do projeto e destaca a aplicação específica de princípios de DDD no contexto de Micro-Frontends.

Divisão Vertical (Vertical Split)

A abordagem de *Vertical Split* oferece menos escolhas, especialmente aquelas familiares aos desenvolvedores front-end acostumados a escrever single-page applications (SPAs). Essa decisão é particularmente útil quando um projeto requer uma evolução consistente da interface do usuário e uma experiência fluida em várias visualizações. A divisão vertical proporciona a experiência de desenvolvimento mais próxima de uma SPA, permitindo a utilização de ferramentas, boas práticas e padrões conhecidos para o desenvolvimento de um micro-frontend. Nessa divisão cada equipe é encarregada de um domínio de negócios específico, como autenticação ou catálogo. Essa abordagem adota o *Domain-Driven Design* (DDD) [16], geralmente aplicado no backend, adaptando-o para o frontend. DDD destaca três tipos de subdomínios: core (núcleo), supporting (de suporte) e generic (genérico), categorizando a essencialidade para a aplicação, a relação com o diferencial e a contribuição genérica à plataforma. Mais adiante neste trabalho, dedicaremos uma subseção para discutir o DDD no contexto do Micro-frontend.

Para Florian Rappl [15] a abordagem de Vertical Split é usada quando uma seção inteira (ou domínio) de um aplicativo muito grande tem requisitos e ciclos de desenvolvimento diferentes de outra seção, e é desenvolvida por uma equipe completamente independente. Por exemplo, em uma plataforma bancária online grande que tem uma seção de blog, essa seção de blog pode ser desenvolvida por uma equipe diferente da equipe que desenvolve outras partes do aplicativo. Nesse caso, a abordagem de Vertical Split é uma boa opção.

De acordo com Mezzalira [1], tecnicamente, é possível servir micro-frontends com divisão vertical por meio de qualquer composição, mas até agora, todas as implementações exploradas adotam uma composição no lado do cliente, onde um *Application Shell* é responsável por montar e desmontar os micro-frontends. A relação entre um micro-frontend e o *Application Shell* é sempre uma para um, o que significa que o *Application Shell* carrega apenas um micro-frontend por vez. O roteamento é geralmente dividido em duas partes, com um roteamento global usado para carregar diferentes micro-frontends, sendo manipulado pelo "Application Shell".

Divisão Horizontal (Horizontal Split)

A abordagem de *Horizontal Split* destaca-se em cenários nos quais um subdomínio de negócios precisa ser apresentado em várias visualizações, sendo especialmente eficaz sua reutilização desse subdomínio. Essa escolha é apropriada para projetos que demandam uma evolução consistente da interface do usuário e uma experiência fluida em diversas visualizações. Além disso, revela-se vantajosa em iniciativas que necessitam de otimização

para mecanismos de busca, adotando uma abordagem de renderização no lado do servidor. A divisão horizontal também é indicada em contextos que envolvem a colaboração de dezenas ou centenas de desenvolvedores, exigindo uma subdivisão mais granular dos subdomínios, e é útil em projetos multi-locatários com personalizações específicas para diferentes clientes. Nessa abordagem, diversos Micro-Frontends coexistem na mesma visualização, cada um sob a responsabilidade de equipes distintas, oferecendo flexibilidade e reutilização em diferentes contextos visuais, mas demandando disciplina e governança para evitar a proliferação descontrolada de Micro-Frontends no mesmo projeto. Caso a decisão seja pela Divisão horizontal deve-se considerar três aspectos: Composição, Roteamento e Comunicação.

Composição:

A composição em micro-frontends refere-se à maneira como os diferentes elementos visuais e funcionais de uma aplicação web são combinados e apresentados ao usuário final. Existem diferentes estratégias de composição em micro-frontends, sendo as principais a composição no lado do cliente, composição no lado da borda, e composição no lado do servidor. Cada abordagem possui vantagens e é escolhida com base nos requisitos específicos do projeto.

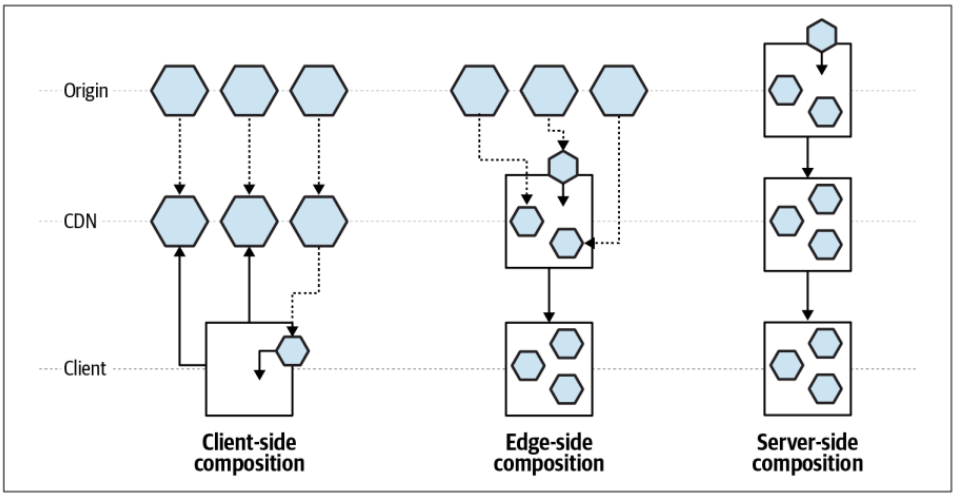


Figura 2.2: Micro-frontends composition diagram [1]

A Figura 2.2 mostra à esquerda, a composição no lado do cliente (client-side composition), onde um aplicativo principal carrega vários micro-frontends (representados por hexágonos) diretamente de um cache em uma *Content Delivery Network* (CDN)[1], que distribui os conteúdos estáticos, aliviando a carga nos servidores de aplicação e melhorando

[1]https://aws.amazon.com/pt/what-is/cdn/

a velocidade de entrega globalmente, ou diretamente da origem, se ainda não estiverem em cache na CDN. No meio do diagrama, a composição carrega os micro-frontends na borda (edge-side composition), ou seja, no nível da CDN, reunindo os micro-frontends da origem e entregando o resultado final ao cliente. À direita, vê-se a composição no nível da origem do servidor (server-side composition), onde os micro-frontends são compostos dentro de uma visualização, armazenados em cache na CDN e, por fim, entregues ao cliente. Veremos em detalhes cada uma dessas abordagens a seguir:

Client-Side Composition: Na situação de composição no lado do cliente, em que um aplicativo principal carrega micro-frontends em seu próprio ambiente, é essencial que esses micro-frontends possuam um arquivo JavaScript ou HTML como ponto de entrada. Isso permite que o aplicativo principal anexe dinamicamente os nós do Document Object Model (DOM) no caso de um arquivo HTML, ou inicialize a aplicação JavaScript com um arquivo JavaScript.

A Figura 2.2 mostra a composição de microsserviços no navegador, ou client-side composition. Segundo Geers [2], cada fragmento é sua própria miniaplicação e pode renderizar e atualizar sua marcação independentemente do restante da página. Thunder.js e Wonder.js representam frameworks como Angular, Vue e React.

Outra abordagem envolve o uso de iframes em combinação para carregar diferentes micro-frontends, ou a utilização de um mecanismo de Transclusão no Lado do Cliente[2], por meio de uma técnica chamada include no lado do cliente. O include no lado do cliente realiza o carregamento tardio de componentes, substituindo marcadores de posição vazios por componentes complexos. Por exemplo, uma biblioteca chamada h-include utiliza marcadores de posição que geram uma solicitação Ajax a uma URL e substituem o conteúdo interno do elemento pela resposta dessa solicitação.

Edge-Side Composition: Com a composição no lado da borda, montamos a visualização no nível do CDN. Muitos provedores de CDN nos oferecem a opção de usar uma linguagem de marcação baseada em XML chamada *Edge Side Includes* (ESI)[3]. ESI não é uma linguagem nova; foi proposta como um padrão pela Akamai e Oracle em 2001. ESI permite dimensionar a infraestrutura web para explorar o grande número de pontos de presença ao redor da estrutura fornecidos por uma rede CDN, em comparação com a capacidade limitada dos data centers em que a maioria do software normalmente é hospedada. Uma desvantagem do ESI é que não é implementado da mesma forma por cada provedor de CDN; portanto, uma estratégia de múltiplos CDNs, assim como a migração do código de um provedor para outro, pode resultar em muitos refatoramentos e potencialmente na necessidade de novas lógicas de implementação.

[2]https://en.wikipedia.org/wiki/Transclusion
[3]https://www.w3.org/TR/esi-lang

A Figura 2.2 mostra na parte central a composição edge-side. Para Geers [2], que a chamou de "Composição Universal", é a combinação de uma técnica de composição no lado do servidor e no lado do cliente. Os servidores na borda da rede utilizam a ESI para compor dinamicamente páginas web. Quando um cliente faz uma solicitação, o servidor na borda compõe o conteúdo da página, incluindo fragmentos de conteúdo de diferentes fontes, como microsserviços, caches ou APIs externas, utilizando ESI. O HTML resultante é então enviado de volta para o navegador do cliente. Isso permite uma entrega eficiente de conteúdo e uma experiência de usuário mais rápida e personalizada, com atualizações dinâmicas no cliente conforme necessário.

Server-Side Composition: A última possibilidade que analisaremos é a composição no lado do servidor, que pode ocorrer em tempo de execução ou em tempo de compilação. Neste caso, o servidor de origem está compondo a visualização ao recuperar todos os diferentes micro-frontends e montar a página final. Se a página for altamente cacheável, o CDN a servirá com uma política de tempo de vida longo. No entanto, se a página for personalizada por um usuário, será necessário considerar outras abordagens em relação à escalabilidade da solução final, especialmente quando há muitas solicitações provenientes de diferentes clientes. Isso ocorre porque a personalização por usuário adiciona complexidade ao processo de composição no lado do servidor, exigindo uma gestão cuidadosa dos recursos para garantir que o sistema possa lidar eficientemente com um grande volume de solicitações. Nesse caso, pode ser necessário explorar técnicas adicionais, como o cache inteligente de conteúdo personalizado ou a distribuição de carga entre vários servidores de borda, para garantir que a infraestrutura seja escalável e possa atender às demandas variáveis dos usuários de forma eficaz.

Ainda na Figura 2.2 ilustra-se a composição no lado do servidor realizada por um serviço posicionado entre o navegador e os servidores de aplicação. Nesse método, a página já está completamente montada quando chega ao navegador do usuário. Isso resulta em alta velocidade de carregamento da primeira página, difíceis de igualar com técnicas exclusivas de integração no lado do cliente. A composição da aplicação no lado do servidor proporciona a base para adotar os princípios de aprimoramento progressivo. As equipes podem decidir adicionar JavaScript no lado do cliente aos fragmentos que melhora a experiência do usuário.

Florian Rappl [15] destaca que, no lado do servidor, existe uma variedade de frameworks disponíveis que podem facilitar a implementação. Alguns deles são apenas bibliotecas ou frameworks para o Express, enquanto outros já vêm na forma de serviços que precisam ser implementados em sua infraestrutura, alguns destes frameworks citados pelo

autor são: Mosaic[4], PuzzleJs[5], Podium[6] e Micromono[7].

Roteamento:

A próxima escolha trata de como rotear as visualizações da aplicação. Essa decisão está diretamente vinculada ao mecanismo de composição de microsserviços que pretendemos usar no projeto, podendo ser feita no lado do servidor, na borda ou no lado do cliente, como mostrado na Figura 2.2.

Client-Side Composition: No contexto de roteamento no lado do cliente, a abordagem consiste em carregar os micro-frontends com base no estado do usuário. Essa estratégia é particularmente eficaz quando empregada em conjunto com um shell de aplicação que carrega um micro-frontend como uma Single Page Application (SPA). Nesse cenário, o shell da aplicação assume a responsabilidade pela lógica de roteamento, tornando-se o ponto central para determinar quais micro-frontends devem ser carregados em resposta às interações do usuário.

A utilização de um shell de aplicação proporciona uma gestão centralizada do roteamento, facilitando a implementação de lógicas avançadas de roteamento, como aquelas baseadas em autenticação, geolocalização ou outras condições específicas do usuário. Esse modelo é particularmente útil em casos nos quais a aplicação apresenta uma estrutura de navegação complexa e dinâmica, exigindo decisões de roteamento que vão além das capacidades tradicionais.

Além disso, ao adotar o roteamento no lado do cliente com um shell de aplicação, as transições entre diferentes estados ou seções da aplicação podem ser realizadas de maneira mais eficiente e responsiva, uma vez que os microsserviços relevantes são carregados dinamicamente conforme necessário. Essa abordagem também se alinha bem com a tendência de construir interfaces de usuário mais interativas e fluidas, oferecendo uma experiência de usuário aprimorada.

Edge-Side Composition: Quando optamos por utilizar a composição no lado da borda, o roteamento é fundamentado na URL da página, e o CDN assume a responsabilidade de montar os microsserviços no nível da borda, antes de entregar a página ao cliente. Essa abordagem tem suas vantagens, especialmente em termos de distribuição eficiente e rápida de conteúdo estático para os usuários.

No entanto, uma das limitações dessa estratégia é a reduzida flexibilidade para criar roteamentos mais inteligentes. O roteamento baseado na URL geralmente implica em uma correspondência direta entre a URL da página solicitada e os microsserviços que

[4]https://www.mosaic9.org/
[5]https://github.com/puzzle-js/puzzle-js
[6]https://podium-lib.io/
[7]https://github.com/lsm/micromono

serão agregados pela CDN. Isso significa que a lógica de roteamento é simplificada e segue uma abordagem mais direta, o que pode ser insuficiente em situações que demandam decisões de roteamento mais complexas.

Em cenários nos quais a aplicação exige lógicas de roteamento mais sofisticadas, como personalização dinâmica com base no usuário, contexto geográfico ou outros parâmetros específicos, a composição no lado da borda pode apresentar limitações. Roteamentos inteligentes, que respondem dinamicamente a diferentes condições, podem ser mais desafiadores de implementar quando o roteamento é rigidamente vinculado à estrutura da URL.

Apesar dessas limitações, a composição no lado da borda continua sendo uma estratégia valiosa, especialmente para conteúdo estático e em situações em que a simplicidade e a distribuição global são prioridades. Ao escolher essa abordagem, é essencial ponderar as necessidades específicas da aplicação e a complexidade do roteamento necessário para garantir que a solução atenda adequadamente aos requisitos do projeto.

Server-Side Composition: Ao escolher a composição de microsserviços no lado do servidor, a lógica de reunir os diferentes fragmentos ou microsserviços acontece nos próprios servidores de aplicação. Isso implica que as solicitações precisam ser direcionadas e processadas no servidor, uma vez que toda a lógica da aplicação está concentrada nesse ambiente. No entanto, essa abordagem pode apresentar desafios significativos ao escalar a infraestrutura, especialmente quando confrontada com tráfego intenso e um grande volume de solicitações por segundo.

A escalabilidade pode se tornar um ponto crítico, uma vez que a demanda por recursos de servidor aumenta proporcionalmente ao número de usuários ou solicitações. Lidar eficientemente com esse aumento de carga pode ser desafiador, exigindo a capacidade de escalar horizontalmente, ou seja, adicionar mais servidores conforme necessário. Essa tarefa pode ser complexa e demandar recursos substanciais, visto que cada servidor de aplicação deve ser capaz de recuperar os microsserviços necessários para compor as páginas a serem servidas.

Uma estratégia para mitigar esses desafios é a utilização de CDNs. No entanto, a eficácia das CDNs pode ser limitada quando lidamos com dados dinâmicos ou personalizados, pois esses dados podem não se beneficiar completamente do cache global proporcionado pelas CDNs.

Portanto, ao adotar a composição no lado do servidor, deve-se considerar cuidadosamente a escalabilidade da infraestrutura e explorar estratégias adicionais, como o uso inteligente de CDNs, para otimizar o desempenho e a capacidade de resposta da aplicação, especialmente em cenários de tráfego intenso.

Essas abordagens de roteamento não são mutuamente exclusivas e podem ser combinadas usando CDN e origem ou cliente e CDN juntos.

Comunicação

A comunicação eficaz entre micro-frontends é um aspecto crítico no desenvolvimento de arquiteturas baseadas nessa abordagem, uma vez que micro-frontends são unidades independentes que coexistem em uma única aplicação. Para Luca Mezzalira [1], em um cenário ideal, cada micro-frontend seria autônomo, operando de forma isolada, sem a necessidade de interações com outros componentes. No entanto, na prática, há momentos em que é essencial estabelecer comunicação entre esses elementos para garantir uma experiência do usuário coesa e funcional. Esta necessidade se torna ainda mais evidente quando múltiplos micro-frontends coexistem na mesma página, exigindo estratégias inteligentes para notificar uns aos outros sobre eventos relevantes, compartilhar dados e garantir a consistência da interface do usuário.

Michael Geers [2] propôs alternativas de comunicação que podem ocorrer em uma arquitetura de micro-frontends, focando especialmente na coesão e eficácia do sistema. A ênfase na indústria recai principalmente na UI Communication, que engloba métodos e ferramentas para interação entre os micro-frontends. Entre esses mecanismos, destacam-se notificações de eventos, troca de informações sobre o estado da aplicação e sincronização de ações na interface.

Dentro desse contexto, destacam-se diferentes modalidades de comunicação, cada uma atendendo a necessidades específicas:

- *Parent to Fragment (Pai para Fragmento):* Nessa modalidade, o componente “pai” comunica informações diretamente ao componente "fragmento". Isso é alcançado por meio de propriedades, eventos personalizados ou outros mecanismos. O componente pai envia dados relevantes ou instruções diretamente ao fragmento, permitindo uma comunicação eficaz e direta.

- *Fragment to Parent (Fragmento para Pai):* Ao contrário da abordagem anterior, aqui, é o componente “fragmento” que inicia a comunicação e envia informações ao componente “pai”. Isso pode ocorrer por meio da emissão de eventos, chamadas de função específicas ou outros métodos, proporcionando ao fragmento a capacidade de informar ou solicitar ação ao componente pai.

- *Fragment to Fragment (Fragmento para Fragmento):* Quando dois ou mais fragmentos precisam se comunicar diretamente entre si, seja para coordenar ações ou compartilhar informações, ocorre uma comunicação fragmento para fragmento. Isso geralmente é realizado por meio de um mecanismo centralizado, como um “event

bus" compartilhado ou uma camada de gerenciamento de estado global, onde os fragmentos emitem eventos ou acessam o estado compartilhado para troca de dados.

2.3.2 Estratégias de Implementação

Esta seção se dedica à exploração das metodologias, técnicas e ferramentas fundamentais para a adoção e implementação bem-sucedida da arquitetura de micro-frontends. Examinaremos técnicas que abrangem desde a escolha de frameworks Javascript, como Angular, Vue e React, até ferramentas especializadas como Single-SPA, Module Federation e conceitos fundamentais como o DDD.

Domain-Driven Design (DDD) e Micro-frontend

No âmbito do Domain-Driven Design (DDD), destaca-se o conceito fundamental de *bounded context* (contexto delimitado), uma fronteira lógica que encapsula os detalhes de implementação, expondo um contrato de API para consumir dados do modelo contido. Essa abordagem traduz as áreas de negócio definidas por domínios e subdomínios em áreas lógicas, onde são delineados o modelo, a estrutura de código e, possivelmente, as equipes responsáveis. O autor Eric Evans [16] fundamenta o DDD explorando conceitos como entidades, agregados, serviços de domínio, subdomínios, eventos e projeções, oferecendo uma visão abrangente e integrada da modelagem. Ele também aborda a organização do código em camadas e apresenta padrões estratégicos, enfatizando a importância de evitar anti-padrões.

Em sistemas novos, os subdomínios frequentemente se sobrepõem aos *bounded contexts*, devido à liberdade de design disponível. No entanto, em sistemas legados, é comum que um único *bounded context* abranja múltiplos subdomínios. Nesse cenário, o ecossistema de micro-frontends oferece diversas abordagens técnicas, como iframes, bibliotecas de componentes ou web components, para auxiliar na implementação eficiente dos micro-frontends, levando em consideração os princípios do Domain-Driven Design (DDD).

No DDD é importante considerar a perspectiva comercial ao abordar um projeto, é viável trabalhar com Single-Page Applications (SPAs), proporcionando uma abordagem mais eficiente. A integração de microarquiteturas, microsserviços e micro-frontends, possibilita entregas independentes, mitigando riscos para a produção. Ao definir todos os *bounded contexts*, obtém-se um mapa do sistema que representa as diferentes áreas que o compõem. Embora a DDD normalmente não considere o frontend, ao adotar a abordagem de micro-frontends e uma divisão vertical, torna-se possível mapear facilmente o frontend e o backend dentro do mesmo *bounded context*.

Single-SPA

O Single-SPA é um framework JavaScript projetado para facilitar a implementação de arquiteturas de micro-frontends. Ele atua como um roteador de alto nível, permitindo o desenvolvimento independente e a implantação de diferentes partes de uma aplicação micro-frontend. Esses micro-frontends possuem ciclos de vida independentes e são carregados sob demanda, utilizando técnicas como lazy loading para otimizar o desempenho da aplicação.

O mecanismo Single-SPA fornece um aplicativo shell carregando o código das aplicações e roteia entre elas, simplificando o registro das aplicações e garantindo a ausência de conflitos de estilo. Utilizando métodos assíncronos, como mount(), unmount() e bootstrap, o Single-SPA controla o ciclo de vida das aplicações, proporcionando agilidade e eficiência no desenvolvimento e na manutenção. Além disso, o framework utiliza a biblioteca SystemJS para melhorar ainda mais o desempenho, atuando como um carregador de módulos na camada inferior [17].

Design System

O Design System [19] fornece uma abordagem coesa para o desenvolvimento e design de interfaces de usuário em sistemas distribuídos. Ele atua como um conjunto de padrões, diretrizes e componentes reutilizáveis, garantindo consistência visual e funcional entre os diferentes micro-frontends. A sua importância é evidente na promoção da eficiência do desenvolvimento, facilitando a manutenção, colaboração entre equipes e a criação de uma experiência de usuário integrada.

Nathan Curtis [19], destaca a relevância de um Design System na construção de interfaces coesas e na garantia da consistência ao longo do tempo. Curtis ressalta que um Design System bem elaborado não apenas acelera o desenvolvimento, mas também fortalece a identidade da marca e melhora a usabilidade.

Um sistema de design funciona como um repositório consolidado de estilos visuais, componentes e outros elementos relacionados ao design, documentados e disponibilizados por indivíduos, equipes ou comunidades na forma de código e ferramentas de design. Seu propósito é aprimorar a eficiência e coesão dos produtos adotados, fornecendo um conjunto padronizado e reutilizável de elementos de design [19]. Alguns Design Systems disponibilizados pela Indústria:

- **Material Design:** Desenvolvido pela Google, o Material Design[8] é um Design System amplamente adotado que oferece diretrizes claras para o design de interfaces, além de uma biblioteca de componentes reutilizáveis. Ele se destaca por seu foco

[8] https://m3.material.io/

na experiência do usuário e consistência visual, sendo aplicável em uma variedade de plataformas.

- **Bootstrap:** O Bootstrap[9] é um popular framework front-end que também pode ser considerado um Design System. Ele fornece uma coleção de componentes e estilos pré-construídos para o desenvolvimento rápido e consistente de interfaces responsivas.
- **Prime:** O Prime[10] é um conjunto de componentes de interface de usuário para Javascript com opções para framework React, Vue e Angular. Ele oferece uma variedade de componentes prontos para uso, seguindo um conjunto consistente de padrões visuais e interativos.

Web Components

Os Web Components[11] são uma tecnologia da web que permite a criação de componentes reutilizáveis e encapsulados, consistindo em HTML, CSS e JavaScript. Eles são projetados para serem interoperáveis e podem ser utilizados em diferentes frameworks ou até mesmo sem nenhum framework específico. A relação entre Web Components e Design Systems está na capacidade dos Web Components de oferecerem uma abordagem modular e padronizada para a construção de interfaces de usuário.

Ao incorporar Web Components em um Design System, os desenvolvedores podem criar componentes visuais e funcionais consistentes que podem ser facilmente integrados em várias partes de uma aplicação ou entre diferentes projetos. Esses componentes encapsulam tanto a lógica de apresentação quanto o estilo, promovendo a reutilização e a consistência visual em toda a aplicação [20].

A natureza independente dos Web Components permite que eles sejam utilizados em conjunção com diversos frameworks ou bibliotecas, o que é benéfico em ambientes nos quais diferentes tecnologias coexistem. Além disso, a capacidade de personalização e estilização dos Web Components contribui para a flexibilidade do Design System, adaptando-se às necessidades específicas de diferentes partes de uma aplicação.

Module Federation

As arquiteturas de micro-frontends foram enriquecidas com o lançamento do webpack 5, trazendo consigo um novo plug-in nativo denominado Module Federation. Este recurso proporciona a carga síncrona ou assíncrona de pedaços de código JavaScript, permitindo que vários desenvolvedores ou equipes operem de forma isolada, cuidando da composição

[9] https://getbootstrap.com/
[10] https://www.primefaces.org/
[11] https://www.webcomponents.org/

da aplicação. Isso possibilita o carregamento dinâmico de diferentes pedaços de código JavaScript nos bastidores durante a execução. O Module Federation permite o carregamento assíncrono de vários micro-frontends, proporcionando ao usuário uma experiência contínua.

Uma aplicação Module Federation é composta por duas partes fundamentais:

- **O Host:** Representa o contêiner que carrega um ou mais micro-frontends ou bibliotecas.
- **O Remoto:** Representa o micro-frontend ou biblioteca que será carregado dentro de um host em tempo de execução. Um remoto expõe um ou mais objetos que podem ser utilizados pelo host quando o remoto é carregado de forma assíncrona em uma aplicação.

O ponto que se destaca no Module Federation é a simplicidade de expor diferentes micro-frontends, ou até mesmo bibliotecas compartilhadas, como um design system, permitindo uma integração assíncrona descomplicada. A experiência do desenvolvedor torna-se mais fluida, proporcionando a capacidade de importar micro-frontends remotos e compor visualizações de maneira flexível, semelhante ao trabalho com uma base de código monolítica.

API Gateway

O padrão do Gateway de API é caracterizado como um serviço que se posiciona entre o cliente e o servidor, proporcionando um único ponto de entrada para os clientes. Geralmente, encaminha solicitações do cliente para um microserviço específico ou recupera dados de vários microserviços em uma única solicitação do cliente.

Para Rousos et al.[23], no contexto de arquiteturas de micro-frontends, o API Gateway assume uma função central como um intermediário vital na comunicação entre os clientes e os diversos micro-frontends, facilitando a orquestração de chamadas de API e direcionando solicitações para os microserviços específicos associados aos fragmentos de interface de usuário. Atuando como um ponto de entrada único, o API Gateway proporciona roteamento eficiente, reduzindo o acoplamento direto entre clientes e microserviços. Essa abordagem não apenas consolida requisições e aprimora o desempenho, mas também centraliza questões de segurança, oferecendo uma camada de autorização e autenticação consistente. Adicionalmente, o API Gateway contribui para logging e monitoramento eficazes, facilitando o diagnóstico, otimização e garantia de confiabilidade na gestão de micro-frontends.

Backends For Frontends (BFF)

O padrão BFF desempenha um papel estratégico ao oferecer uma abordagem especializada para a comunicação eficiente entre clientes e servidores. Enquanto o padrão API Gateway aborda muitos cenários com sucesso, situações específicas dentro da arquitetura de micro-frontends demandam uma consideração cuidadosa, destacando o papel específico do BFF.

Newman [24] destaca que o BFF está fortemente acoplado a uma experiência de usuário específica e normalmente será mantido pela mesma equipe que a interface do usuário, tornando mais fácil definir e adaptar a API conforme a UI exige, ao mesmo tempo que simplifica o processo de alinhamento do lançamento de ambos os componentes cliente e servidor.

Em cenários onde interfaces de usuário necessitam da agregação de múltiplas APIs para compor uma visualização única, como é o caso de um painel financeiro que reúne dados de diversos endpoints, o BFF se destaca como uma solução eficaz. Tradicionalmente, a agregação desses dados ocorre no lado do cliente, envolvendo a requisição de múltiplos endpoints e a interpolação de informações para atualizar a visualização. O BFF surge como um aliado estratégico, possibilitando a criação de backends específicos para cada experiência do usuário. Essa abordagem não apenas melhora o desempenho ao agregar dados no lado do servidor, mas também reduz a complexidade para os clientes de micro-frontends.

Luza Mezzalira [1], destacou a importância do BFF ao proporcionar uma entrada única para grupos específicos de dispositivos, como mobile e web. Essa abordagem permite a agregação de respostas de API antes de servir ao cliente, reduzindo a sobrecarga de comunicação entre clientes e o backend. O BFF otimiza o tráfego, entregando apenas as informações necessárias para preencher a estrutura da visualização. Uma das características mais evidentes dessa abordagem é encapsular a complexidade da arquitetura de micro-frontends no BFF. Dessa forma, proporciona um ponto de entrada único, eliminando a necessidade de os clientes compreenderem a intricada arquitetura de micro-frontends. Isso simplifica a interação dos clientes com o sistema, tornando a experiência mais intuitiva e eficiente.

Implantação da Aplicação

Para Martin Fowler [25], no contexto de desenvolvimento de micro-frontends, a implantação (deploy) da aplicação emerge como uma etapa crucial, destacando-se por sua principal característica: a independência. A capacidade de realizar implantações independentes de cada micro-frontend é essencial para mitigar riscos associados a implantações em larga es-

cala, permitindo que cada equipe de desenvolvimento entregue suas atualizações de forma autônoma.

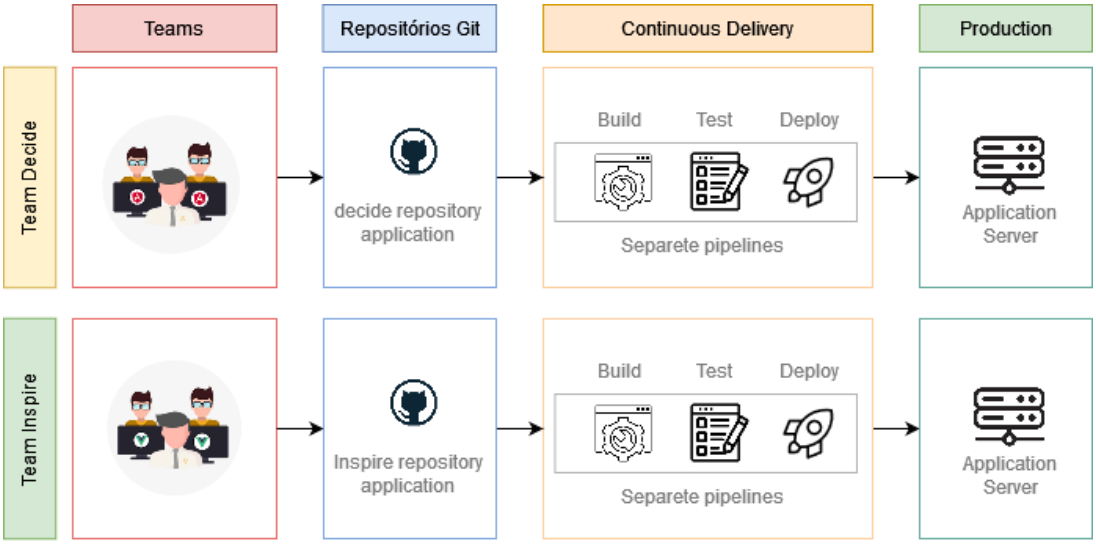


Figura 2.3: Implantação (Deploy) - Adaptado[2]

A Figura 2.3 apresenta o cenário no qual as equipes “Team Decide” e “Team Inspire” operam em seus próprios repositórios de código-fonte (Repository Decide e Repository Inspire), possuindo pipelines de integração separados e a capacidade de realizar implantações independentes até o ambiente de produção. Essa abordagem promove a agilidade, flexibilidade e independência entre os componentes do sistema, em conformidade com os princípios fundamentais dos micro-frontends.

Mezzalira [1] ressalta que trabalhar com micro-frontends exige uma constante melhoria na automação do pipeline, sendo essencial a incorporação de ferramentas específicas para garantir uma integração contínua, entrega contínua e implantação contínua eficazes.

Ao voltarmos a atenção à Figura 2.3 podemos notar que a implantação inclui ferramentas como o Git para controle de versão, plataformas de integração contínua como Jenkins ou Travis CI para automação de construção e distribuição, e Docker para empacotamento consistente em ambientes isolados. O uso de ferramentas como Webpack ou Parcel facilita o gerenciamento de dependências e a criação de bundles otimizados. Testes são automatizados com frameworks como Jest, Jasmine, Mocha e ferramentas como Selenium ou Cypress para garantir a qualidade da interface do usuário. Além disso, a análise contínua de código é possível com ferramentas como SonarQube, enquanto Ansible ou Terraform auxiliam no provisionamento da infraestrutura como código. Para monitoramento e registro, Prometheus ou ELK Stack são opções viáveis. A integração harmoniosa dessas ferramentas é fundamental para o sucesso de estratégias de desenvolvimento contínuo.

Controle de Versionamento:

O Git desempenha um papel fundamental na implantação contínua, fornecendo um sistema robusto de controle de versão distribuído. Permitindo que os desenvolvedores rastreiem e controlem eficientemente as alterações em seus códigos por meio de commits, o Git também facilita a prática eficiente de branches e merges, possibilitando o desenvolvimento simultâneo de recursos isolados e correções de bugs. Além disso, ao integrar o Git a sistemas de integração contínua, como Jenkins ou GitHub Actions, as equipes podem automatizar a compilação, teste e implantação do código em ambientes de teste após cada push para o repositório [26].

Além disso, o Git desempenha um papel vital na gestão de releases e rollbacks eficientes. A marcação de versões específicas do código por meio de tags no Git permite a referência fácil para implantações em ambientes de produção. Em casos de problemas pós-implantação, a natureza distribuída do Git possibilita rollbacks rápidos, revertendo para versões anteriores. Combinando essas capacidades, o Git não apenas facilita a colaboração distribuída entre equipes, mas também fortalece a prática de implantação contínua, promovendo ambientes de desenvolvimento ágeis e controlados.

Integração e Entrega Contínua (CI/CD):

CI/CD significa Integração Contínua e Entrega Contínua. É um conjunto de práticas que visam automatizar o processo de desenvolvimento de software, desde a escrita do código até a sua entrega em produção [27]. A CI/CD visa:

- Acelerar o desenvolvimento: Automatizando tarefas repetitivas, os desenvolvedores podem se concentrar em escrever código e inovar.
- Melhorar a qualidade do código: Testes automatizados executados com frequência identificam e corrigem bugs antes que cheguem à produção.
- Implantar com mais frequência: Implantações mais frequentes permitem que os usuários acessem novos recursos e correções de bugs mais rapidamente.
- Colaboração aprimorada: A automação facilita a colaboração entre os membros da equipe, pois as alterações no código são automaticamente testadas e implantadas.

A escolha da ferramenta ideal depende das suas necessidades e preferências específicas. A Tabela 2.1 apresenta características de algumas opções de ferramentas utilizadas

na indústria de software para o CI/CD: Jenkins[12], CircleCI[13], GitHub Actions[14], Azure DevOps[15] e Bitbuket[16].

Características	**Jenkins**	**CircleCI**	**GitHub Actions**	**Azure DevOps**	**Bitbucket**
Open Source	Yes	No	No	No	No
Free Version	Yes	Yes	Yes	No	Yes
Supported OSs	Windows, macOS, Linux	Windows, macOS, Linux	Linux	Windows	Windows, macOS, Linux
Git Integration	Yes	Yes	Yes	Yes	Yes
Hosting Options	On-Premise, Cloud	Cloud	Cloud	Cloud	Cloud

Tabela 2.1: Comparação de Ferramentas de CI/CD Populares

Orquestração:

A orquestração de micro-frontends é o processo de coordenar e gerenciar as várias partes de uma aplicação frontend distribuída. Isso envolve a inicialização da aplicação, roteamento entre os micro-frontends, compartilhamento de configurações e dados entre eles, gerenciamento do ciclo de vida dos micro-frontends e comunicação entre o orquestrador e os micro-frontends [28].

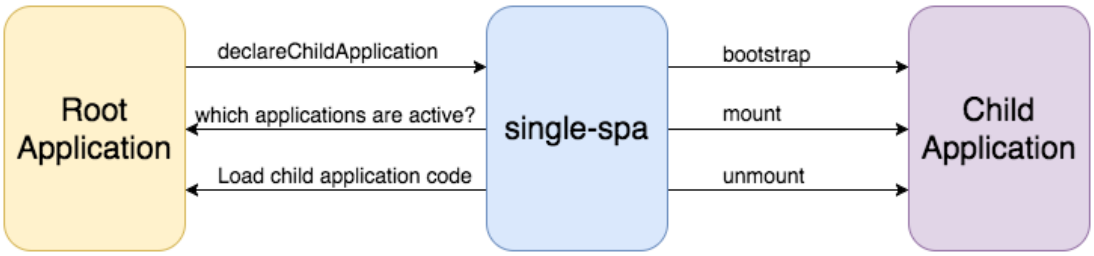


Figura 2.4: Orquestração com Framework Single-SPA

A abordagem A Figura 2.4, apresenta o fluxo da orquestração usando o framework Single-SPA como exemplo. O Root Application (à direita da figura) é responsável por orquestrar a integração e a renderização dos micro-frontends. Quando a aplicação root é carregada no navegador, ela inicializa e configura o recursos do single-spa (no meio da figura) para monitorar mudanças de rota. Quando uma mudança de rota ocorre, o single-spa determina qual Child Application (à esquerda de figura) deve ser carregada

[12]https://www.jenkins.io/
[13]https://circleci.com/
[14]https://docs.github.com/pt/actions
[15]https://azure.microsoft.com/pt-br/products/devops
[16]https://bitbucket.org/product/

dinamicamente, inicializando-a, montando-a no DOM[17] e exibindo-a para o usuário. Posteriormente, quando ocorre outra mudança de rota ou a aplicação filha é desmontada, o single-spa gerencia o processo de desmontagem da aplicação filha do DOM e liberação de recursos.

2.4 Síntese do Capítulo

No capítulo dedicado ao Referencial Teórico, abordamos a arquitetura de micro-frontends, destacando as decisões necessárias para a adoção e implementação. Essas decisões foram divididas em categorias fundamentais, compreendendo Identificação, Roteamento, Composição e Comunicação, proporcionando uma visão holística da abordagem micro-frontends. Além disso, exploramos estratégias de implementação essenciais, como DDD, SPA, Design System, Module Federation, API Gateway e BFF, fornecendo insights valiosos sobre como essas estratégias moldam a arquitetura. No contexto de implantação, destacamos a necessidade de considerar a integração contínua, entrega contínua, enfatizando a importância da automação do pipeline. Essas informações consolidam uma compreensão das bases teóricas da arquitetura de micro-frontends e das estratégias associadas à ela, servindo como guia essencial para a implementação bem-sucedida dessas práticas inovadoras no desenvolvimento frontend.

[17] https://developer.mozilla.org/pt-BR/docs/Glossary/DOM

Capítulo 3

Revisão Sistemática da Literatura

Neste trabalho foi realizado uma Revisão Sistemática de Literatura (RSL), conforme as diretrizes delineadas por Kitchenham e Charters [29], com o objetivo de identificar os impactos e desafios associados à implementação da arquitetura micro-frontend. A RSL é um método holístico que engloba a identificação, avaliação e interpretação de estudos relevantes em uma área ou questão de pesquisa específica [30]. Este processo é estruturado em fases distintas, conforme definido por Kitchenham e Charters[29]:

1. **Planejamento:** Identificação da necessidade de revisão. Estabelecimento de objetivos claros. Definição do protocolo de revisão, que abrange elementos como questões de pesquisa, sequência de busca, critérios de seleção dos estudos, lista de dados a serem extraídos e um checklist de avaliação da qualidade.
2. **Condução:** Implementação prática do protocolo de revisão. Utilização dos artefatos desenvolvidos na fase anterior para filtrar os estudos relevantes. Extração de informações significativas dos estudos selecionados.
3. **Relatório:** Documentação dos resultados da revisão, apresentados neste caso como um artigo de pesquisa.

3.1 Questões de Pesquisa

A razão subjacente à realização desta RSL reside na disseminação de informações sobre os impactos e desafios inerentes à implementação da arquitetura micro-frontend. Nosso objetivo é fornecer insights acerca das visões e padrões arquiteturais associados ao micro-frontend e entender de que maneira esses elementos reverberam no ambiente de desenvolvimento.

É fundamental destacar que nosso escopo de interesse está exclusivamente voltado para os aspectos relacionados à implementação da arquitetura micro-frontend, e não nos

aprofundamos em questões técnicas específicas de desenvolvimento como linguagens de programação e frameworks.

As questões de pesquisa apresentadas na Tabela 3.1 direcionam nosso esforço de investigação, orientando a análise para aspectos específicos da implementação da arquitetura micro-frontend. Este foco estratégico visa proporcionar uma compreensão mais clara e holística dos desafios e impactos enfrentados, contribuindo para o entendimento desta arquitetura e suas implicações no desenvolvimento de software. Assim, as questões de pesquisas que nortearam essa pesquisa são apresentadas na Tabela 3.1.

ID	Questões de Pesquisa (RQ)
RQ.1	Quais os padrões arquiteturais utilizados na implementação de micro-frontends?
RQ.2	Quais são os impactos da adoção de micro-frontends em projetos de desenvolvimento de software?
RQ.3	Quais são os desafios enfrentados pelas equipes de desenvolvimento de software na adotação de micro-frontends?

Tabela 3.1: Questões de Pesquisa

3.2 String de Busca

A elaboração da string de busca seguiu o método PICOC (População, Intervenção, Comparação, Resultado, Contexto) [31], no qual cada componente possui um papel específico:

- **População:** Refere-se ao objeto de estudo, a quem ou a quê a intervenção está relacionada.
- **Intervenção:** Representa os meios utilizados ou causados pela população para atingir um objetivo específico.
- **Comparação:** Indica o que está sendo comparado com a intervenção, embora, neste contexto, não seja aplicável, pois não estamos contrastando a intervenção com outra variável.
- **Outcome (Resultado):** Refere-se aos efeitos ou resultados decorrentes da intervenção.
- **Contexto:** Engloba o foco do estudo, incluindo suas restrições e limitações, proporcionando um entendimento mais amplo do ambiente em que a intervenção ocorre.

A Tabela 3.2 apresenta a definição final dos termos PICOC, utilizados para a seleção dos estudos, ressaltando a importância de cada componente nos critérios de seleção e exclusão no processo de triagem dos estudos.

PICOC	Keywords	Related Words
Population	micro-frontend	indústria de software, times de desenvolvimentos, adoção de arquitetura micro-frontend
Intervention	micro-frontend	implementação, padrões, adoção
Comparison	*Not applicable*	*Not applicable*
Outcome	micro-frontend	Impactos, desafios, coordenação de times, integração contínua, entrega contínua, dependências
Context	micro-frontend	desenvolvimento de software, implementação, paradigmas de frontend

Tabela 3.2: Termos PICOC

Na formulação da string de busca, optamos por uma abordagem que prioriza a especificidade, restringindo os termos de pesquisa. Essa decisão foi tomada considerando que as buscas iniciais, que incluíam palavras do PICOC ou outras relacionadas, como microsserviços e monolitos, resultaram na obtenção de artigos com contextos diversos, menos alinhados ao foco específico desta pesquisa. Outrossim, observamos nesta pesquisa que ainda não existe um consenso em relação ao termo exato para o micro-frontend. Alguns autores preferem usar “microfrontend”, enquanto outros escolhem “micro frontend” e ainda há quem utilize “micro-frontend”. No entanto, neste estudo, decidimos adotar o termo "micro-frontend", tendo em vista sua prevalência na literatura contemporânea e aceitação na comunidade técnica. Optamos por não empregar o operador lógico “AND” para incorporar termos adicionais à busca, uma vez que tal método demonstrou ser ineficaz para fins de filtragem. Em determinadas bases de dados, utilizaremos as três formas de string de busca. Assim, a string base final foi estabelecida como:

(micro-frontend OR microfrontend OR “micro frontend”)

Selecionamos as bases de dados digitais ACM Digital Library, IEEE Xplore, Science@Direct’s, Scopus e Springer Link para aplicar a string de busca. A escolha dessas plataformas baseou-se em sua relevância nas pesquisas da área de Engenharia de Software [32]. Essas bases indexam um amplo espectro de conferências e periódicos, e, fundamentalmente, têm a capacidade de executar nossa string de pesquisa genérica na íntegra. A Tabela 3.3 apresenta a string de busca utilizada para cada base de dados digital.

Costal et al.[33], sugere a utilização das funcionalidades de filtragem e strings disponíveis em cada fonte para aplicar determinados critérios de exclusão, automatizando assim o descarte de artigos indesejados. Dessa forma, as strings customizadas para cada base de dados foram:

Source	String
ACM Digital Library	("micro-frontend"OR "micro frontend"OR "microfrontend")
IEEE Xplore	("micro-frontend"OR "micro frontend"OR "microfrontend")
Science Direct	("micro-frontend"OR "micro frontend"OR "microfrontend")
Scopus	("micro-frontend"OR "micro frontend"OR "microfrontend")
Springer Link	("micro-frontend"OR "micro frontend"OR "microfrontend")

Tabela 3.3: String de Busca por Base de Dados

3.3 Critérios de Seleção

Conforme orientado por Kitchenham e Charters [29], estabelecemos critérios de inclusão e exclusão para delinear claramente os parâmetros de seleção. Os critérios de inclusão dizem respeito ao tema central dos estudos e àqueles que devem ser considerados relevantes para o tema deste trabalho. A Tabela 3.4 apresenta a delimitação dos estudos segundo os critérios de inclusão (CI).

ID	Critérios
CI1	Tecnologias ou metodologias usadas na implementação de micro-frontends
CI2	Desafios relacionados à implementação de micro-frontends
CI3	Impactos na organização dos times quando se implementa as tecnologias e métodos relacionados ao micro-frontend

Tabela 3.4: Critérios de Inclusão

Os critérios de inclusão (CI) delineados para a seleção de estudos proporcionam uma visão ampla, contemplando não apenas aspectos técnicos, mas também desafios práticos e impactos organizacionais inerentes à implementação de micro-frontends [2]. Essa abordagem multidimensional visa promover uma análise holística e aprofundada do tema em questão, possibilitando uma compreensão mais completa e integrada das complexidades envolvidas na adoção de micro-frontends.

Por sua vez, os critérios de exclusão têm como finalidade eliminar estudos que, embora se alinhem aos critérios de inclusão, careçam de metodologia, enfoque, aspecto ou abordagens específicas[33]. Na Tabela 3.5, são apresentados os critérios de exclusão (CE).

O critério CE1 direciona a atenção para a exclusão de estudos cujo foco não está na adoção da arquitetura micro-frontend, garantindo que a pesquisa mantenha uma coesão temática. Além disso, o critério CE2 relacionado à linguagem, exclui artigos escritos em idiomas não compreendidos pelos autores, como português, espanhol e inglês, assegurando a compreensibilidade e acessibilidade da pesquisa. Para delimitar a seleção dos estudos, o critério CE3 exclui documentos que não se enquadram na categoria de pesquisa primária

ID	Critérios
CE1	Estudos que não tratam explicitamente da adoção da arquitetura de micro-frontends, deixando lacunas ou falta de enfoque claro sobre sua implementação e utilização
CE2	Publicado em idioma diferente daquele compreendido pelos autores (português, espanhol e inglês)
CE3	Não ser um estudo de pesquisa primário completo (por exemplo, capítulos de livros, artigos de revistas, dissertações, teses, revisões de literatura, trabalhos em andamento, documentos de posicionamento, trabalhos duplicados)
CE4	Publicado antes de 2016

Tabela 3.5: Critérios de Exclusão

completa, como capítulos de livros, artigos de revistas, dissertações e teses. Adicionalmente, a um critério temporal CE4 foi definido o intervalo para os estudos aceitos, uma vez que o termo micro-frontend foi cunhado pela primeira vez em 2016, assim, somente trabalhos a partir desta data foram coletados[10].

Esses critérios de seleção trabalham sinergicamente para garantir que apenas artigos relevantes, coesos e recentes sejam considerados na análise, fortalecendo a robustez da pesquisa.

3.4 Garantia de Qualidade

Embora os critérios de exclusão já nos forneçam trabalhos relevantes, é importante realizar uma verificação de qualidade adicional para garantir que as práticas sejam validadas e que seus impactos tenham sido medidos de maneira adequada e científica. Conforme preconizado por Peterson et al. [34], a avaliação da qualidade dos artigos identificados é importante em estudos de mapeamento para assegurar a disponibilidade de informações suficientes para a extração de dados. Assim, realizamos um processo de avaliação da Garantia de Qualidade (GQ), analisando a robustez global dos artigos selecionados e assegurando a integridade de nossa revisão. Os critérios de avaliação adotados estão resumidos na Tabela 3.6, juntamente com as pontuações correspondentes.

ID	Critério	Score
GQ1	O estudo aborda claramente algum aspecto referente a adoção de arquiteturas de micro-frontends?	0, 0.5, 1
GQ2	O estudo propõe uma solução clara para enfrentar eventuais desafios e impactos na adoção da arquitetura de micro-frontends?	0, 0.5, 1

Tabela 3.6: Critérios de Garantia de Qualidade

Cada estudo foi avaliado por abordar distintamente alguns aspectos relacionados à adoção da arquitetura de micro-frontends. No que tange ao primeiro critério de garantia de qualidade GQ1, o estudo deveria abordar elementos fundamentais do micro-frontend, proporcionando uma compreensão clara de sua adoção, a essa questão é atribuída nota 1 aos estudos que apresentam uma solução detalhada sobre os elementos fundamentais, a pontuação 0,5 é atribuída a estudos que fornecem apenas uma visão geral dos fundamentos do micro-frontend e a pontuação 0 é atribuída quando o estudo não se manifesta claramente sobre o tema.

Para o critério GQ2, a pontuação 1 é atribuída a estudos que apresentam uma solução detalhada na identificação de desafios e impactos na adoção de arquiteturas de micro-frontends, a pontuação 0,5 é atribuída a estudos que fornecem apenas uma visão geral sobre esta adoção, e a pontuação 0 é atribuída a estudos sem identificações claras sobre o tema.

Ambos os critérios são tratados de maneira equitativa, e a pontuação total de cada estudo é calculada somando-se os valores correspondentes. Somente estudos com uma pontuação de qualidade igual ou superior a 1 foram incluídos nesta análise.

3.5 Condução

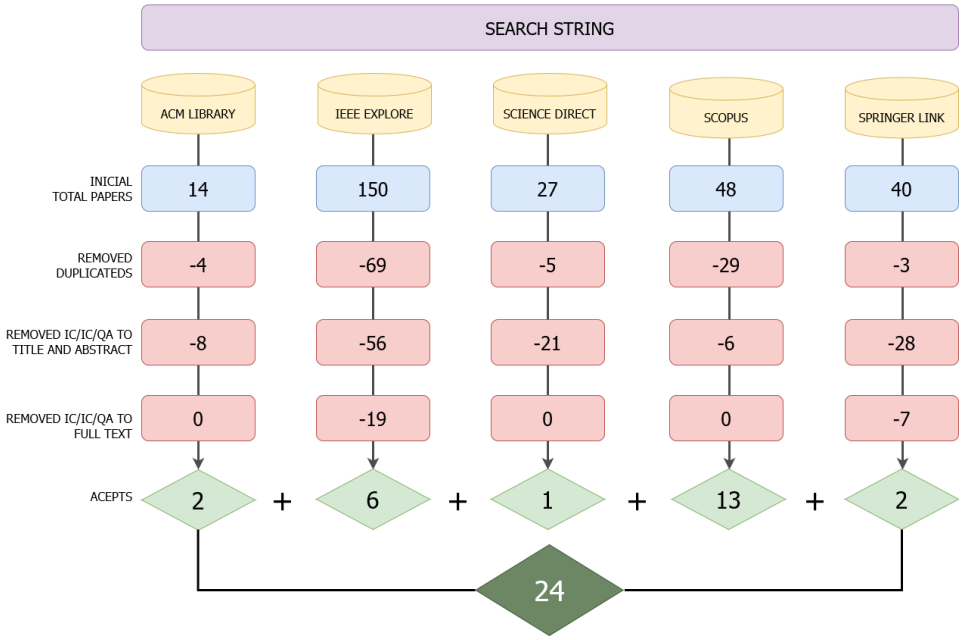
Para realizar a revisão, utilizamos a ferramenta Parsifal [1], uma plataforma web de código aberto dedicada ao suporte de Revisão Sistemática da Literatura (RSL). A escolha do Parsifal se deu em virtude de seus recursos e fluxo de trabalho, os quais foram modelados com base no processo RSL adotado neste estudo, conforme proposto por Kitchenham e Charters [29]. O uso da ferramenta agilizou o processo de revisão, proporcionando uma navegação fácil e rápida por títulos e resumos durante as fases de filtragem, além de detectar automaticamente estudos duplicados. A Figura 3.1 apresenta a quantidade de estudos selecionados em cada etapa da revisão.

Os estudos foram coletados através de buscas realizadas nas bases de dados até julho de 2024, utilizando as strings apresentadas na Tabela 3.3. A Figura 3.1 identifica cada fase do processo de seleção dos artigos.

Fase de Identificação de Artigos

Foram identificados um total de 279 artigos, distribuídos da seguinte forma: 14 da ACM Library, 150 da IEEE Xplore, 27 da Science@Direct's, 48 da Scopus e 40 da Springer Link.

[1] https://parsif.al



IC=Inclusion Criteria EC=Exclusion Criteria QA=Quality assessment

Figura 3.1: Estudos selecionados para extração de dados

Fase de Remoção de Duplicados

Durante esta fase, 110 estudos duplicados foram identificados e removidos, resultando em um conjunto final de 169 estudos para análise. Nesta seleção retaram nas sequintes quantidades de arquitos: 10 da ACM Library, 81 da IEEE Xplore, 22 da Science@Direct's, 19 da Scopus e 30 da Springer Link.

Fase de Aplicação de Critérios de Seleção aos Títulos e Resumos

Os critérios de inclusão e exclusão e garantia de qualidade foram aplicados aos 169 artigos restantes por meio da revisão dos títulos e resumos. Apesar de os artigos inicialmente selecionados abordarem tópicos relacionados à string de busca, 119 estudos foram excluídos por não atenderem aos critérios de seleção estabelecidos ou não terem relação com o contexto desta dissertação, resultando em um total de 50 artigos para análise da próxima fase.

Fase de Aplicação de Critérios de Seleção ao Texto Completo

Na última fase, realizamos a leitura completa dos estudos selecionados. Nessa etapa, aplicamos todos os critérios de seleção estabelecidos em um total de 24 artigos, sendo 2 provenientes da ACM Library, 6 da IEEE Xplore, 1 da Science Direct, 13 da Scopus e 2 da Springer Link.

3.6 Extração de Dados

Na fase de extração de dados, nosso objetivo é garantir a completude da Revisão Sistemática da Literatura (RSL). Para isso, identificamos, avaliamos e interpretamos de forma sistemática todos os trabalhos selecionados. Os dados extraídos servem como base para as análises quantitativas e qualitativas. Assim, a pesquisa não se limitou à mera classificação dos artigos, mas também os avaliou detalhadamente, utilizando-os como ponto de partida para as demais fases da investigação. A Tabela 3.7 resume os metadados mais relevantes dos artigos, incluindo o ano de publicação (Year), a quantidade de citações disponíveis nas bases de dados (#Cite), o tipo de publicação (Artigo de Jornal = J ou Artigo de Conferência = C), o nome do periódico ou conferência em que o estudo foi publicado (Conference/Journal) e o score obtido com base nos critérios de garantia de qualidade (GQ).

Study	Year	#Cite	Type	Conference/Journal	Score	Ref.
PS1	2021	21	C	Information and Software Technology	2.0	[35]
PS2	2020	0	C	Journal of Internet Services and Information Security	1.5	[36]
PS3	2022	1	C	AICMHI 2022	2.0	[37]
PS4	2023	0	C	CLOSER 2023	1.5	[38]
PS5	2019	11	J	IOP Publishing Ltd	1.5	[39]
PS6	2023	0	C	Politechnica University of Bucharest	2.0	[40]
PS7	2020	5	J	ETFA 2020	1.0	[41]
PS8	2021	0	C	ETFA 2021	2.0	[42]
PS9	2023	0	J	International Journal of Web Engineering and Technology	1.5	[43]
PS10	2022	0	J	Journal of Information Processing	2.0	[44]
PS11	2021	0	C	Budapest Tech Polytechnical Institution	2.0	[45]
PS12	2022	1	J	ICTI 2022	1.5	[46]
PS13	2021	1	J	ICITCS 2021	1.5	[47]
PS14	2022	0	C	SummerSOC 2022	2.0	[48]
PS15	2021	11	C	SN Computer Science	2.0	[49]
PS16	2020	4	J	IEEE-HYDCON 2020	1.5	[50]
PS17	2023	1	C	ICSA 2023	1.0	[51]
PS18	2023	1	J	AICT 2023	2.0	[52]
PS19	2019	17	C	IEEE Access Journal	2.0	[53]
PS20	2022	1	C	ACM SIGSOFT Softw. Eng. Notes	2.0	[54]
PS21	2023	1	J	Web3D 2023	1.0	[55]
PS22	2020	9	C	KES-2020	2.0	[56]
PS23	2023	0	C	ISCSIC	2.0	[57]
PS24	2024	0	C	MECO	2.0	[58]

Tabela 3.7: Extração de Dados

Seguindo as diretrizes propostas por Kuhrmann et al.[59], utilizamos uma nuvem de palavras como uma abordagem adicional para avaliar a relevância do nosso conjunto de resultados dos estudos primários. Na Figura 3.2, são destacadas as palavras mais frequentes extraídas dos títulos, palavra-chaves e resumos dos estudos selecionados. A representação visual evidencia o predomínio de termos como “Micro”, “Frontend”, “Web”, “Architecture” e “Microservice”.

O Gráfico 3.3 apresenta dados sobre a quantidade de estudos identificados e selecionados nas bases de dados digitais onde aplicamos a string de busca. O Gráfico nos mostra

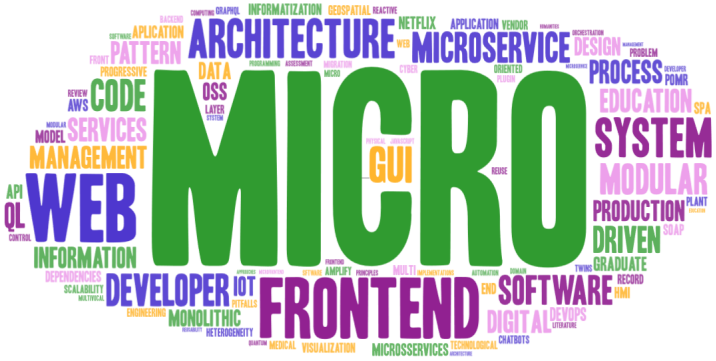


Figura 3.2: Nuvem de Palavras identificadas nos estudos selecionados

que a ACM Digital Library registrou 14 artigos identificados e apenas 2 selecionados. Por outro lado, a IEEE Xplore destacou-se inicialmente com 150 artigos identificados, mas, após a aplicação dos critérios de seleção, apenas 6 foram mantidos. A Science Direct resultou em 27 artigos identificados, dos quais apenas 1 foi selecionado. A Scopus, por sua vez, apresentou 48 estudos identificados, reduzindo para 13 na fase de seleção. Por fim, a Springer Link registrou 40 artigos identificados, mas apenas 2 foram considerados na seleção. Essa análise evidencia a variação significativa entre os números de identificação e seleção em cada base de dados, ressaltando a importância de critérios rigorosos no processo de seleção de artigos como demonstrado na Figura 3.1.

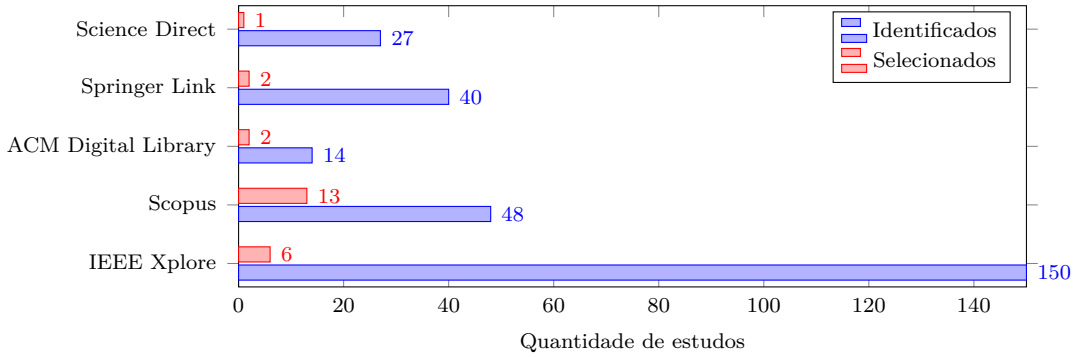


Figura 3.3: Quantidade de estudos identificados e selecionados por base de dados digital

3.7 Resultados da RSL

Nesta seção, apresentamos os resultados da extração de dados. Ao abordar as questões de pesquisa, a interpretação dos resultados destacados nos artigos mostra não apenas as práticas conversacionais adotadas nos estudos selecionados, mas também os impactos e desafios inerentes à implementação de micro-frontends.

A coluna “Visão” destaca os aspectos arquiteturais no processo de tomada de decisões que definem a estrutura e comportamento do software, atendendo seus requisitos funcionais e não funcionais PS21 como por exemplo Single-SPA, Orquestração e *Backend For Frontend* (BFF). Além disso, a Figura 3.4 ilustra a quantidade de artigos que abordaram essa estratégia no contexto da arquitetura micro-frontend.

Já a coluna “Padrões” aborda aspectos arquiteturais que descrevem problemas recorrentes e genéricos em um projeto de software, apresentando, por exemplo, soluções padronizadas [60] como frameworks e linguagens de programação. Também adicionamos a Figura 3.5 para auxiliar na compreensão da relevância do uso desses aspectos para os estudos selecionados.

A coluna “Resultados” apresenta as soluções encontradas pelos autores na tentativa de resolver os impactos e desafios da adoção e implementação da arquitetura micro-frontend, evidenciando os prós e contras de cada solução relacionando-as com as visões e padrões mencionadas nos artigos.

RQ.1. Quais os padrões arquiteturais utilizados na implementação de micro-frontends?

Para responder a RQ.1, nós utilizamos os dados dos estudos selecionados apresentados na Tabela 3.8. Nos estudos foram identificados as visões e padrões arquiteturais utilizados na implementação de micro-frontends, incluindo tecnologias como frameworks, linguagem de programação e ferramentas para desenvolvimento e deploy. Devido a abrangência da da RQ.1, dividimos em dois tópicos: visões arquiteturais e padrões arquiteturais.

Visões Arquiteturais

Durante a análise dos estudos selecionados, identificamos os padrões arquiteturais mais utilizados nas implementações da arquitetura de micro-frontends. Esses padrões representam as estratégias adotadas pelas equipes de desenvolvimento para enfrentar desafios rotineiros, como a escolha de linguagens de programação, frameworks, bibliotecas e decisões específicas sobre a arquitetura das micro-aplicações, como composição e roteamento. Na Figura 3.5, apresentamos cada um desses padrões e o número de artigos que os abordaram.

- **Visão API:** Abordada em 20 estudos (PS1, PS2, PS3, PS4, PS5, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS15, PS16, PS17, PS19, PS20, PS21, PS23 e PS24), esta visão demonstra o papel das APIs como um ponto de entrada único para solicitações de backend, permitindo que os micro-frontends permaneçam independentes e

Study	Views	Patterns	Results
PS1	API, App Container, Devops, Orchestration, Single-SPA	Frameworks(Angular, React and Vue), Languages(Javascript, HTML, CSS), DDD, Application Shell, Communication, Composition, Iframe, Routing	Team scalability, Deployment independence
PS2	API, App Container, Devops, Orchestration, Single-SPA, Web Components	Frameworks(Angular and React), Languages(Javascript, HTML, CSS, Typescript), Composition, Iframe, VCS, Routing, MVC	Operational complexity, increased efficiency in deliveries
PS3	API, BFF, Module Federation, Single-SPA	Framework(React), Composition	Presents the implementation of chatbots with Micro-frontend architecture
PS4	API, App Container, BFF, Orchestration, Single-SPA	Communication	Data optimization, network efficiency, and resource savings
PS5	API, Single-SPA, Web Component	Frameworks(Angular and React), Languages(Javascript, HTML, CSS), Design System, Communication, Iframe, Routing, CMS	Team independence, deployment efficiency, CSS conflicts, and code redundancy
PS6	Module Federation, Single-SPA, Web Components	Languages(Javascript, HTML, CSS), Application Shell, Design System, Communication, Composition, Iframe, Routing	Deployment independence and team autonomy
PS7	App Container, Single-SPA	Framework(Vue), Programming Languages (HTML, CSS), Communication	Development of a module in micro-frontend for HMI
PS8	API, Web Components	Framework(Angular), Programming Languages (Typescript), Design System, Communication	Proposes micro-frontend for modular industrial plant processes
PS9	API, Single-SPA	Frameworks(Angular and Vue), Languages(Javascript, Typescript, HTML, CSS), Design System, Routing, MVC	Prototype development, high performance, reduction of development complexity
PS10	API, Single-SPA, Web Components	Frameworks(React and Vue), Languages(Javascript, CSS), Routing	Inefficiency in team management for small projects, increased operational complexity
PS11	API, App Container, BFF, Devops, Single-SPA, Web Component	Frameworks(Angular, React and Vue), Languages(Javascript, HTML, CSS), Shell Application, Iframe, VCS, Routing, Composition	Implementing migration to micro-frontend and CSS conflicts
PS12	API, App Container, Module Federation, Single-SPA, Web Components	Frameworks(Angular, React and Vue), Languages(Javascript, HTML, CSS), DDD, Routing, Composition, Communication, Iframe	Efficiency in code reuse
PS13	API, Single-SPA	Framework(Angular), Languages(HTML, CSS), Communication, Composition	Efficiency in team management and CSS conflicts
PS14	API, Web Components	Frameworks(Angular and React), Languages(Javascript, Typescript, HTML, CSS), Communication, Composition	Reusability with plugins
PS15	API, DevOps	DDD, Communication, Composition, Design System	Development of a micro-frontend solution for SMEs
PS16	API	Application Shell	Development of a platform to organize and identify functional requirements
PS17	API, Single-SPA, Web components	Languages(Javascript, HTML), Application Shell, Design system, Communication	Development of a micro-frontend solution for small companies
PS18	Module Federation, Orchestration, Single-SPA	Frameworks(Angular, React and Vue), Languages(Javascript, HTML, CSS), Design System, Communication, Composition	Efficiency in team management
PS19	API, Orchestration, Web Components	Frameworks(Angular, React and Vue), Communication, Composition, Iframe, Routing	Effectiveness in implementing dynamic interfaces.
PS20	API, Module Federation, Orchestration, Single-SPA, Web Components	Frameworks(Angular), Languages(Javascript, HTML), Application Shell, Design System, Communication, Composition, Iframe, Routing	Increase in operational complexity
PS21	API, Module Federation, Orchestration, Single-SPA	Frameworks(Angular, React and Vue), Languages(Javascript, Typescript, HTML and CSS), DDD, Routing	Greater scalability and increased operational complexity
PS22	App Container, Single-SPA, Web Component	Frameworks(Angular, React and Vue), Languages(Javascript, Typescript, HTML and CSS), DDD, Communication, Iframe, VCS, Routing, MVC	Improved user experience, decoupling, application scalability, and cost reduction
PS23	API	Routing	Proposes a method based on a bus for micro-frontend
PS24	API, Single-SPA, Web Components, Module Federation	Languages(Javascript, Typescript, HTML and CSS), DDD, Communication, Composition, Iframe, Routing	Explores aspects of microfrontends, such as their benefits and challenges

Tabela 3.8: Views and Architectural Patterns Used in Micro-frontends Implementation

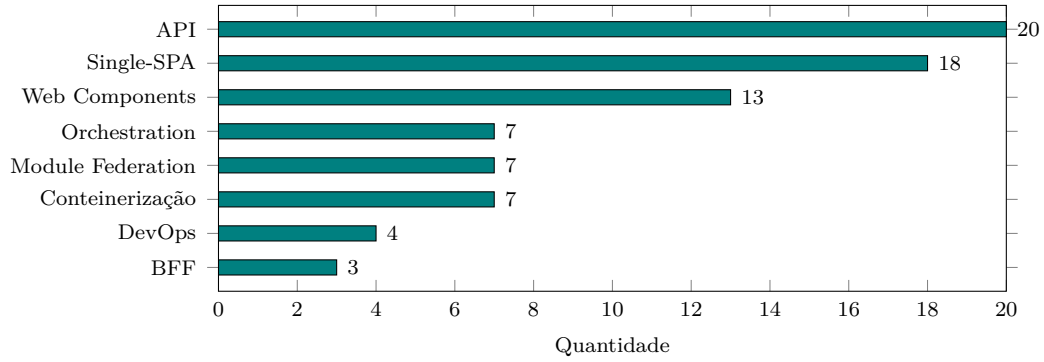


Figura 3.4: Quantidade de artigos por visões arquiteturais abordadas

fracamente acoplados, facilitando a comunicação entre componentes e otimizando o desenvolvimento e a manutenção.

- **Visão Single-SPA:** Mencionada em 18 estudos (PS1, PS2, PS3, PS4, PS5, PS6, PS7, PS9, PS10, PS11, PS12, PS13, PS17, PS18, PS20, PS21, PS22 e PS24), esta visão apoia a escalabilidade e modularidade da aplicação, permitindo a adição de novos micro-frontends sem impactar as partes existentes da aplicação.
- **Web Components:** Discutida em 13 estudos (PS2, PS5, PS6, PS8, PS10, PS11, PS12, PS14, PS17, PS19, PS20, PS22 e PS24). A implementação nos estudos utilizou tecnologias como Custom Elements, HTML Templates, Shadow DOM, HTML e JavaScript sem a necessidade de uma biblioteca ou framework específico.
- **Visões de Container e Orquestração de Aplicações:** A visão App Container (conteinerização) foi referenciada em 7 estudos (PS1, PS2, PS4, PS7, PS11, PS12, PS22). Da mesma forma, a visão Orquestração também foi mencionada em 7 estudos (PS1, PS2, PS4, PS18, PS19, PS20, PS21). Esses termos estão relacionados à forma como as aplicações são encapsuladas (App Container) e roteadas (Orquestração), muitas vezes envolvendo plataformas como Docker e Kubernetes.
- **Module Federation:** Identificada em 7 estudos (PS3, PS6, PS12, PS18, PS20, PS21 e PS24), esta visão enfatiza flexibilidade, compartilhamento de módulos e reutilização de código.
- **Visões DevOps e BFF:** Embora importantes, essas visões foram mencionadas com menos frequência. DevOps foi discutido em 4 estudos (PS1, PS2, PS11 e PS15), sendo notado por sua eficiência em integração e entrega contínuas (CI/CD). A visão BFF apareceu em 3 estudos (PS3, PS4 e PS11), destacando que diferentes partes da interface do usuário podem se comunicar com backends específicos, garantindo uma comunicação direcionada e eficiente.

Essas descobertas destacam as várias visões arquiteturais empregadas na implementação de micro-frontends, enfatizando sua importância em promover modularidade, escalabilidade e comunicação eficaz.

Padrões Arquiteturais

Durante a análise dos artigos, procuramos identificar os padrões arquiteturais mais empregados nas implementações da arquitetura de micro-frontends. Esses padrões representam as estratégias adotadas pelas equipes de desenvolvimento para solucionar desafios rotineiros, como a escolha da linguagem de programação, frameworks, bibliotecas e decisões específicas da arquitetura de micro-aplicações, como composição e roteamento. Na Figura 3.5, apresentamos cada um desses padrões e a quantidade de artigos que os abordaram. A seguir, examinaremos detalhadamente cada padrão, estabelecendo relações com os respectivos artigos.

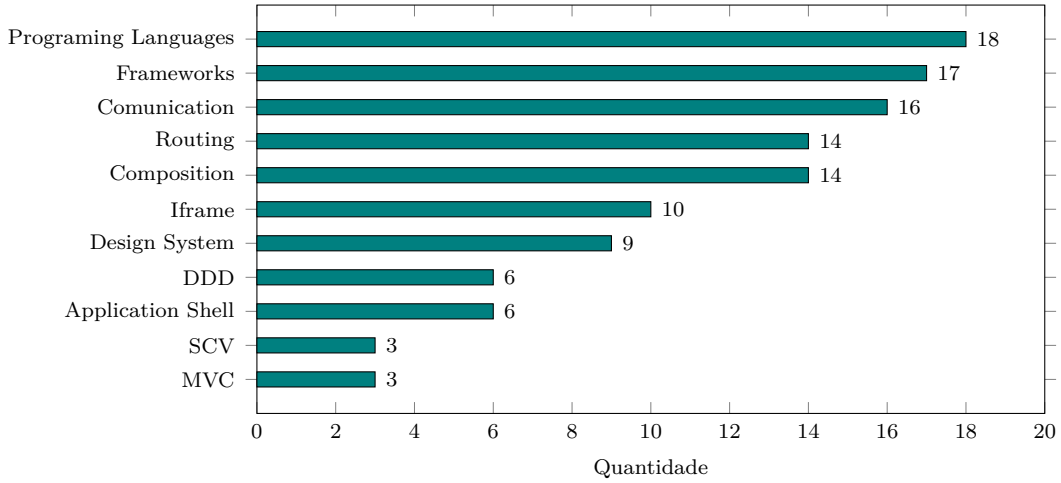


Figura 3.5: Quantidade de artigos por padrões arquiteturais abordados

- **Linguagens de Programação**: Mencionadas em 18 estudos, Javascript, Typescript, HTML e CSS são amplamente utilizadas na camada frontend. Os estudos que abordaram essas linguagens incluem PS1, PS2, PS5, PS6, PS7, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS17, PS18, PS20, PS21, PS22 e PS24. Essas linguagens são frequentemente usadas em conjunto com frameworks como Angular, React e Vue.
- **Frameworks**: A escolha dos frameworks Angular, React e Vue foi a mais abordada nos estudos PS1, PS2, PS3, PS5, PS7, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS18, PS19, PS20, PS21 e PS22. Esses 17 estudos indicaram a flexibilidade proporcionada pelos micro-frontends, permitindo que as equipes adotem suas tecnologias preferidas para desenvolver componentes de forma independente.

- **Padrão de Comunicação**: Utilizado nos estudos PS1, PS4, PS5, PS6, PS7, PS8, PS12, PS13, PS14, PS15, PS17, PS18, PS19, PS20, PS22 e PS24, este padrão promove uma integração mais eficiente entre micro-aplicações. Ele melhora a comunicação entre os componentes, especialmente por meio de roteamento dinâmico, oferecendo um controle e gerenciamento mais eficaz do estado dos componentes.
- **Padrão de Roteamento**: Mencionado nos estudos PS1, PS2, PS5, PS6, PS9, PS10, PS11, PS12, PS19, PS20, PS21, PS22, PS23 e PS24, este padrão melhora a eficiência do carregamento inicial da aplicação ao carregar dinamicamente os micro-frontends. Pode ser gerenciado no lado do servidor, reduzindo o tempo de carregamento e aprimorando a experiência do usuário [1]. O padrão está associado à independência de implantação, autonomia da equipe e facilidade de manutenção do código, embora o estudo PS9 tenha notado uma complexidade operacional aumentada quando usado com SPA e o padrão *Static Assets*.
- **Padrão de Composição**: Analisado nos estudos PS1, PS2, PS3, PS6, PS11, PS12, PS13, PS15, PS14, PS15, PS18, PS19, PS20 e PS24, este padrão inclui Composição no Lado do Cliente, Composição no Lado do Servidor e Composição no Lado da Edge. *Composição* está associada ao aumento da escalabilidade da aplicação e complexidade operacional.
- **Padrão Iframe**: Destacado nos estudos PS1, PS2, PS5, PS6, PS11, PS12, PS19, PS20, PS22 e PS24, este padrão simplifica o carregamento de múltiplas aplicações dentro do navegador, promovendo independência e agilidade nas implantações das equipes. No entanto, pode levar a potenciais conflitos de estilo CSS e redundância de código.
- **Sistema de Design**: Mencionado nos estudos PS5, PS6, PS8, PS9, PS15, PS17, PS18, PS20 e PS24, este padrão aborda a reutilização de código, padronização da interface e redução da complexidade no desenvolvimento de aplicações frontend. O estudo PS24, embora não use explicitamente o termo Sistema de Design, refere-se a ele como Biblioteca de Componentes e Experiência do Usuário (UX), contextualizando-o com padrões de design.
- **Padrão DDD**: Mencionado nos estudos PS1, PS12, PS15, PS21, PS22 e PS24, este padrão alinha estratégias de negócios, unificando produto e tecnologia para maior eficiência operacional.
- **Shell da Aplicação**: Referenciado em 6 estudos (PS1, PS11, PS16, PS17, PS20 e PS24), este padrão monta e desmonta um micro-frontend, melhorando o desempenho

ao carregar rapidamente a estrutura básica da aplicação para uma resposta inicial mais rápida.

- **Model-View-Controller (MVC)**: Abordado nos estudos PS2, PS9 e PS22, este padrão melhora a organização do código-fonte e a eficiência de manutenção.
- **Padrão VCS**: Discutido nos estudos PS2, PS9 e PS22, este padrão é essencial para garantir o controle das versões da aplicação.

Os padrões arquiteturais identificados refletem estratégias e ferramentas diversas empregadas pelas equipes de desenvolvimento para enfrentar desafios comuns nas arquiteturas de micro-frontends.

RQ.1 Sumário: Para responder à RQ.1, foram utilizados dados dos estudos apresentados na Tabela 3.8, que identificaram visões e padrões arquiteturais em micro-frontends, incluindo tecnologias e estratégias. A análise foi dividida em duas partes: A primeira parte tratou das visões arquiteturais que incluiram Single-SPA, BFF, ModuleFederation, API, Orchestration, Web Components, DevOps e Conteinerização conforme Figura 3.4; e a segunda parte tratou dos padrões arquiteturais *Domain-Driven Design* (DDD), Model View Controller (MVC), Routing, System Control Version (SCV), Communication, Iframe, Composition, Design System, Frameworks, Programn Languages e Application Shell. Esses padrões e visões fornecem insights sobre as estratégias adotadas pelas empresas na implementação de micro-frontends, destacando aspectos como escalabilidade, comunicação eficiente e flexibilidade no desenvolvimento.

RQ.2. Quais são os impactos da adoção de micro-frontends em projetos de desenvolvimento de software?

A implementação de micro-frontends apresenta a perspectiva de trazer benefícios tais como escalabilidade, flexibilidade e a capacidade de desenvolvimento independente. Contudo, é importante avaliar atentamente os impactos negativos que podem surgir, notadamente em relação à complexidade na configuração e coordenação. A Figura 3.6 apresenta a proporção de estudos que abordam tanto os impactos positivos quanto os negativos, categorizando-os para uma análise holística e nos parágrafos seguintes analisaremos cada estudo e sua relação com estes impactos. É importante ressaltar que nesta Figura 3.6, agrupamos as estratégias de Desacoplamento, Escalabilidade, Produtividade, Flexibilidade e Performance no item “Eficiência da Aplicação” e as estratégias Componentes, Conflitos e Redundância de Códigos no item “Reusabilidade”.

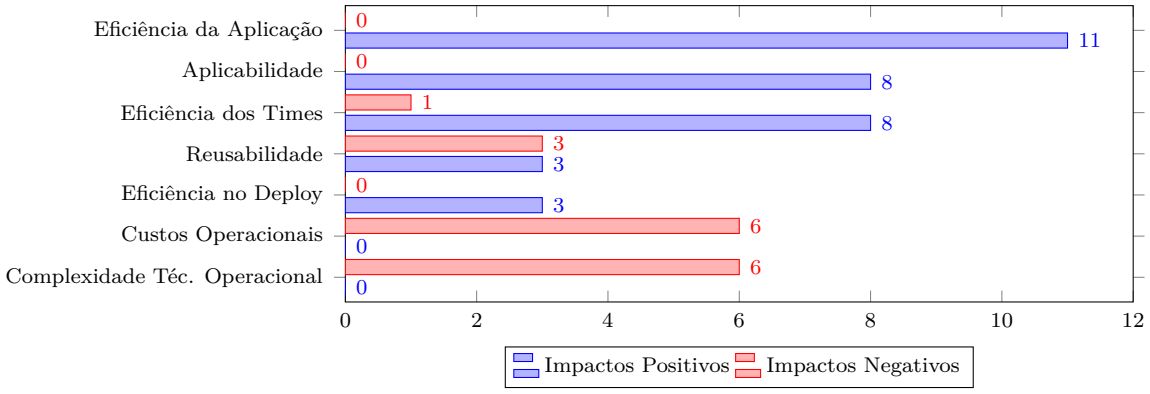


Figura 3.6: Impactos positivos e negativos

Impactos Positivos

Com relação à eficiência da aplicação (Figura 3.6), os estudos PS1, PS8, PS9, PS10, PS11, PS12, PS15, PS20, PS21, PS22 e PS24. Como destaque para o estudo PS20 que trouxe pontos positivos sobre desacoplamento e escalabilidade da aplicação, redução de custos e melhoria na experiência do usuário. Os estudos PS1 e PS15 também mencionaram melhorias na escalabilidade e redução dos custos operacionais. Um estudo em particular, PS21, citou a melhoria no desempenho da aplicação como um destaque positivo, enquanto o estudo PS5 ressaltou a importância da eficácia na implementação de interfaces dinâmicas ao adotar a abordagem de micro-frontends.

No contexto da *Aplicabilidade* dos micro-frontends, os estudos PS3, PS4, PS6, PS14, PS16, PS19, PS23 e PS24 desenvolveram protótipos e casos de uso em cenários de negócios onde os micro-frontends foram adotados com sucesso. Esses estudos destacaram resultados positivos ao aplicar micro-frontends em diversos setores, incluindo a indústria de Interface Humano-Máquina (HMI) (PS14), plantas industriais (PS3) e Internet das Coisas (IoT) (PS16). O estudo PS6 apresentou benefícios na implementação eficaz de chatbots. O estudo PS21 relatou a construção bem-sucedida de plataformas destinadas a organizar funcionalidades associadas a microserviços e micro-frontends. Por outro lado, o estudo PS23 destacou a aplicabilidade do método proposto para melhorar sistemas integrados usando micro-frontends, relatando aumento de eficiência e estabilidade.

A *Eficiência dos Times* resultante de maior independência, escalabilidade e entregas mais rápidas destacou-se como um dos impactos mais mencionados pelos autores dos estudos (PS1, PS8, PS9, PS10, PS11, PS12, PS22 e PS24).

Além disso, a *Reusabilidade* de componentes e plugins (PS2, PS5 e PS17) foi identificada como uma prática influente na velocidade da equipe e padronização da interface.

Finalmente, os estudos PS10, PS12 e PS22 também mostraram impactos positivos na *Eficiência no Deploy*, permitindo implementações independentes, redução de riscos,

facilitação de rollback e atualizações contínuas. Esses benefícios contribuem para um ciclo de desenvolvimento mais ágil e uma aplicação mais resiliente às mudanças.

Impactos Negativos

As maiores ocorrências de impactos negativos, segundo os estudos selecionados, estão associados a complexidades técnicas e operacionais, aos custos operacionais e a reusabilidade de códigos(Figura 3.6). Nos estudos PS2, PS10, PS12, PS20, PS21 e PS24 houve destaque no aumento na Complexidade Operacional como um impacto negativo resultante da adoção da arquitetura de micro-frontends. Por exemplo, o estudo PS12 comparou duas estratégias de implementação de micro-frontends utilizando os padrões Iframes e Module Federation. Os autores identificaram que ter uma arquitetura descentralizada exige escalabilidade nos processos de liberação e implantação para suportar múltiplas aplicações, o que introduz um novo tipo de complexidade. Esse aumento na complexidade foi atribuído, em parte, ao considerável esforço necessário para configurar ambientes, conforme mencionado pelo estudo PS20. Enquanto o estudo PS24 enfatizou a complexidade combinada com a necessidade de planejamento e criação de modelos de comunicação mais eficazes. Essa complexidade pode resultar em uma curva de aprendizado acentuada e exigir práticas eficientes de gerenciamento de configuração.

Por sua vez o aumento nos *Custo Operacionais* foram apontados pelos estudos PS2, PS10, PS12, PS20, PS21 e PS24 destacam que a Complexidade Operacional aumenta devido à necessidade de escalabilidade nos processos de liberação e implantação, o que introduz novos desafios e esforços consideráveis para configurar e manter ambientes. O estudo PS12, por exemplo, comparou diferentes estratégias e encontrou que uma arquitetura descentralizada exige mais esforço, o que pode aumentar os custos operacionais. Além disso, o estudo PS24 enfatizou que a complexidade também está relacionada ao planejamento e criação de modelos de comunicação mais eficazes, resultando em custos adicionais e exigindo práticas de gerenciamento de configuração mais eficientes.

Em relação à *Reusabilidade*, o impacto negativo foi enfatizado pela presença de redundância e conflitos no código-fonte, como evidenciado nos estudos PS5, PS11 e PS13. Esses problemas destacam a importância de adotar técnicas e métodos eficazes de codificação e reutilização como a componentização, o uso de bibliotecas e plugins, como mencionado no estudo PS6, surgem como abordagens destinadas a mitigar a redundância e minimizar os conflitos no código.

Quanto à *Eficiência da Equipe*, embora a maioria dos estudos destaque melhorias na gestão de equipes de desenvolvimento de software - como a autonomia que permite à equipe focar em seu domínio específico, tomando decisões de arquitetura e implementação de forma independente - o estudo PS7 adotou uma perspectiva contrária, relatando

ineficiências nesse aspecto devido ao tamanho relativamente pequeno da aplicação. De acordo com Geers [2], a arquitetura de micro-frontends seria ideal para aplicações de médio a grande porte.

RQ.2 Sumário: Para responder à RQ.2, foram utilizados dados dos estudos apresentados na Tabela 3.8 e o Gráfico 3.6. A análise ressaltou que a adoção de micro-frontends pode impactar positivamente projetos de desenvolvimento de software, proporcionando benefícios como maior eficiência dos times, otimização no processo de deploy, aplicabilidade em diversos setores e melhorias na eficiência da aplicação. No entanto, também destacou impactos negativos, como aumento da complexidade técnica, desafios relacionados ao tamanho da aplicação e problemas de reusabilidade.

RQ.3. Quais são os desafios enfrentados pelas equipes de desenvolvimento de software na adotação de micro-frontends?

De acordo com os estudos selecionados (Tabela 3.8), os desafios enfrentados pelas equipes de desenvolvimento de software ao adotar micro-frontends incluem:

Desafios na Coordenação da Equipe. Como diferentes equipes podem estar desenvolvendo micro-frontends independentes, a coordenação entre essas equipes pode se tornar complexa. É essencial estabelecer estratégias eficazes de comunicação e colaboração para prevenir conflitos e garantir a coesão do software. Apenas o estudo PS7 relatou a coordenação da equipe como um desafio, provavelmente devido à pequena escala do projeto.

Desafios na Configuração de Ambiente. Configurar ambientes para suportar a execução e integração contínua de múltiplos micro-frontends pode ser desafiador. Garantir que todos os ambientes estejam alinhados e funcionando corretamente é importante para a consistência e estabilidade do sistema. A configuração de ambientes para micro-frontends envolve tanto complexidade técnica quanto custos operacionais, conforme apresentado na Figura 3.6. Esses desafios também foram relatados nos estudos PS9, PS7, PS18, PS15 e PS12. Além disso, os estudos PS1, PS3, PS13, PS16, PS5 e PS15 enfatizaram a importância de uma abordagem estratégica e proativa para superar os desafios de orquestração, considerando fatores específicos do projeto, como requisitos, tecnologias envolvidas (Docker e Kubernetes) e escalabilidade pretendida.

Desafios na Eficiência da Aplicação. Esse desafio inclui aspectos inerentes ao desenvolvimento de software, como as visões e padrões apresentados na Tabela 3.8, usados para alcançar resultados como desacoplamento, escalabilidade, produtividade, flexibilidade e desempenho. Os aspectos incluem gerenciamento de estado, padrões de comunica-

ção, navegação e roteamento, composição e experiência do usuário, incluindo a experiência do desenvolvedor. Esses desafios foram identificados nos estudos PS1, PS10, PS15, PS18, PS19, PS20, PS21 e PS24.

Desafios na Reusabilidade. Os estudos PS22, PS13 e PS8 identificaram desafios relacionados à complexidade do código e à reutilização de componentes, incluindo dificuldade em gerenciar eficientemente conflitos de estilo CSS e redundância de código (PS13). Esses desafios destacam a dificuldade inerente em integrar vários frontends e equipes trabalhando em um único sistema, mesmo quando divididos entre diferentes equipes e micro-aplicações, já que o sistema deve, em última análise, parecer integrado para uma melhor experiência do usuário. Isso ressalta a importância de aderir a estratégias arquitetônicas eficazes para o contexto de micro-frontends, mitigando conflitos de estilo e evitando duplicação de código. Uma solução, por exemplo, é a adoção de um Design System, conforme mencionado nos estudos PS10, PS12, PS3, PS21, PS13, PS2, PS11, PS18 e PS15.

RQ.3 Sumário: Para responder à RQ.3, foram utilizados dados dos estudos apresentados na Tabela 3.8 e o Gráfico 3.6. Identificamos desafios que abordaram questões como a coordenação entre equipes, configuração de ambientes, eficiência da aplicação e reusabilidade. A análise mostrou que coordenação entre equipes pode tornar-se complexa devido ao desenvolvimento independente de micro-frontends, exigindo comunicação eficaz e estratégias de colaboração. Outrossim a configuração de ambientes para suportar múltiplos micro-frontends apresenta desafios técnicos e operacionais, destacando-se nos estudos como um ponto crítico. Também foram identificados na análise desafios na eficiência da aplicação que abrangem a gestão de estados, padrões de comunicação e experiência do usuário, enquanto a reusabilidade enfrenta dificuldades na integração de códigos e componentes, especialmente em relação a conflitos de estilo e redundância de código. A necessidade de estratégias arquiteturais eficazes, como a adoção de Design Systems, é ressaltada como uma abordagem para mitigar esses desafios.

3.8 Ameaças à Validade

Nesta seção, seguindo a estrutura proposta por Wohlin et al. [61], abordamos as principais ameaças à validade em relação à Revisão Sistemática da Literatura (RSL). Inicialmente, focalizaremos a Validade de Construção, assegurando a precisão e consistência nas definições de conceitos e na escolha de métricas. Em seguida, exploramos a Validade Interna, identificando e controlando possíveis ameaças à conclusão causal, enfatizando a interpre-

tação criteriosa das relações entre variáveis. Na sequência, direcionamos nossa atenção à Validade Externa, avaliando a generalização dos resultados para diferentes contextos e a representatividade da amostra. Por fim, abordamos a Confiabilidade, visando garantir a consistência e reprodutibilidade dos procedimentos adotados ao longo da RSL.

3.8.1 Validade de Construção

Ao explorar as ameaças à validade de construção em uma revisão sistemática centrada em micro-frontends, é importante considerar a clareza nas definições de conceitos, termos e métricas utilizadas nos estudos primários. Esta subseção aborda as estratégias adotadas para manter a validade de construção ao longo do processo de revisão.

Consistência nas Definições de Micro-Frontends

Para garantir a validade de construção nesta revisão sistemática, dedicamos especial atenção à consistência nas definições e concepções de micro-frontends presentes nos estudos selecionados. Identificamos a ameaça potencial de divergências conceituais, como por exemplo diferentes interpretações sobre comunicação entre componentes, variação de interpretação entre estratégias específicas (Single-SPA ou Module Federation) e perspectivas organizacionais. Demos prioridade à inclusão daqueles que adotaram definições alinhadas com práticas e conceitos amplamente aceitos no cenário de micro-frontends.

Além disso, cada estudo foi minuciosamente analisado em seu contexto de negócio, levando em consideração as distintas perspectivas conceituais presentes em suas visões e padrões adotados. Esta abordagem permitiu-nos mapear as relações entre tais perspectivas e os resultados encontrados em cada estudo apresentado na Tabela 3.8, proporcionando uma compreensão holística das interpretações atribuídas ao conceito de micro-frontends na literatura.

Ambiguidade nas Métricas Utilizadas

A variabilidade das métricas e indicadores utilizados para avaliar práticas de micro-frontends pode introduzir ambiguidade e comprometer a validade de construção. Adotamos uma abordagem sistemática ao extrair dados dos estudos primários, identificando e padronizando métricas relevantes por tipo de estratégia, conforme apresentado nos gráficos de visões (Figura 3.4) e padrões (Figura 3.5)mencionados nos estudos selecionados. Isso envolveu a identificação de cada métrica utilizada pela literatura.

Definições Claras de Termos-Chave

Para mitigar a ameaça à validade de construção relacionada à utilização ambígua ou interpretações diversas de termos-chave, ao definir a string de busca, optamos pelo termo mais abrangente e frequentemente utilizado: *(micro-frontend OR microfrontend OR "micro frontend")*. Essa escolha visou abranger as diferentes formas de referência à arquitetura de micro-frontends.

Reconhecendo a sensibilidade na manipulação dos operadores lógicos, optamos por restringir nossa escolha ao operador OR durante a definição da string de busca. Essa decisão foi motivada pela constatação de que a inclusão do operador AND resultava em filtragens excessivas, uma vez que o AND só traria artigos que tivessem a combinação de todos os termos, levando à exclusão de estudos relevantes e, ao mesmo tempo, à incorporação de estudos não pertinentes ao contexto de micro-frontends. Essa estratégia visou alinhar as interpretações dos termos com o contexto específico, reduzindo ambiguidades e assegurando a coleta de estudos aderentes ao contexto pesquisado. Dessa forma, a definição de termos-chave fortaleceu a validade de construção da revisão sistemática.

Estratégias de Mitigação Específicas

As estratégias de mitigação adotadas incluíram a revisão das definições e métricas utilizadas nos estudos primários, buscando uma compreensão clara e consistente das metodologias dos estudos aplicando inicialmente a fase de critérios de seleção aos títulos e resumos e posteriormente a leitura completa dos artigos selecionados. A comunicação sobre as decisões de inclusão, métricas padronizadas e definições de termos na revisão sistematizou o processo, fortalecendo a validade de construção ao proporcionar uma base sólida para a interpretação dos resultados.

3.8.2 Ameaças à Validade Interna

Dentro do contexto específico de micro-frontends, é importante explorar e abordar possíveis ameaças à validade interna que possam comprometer a interpretação dos resultados obtidos nesta revisão sistemática.

Viés de Seleção e Causalidade

A ameaça à validade interna relacionada ao viés de seleção pode surgir quando a escolha dos estudos primários não é aleatória ou representativa. No caso de micro-frontends, a variação nas práticas de implementação pode introduzir viés na seleção dos estudos. Para mitigar essa ameaça, estabelecemos critérios de inclusão, buscando uma representação equilibrada das diferentes abordagens utilizadas na implementação de micro-frontends.

Além disso, ao interpretar os resultados durante a fase de extração e análise dos dados e ao respondermos as questões, consideramos a possibilidade de outros fatores (visões e padrões) que podem influenciar as relações causais, promovendo uma análise mais abrangente.

Ameaças à Conclusão Causal

Ao investigar a relação de causa e efeito entre aspectos específicos de micro-frontends, questões como a existência de fatores de confusão não identificados podem distorcer as conclusões. Para lidar com essa ameaça, discutimos possíveis variáveis de confusão que poderiam afetar os resultados. Além disso, destacamos a natureza observacional dos estudos incluídos, reconhecendo as limitações inerentes à inferência causal em contextos como o desenvolvimento de micro-frontends. A Tabela 3.9 tem como base a relação dos impactos apresentados na Figura 3.6 e nos ajudou a identificar os fatores de confusão:

Aspectos	**Fator de Confusão**	**Impacto Potencial**
Reusabilidade	Divergências Conceituais na Adoção de Padrões	Consistência na Definição de Práticas e Conceitos, Aumento da Reusabilidade
Aplicabilidade	Uso da Arquitetura em Diferentes Contextos de Negócio	Adequação da Arquitetura a Diferentes Contextos
Complexidade Técnica Operacional	Variação nas Estratégias de Implementação	Aumento da Complexidade, Necessidade de Treinamento Adicional
Custos Operacionais	Diversidade nas Implementações	Variação nos Custos, Necessidade de Recursos Adicionais
Eficiência da Aplicação	Escala e Complexidade do Projeto	Desempenho, Utilização Eficiente de Recursos
Eficiência no Deploy	Variação nas Estratégias Operacionais de DevOps	Velocidade de Implantação, Consistência nas Entregas
Eficiência dos Times	Ambiguidade nas Metodologias de Colaboração	Cooperação Efetiva, Produtividade da Equipe

Tabela 3.9: Aspectos e Fatores de Confusão em Micro-Frontends

A Tabela 3.9 apresenta os aspectos críticos e fatores de confusão em micro-frontends, sendo essenciais para a compreensão da validade das conclusões causais desta pesquisa. No que se refere à reusabilidade, a divergência conceitual na adoção de padrões pode criar ambiguidades nos resultados, comprometendo a consistência na definição de práticas e conceitos. Contudo, um aumento na reusabilidade pode ser um indicador positivo, desde que as divergências conceituais sejam devidamente tratadas durante a análise.

A aplicabilidade da arquitetura em diferentes contextos de negócio é um fator crítico que pode influenciar a validade das conclusões causais. Se a arquitetura apresentar variações significativas em sua aplicação em contextos diversos, a generalização dos resultados pode ser desafiadora, destacando a importância de avaliar a adequação da arquitetura em diferentes cenários.

A complexidade técnica operacional, influenciada pela variação nas estratégias de implementação, representa outro ponto crucial. Diferenças nas abordagens podem impactar a interpretação dos resultados, especialmente quando se considera o aumento da complexidade e a possível necessidade de treinamento adicional. Os custos operacionais, por sua vez, são suscetíveis à diversidade nas implementações, introduzindo variações nos custos que devem ser cuidadosamente consideradas. Entender a relação entre diversidade nas implementações e custos operacionais é essencial para mitigar ameaças à validade das conclusões.

A eficiência da aplicação, em termos de desacoplamento, escalabilidade, produtividade, flexibilidade e performance, pode ser afetada pela escala e complexidade do projeto. Projetos maiores e mais complexos podem ter requisitos de eficiência distintos, evidenciando a necessidade de avaliação contextualizada. A eficiência no deploy, relacionada à variação nas estratégias operacionais de DevOps, representa um aspecto crítico na validação das conclusões. A consistência nas entregas e a velocidade de implantação são indicadores-chave que demandam uma análise cuidadosa em face da diversidade nas estratégias. Por fim, a eficiência dos times, vinculada à ambiguidade nas metodologias de colaboração, destaca a importância da cooperação efetiva e da produtividade da equipe. Uma compreensão aprofundada desses aspectos é fundamental para garantir a validade das conclusões causais em estudos relacionados a micro-frontends.

Estratégias de Mitigação Específicas

Para atenuar as ameaças à validade interna, ressaltamos a importância da seleção dos estudos primários e buscamos interpretar as relações de causa e efeito entre estratégias abordadas pelos estudos e os resultados evidenciados. Reconhecendo a complexidade dessas relações, buscamos fornecer uma análise robusta, considerando possíveis variáveis e destacando a necessidade de futuras pesquisas experimentais para validar causalmente as práticas observadas.

3.8.3 Ameaças à Validade Externa

Ao explorar as ameaças à validade externa nesta revisão sistemática centrada em micro-frontends, consideramos as nuances específicas desse domínio em constante evolução. As características singulares do desenvolvimento de micro-frontends podem influenciar a generalização dos resultados e a representatividade da amostra de estudos primários.

Representatividade da Amostra

As estratégias utilizadas na adoção e implementação da arquitetura de micro-frontends podem variar, conforme apresentado nas Figuras 3.5 e 3.4. A diversidade de abordagens e tecnologias utilizadas por diferentes organizações pode introduzir vieses na representatividade da amostra. Para contornar essa ameaça, seguimos os critérios de seleção, orientado por Kitchenham e Charters [30]. Estabelecemos os critérios de inclusão e exclusão e delineamos os parâmetros de seleção que contemplam uma variedade de implementações de micro-frontends segundo as aplicabilidades encontradas nos estudos PS3, PS7, PS8, PS15, PS16, PS17 e PS23, garantindo a representatividade da amostra em relação às práticas mais difundidas na área.

Generalização dos Resultados

Dada a natureza modular e descentralizada da arquitetura baseada em micro-frontends, os resultados obtidos em um contexto específico podem não ser diretamente aplicáveis a outros ambientes. Adotamos uma abordagem transparente ao discutir os resultados e os contextos específicos abordados pelos estudos primários, oferecendo insights sobre a extensão da aplicabilidade dos achados em diversas configurações de micro-frontends. Isso assegura a representatividade da amostra, abordando visões e padrões aplicados, como Single-SPA, BFF, Module Federation, API, Orchestration, Web Components, Composição, Design System, Frameworks, entre outros, conforme apresentado na Tabela 3.8.

Estratégias de Mitigação Específicas

Para atenuar possíveis ameaças à validade externa nesse contexto, ressaltamos a relevância de contemplar a diversidade nas implementações de micro-frontends ao interpretar e generalizar os resultados. Os estudos selecionados oferecem cenários diferentes aplicados à arquitetura de micro-frontends, proporcionando insights valiosos e distintos sobre sua eficácia e aplicabilidade.

O estudo de PS22 concentra-se no sistema de gestão educacional, enquanto PS3 explora a implementação em registros médicos. PS5 aborda o desenvolvimento de um sistema de gerenciamento de conteúdos, e PS7 direciona-se à indústria de *Human Machine Interface* (HMI). A pesquisa de PS8 oferece insights valiosos sobre plantas de processos modulares, enquanto PS11 discute migrações para micro-frontends. Já o estudo PS17 se concentra na adoção de micro-frontends em pequenos projetos, PS19 explora a combinação de dados geoespaciais na indústria da *Internet of Things* (IoT), e PS21 aborda a indústria de Digital Twin (DT). Os demais estudos selecionados, PS1, PS2, PS4, PS6, PS9, PS10, PS12, PS13, PS14, PS15, PS16, PS18, PS20 e PS23 focaram seus esforços nos detalhes técnicos

da adoção e implementação da arquitetura. Essa diversidade de contextos fortalece a robustez e a representatividade da análise realizada, proporcionando uma compreensão holística da arquitetura de micro-frontends.

3.8.4 Confiabilidade

No contexto específico de micro-frontends, abordar a confiabilidade é importante para avaliar até que ponto os dados e a análise apresentados são dependentes dos pesquisadores envolvidos na revisão sistemática. Esta subseção explora os aspectos relacionados à confiabilidade dos resultados e como ameaças a essa dimensão da validade foram mitigadas ao longo do processo. A confiabilidade neste contexto refere-se à consistência e reprodutibilidade dos resultados se o estudo fosse conduzido por diferentes pesquisadores. Considerando a natureza complexa das arquiteturas de micro-frontends, a interpretação dos dados coletados e a análise das práticas podem estar sujeitas a variações entre pesquisadores. Buscamos garantir a confiabilidade dos resultados por meio dos critérios de inclusão, exclusão, e critérios de garantia de qualidade para assegurar uma compreensão consistente das práticas de micro-frontends.

As estratégias adotadas para garantir a confiabilidade incluíram a documentação dos processos de coleta e análise de dados, discussões regulares entre os revisores para esclarecimento de dúvidas e a padronização de critérios de inclusão. Além disso, reconhecemos a importância da transparência nas decisões metodológicas para permitir que outros pesquisadores repliquem e avaliem a consistência dos resultados.

3.9 Síntese do Capítulo

Este capítulo apresentou o protocolo empregado na condução da Revisão Sistemática da Literatura (RSL), fornecendo uma visão das etapas adotadas. Os resultados foram sintetizados e apresentados ao longo do Capítulo. A busca nas bases de dados digitais resultou em um conjunto inicial de 279 artigos, os quais foram refinados por meio da remoção de duplicatas e da aplicação de filtros baseados em título, resumo e texto completo. O processo de seleção resultou na seleção de 24 estudos primários.

Capítulo 4

Guia para Adoção de Micro-Frontends (GAM)

Com os dados extraídos dos estudos selecionados na RSL, foi possível propor um conjunto estruturado de diretrizes, práticas recomendadas, ferramentas e recursos para auxiliar e orientar as organizações na adoção eficaz de arquiteturas baseadas em Micro-frontends. O primeiro passo foi criar uma Big Picture (Figura 4.1), a mesma expressão usada por Geers [2] para representar a ideia do micro-frontend, que sintetiza os achados dos estudos selecionados. Com base nessa Figura, desenvolvemos um guia em formato de página web que apresenta as práticas conversacionais e em que situações elas devem ser aplicadas. Esse guia serve como parte integrante do Guia de Adoção de Micro-frontends (GAM), o qual está disponível nos endereços https://github.com/giovanniamorim/gam e https://giovanniamorim.github.io/gam/.

4.1 Introdução ao Guia

A abordagem de Micro-Frontends emerge como um paradigma fundamental para a construção de sistemas frontend escaláveis e flexíveis. Neste capítulo, apresentamos o GAM, um conjunto de princípios e recomendações que reúne o conhecimento consolidado na indústria de software sobre a arquitetura micro-frontend, enriquecido pelos achados dessa pesquisa e as estratégias abordadas por Luca Mezzalira [1] no livro *Building Micro-frontends* e Michael Geers [2] no seu livro *Micro-frontend In Action*. Além disso, incorporamos nos passos da metodologia para o desenvolvimento do GAM, princípios fundamentais da engenharia de software presentes no Guia SWEBOK [21].

Também incluímos a Implementação Prática, envolvendo decisões operacionais e de integração, até a expansão, que engloba aspectos de escalabilidade e manutenibilidade. O

GAM se propõe a guiar equipes de desenvolvimento, arquitetos de software e líderes de projeto em tomadas de decisões mais assertivas para o êxito da adoção desta abordagem.

The Big Picture

Ao compreender a jornada da adoção de Micro-Frontends, deve-se enxergar o cenário de forma abrangente, capturando as nuances críticas que moldam o sucesso dessa adoção arquitetural. Essa perspectiva holística, conhecida como "Big Picture", oferece um olhar amplo e interconectado sobre as etapas-chave envolvidas, delineando um fluxo estruturado que orienta desde o Estudo da Viabilidade até a expansão.

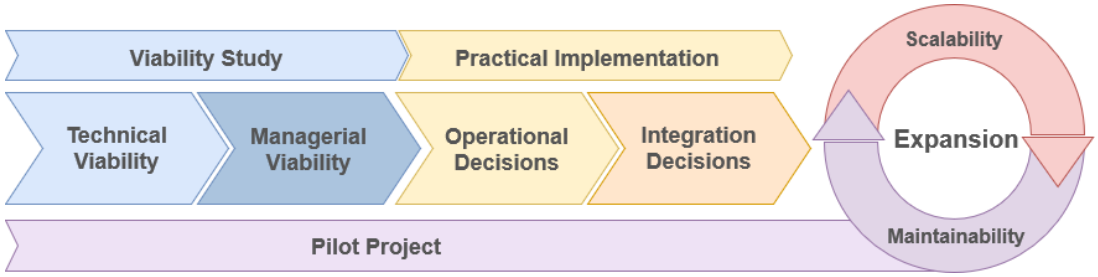


Figura 4.1: The Big Picture - GAM, Fonte: Autor

O ponto de partida, o "Estudo da Viabilidade", tem como papel avaliar fatores como o tamanho do projeto, a eficácia da comunicação da equipe e a complexidade funcional. Essas considerações fornecem a base para tomada de decisões fundamentais, direcionando o rumo da arquitetura a ser adotada. Esta fase foi fortemente influenciada pelo primeiro capítulo do livro de Gears [2], "Micro-Frontend in Action". Neste capítulo, o autor aborda a questão decisiva de determinar a necessidade de adoção de Micro-Frontends, questionando: "*When does Micro-Frontend make sense?*".

A "Implementação Prática" surge como a materialização das decisões operacionais e de integração. Este estágio foca (mas não se limita) à escolha das decisões operacionais e de integração, enquanto um projeto piloto é desenvolvido pela equipe. Neste ponto, delineamos não apenas o "o quê" da arquitetura, mas também o "como" ela será integrada ao ambiente existente.

A etapa de "Expansão" mostra a ideia de escalabilidade e manutenibilidade contínuas. Ela reconhece que o aprimoramento da aplicação é um processo iterativo e que, à medida que a necessidade de escala se manifesta, as decisões tomadas anteriormente devem ser reavaliadas e ajustadas. A escalabilidade e manutenibilidade tornam-se, assim, elementos dinâmicos e essenciais, moldando a evolução contínua da arquitetura.

Por fim, a etapa do "Projeto Piloto", representa um ambiente controlado, onde as fases anteriores previamente delineadas serão aplicadas de maneira tangível. Ele se aplica tam-

bém à fase de expansão quando a equipe avalia na práticas as estratégias de escalabilidade e manutenibilidade do sistema.

4.2 Estrutura do Guia

O GAM foi estruturado de forma a fornecer uma orientação abrangente e prática para equipes de desenvolvimento, arquitetos de software e líderes de projeto. A seguir, destacamos as principais seções que compõem a estrutura do GAM, delineando como o guia está organizado para oferecer diretrizes claras e práticas recomendadas.

4.2.1 Estudo da Viabilidade

A etapa de Estudo da Viabilidade do GAM é a fase que busca analisar as viabilidades gerenciais e técnicas antes de decidir sobre a adoção da arquitetura de Micro-Frontends. Essa avaliação é subdividida em dois momentos distintos: Viabilidade Técnica e Viabilidade Gerencial. É fundamental destacar que o estudo da viabilidade técnica precede o estudo da viabilidade gerencial. Essa abordagem garante que a gestão tenha artefatos e informações suficientes para analisar, por exemplo, os custos associados à transição, contribuindo para uma decisão fundamentada. A Figura 4.2 ilustra o fluxo das análises necessárias para que a equipe decida pela adoção ou não da arquitetura.

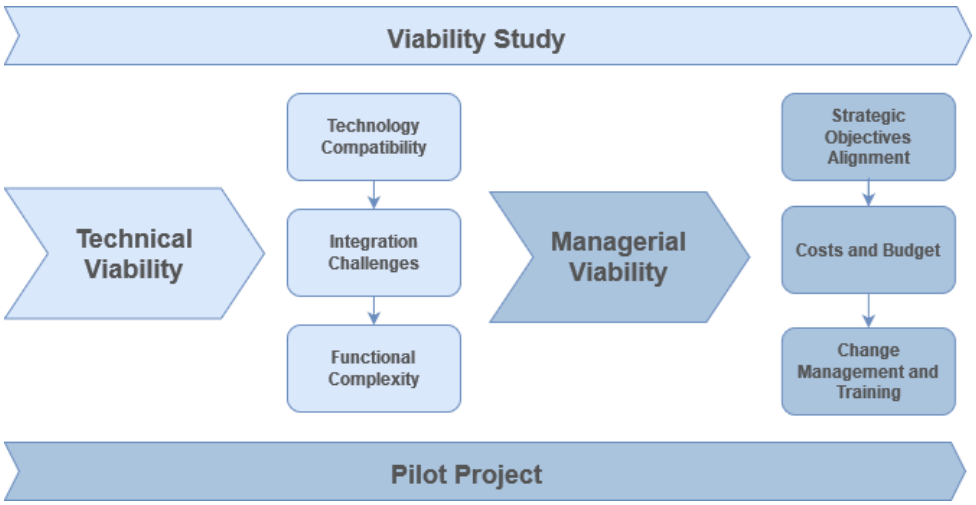


Figura 4.2: GAM - Fase do Estudo da Viabilidade, Fonte: Autor

Viabilidades Técnicas

- **Compatibilidade Tecnológica:**

- **Objetivo:** Garantir que a arquitetura de Micro-Frontends seja compatível com as tecnologias existentes na infraestrutura da empresa, minimizando conflitos e promovendo uma transição suave.
- **Recomendações:**
  * *Avaliação da Infraestrutura Atual:* Realizar uma análise detalhada da infraestrutura tecnológica existente, identificando as tecnologias, frameworks e bibliotecas em uso.
  * *Identificação de Conflitos Potenciais:* Identificar possíveis conflitos entre as tecnologias utilizadas atualmente e aquelas associadas à arquitetura de Micro-Frontends.
  * *Padronização de Tecnologias:* Considerar a padronização de tecnologias que são comuns tanto na infraestrutura atual quanto na arquitetura de Micro-Frontends para facilitar a integração.
  * *Testes de Compatibilidade:* Implementar testes específicos usando o projeto piloto para verificar a compatibilidade entre os componentes de Micro-Frontends e a infraestrutura existente.

Exemplo Prático: Suponha que a empresa utilize o Angular como framework principal em seus sistemas existentes. Ao iniciar a adoção de Micro-Frontends, a equipe de análise de compatibilidade avalia a versão específica do Angular em uso e confirma sua compatibilidade com os requisitos do Micro-Frontend. Além disso, ao identificar a necessidade de uma biblioteca de gerenciamento de estado, a equipe opta por padronizar o uso do Redux, que é amplamente aceito tanto na infraestrutura atual quanto na arquitetura de Micro-Frontends. Durante o projeto piloto, são conduzidos testes específicos para verificar a interoperabilidade entre os Micro-Frontends e os serviços legados, garantindo uma transição suave e minimizando conflitos.

- **Desafios de Integração:**
  - **Objetivo:** Identificar e abordar proativamente os desafios de integração que possam surgir durante a adoção da arquitetura de Micro-Frontends.
  - **Recomendações:**
    * *Mapeamento de Interfaces:* Realizar um mapeamento claro das interfaces entre os Micro-Frontends e outros componentes do sistema.
    * *Definição de Contratos:* Estabelecer contratos claros e documentados entre os Micro-Frontends e os serviços backend, garantindo consistência nas interações.
    * *Monitoramento Contínuo:* Implementar mecanismos de monitoramento contínuo para identificar e resolver rapidamente problemas de integração.

Exemplo Prático: Durante a implementação de Micro-Frontends, um desafio de integração foi identificado na comunicação entre um Micro-Frontend específico e um serviço backend essencial. Utilizando o mapeamento de interfaces previamente documentado, a equipe consegue rapidamente diagnosticar o problema e atualizar o contrato de API. O monitoramento contínuo alerta a equipe sobre a anomalia, permitindo uma intervenção rápida e minimizando o impacto nos usuários finais.

- **Complexidades Funcionais:**
  - **Objetivo:** Entender e mitigar as complexidades funcionais associadas à arquitetura de Micro-Frontends, garantindo a entrega de funcionalidades de forma eficiente.
  - **Recomendações:**
    * *Análise de Requisitos:* Realizar uma análise aprofundada dos requisitos funcionais da aplicação, identificando possíveis complexidades.
    * *Design Modular:* Adote um design modular para os Micro-Frontends, dividindo a aplicação em componentes independentes para facilitar o gerenciamento das complexidades.
    * *Testes Unitários e de Integração:* Implementar testes unitários e de integração para validar a funcionalidade dos Micro-Frontends individualmente e em conjunto.
    * *Documentação Clara:* Mantenha uma documentação clara e abrangente que descreva as interações funcionais entre os Micro-Frontends e outros componentes

Exemplo Prático: Durante a análise de requisitos foi identificado um requisito funcional complexo relacionado à gestão de permissões de usuários em diferentes Micro-Frontends. Ao adotar um design modular, a equipe decide criar um Micro-Frontend dedicado exclusivamente à gestão de permissões. Testes unitários e de integração são implementados para garantir que essa funcionalidade se integre sem problemas aos demais Micro-Frontends. A documentação clara detalha os fluxos de interação, simplificando a compreensão da complexidade funcional pela equipe.

Viabilidades Gerenciais

- **Alinhamento aos Objetivos Estratégicos**
  - **Objetivo:** Assegurar que a implementação da arquitetura Micro-frontend esteja alinhada aos objetivos estratégicos da organização.
  - **Recomendações:**

    * *Análise de Impacto na Gestão:* Realizar uma análise do impacto da introdução de micro-frontends de forma progressiva nos objetivos estratégicos da gestão. Avaliar a viabilidade de alinhamento gradual, sem forçar uma transição imediata.
    * *Mapeamento de Benefícios:* Identificar os benefícios potenciais da adoção gradual de micro-frontends para os objetivos estratégicos da empresa, priorizando a harmonização com as metas existentes.
    * *Ajustes de Metas Estratégicas:* Analisar a necessidade de ajustar ou realinhar as metas estratégicas para otimizar a adoção de micro-frontends, assegurando que estejam alinhadas com a visão a longo prazo da organização.

> **Exemplo Prático**: Para alinhar a adoção da arquitetura de micro-frontends aos objetivos estratégicos, a gestão deve identificar metas específicas, envolver stakeholders, estabelecer indicadores-chave de desempenho (KPIs) mensuráveis, implementar metodologias ágeis, realizar avaliações contínuas com feedback dos usuários finais, ajustar estratégias conforme necessário e manter uma comunicação transparente sobre o progresso e desafios.

- **Custos e Orçamento**
  - **Objetivo:** Avaliar os impactos financeiros relacionados à adoção da arquitetura de micro-frontends no contexto do GAM, identificando custos associados e garantindo uma gestão orçamentária eficaz
  - **Recomendações:**
    * *Levantamento Detalhado de Custos:* Realizar um levantamento dos custos associados à implementação de micro-frontends. Incluir despesas relacionadas ao treinamento da equipe, possíveis atualizações de infraestrutura e aquisição de ferramentas específicas.
    * *Orçamento Flexível:* Estabelecer um orçamento flexível que permita adaptações conforme a evolução do projeto de adoção de micro-frontends. Incorporar uma margem para imprevistos e ajustes durante as fases iniciais da implementação.
    * *Avaliação de Retorno sobre Investimento (ROI):* Desenvolver métricas para avaliação do retorno sobre o investimento. Monitorar indicadores-chave para garantir que os benefícios esperados estejam alinhados aos recursos financeiros empregados.

Exemplo Prático: Ao adotar a arquitetura de micro-frontends, a empresa XYZ precisa avaliar os custos associados à transição. Isso inclui despesas com a capacitação da equipe, aquisição de ferramentas específicas para o desenvolvimento e manutenção da nova arquitetura, e possíveis custos de licenciamento de plataformas. Recomenda-se a realização de uma análise minuciosa para estimar esses custos, permitindo a alocação adequada de recursos no orçamento. Além disso, a criação de cenários alternativos, como a continuidade da estrutura atual ou a avaliação de outras arquiteturas, proporciona uma visão abrangente das opções disponíveis, contribuindo para uma decisão estratégica e financeiramente sólida.

- **Gestão de Mudanças e Treinamentos**
  - **Objetivo:** avaliar a capacidade da organização em lidar com as mudanças que a adoção da arquitetura de Micro-Frontends pode trazer. Busca-se identificar os impactos nos processos, na cultura organizacional e na estrutura de equipe, além de preparar a equipe para a transição.
  - **Recomendações:**
    * *Comunicação Efetiva:* Estabelecer um plano de comunicação claro e aberto para informar os membros da equipe sobre as mudanças planejadas. Comunique os benefícios da nova arquitetura, destacando como ela contribuirá para os objetivos estratégicos da empresa.
    * *Treinamento Personalizado:* Desenvolver programas de treinamento específicos para as equipes que lidarão diretamente com os Micro-Frontends. Certifique-se de que os membros da equipe tenham as habilidades necessárias para trabalhar eficientemente com a nova arquitetura.
    * *Definir Marcos e Métricas:* Estabelecer marcos claros e métricas mensuráveis para avaliar o progresso durante a transição. Definir indicadores de sucesso ajudará a monitorar a eficácia das estratégias de gerenciamento de mudanças e identificar áreas que precisam de ajustes.

Exemplo Prático: Uma empresa que deseja adotar a arquitetura de Micro-Frontends, precisa avaliar a capacidade organizacional de lidar com as mudanças decorrentes dessa transição. Para garantir uma transição suave, a empresa adota práticas como comunicação efetiva, realizando reuniões regulares para destacar os benefícios da nova arquitetura, além de oferecer treinamento personalizado para as equipes envolvidas. Paralelamente, são estabelecidos marcos claros e métricas mensuráveis para monitorar o progresso, proporcionando uma avaliação contínua e a capacidade de ajustar estratégias de gerenciamento de mudanças conforme necessário. Essas medidas visam preparar a equipe e a estrutura organizacional para as transformações advindas da adoção de Micro-Frontends.

4.2.2 Implementação Prática

A fase de Implementação Prática do GAM marca a transição do planejamento teórico para a aplicação concreta da arquitetura de Micro-Frontends. Dividida em decisões Operacionais e de Integração, esta etapa visa guiar os arquitetos e desenvolvedores na execução efetiva do paradigma arquitetônico. As estratégias delineadas nas fases anteriores serão agora aplicadas no Projeto Piloto, representando um ambiente controlado para validar e consolidar as decisões tomadas previamente. A implementação prática é o ponto em que as escolhas teóricas começam a se materializar, e cada decisão tem impacto direto na eficiência, escalabilidade e manutenibilidade do sistema. A Figura 4.3 ilustra o fluxo das decisões operacionais e de integração durante esta fase.

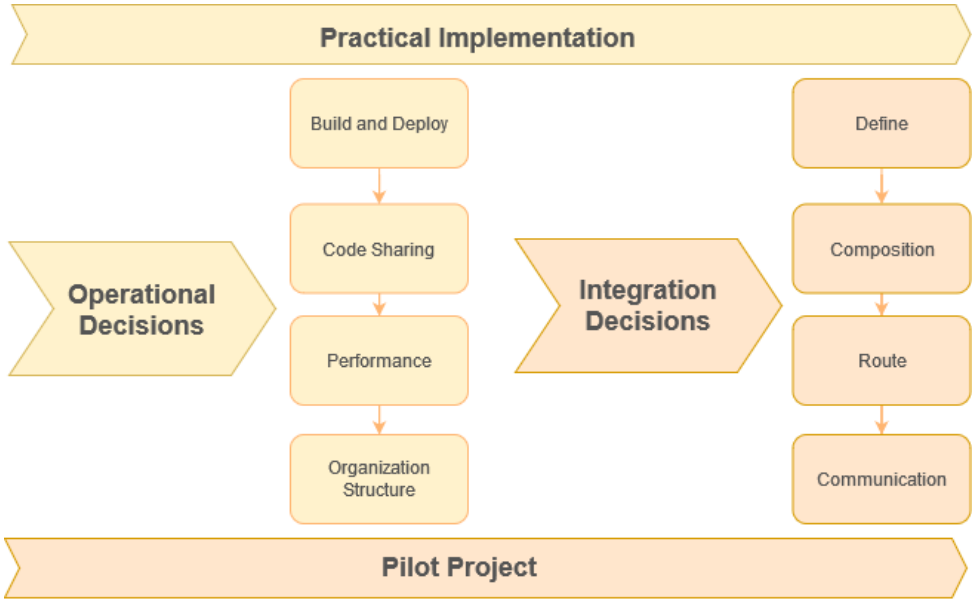


Figura 4.3: GAM - Fase de Implementação Prática, Fonte: Autor

Decisões Operacionais

As decisões operacionais focam nas escolhas estratégicas das áreas de compilação e implantação, compartilhamento de código, otimização de desempenho e a estrutura organizacional. No contexto dos micro-frontends, a tomada de decisões operacionais práticas e bem-informadas é fundamental para assegurar a agilidade, a colaboração eficiente entre equipes e a adaptação contínua às demandas dinâmicas do desenvolvimento de software distribuído.

- **Compilação e Implantação:**

- **Objetivo:** Implementar um processo eficiente de compilação e implantação para permitir atualizações rápidas e independentes de cada micro-frontend. [1].
- **Recomendações:**
  * **Automação**
    - *Integração e Entrega contínua (CI/CD):* Utilizar ferramentas de integração contínua e entrega contínua (CI/CD) para automatizar o processo de compilação, testes e implantação. Garanta que cada alteração no código seja submetida a um pipeline automatizado para validar e implantar automaticamente.
    - *Pipeline:* Configurar pipelines de entrega contínua para garantir uma integração suave e implantações rápidas. Automatize a geração de artefatos, como contêineres ou pacotes, prontos para a implantação em ambientes de produção.
    - *Orquestração:* Explorar ferramentas como Kubernetes para orquestração de contêineres e escalabilidade.
  * **Observabilidade**
    - *Logs e rastreamento:* Integrar sistemas de registro (logs) e rastreamento para acompanhar eventos durante o processo de compilação e implantação. Isso facilita a identificação rápida de problemas e a análise de desempenho.
    - *Métricas de Desempenho:* Implementar métricas de desempenho durante a compilação e a implantação para monitorar a eficiência do processo. Utilize ferramentas que ofereçam visibilidade em tempo real e alertas para eventos críticos.

Exemplo Prático: Ao implementar a automação, a equipe decide integrar uma ferramenta CI/CD, como Jenkins [a], ao repositório de código-fonte. Cada alteração no código aciona automaticamente um pipeline de entrega contínua, que inclui testes automatizados e a geração de contêineres Docker [b] prontos para implantação. Além disso, optam por utilizar Kubernetes para orquestrar esses contêineres, garantindo uma escalabilidade eficiente. Em termos de observabilidade, integram ferramentas de logs e rastreamento, como ELK Stack [c], para monitorar eventos durante o processo de compilação e implantação. Métricas de desempenho são configuradas usando Prometheus, oferecendo visibilidade em tempo real e alertas para eventos críticos. Essas decisões operacionais promovem uma compilação e implantação ágeis e eficientes dos micro-frontends.

[a] https://www.jenkins.io/
[b] https://www.docker.com/
[c] https://www.elastic.co

- **Compartilhamento de Código:**
  - **Objetivo:** Facilitar e otimizar o compartilhamento de código entre os diferentes micro-frontends, promovendo uma abordagem eficiente e colaborativa no desenvolvimento distribuído.
  - **Recomendações:**
    * *Identificação de Funcionalidades Comuns:* Identificar funcionalidades comuns que possam ser usadas por bibliotecas;
    * *Web Components e Design System:* Criar Web Components para encapsular funcionalidades específicas, garantindo modularidade e reusabilidade e integrar ao Design System da organização, garantindo consistência visual e de interação em todos os micro-frontends.
    * *Controle de Versão:* Implementar um sistema de controle de versão como git utilizando ferramentas de gestão de pacotes npm ou yarn.

Exemplo Prático: A equipe inicia o processo identificando funcionalidades comuns que podem ser encapsuladas em bibliotecas. Desenvolvem Web Components reutilizáveis para modularidade e reusabilidade, integrando-os ao Design System da organização para garantir consistência visual e de interação. Optam por utilizar Git [a] como sistema de controle de versão, combinado com npm [b] ou yarn [c] para gerenciamento eficiente de pacotes. Essas decisões visam promover o compartilhamento de código de forma eficiente entre os micro-frontends, facilitando o desenvolvimento colaborativo e distribuído.

[a] https://git-scm.com/
[b] https://www.npmjs.com/
[c] https://yarnpkg.com/

- **Performance:**

- **Objetivo:** Garantir um desempenho eficiente na arquitetura de Micro-Frontends, otimizando a carga, renderização e interação das interfaces para proporcionar uma experiência do usuário ágil e responsiva.
- **Recomendações:**
  * *Carregamento Assíncrono de Módulos:* Adotar estratégias de carregamento assíncrono, como Lazy Loading, para carregar módulos somente quando necessário, reduzindo o tempo de carregamento inicial.
  * *Otimização de Renderização:* Utilizar técnicas como Virtual DOM ou Shadow DOM para otimizar a renderização, minimizando as atualizações desnecessárias na interface do usuário.
  * *Gestão de Estado:* Implementar uma gestão de estado escolhendo uma solução que minimize a reatividade excessiva e mantenha um estado coerente entre os micro-frontends.
  * *Cache de Recursos:* Implementar estratégias de cache para recursos estáticos e dinâmicos, reduzindo a necessidade de buscar repetidamente os mesmos dados do servidor.

Exemplo Prático: A equipe decide adotar o Lazy Loading para carregar módulos de forma assíncrona, especialmente aqueles que não são essenciais na carga inicial. Implementam o Virtual DOM para otimizar a renderização, garantindo atualizações eficientes. Escolhem o Redux como solução de gestão de estado devido à sua eficiência com micro-frontends. Adicionalmente, estabelecem estratégias de cache usando armazenamento local para recursos estáticos e cache de servidor para dados dinâmicos, promovendo um desempenho eficiente na arquitetura de Micro-Frontends.

- **Estrutura Organizacional:**
  - **Objetivo:** Estabelecer uma estrutura organizacional eficiente para a implementação de Micro-Frontends, promovendo uma colaboração fluida entre equipes e garantindo a coesão na entrega de funcionalidades.
  - **Recomendações:**
    * *Modelo de Squad:* Implementar o modelo de Squads, onde equipes multifuncionais e autônomas são responsáveis por recursos específicos. Isso promove a especialização e agilidade, favorecendo a entrega independente de micro-frontends.
    * *Metodologia Ágil:* Adotar metodologias ágeis, como Scrum ou Kanban, para gerenciar o desenvolvimento de micro-frontends. Sprints regulares,

reuniões de retrospectiva e práticas ágeis fortalecem a colaboração e a adaptação contínua.

* *Ferramentas de Colaboração:* Utilizar ferramentas de colaboração, como Slack, Microsoft Teams ou outras plataformas de comunicação. Essas ferramentas facilitam a comunicação entre equipes distribuídas, promovendo a troca de informações de forma eficaz.
* *Cultura de DevOps:* Cultivar uma cultura de DevOps, integrando desenvolvimento e operações. Isso inclui automação de processos, entrega contínua e monitoramento, garantindo eficiência na implementação e manutenção.

Exemplo Prático: A equipe decide adotar o modelo de Squads, formando equipes dedicadas a diferentes micro-frontends. Escolhem a metodologia Scrum para gerenciar o desenvolvimento, realizando sprints regulares e reuniões de retrospectiva. Utilizam o Teams como plataforma de comunicação para facilitar a colaboração entre as equipes distribuídas. Essas escolhas estruturais contribuem para uma implementação prática e colaborativa de Micro-Frontends. Além disso, a integração das práticas de DevOps garante um fluxo contínuo de desenvolvimento e operações, promovendo automação, monitoramento constante e entrega contínua.

Decisões de Integração

As Decisões de Integração desempenham um papel fundamental na fase de Implementação Prática do GAM. Neste contexto, as escolhas relacionadas às Decisões-Chave incluem a definição das estratégias de Composição, Roteamento e Comunicação. Quanto à Orientação, inclui a decisão entre Divisão Vertical ou Divisão Horizontal. No que diz respeito à composição e roteamento inclui decidir sobre o client-side, edge-side ou server-side [1]. Cada decisão nesta categoria impacta diretamente a estrutura da aplicação, influenciando a independência de desenvolvimento e implantação, a lógica de organização dos módulos e a escolha de padrões arquiteturais específicos.

- **Definição de Divisão:**
  - **Objetivo:** Identificar e definir o micro-frontend do ponto de vista de sua divisão. Podemos decidir ter vários micro-frontends na mesma visualização (Divisão Horizontal) ou ter apenas um micro-frontend por visualização (Divisão Vertical) [1].
  - **Recomendações:**
    * **Divisão Horizontal**

· *Coordenação Efetiva:* Dada a presença de vários micro-frontends na mesma visualização, é essencial estabelecer canais eficazes de comunicação e coordenação entre as equipes para garantir a coesão na experiência do usuário.

· *Governança Rigorosa:* Devido à flexibilidade oferecida por esta abordagem, é crucial implementar uma governança rigorosa para evitar a proliferação descontrolada de micro-frontends. Estabeleça diretrizes claras para garantir consistência e qualidade.

∗ **Divisão Vertical**

· *Responsabilidade por Domínio:* Ao adotar a divisão vertical, cada equipe é responsável por um domínio de negócios específico. Recomenda-se aplicar princípios de *Domain-Driven Design* (DDD) [16] para garantir uma arquitetura alinhada aos requisitos de negócios.

· *Exploração do Domain-Driven Design (DDD):* Embora não seja comum em arquiteturas de frontend, a divisão vertical oferece uma boa razão para explorar os princípios do *Domain-Driven Design* (DDD). Considere aplicar conceitos como agregados e bounded contexts para melhorar a modelagem e a coesão.

Exemplo Prático: Na abordagem de Orientação Horizontal, temos como exemplo o cenário de comércio eletrônico, onde diferentes equipes são responsáveis por componentes distintos, como a barra de navegação, o carrinho de compras e os produtos exibidos. Cada equipe pode desenvolver e evoluir seu micro-frontend de forma independente, facilitando a coordenação entre elas. Por outro lado, na Orientação Vertical, cada equipe seria encarregada de um domínio específico, como autenticação, garantindo que todos os aspectos relacionados a esse domínio estejam encapsulados. Por exemplo, uma equipe pode lidar com todo o processo de autenticação, desde o login até a segurança, proporcionando uma visão holística do domínio. Ambas as abordagens têm suas vantagens, e a escolha dependerá dos requisitos específicos do projeto e da necessidade de flexibilidade versus foco em domínio.

- **Composição**
  - **Objetivo:** Escolher a estratégia de composição dos Micro-frontends, abrangendo Client-Side, Server-Side ou Edge-Side
  - **Client-Side:**
    - **Objetivo:** Facilitar a integração dinâmica de micro-frontends no shell de aplicativos do cliente, permitindo que diferentes partes da interface sejam carregadas de maneira eficiente.
    - **Recomendações:**

- *Utilização de Iframes:* Garantir que cada micro-frontend tenha um arquivo JavaScript ou HTML definido como ponto de entrada, facilitando a dinâmica adição ao Modelo de Objeto de Documento (DOM) pelo shell do aplicativo.
- *Single-SPA:* Possibilitar a composição e entrega de micro-frontends por meio da integração baseada em JavaScript usando frameworks do tipo Single Page Applications (Single-SPA) como Angular, React e Vue.
- *Web Components:* Entregar micro-frontends por meio da criação de elementos personalizados reutilizáveis com web components.
- *Transclusão no Lado do Cliente:* Avaliar a técnica de inclusão no lado do cliente (clientside include), utilizando bibliotecas como h-include [1], para carregar componentes de forma tardia e dinâmica.

Exemplo Prático: Em um aplicativo de notícias online, o objetivo é integrar dinamicamente diferentes módulos, como manchetes, artigos relacionados e comentários, proporcionando uma experiência de leitura fluida. Para alcançar isso, cada módulo será desenvolvido como um micro-frontend com um ponto de entrada definido, permitindo a incorporação dinâmica no shell do aplicativo usando elementos <iframe>. O uso do framework Single-SPA facilitará a integração de micro-frontends desenvolvidos em Angular, React e Vue. Elementos personalizados, como <headline-module> e <comment-module>, serão criados como Web Components, garantindo modularidade e reusabilidade.

– **Edge-Side:**

* **Objetivo:** Montar a visualização no nível do CDN, otimizando a entrega global da aplicação por meio de estratégias como Edge Side Includes (ESI).
* **Recomendações:**
  - *Uso de ESI:* Considerar o uso de Edge Side Includes (ESI) [2] para composição no lado da borda, aproveitando as capacidades de dimensionamento oferecidas por CDNs [3] e facilitando a montagem da visualização.
  - *Atenção à Variação entre Provedores:* Conscientizar-se de que o ESI pode ser implementado de maneira diferente por diferentes provedores de CDN, exigindo considerações especiais em uma estratégia multi-CDN ou migrações entre provedores.

[1] https://github.com/gustafnk/h-include
[2] https://www.w3.org/TR/esi-lang/
[3] https://aws.amazon.com/pt/what-is/cdn/

Exemplo Prático: Em um site de comércio eletrônico global, o objetivo é otimizar a entrega de conteúdo, montando a visualização no nível do CDN. Adota-se a estratégia Edge Side Includes (ESI) para compor dinamicamente elementos como carrinho de compras, promoções e recomendações com base na localização do usuário. O uso de ESI permite uma montagem eficiente na borda da rede, aproveitando as capacidades de dimensionamento oferecidas por CDNs. A implementação cuidadosa considerará as variações entre provedores de CDN, garantindo uma estratégia eficaz em ambientes multi-CDN ou em migrações entre provedores.

- **Server-Side:**
  - **Objetivo:** Permitir que o servidor de origem componha a visualização, recuperando diferentes micro-frontends e montando a página final.
  - **Recomendações:**
    - *Análise de Casos de Uso:* Antes de optar pela composição no lado do servidor, deve-se realizar uma análise aprofundada dos casos de uso da aplicação para compreender a natureza e os requisitos das páginas que serão compostas.
    - *Escalabilidade:* Desenvolver uma estratégia clara de escalabilidade para os servidores com o objetivo de lidar com um grande volume de solicitações, especialmente quando há personalização da página para cada usuário, a fim de evitar períodos de inatividade para esses usuários.
    - *Cache:* Avaliar o grau de cacheabilidade da página para determinar a eficácia da composição no lado do servidor. Se a página for altamente cacheável, pode-se aproveitar as políticas de tempo de vida longo oferecidas pela CDN, otimizando o desempenho e a eficiência na entrega de conteúdo.

Exemplo Prático: Em um sistema de gerenciamento de conteúdo personalizado, onde diferentes clientes exigem visualizações específicas, o objetivo é permitir que o servidor de origem componha a visualização, recuperando micro-frontends personalizados e montando a página final. Realiza-se uma análise detalhada dos casos de uso para entender a natureza e os requisitos das páginas a serem compostas. Adota-se uma estratégia de escalabilidade considerando a personalização da página para cada usuário. Avalia-se também o grau de cacheabilidade da página o que permite a eficiência na composição do lado do servidor otimizando o desempenho e a eficiência na entrega de conteúdo.

- **Roteamento:**

- **Objetivo:** Garantir uma experiência de usuário coesa e ágil, alinhada à estratégia de composição escolhida (origin, edge-side ou client-side). Isso envolve direcionar corretamente as solicitações de páginas para os micro-frontends apropriados, considerando as características específicas do projeto.
- **Recomendações:**
  - *Roteamento no Lado do Servidor(Server-Side):* Considere o uso de servidores de aplicativos, como Nginx, para lidar com o roteamento eficiente e a distribuição de carga.
  - *Roteamento na Borda (Edge-Side):* Adotar CDNs como Cloudflare [4] ou Akamai [5], que ofereçam suporte a Edge Side Includes (ESI) para compor micro-frontends na borda.
  - *Roteamento no Lado do Cliente (Client-Side):* Utilize bibliotecas de roteamento client-side, como React Router ou Vue Router, para uma navegação eficiente entre micro-frontends. Se optar por um shell de aplicativo SPA, frameworks como Angular ou React podem gerenciar o roteamento interno entre micro-frontends.

Exemplo Prático: Em uma aplicação de comércio eletrônico com micro-frontends que abrangem diferentes funcionalidades, como catálogo de produtos, carrinho de compras e área de pagamento, se a estratégia escolhida for o roteamento no lado do cliente, o objetivo seria carregar dinamicamente esses micro-frontends com base nas ações do usuário. Utilizando React como framework no lado do cliente, React Router poderia ser implementado para gerenciar as transições entre os micro-frontends. O webpack [a] pode ser usado para empacotar e otimizar os micro-frontends antes do carregamento no navegador, garantindo uma experiência de usuário eficiente.

[a] https://webpack.js.org/

- **Comunicação:**
  - **Objetivo:** Garantir uma interação consistente entre os diferentes componentes, mesmo quando estão distribuídos em várias equipes ou partes da aplicação.
  - **Recomendações:**
    - *Event Emitter:* Implementar um Event Emitter ou a funcionalidade de eventos nativos do JavaScript para permitir a comunicação assíncrona entre os micro-frontends.

[4] https://www.cloudflare.com
[5] https://www.akamai.com

* *Custom Events:* Adotar eventos personalizados como CustomEvent [6] para notificar outros micro-frontends sobre eventos específicos.
* Injeção de EventBus: Considerar a injeção de um EventBus, no contêiner de micro-frontends para facilitar a comunicação global.
* *Armazenamento Local (Web Storage):* Utilize o armazenamento local (localStorage) para compartilhar dados persistentes entre os micro-frontends.
* *Query Strings:* Explorar a passagem de dados via query strings para casos específicos, como identificação de produtos ou informações de página. Evitar transmitir dados sensíveis como nomes de usuário e senha por meio de query strings e reserve para informações não confidenciais.

Exemplo Prático: Um micro-frontend responsável pelo processo de autenticação precisa notificar outro micro-frontend sobre a conclusão bem-sucedida da autenticação, fornecendo um token de acesso. Nesse caso, o uso de eventos personalizados seria apropriado. O micro-frontend de autenticação despacha um evento personalizado, como "userAuthenticated", com detalhes contendo o token. Outros micro-frontends interessados podem adotar ouvir esse evento e agir de acordo, garantindo uma comunicação eficiente e desacoplada entre os componentes. O EventBus [a] injetado no contêiner de micro-frontends facilitaria essa comunicação global.

[a] https://www.npmjs.com/package/js-event-bus

4.2.3 Expansão

A Fase de Expansão do GAM representa o período em que a arquitetura de Micro-Frontends evolui e se estende para além do seu ponto inicial de implementação. Esta etapa é marcada pela aplicação contínua dos princípios e diretrizes estabelecidos nas fases anteriores, com foco na ampliação controlada da adoção da arquitetura em diferentes áreas do sistema.

Durante a Expansão, as decisões tomadas na Avaliação Inicial e Implementação Prática continuam a moldar a jornada da equipe de desenvolvimento. O objetivo é escalar a aplicação da arquitetura de Micro-Frontends, garantindo uma transição suave e minimizando os riscos associados.

Esta fase abrange desde a seleção de projetos pilotos até a ampliação para outros componentes do sistema. Aspectos práticos, aprendizado contínuo, treinamento da equipe e monitoramento constante são elementos-chave para garantir o sucesso da expansão. A equipe é incentivada a coletar feedback, realizar ajustes na estratégia conforme necessário

[6] https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent/CustomEvent

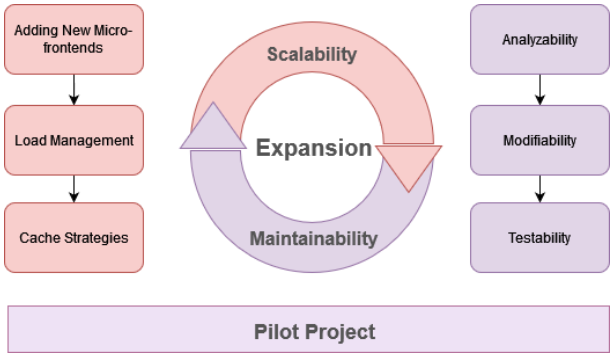


Figura 4.4: GAM - Fase de Expansão, Fonte: Autor

e aprimorar continuamente a implementação com base nas lições aprendidas ao longo do processo. A Figura 4.4 ilustra as estratégias sugeridas para a implementação desta fase considerando a escalabilidade e a manutenibilidade do sistema em que se pretende adotar a arquitetura de micro-frontend.

Escalabilidade

Esta etapa visa capacitar a aplicação a crescer de maneira eficiente e adaptar-se às demandas variáveis do ambiente. As principais áreas de foco são a adição de novos micro-frontends, o gerenciamento de carga e as estratégias de cache.

- **Adição de Novos Micro-Frontends:**
  - **Objetivo:** Facilitar a incorporação de novos micro-frontends ao sistema existente.
  - **Recomendações:**
    - *Modularização Dinâmica:* Desenvolver arquitetura modular que suporte a adição dinâmica de novos micro-frontends sem impactar os existentes.
    - *Balanceamento de Carga:* Implementar sistema de balanceamento de carga automático como serviços de orquestração, usando ferramentas como Kubernetes, para facilitar o dimensionamento automático com base nas métricas de tráfego.
    - *Gestão de Cache:* Adotar estratégias de cache, como Redis, para armazenar temporariamente as respostas de micro-serviços, melhorando o throughput da composição de micro-frontends. Explorar o armazenamento de DOM de micro-frontends em caches in-memory para reduzir a necessidade de composição a cada requisição.

Exemplo Prático: Em um cenário de comércio eletrônico baseado em micro-frontends, um novo micro-frontend chamado "Recomendações Personalizadas"foi adicionado. Implementa-se um sistema de balanceamento de carga automático com Kubernetes para escalabilidade dinâmica com base no tráfego. Para aprimorar o desempenho, utiliza-se estratégias de cache, incluindo Redis, para armazenar temporariamente respostas de micro-serviços e o DOM de micro-frontends em caches in-memory, reduzindo a necessidade de recomposição a cada requisição.

- **Gerenciamento de Carga:**
  - **Objetivo:** Garantir a eficiência operacional e o desempenho otimizado da aplicação, mesmo diante de demandas crescentes.
  - **Recomendações:**
    * *Autoescalabilidade na Nuvem:* Utilizar funcionalidades de autoescalabilidade oferecidas por provedores de nuvem para ajustar dinamicamente a infraestrutura com base nos padrões de tráfego. Escolha camadas de computação eficientes, como contêineres, para rápida execução, e considere opções gerenciadas, como serviços serverless, para simplificar a operacionalização da infraestrutura.
    * *Previsão de Carga e Ajuste Manual:* Estabelecer uma infraestrutura de linha de base capaz de lidar com cargas previsíveis, como vendas da *Black Friday*, adotando práticas de comparação entre diferentes serviços e opções plug-and-play.
    * *Otimização de Latência com CDN:* Utilizar uma CDN para aumentar a velocidade de entrega das páginas da web, reduzindo a latência entre o cliente e o conteúdo solicitado.

Exemplo Prático: Para otimizar o desempenho da plataforma de comércio eletrônico, foi aproveitado a autoescalabilidade na nuvem, utilizando serviços serverless e contêineres para ajustar dinamicamente a infraestrutura com base em padrões de tráfego, garantindo rápida execução e eficiência operacional. Para lidar com picos previsíveis, como as vendas da "Black Friday", estabeleceu-se uma infraestrutura de linha de base com ajustes manuais quando necessário, utilizando práticas de comparação entre diferentes serviços e opções plug-and-play. Além disso, incorporou-se uma CDN para otimizar a latência, acelerando a entrega de páginas web e aprimorando a experiência do usuário em cenários de alta demanda.

- **Estratégias de Cache:**
  - **Objetivo:** Otimizar o desempenho da aplicação reduzindo a carga nos servidores e acelerando o tempo de resposta.

- **Recomendações:**
  - *Cache de Conteúdo Estático:* Implementar cache para conteúdo estático, como imagens, folhas de estilo e scripts, reduzindo a latência de carregamento.
  - *Cache de Dados Dinâmicos:* Utilizar estratégias de cache para dados dinâmicos, minimizando consultas frequentes ao servidor.
  - *Invalidação de Cache Eficiente:* Implementar métodos eficazes de invalidação de cache para garantir que os usuários recebam informações atualizadas.
  - *Cache de Respostas:* Implementar caches para armazenar temporariamente as respostas de micro-frontends, utilizando soluções como Redis, armazenando temporariamente as respostas de micro-serviços para aumentar o throughput.
  - *Armazenamento de DOM em Cache:* Armazenar o DOM completo de micro-frontends em caches in-memory para evitar composição a cada requisição.

Exemplo Prático: Em uma aplicação web introduziu-se caches para conteúdo estático (imagens e estilos), dados dinâmicos e respostas de micro-frontends e micro-serviços, utilizando soluções como Redis. Essas abordagens reduzem a latência de carregamento, minimizam consultas frequentes aos servidores e aumentam o throughput. Além disso, adotou-se métodos invalidação de cache para garantir informações atualizadas e armazenou-se o DOM de micro-frontends em caches in-memory, evitando composição excessiva a cada requisição.

Manutenibilidade

A ISO/IEC 25010 [62] define a manutenibilidade como a capacidade de um sistema ser modificado, atualizado e adaptado. Suas subcaracterísticas principais incluem Modularidade, Reusabilidade, Analisabilidade, Modificabilidade e Testabilidade. No contexto do GAM, a manutenibilidade dá ênfase ao código, adaptamos os conceitos para focar em Analisabilidade, Modificabilidade e Testabilidade, destacando a importância da análise eficiente, modificação eficaz e testagem robusta do código-fonte em ambientes de micro-frontends.

- **Analisabilidade:**
  - **Objetivo:** Compreender a estrutura e funcionamento dos micro-frontends para diagnosticar problemas de forma rápida e precisa.
  - **Recomendações:**

* *Padrões de Nomenclatura:* Adotar convenções de nomenclatura consistentes para facilitar a identificação e compreensão de componentes.
* *Documentação*: Manter documentação detalhada sobre a arquitetura, interações e responsabilidades de cada micro-frontend.
* *Ferramentas de Análise Estática:* Utilizar ferramentas como ESLint e TSLint para análise estática de código, identificando padrões, complexidade e possíveis problemas. Ferramentas mais abrangentes como Sonar podem oferecer uma visão holística da qualidade do código, incluindo métricas de código-fonte, cobertura de código e detecção de possíveis problemas de segurança.

Exemplo Prático: Um desenvolvedor é designado para atualizar a interface da página de detalhes do produto. Durante o processo, o desenvolvedor segue os padrões de nomenclatura, consulta a documentação para entender a estrutura e as interações, enquanto as ferramentas de análise estática (ESLint [a] e TSLint [b]) sinalizam potenciais melhorias no código. O Sonar [c] complementa essa análise, fornecendo métricas detalhadas e identificando áreas de atenção, resultando em modificações eficientes e seguras no micro-frontend.

[a] https://eslint.org/
[b] https://palantir.github.io/tslint/
[c] https://www.sonarsource.com/

- **Modificabilidade:**
  - **Objetivo:** Permitir modificações eficientes nos micro-frontends sem introduzir defeitos ou degradar a qualidade.
  - **Recomendações:**
    * *Design Modular:* Estruturar os micro-frontends de forma modular, minimizando acoplamento entre eles.
    * *Refatoração Contínua:* Incentivar práticas de refatoração para manter o código limpo e adaptável.
    * *Controle de Versão:* Utilizar sistemas de controle de versão eficientes, como Git, para rastrear alterações e facilitar reversões se necessário. Ferramentas como o Veracode podem ser incorporadas para realizar análises de segurança estática e dinâmica, ajudando a garantir modificações seguras.

Exemplo Prático: Uma nova funcionalidade precisa ser adicionada a um dos micro-frontends. Com o design modular, o desenvolvedor pode isolar a implementação da nova funcionalidade sem afetar outras partes do sistema. Durante o processo, práticas contínuas de refatoração garantem que o código permaneça limpo e adaptável. O uso eficiente do controle de versão permite que o desenvolvedor rastreie e gerencie as alterações de maneira organizada. A integração do Veracode[a] verifica possíveis vulnerabilidades de segurança, garantindo modificações seguras antes da implementação.

[a]https://www.veracode.com/

- **Testabilidade:**
  - **Objetivo:** Garantir a efetividade dos testes, permitindo a detecção precoce de falhas e mudanças seguras.
  - **Recomendações:**
    - *Testes Unitários e de Integração:* Implementar testes unitários (usando Jest e Enzyme para React, por exemplo) e de integração para verificar a funcionalidade isolada e a interação entre os micro-frontends.
    - *Automação de Testes:* Utilize ferramentas de automação de testes, como Cypress, para agilizar o processo de verificação.
    - *Ambientes de Teste Similares à Produção:* Manter ambientes de teste que reproduzam fielmente as condições de produção para validar o comportamento real do sistema.

Exemplo Prático: Uma nova funcionalidade é adicionada a um micro-frontend específico. Os testes unitários garantem que a funcionalidade isolada funcione conforme o esperado, utilizando ferramentas como Jest[a] para validar componentes React. Testes de integração, incorporando ferramentas como Cypress[b], verificam a interação entre os diferentes micro-frontends. A automação desses testes agiliza o processo de verificação, permitindo que a equipe de desenvolvimento detecte precocemente quaisquer falhas potenciais. Ao manter ambientes de teste semelhantes à produção, a equipe assegura que as condições reais do sistema sejam simuladas, validando de forma abrangente o comportamento do micro-frontend antes da implantação.

[a]https://jestjs.io
[b]https://www.cypress.io/

4.2.4 Projeto Piloto

A Seção Projeto Piloto do GAM representa um marco no processo de implementação prática da arquitetura de Micro-Frontends. Nesta fase, utiliza-se um ambiente controlado, onde o estudo das viabilidades técnicas, as decisões operacionais e de integração

são previamente delineadas e aplicadas de maneira tangível. O Projeto Piloto serve como um teste de validação, permitindo que as equipes de desenvolvimento, arquitetos de software e líderes de projeto observem como as estratégias teóricas se traduzem na realidade operacional. A Figura 4.5 fornece uma visão geral do papel central do Projeto Piloto no contexto do GAM.

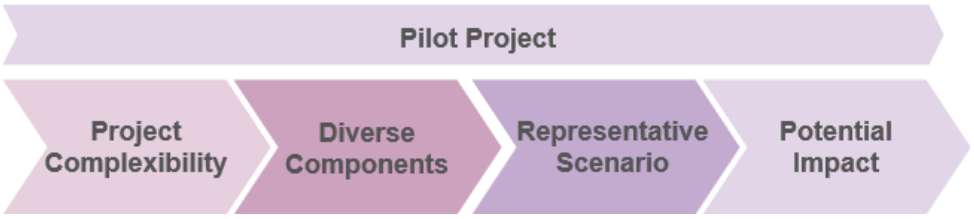


Figura 4.5: GAM - Fase do Projeto Piloto, Fonte: Autor

Durante esta fase, destacaremos não apenas a implementação técnica, mas também as lições aprendidas e os ajustes necessários à medida que a equipe ganha *insights* valiosos na aplicação prática do paradigma de Micro-Frontends. O Projeto Piloto oferece uma oportunidade única para avaliar a eficácia das escolhas arquiteturais e de integração em um contexto controlado, antes de uma expansão mais ampla para outros projetos ou áreas do sistema. A escolha do Projeto Piloto é um ponto fundamental para o sucesso da adoção de Micro-Frontends. Nesta etapa, os stakeholders devem considerar os seguintes aspectos:

- **Seleção do Projeto Piloto:**
  - **Objetivo:** A escolha do Projeto Piloto é um ponto fundamental para o sucesso da adoção de Micro-Frontends. Nesta etapa, os stakeholders devem considerar os seguintes aspectos:
  - **Considerações:**
    - *Complexidade do Projeto:* Avaliar a complexidade geral do projeto em termos de funcionalidades, interações e número de componentes. Projetos mais complexos e extensos podem oferecer uma melhor compreensão dos desafios e benefícios do uso de Micro-Frontends. É fundamental ressaltar, porém, que é prudente limitar os testes a alguns componentes ou módulos do projeto, especialmente se este for de grande importância para a empresa, a fim de evitar comprometimentos indesejados.
    - *Diversidade de Componentes:* Buscar projetos que possuam uma variedade representativa de componentes. Isso permite explorar como diferentes tipos de funcionalidades podem ser encapsulados em Micro-Frontends e integrados de maneira eficiente.

* *Representatividade do Cenário:* Certifique-se de que o projeto selecionado seja representativo do ambiente mais amplo da organização. Um Piloto que reflita os desafios e requisitos típicos do desenvolvimento frontend da empresa proporcionará insights mais aplicáveis.
* *Impacto Potencial:* Considerar o impacto potencial do Projeto Piloto na organização. Escolher um projeto com impacto significativo, mas controlado, permite avaliar os resultados sem comprometer a estabilidade geral do sistema.

Exemplo Prático: Durante a fase de implementação do projeto piloto, a equipe opta por selecionar a página de carrinho de compras como o cenário inicial, considerando a complexidade do projeto, pois a página envolve diversas funcionalidades, desde a exibição dos itens no carrinho até a gestão de cupons e opções de pagamento. A diversidade de componentes é assegurada, abrangendo desde elementos simples, como ícones, até componentes mais complexos, como a lógica de aplicação de descontos. A representatividade do cenário é garantida, pois a página de carrinho de compras reflete os desafios típicos do frontend, incluindo interações com o banco de dados de inventário, cálculos de preços e integrações com gateways de pagamento. Quanto ao impacto potencial, o projeto possui relevância significativa, uma vez que melhorias na experiência do usuário nessa etapa do processo de compra podem ter um impacto notável, mas controlado, permitindo uma avaliação aprofundada antes de uma expansão mais ampla da arquitetura de Micro-Frontends para outras áreas do sistema.

4.3 Verificação e Validação

A fase de verificação e validação desempenha um papel essencial na garantia da eficácia e utilidade prática do GAM. Verificação refere-se ao processo de avaliar se o GAM atende aos requisitos e especificações propostas, enquanto validação visa determinar se o GAM realmente cumpre seu propósito e é útil na prática. Por meio de um survey direcionado a profissionais experientes da área de Tecnologia da Informação, nós buscamos confirmar a conformidade do GAM com as expectativas estabelecidas e identificar possíveis áreas de melhoria. Essa abordagem colaborativa visa assegurar a relevância do GAM ao incorporar insights e perspectivas dos profissionais, além de promover sua aceitação e aplicabilidade no contexto da adoção de novas tecnologias.

Foram utilizados 2 surveys para realizar a verificação e validação do GAM. Inicialmente, aplicamos o survey “A” ao público alvo entre os dias 23 a 26 de janeiro de 2024, com o objetivo de capturar as percepções dos profissionais que atuam nos cargos de gestores, arquitetos e desenvolvedores. As percepções foram analisadas em relação à experiencia e conhecimento dos participantes em relação à adoção e implementação de arquiteturas

emergentes, incluindo o micro-frontends. Contudo, é relevante salientar que as respostas coletadas por meio deste survey não têm a intenção de aprofundar nos detalhes específicos de cada projeto de software em que o participante atuou. Todas as questões do survey "A" são apresentadas na Tabela 4.1.

ID	Question
Q1	What is your educational level?
Q2	What is the nature of your work and/or organization?
Q3	How long have you been working in the technology/software engineering field?
Q4	What is your current position in your organization?
Q5	How would you describe your level of involvement in frontend development in your organization?
Q6	How would you classify your level of involvement in frontend architectural decisions in your organization?
Q7	Regarding the number of people working in your team, can you say that there are:
Q8	What is your level of knowledge about monolithic architecture?
Q9	What is your level of knowledge about micro-frontend architecture?
Q10	Please indicate the programming languages you have experience with.
Q11	Please indicate the frameworks you have experience with.
Q12	What is your perception of solutions in the micro-frontends architecture?
Q13	Regarding the size of the project where you work, can you say that:
Q14	Have you had the opportunity to participate in feasibility studies for the adoption of any technology?
Q15	How do you rate your experience with Web Components?
Q16	How do you rate your experience with using JavaScript Frameworks for Micro-frontends?
Q17	How do you rate your experience with integrating APIs and Microservices in the context of micro-frontends?
Q18	Was there any consideration about something that was not addressed in this survey?

Tabela 4.1: Survey Questions A

Survey A

Realizamos um primeiro survey (Survey A) com um grupo de profissionais da área de TI para coletar informações gerais sobre seus conhecimentos em arquitetura de micro-frontends, bem como dados demográficos. O objetivo foi compreender melhor o perfil dos participantes e obter insights que pudessem validar o desenvolvimento inicial do Guia.

Análise dos Dados

A Tabela 4.2 apresenta as características demográficas dos participantes do survey A. Primeiramente, em relação ao nível educacional dos participantes (Q1), 85% completaram ou estão atualmente cursando um mestrado, enquanto os 15% restantes são graduados. Quanto à natureza de suas organizações (Q2), aproximadamente 45% trabalham em empresas privadas, 40% em empresas públicas e 15% como trabalhadores autônomos.

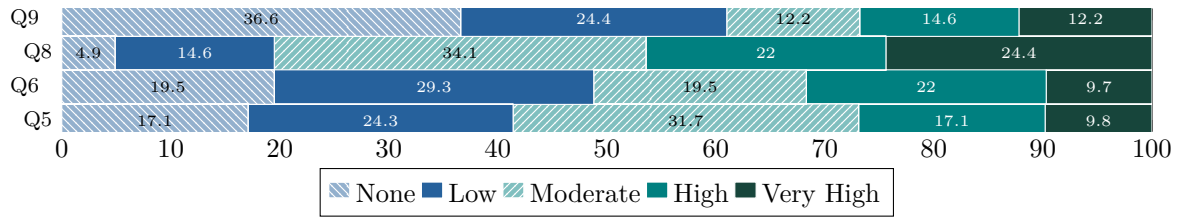


Figura 4.6: Level of engagement in frontend development and knowledge level in monolithic and micro-frontend architecture

Quanto à experiência prática (Q3), observa-se que aproximadamente 45% dos participantes têm entre 7 e 9 anos de experiência, 40% têm mais de 10 anos e os 15% restantes estão distribuídos em outras faixas de experiência. Em relação às ocupações (Q4), destacam-se Líderes Técnicos (25%), Desenvolvedores de Software (20%) e Gerentes de Projetos (15%), enquanto os 40% restantes desempenham vários papéis.

17,1% dos participantes afirmaram que não têm envolvimento em desenvolvimento frontend, enquanto 24,3% alegaram ter baixo envolvimento nessa área. 31,7% dos participantes relataram ter um nível moderado de envolvimento em desenvolvimento frontend. Os participantes que indicaram níveis alto e muito alto de envolvimento representam 17,1% e 9,8% dos participantes, respectivamente, como mostrado em Q5 da Figura 4.6. Este resultado permite inferir que os participantes da pesquisa são especialistas e/ou líderes estratégicos em projetos frontend.

Quanto ao nível de envolvimento dos participantes da pesquisa em decisões arquitetônicas de frontend, 19,5% afirmaram que não têm envolvimento. Além disso, 29,3% dos praticantes classificaram seu nível de envolvimento como baixo. Enquanto isso, 19,5% relataram ter um nível moderado de envolvimento. Por outro lado, 22% relataram um nível alto de envolvimento e 9,7% afirmaram ter um nível muito alto de envolvimento em decisões arquitetônicas, conforme apresentado em Q6 da Figura **??**. Este resultado pode indicar que uma parte considerável dos participantes tem um papel mais limitado ou menos envolvimento nas decisões, enquanto os praticantes que relataram envolvimento alto ou muito alto podem desempenhar papéis mais estratégicos e influentes na arquitetura de frontend de suas organizações.

No que diz respeito ao ambiente de trabalho (Q7), cerca de 35% dos participantes colaboram em equipes com mais de 10 membros, 50% trabalham em equipes de 5 a 10 membros, e os 15% restantes integram equipes menores.

Foi investigado o nível de conhecimento dos praticantes em relação à arquitetura monolítica. Esta questão é importante para entender a transição da arquitetura monolítica para a arquitetura de micro-frontends. Apenas 4,9% dos praticantes afirmaram não ter conhecimento sobre arquitetura monolítica, 14,6% relataram ter um nível baixo de conhe-

cimento e 34,1% classificaram seu nível de conhecimento como moderado. Além disso, 22% dos praticantes relataram ter um alto nível de conhecimento, enquanto 24,4% declararam ter um nível muito alto de conhecimento, conforme apresentado em Q8 da Figura 4.6.

Na questão relacionada ao conhecimento da arquitetura de micro-frontends servir como um referencial para compreender a interação do participantes com as escolhas das questões relacionadas à experiência, atuação e preferências tecnológicas, 36.6% dos participantes afirmaram não possuir conhecimento sobre a arquitetura de micro-frontends. 24.4% deles afirmaram que o conhecimento é classificado como baixo, enquanto 12.2% e 14.6% dos participantes afirmaram possuir conhecimento moderado e alto, respectivamente. Apenas 12.2% dos participantes afirmaram ter um nível muito alto de conhecimento sobre a arquitetura de micro-frontends, conforme apresentado na Q9 da Figura 4.6.

Quanto às linguagens de programação (Q10), JavaScript e Java são as mais conhecidas pelos participantes do survey, totalizando aproximadamente 50%, seguidas pelo Python, com cerca de 30%, e os 20% restantes possuem experiência com diversas outras linguagens, conforme apresentado na Tabela 4.2.

Em relação aos frameworks com os quais os profissionais possuem experiência (Q11), a Figura 4.7 revela que o React é o framework mais adotado (51,2%), seguido pelo Spring Boot (48,8%). Além disso, 46,3% dos participantes afirmaram ter experiência com o Angular, enquanto 12% mencionaram ter experiência com outros frameworks, como Zend, Slim, Next, Django, Laravel e JSF. Adicionalmente, 9,8% deles têm experiência com o Vue, e 7,3% afirmaram não ter experiência com nenhum dos frameworks identificados na literatura.

O levantamento também indicou que o uso combinado dos frameworks JavaScript e Java, Angular e Spring Boot, corresponde a 46,3% dos participantes; React e Spring Boot, a 48,8%; e Angular, Vue, React e Spring Boot são utilizados por 9,8% dos participantes. Esse resultado sugere que esses profissionais possuem habilidades associadas ao perfil de desenvolvedores full stack. Equipes compostas por profissionais com esse perfil têm uma maior probabilidade de eficácia na implementação de abordagens como o micro-frontend.

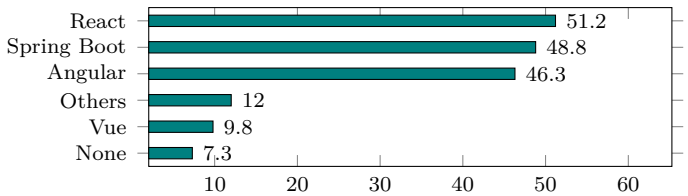


Figura 4.7: Frameworks that participants have experience with

A questão Q12 da Tabela 4.1 investiga a percepção dos participantes sobre as soluções da arquitetura de micro-frontends. 56,1% dos participantes afirmaram que essa abor-

dagem facilita a manutenção e a implementação de novas funcionalidades e 41,5% dos participantes afirmaram que o principal benefício da arquitetura é proporcionar uma melhor escalabilidade. 34,1% dos participantes afirmaram que a arquitetura micro-frontend oferece maior flexibilidade na escolha de tecnologias. Entretanto, 34,1% dos participantes afirmaram não saber opinar sobre o assunto, conforme apresentado na Figura 4.8. Esses resultados indicam uma percepção positiva em relação aos benefícios da arquitetura de micro-frontends, especialmente no que diz respeito à agilidade no desenvolvimento e à capacidade de adaptação a diferentes contextos tecnológicos.

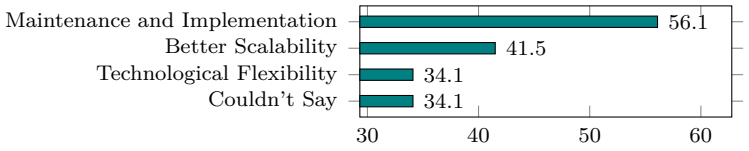


Figura 4.8: Participants perception of micro-frontend solutions

O tamanho de um projeto pode influenciar o número de membros da equipe que colaboram simultaneamente (Q13). Embora não haja um consenso preciso sobre o que caracteriza um projeto de software como grande ou médio, a metodologia Scrum oferece algumas métricas para essa definição [7]. Essa incerteza é um dos desafios que a arquitetura de micro-frontends enfrenta na gestão de equipes. Determinar o tamanho do projeto e sua possível divisão deve ser avaliado junto com os gerentes de projeto e a equipe técnica antes de adotar a arquitetura de micro-frontends. Entre os participantes, 39% afirmaram que estão envolvidos em projetos grandes, com mais de 5 equipes, enquanto 29,3% deles estão envolvidos em projetos médios, envolvendo até 5 equipes. Além disso, 26,8% dos participantes afirmaram que estão envolvidos em projetos pequenos, com apenas uma equipe. Por outro lado, 4,9% dos participantes não conseguiram expressar uma opinião sobre o tamanho do projeto em que estão envolvidos.

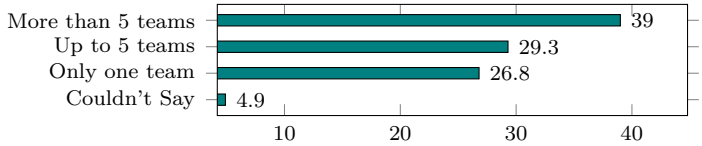


Figura 4.9: Sizes of projects in which the participant has worked

A participação em estudos de viabilidade para a adoção de novas tecnologias é uma prática comum nos processos de desenvolvimento de software (Q14). Dos participantes, 56,1% afirmaram que já tiveram a oportunidade de participar ativamente de estudos de viabilidade para adotar novas tecnologias na organização em que atuam. Por outro

[7]https://www.scrum.org

lado, 24,4% afirmaram que não participaram de nenhum estudo de viabilidade, mas têm interesse em participar de tais estudos para a adoção de novas tecnologias na organização. Além disso, 19,5% dos participantes afirmaram ter participado como observadores, indicando uma participação, mesmo que de forma passiva.

Incluímos uma questão sobre Web Components (Q15) com o intuito de investigar as estratégias de reutilização de código, já que os componentes podem ser facilmente compartilhados e incorporados em diferentes partes de uma aplicação ou em diferentes aplicações. Dos participantes, 48,8% afirmaram ter uma experiência positiva com os Web Components, enquanto 41,5% disseram não ter experiência com eles. Apenas 9,8% dos participantes afirmaram não perceber impacto em seu uso, conforme apresentado na Figura 4.10.

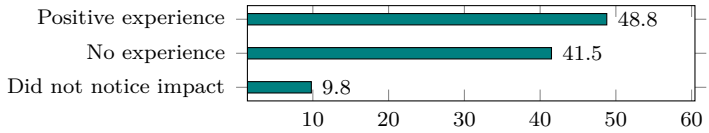


Figura 4.10: Participants Experience with Web Components

Também investigamos como os profissionais avaliam sua experiência com os Frameworks JavaScript para Micro-frontend (Q16), pois o conhecimento e a experiência com frameworks são indicadores importantes para compreender as vantagens e desvantagens da adoção da arquitetura de Micro-frontends. Entre os participantes, 43,9% afirmaram não ter experiência com o framework JavaScript. Além disso, 26,8% dos participantes mencionaram ter adotado o framework "Sem preferência, adotei conforme a equipe", devido à flexibilidade da arquitetura e às necessidades da equipe. Por outro lado, 22% dos participantes afirmaram ter utilizado mais de um framework, conforme a necessidade da equipe ou do projeto. Por fim, 7,3% dos participantes afirmaram que a escolha do framework JavaScript foi devido à preferência por um framework específico para atender às necessidades da equipe ou do projeto, conforme apresentado na Figura **??**.

Sobre a percepção dos participantes de suas experiências com relação a integração de APIs e microsserviços no contexto de micro-frontends aplicamos a questão (Q17) para identificar o conhecimento do participante sobre tecnologias que fazem parte do contexto da arquitetura de micro-frontend, ou seja, não fazem parte da arquitetura da camada frontend, mas podem impactar na sua implementação. 48,8% dos participantes afirmaram não ter experiência com APIs e microsserviços no contexto dos micro-frontends, 34,1% dos participantes afirmaram que esses recursos facilitam a integração e comunicações entre os componentes, 14,6% dos participantes afirmaram que enfrentaram alguns desafios, mas que eles foram superados. Por fim, 2,4% dos participantes afirmaram que enfrentaram

ID	Education	Experience	Role	Programming Languages
P1	Master	7 to 9 years	Tech Lead	Python, Java, C#
P2	Graduated	1 to 3 years	Software Developer	JavaScript, Java, Typescript, C#, C++
P3	Master	10+ years	Tech Lead	JavaScript, Python, Java, C#, C++, C, PHP, Rust
P4	Pos-Graduated	10+ years	Software Architect	JavaScript, Python, Java, Typescript, PHP
P5	Pos-Graduated	10+ years	Software Developer	JavaScript, Java, Typescript
P6	Master	10+ years	Software Developer	JavaScript, Java, Typescript, C#
P7	Master Student	4 to 6 years	Project Manager	JavaScript, Python, Java, Typescript, C++, C
P8	Pos-Graduated	10+ years	Project Manager	JavaScript, Python, Java, Typescript, C#, PHP, Asp
P9	Pos-Graduated	10+ years	Project Manager	JavaScript, Java, Typescript, C#, Delphi
P10	master	10+ years	Tech Lead	Python, Cobol
P11	master	10+ years	Project Manager	Python, C, PHP, R
P12	Pos-Graduated	4 to 6 years	Tech Lead	Python, R, SQL
P13	master	4 to 6 years	Project Manager	JavaScript, Python, Typescript, C#, C++, C
P14	Pos-Graduated	10+ years	Software Developer	JavaScript, Python, Java
P15	master	10+ years	Project Manager	JavaScript, Python, Typescript, C++, C, PHP
P16	master	1 to 3 years	Project Manager	Python
P17	Master	10+ years	Project Manager	Python, C#, C++, C, Basic
P18	Master	10+ years	Software Developer	Python, C#, C++, C
P19	Graduated	7 to 9 years	Tech Lead	JavaScript, Python, Java, C
P20	Pos-Graduated	10+ years	Software Developer	Python, C, PHP
P21	Master	10+ years	Project Manager	JavaScript, Python, Java, Typescript
P22	Master	10+ years	Project Manager	Java, C#, PHP, Perl
P23	Master	7 to 9 years	Software Developer	Python, R
P24	Graduated	4 to 6 years	Software Developer	JavaScript, Python, Typescript, C#
P25	Master	7 to 9 years	Tech Lead	Python, Java, C#
P26	Master	10+ years	Software Developer	JavaScript, Python, Java, Typescript, C#, C, PHP, Kotlin
P27	Master	10+ years	Tech Lead	Python, Java, C#, C
P28	Master	10+ years	Software Developer	JavaScript, Python, Java, Typescript, C++, C, PHP
P29	Pos-Graduated	1 to 3 years	Software Developer	Java
P30	Pos-Graduated	10+ years	Software Developer	JavaScript, Python, Java, Typescript, C#, C++, C, PHP, PERL
P31	Graduated	10+ years	Software Developer	JavaScript,Java,Typescript, C#, PHP
P32	Pos-Graduated	10+ years	Software Developer	JavaScript, Python, Java, Typescript, C#, PHP
P33	Master	10+ years	Tech Lead	JavaScript, Python, Java, C#
P34	Master	7 to 9 years	Project Manager	Python
P35	Graduated	1 to 3 years	Software Developer	JavaScript, Typescript, Ruby
P36	Graduated	10+ years	Software Developer	JavaScript, Python, Java, Typescript, C, PHP
P37	Master	4 to 6 years	Project Manager	JavaScript, Python, PHP
P38	Pos-Graduated	10+ years	Project Manager	Python, C
P39	Master	10+ years	Software Developer	JavaScript, Python, Java, Typescript, PHP
P40	Master	4 to 6 years	Tech Lead	JavaScript, Python, Java, C++, C
P41	Master	10+ years	Software Architect	JavaScript, Python, Java, C++, C, PHP

Tabela 4.2: Participant Demographics - Survey A

dificuldades significativas na integração de APIs e microsserviços no contexto de micro-frontends, conforme apresentado na Figura 4.11.

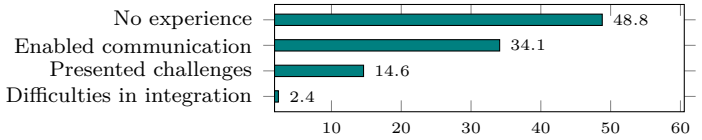


Figura 4.11: Participants Experience with APIs and Microservices

O Survey A possuía apenas uma questão aberta (Q18 da Tabela 4.1) para que os participantes pudessem expressar qualquer aspecto que julgassem relevante e que não tivesse sido abordado nas questões do survey. Apenas três participantes (P4, P36 e P41) responderam a essa pergunta:

> *#P4:"A arquitetura de micro-frontend facilitou as atividades de desenvolvimento dos projetos, mas depois apresentou desafios na integração, manutenção e correções. Em relação a performance, o uso da arquitetura foi satisfatório, uma vez que ter uma arquitetura única em todos os componentes melhora o desempenho. Além disso, quando ocorrem problemas iguais em projetos diferentes é mais fácil corrigir e conseguir auxílio dos outros membros da equipe. Em termos organizacionais foi interessante manter uma base de conhecimento das melhores práticas, desafios enfrentados e soluções adotadas durante a implementação da arquitetura."*

O comentário do participante #P4 reforça que a incorporação de novas tecnologias, incluindo aquelas associadas à arquitetura de micro frontends, pode enfrentar desafios. No entanto, destaca-se que uma implementação eficaz pode proporcionar vantagens, tais como a facilitação da manutenção e a maximização do reaproveitamento de código. A facilidade de lidar com problemas semelhantes em projetos distintos destaca a vantagem de ter uma abordagem consistente. Além disso, a criação de uma base comum de melhores práticas e soluções pelas organizações demonstra uma iniciativa efetiva para gerenciar e disseminar conhecimento, contribuindo com as atividades a serem realizadas pelos membros das equipes.

> *#P36: "Para ser sincero, eu tenho algumas desconfianças quanto a micro-frontends. Eu nunca vi sendo usado na prática, e me parece uma maneira de deixar as aplicações web ainda mais lentas e pesadas. Como eu não tive nenhuma experiência prática com a arquitetura, talvez eu possa estar enganado "*

O comentário do participante P#36 reflete uma atitude inicialmente cética em relação à adoção de micro-frontends, baseada na falta de experiência prática com a arquitetura, o que pode ser evidenciado nas Figuras 4.6, 4.10 e 4.11. O participante expressa preocupações sobre possíveis impactos negativos, como o aumento da lentidão e do peso das

aplicações web. No entanto, ele reconhece a limitação da própria experiência e mantém uma postura aberta à possibilidade de estar equivocado. Essa atitude é positiva, pois indica disposição para reconsiderar as opiniões à medida que se ganha mais experiência e conhecimento sobre a arquitetura de micro-frontends.

> *#P41: "Hoje eu entendo que devemos considerar a afinidade da equipe com o framework a ser adotado na arquitetura micro-frontend. "*

4.4 Síntese do Capítulo

Neste capítulo, destacou-se a criação do GAM, um conjunto estruturado de diretrizes, práticas recomendadas, ferramentas e recursos essenciais para orientar organizações na eficaz adoção de arquiteturas baseadas em micro-frontends. Utilizando dados extraídos de estudos selecionados na Revisão Sistemática da Literatura (RSL), identificaram-se visões e padrões arquiteturais fundamentais que embasaram o desenvolvimento do guia. Adicionalmente, foi conduzido um survey junto a profissionais experientes da área de Tecnologia da Informação, visando a verificação e validação do conteúdo do guia. Os dados coletados nesse survey foram analisados buscando validar a eficácia e relevância do GAM e garantir sua utilidade prática para os usuários finais.

Capítulo 5

Estudo de Caso

Este capítulo foi dedicado à investigação dos impactos e desafios associados à adoção da arquitetura micro-frontend, centrando-se no projeto piloto como estudo de caso. A metodologia adotada segue as diretrizes definidas por Yin et al. [63], que propõe a seguinte estrutura: Definição do Problema, Seleção de Caso, Coleta de Dados, Análise de Dados, Relatório dos Resultados e Validação dos Resultados.

5.1 Definição do Problema

A adoção de micro-frontend na Serasa Experian foi resultado de um processo de análise das necessidades da empresa, seguido por reuniões e discussões entre os principais stakeholders. O contexto competitivo do mercado exigia que a Serasa Experian respondesse de forma ágil e eficiente às demandas dos clientes, mantendo a qualidade e a escalabilidade de seus produtos. No entanto, a estrutura monolítica existente estava se tornando um obstáculo para alcançar esses objetivos.

Análise das Necessidades e Motivações para Adoção

Antes de iniciar qualquer discussão sobre a adoção de micro-frontends, a equipe de liderança da Serasa Experian realizou uma análise das limitações e desafios enfrentados pela empresa. Foi identificado que a arquitetura monolítica existente estava resultando em dificuldades significativas no desenvolvimento, manutenção e implementação de novos recursos. As equipes de desenvolvimento frequentemente se viam limitadas pela complexidade da integração de código e pela dificuldade de escalar componentes individuais sem afetar o sistema como um todo.

Além disso, a Serasa Experian busca constantemente a manutenção da competitividade em um mercado onde a agilidade e a entrega rápida de novos recursos são essenciais para atrair e reter clientes. Diante desses desafios, tornou-se evidente que uma abordagem mais

modular e distribuída para o desenvolvimento front-end poderia oferecer os benefícios necessários para impulsionar a inovação e a eficiência da organização.

Reuniões e Discussões sobre a Adoção

Com base na análise das necessidades e das vantagens potenciais da adoção de micro-frontends, foram organizadas uma série de reuniões e discussões entre os principais membros da equipe técnica, gerentes de projeto e outros stakeholders relevantes. Durante essas reuniões, foram abordados tópicos como os desafios atuais enfrentados pela equipe de desenvolvimento, os benefícios esperados da adoção de micro-frontends e as estratégias para implementar essa mudança de arquitetura.

As discussões foram fundamentais para garantir que todos os envolvidos entendessem os motivos por trás da proposta de adoção de micro-frontends e estivessem alinhados em relação aos objetivos e às expectativas. As preocupações e os pontos de vista de cada parte interessada foram levados em consideração para garantir que a transição para uma arquitetura de micro-frontends fosse suportada por todos os departamentos da organização, bem como pelas equipes envolvidas.

5.2 Seleção de Caso

Para preservar a segurança das informações da empresa e seguir suas normas, este projeto piloto não foi desenvolvido dentro da infraestrutura tecnológica da empresa. Além disso, nenhuma parte do código-fonte, seja no todo ou em parte, foi extraída de aplicações reais. Portanto, os nomes dos componentes, métodos ou funções não correspondem a partes dessas aplicações. No entanto, o projeto piloto foi baseado nos principais módulos de um sistema frontend monolítico que denominaremos neste trabalho de “Produtos”. Essa aplicação serve como plataforma para oferta e gestão de uma ampla variedade de produtos digitais em um e-commerce especializado. Os serviços oferecidos incluem consultas de crédito para os vários segmentos da organização, histórico de pagamentos, verificações de identidade, certificados digitais e consultas de crédito para pessoas físicas e jurídicas. Este sistema é composto por vários componentes, como cabeçalho, rodapé, menu superior, barra lateral, carrinho de compras, página de autenticação, página de recuperação de senha, painel de controle, perfil do cliente, entre outros.

Planejamento e Análise de Requisitos

Durante esta fase fez a análise dos requisitos do sistema e planejamento da arquitetura micro-frontend. Isso envolveu identificar os diferentes componentes da aplicação, determi-

nar quais partes poderiam ser separadas em micro-frontends e definir as interações entre eles.

1. Identificação de Componentes: A Figura 5.1 apresenta os componentes construídos como micro-frontend:

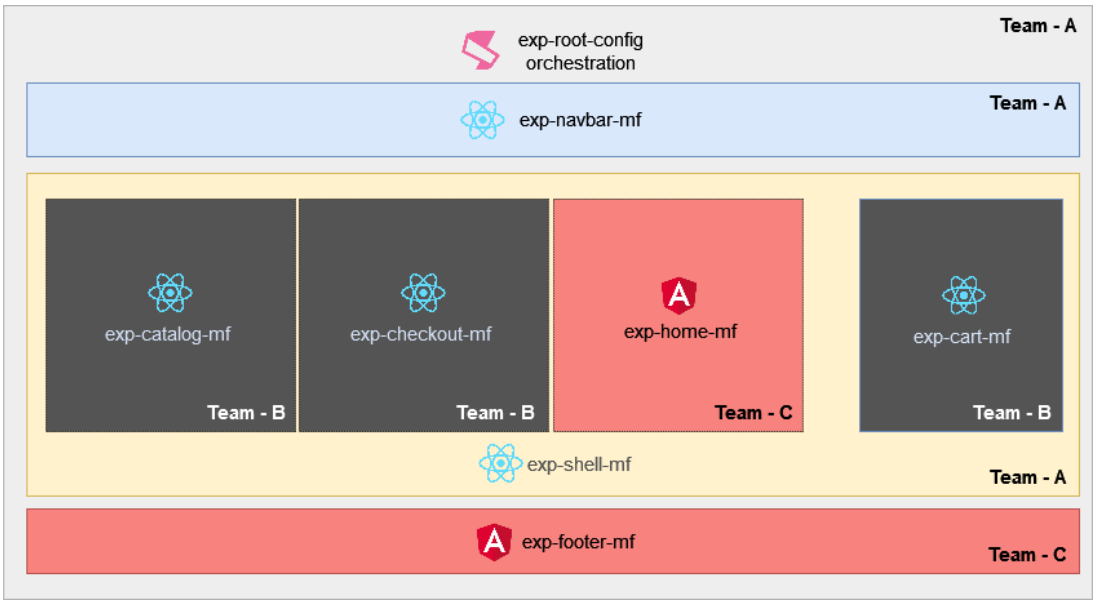


Figura 5.1: Layout do Projeto Piloto

- Root(exp-root-config): É o micro-frontend raiz que faz o papel do orquestrador. Mantido pelo time A, seu papel é configurar a aplicação usando tendo como base n framework Single-Spa e defininir os pontos de entrada para os demais micro-frontends.
- Shell(exp-shell-mf): É o micro-frontend responsável por fornecer o layout básico da aplicação, incluindo a estrutura da página, como cabeçalho, rodapé e navegação. O Shell carrega dinamicamente outros micro-frontends com base na URL atual. Foi desenvolvido com o framework React e é mantido pelo time A.
- Navbar(exp-navbar-mf): É o micro-frontend responsável por exibir a barra de navegação da aplicação. É mantido pelo time A e utiliza o framework React.
- Catalog(exp-catalog-mf): É o micro-frontend responsável por exibir o catálogo de produtos. Permite que os usuários visualizem os produtos disponíveis, adicionem produtos ao carrinho e publica eventos quando um produto é adicionado ao carrinho. Foi utilizado o framework React, e é mantido pelo time B.
- Cart(exp-cart-mf): É o micro-frontend responsável por exibir o carrinho de compras. Recebe eventos quando um produto é adicionado ao carrinho pelo

catálogo, mantém o estado dos itens no carrinho, atualizando-o conforme os produtos são adicionados, calcula o total da compra com base nos itens no carrinho e oferece a funcionalidade de limpar o carrinho. Foi utilizado o framework React e é mantido pelo time B.

- Checkout(exp-checkout-mf): É o microfrontend responsável por exibir a página de checkout. Exibe informações de contato e opções de pagamento e permite ao usuário finalizar a compra. Foi utilizado o framework React e é mantido pelo time B.
- Home(exp-home-mf): É o micro-frontend responsável por exibir a página inicial da aplicação. Pode incluir informações gerais sobre a loja, banners promocionais ou categorias de produtos em destaque. Foi utilizado o framework Angular e é mantido pelo time C.
- Footer(exp-footer-mf): É o micro-frontend responsável por exibir o rodapé da aplicação. Foi utilizado o framework Angular e é mantido pelo time C.

2. Decisão sobre Micro-frontends: Optou-se pela escolha da Decisão Horizontal, que é apropriada para projetos que demandam uma evolução consistente da interface do usuário e uma experiência fluida em diversas visualizações. A Figura 5.1 apresenta a decisão horizontal onde vários micro-frontends coexistem na mesma visão, independente do framework utilizado (Angular, React, etc.).

Configuração do Ambiente de Desenvolvimento

Para configurar o projeto utilizando o single-spa, é importante atender aos seguintes pré-requisitos de configuração: Ambiente de Desenvolvimento: Instalação do Node.js[1], npm ou yarn, e um editor de código, por exemplo, Visual Studio Code[2].

Implementação do Root (exp-root-config)

Para implementar o micro-frontend root usando single-spa, seguiu-se os seguintes passos:

1. Instalação do Single-spa: Foi instalado no ambiente de desenvolvimento via npx utilizando o seguinte comando:

[1]https://nodejs.org/en
[2]https://code.visualstudio.com/

```
npx create-react-app exp-navbar-mf
? Directory for new project (.)  exp-navbar-mf
  single-spa application / parcel
  in-browser utility module (styleguide, api cache, etc)
>  single-spa root config
? Which package manager do you want to use?
  yarn
> npm
  pnpm
? Will this project use Typescript? No
? Would you like to use single-spa Layout Engine (Y/n) No
? Organization name (can use letters, numbers, dash or
   underscore) experian
```

2. Configuração do micro-frontend Root(Orquestrador): Foi criado um arquivo JavaScript para configurar o micro-frontend root chamado experian-root-config.js, responsável por inicializar o single-spa e definir os pontos de entrada para os outros micro-frontends.

```
import { registerApplication, start } from "single-spa";

registerApplication({
  name: "@experian/exp-navbar-mf",
  app: () => System.import("@experian/exp-navbar-mf"),
  activeWhen: ["/"],
});

# Others mf can be configured here

start({
  urlRerouteOnly: true,
});
```

Também foi necessário incluir o arquivo index.ejs na aplicação root do single-spa. Ele é o ponto de entrada para a renderização da página da web e desempenha um papel fundamental na configuração e inicialização da aplicação single-spa:

```
<% if (isLocal) { %>
  <script type="systemjs-importmap">
    {
      "imports": {
        "@experian/root-config":
    "//localhost:9000/experian-root-config.js",
        "@experian/exp-shell-mf":
    "//localhost:8000/experian-exp-shell-mf.js",
        "@experian/exp-navbar-mf":
    "//localhost:8100/experian-exp-navbar-mf.js",
        ... others mf
      }
    }
  </script>
  <% } %>
```

Importante: O código completo da aplicação micro-frontend exp-root-config pode ser encontrada no repositório: https://github.com/giovanniamorim/exp-root-config

Implementação do Shell (exp-shell-mf)

É a aplicação micro-frontend desenvolvida no framework React responsável por fornecer o layout básico da aplicação, incluindo a estrutura da página, como cabeçalho, rodapé e navegação. Ele carrega dinamicamente outros micro-frontends com base na URL atual.

1. Criação de um novo projeto React: Executar o seguinte comando no terminal para criar um novo projeto React e responder as questões:

```
npx create-react-app exp-navbar-mf
? Directory for new project (.)  exp-navbar-mf
> single-spa application / parcel
? Which framework do you want to use? (Use arrow keys)
> react
```

```
? Which package manager do you want to use?
> npm
? Will this project use Typescript? No
? Organization name (can use letters, numbers, dash or
   underscore) experian
? Project name (can use letters, numbers, dash or underscore)
   exp-navbar-mf
```

2. Desenvolvimento do Componente: Dentro da pasta do projeto, foi criado os arquivos como src/Shell.js e Shell.css com os conteúdos javascript, html e css do shell.

Importante: O código completo da aplicação micro-frontend exp-navbar-mf pode ser encontrado no repositório: https://github.com/giovanniamorim/exp-shell-mf

Implementação do Navbar (exp-navbar-mf)

Para criar e instalar o micro-frontend exp-navbar-mf em React, foram seguidos os seguintes passos:

1. Criação de um novo projeto React: Foram seguidos os mesmo passos da implementação da aplicação Shell, alterando apenas o nome da aplicação:

2. Instalação de Dependências Adicionais: Foi necessário a instalação da dependência para roteamento:

```
npm install react-router-dom
```

3. Integração com Root: O micro-frontend exp-navbar-mf foi integrado com o Root single-spa para que ele possa ser carregado dinamicamente em sua aplicação modular. O seguinte trecho de código foi adicionado no arquivo experian-root-config.js:

```
registerApplication({
  name: "@experian/exp-navbar-mf",
  app: () => System.import("@experian/exp-navbar-mf"),
  activeWhen: ["/"],
});
```

Também foi adicionado no arquivo index.ejs a entrada para o micro-frontend ex-navbar-mf:

```
"@experian/exp-navbar-mf":
   "//localhost:8100/experian-exp-navbar-mf.js",
```

4. Desenvolvimento do Componente: Dentro da pasta do projeto, foi criado os arquivos como src/Navbar.js e NavBar.css com os conteúdos javascript, html e css do navbar. A Figura 5.2 apresenta resultado no navegador foi:

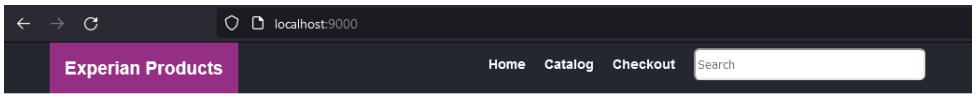


Figura 5.2: Micro-frontend Navbar

Importante: O código completo da aplicação micro-frontend exp-navbar-mf pode ser encontrada no repositório: https://github.com/giovanniamorim/exp-navbar-mf

Implementação do Catalog (exp-catalog-mf)

Para criar e instalar o micro-frontend exp-catalog-mf em React, foram seguidos os mesmos passos da implementação do Navbar alterando apenas o nome do componente. A Figura 5.3 apresenta o resultado da renderização do componente no navegador:

Importante: O código completo da aplicação micro-frontend exp-catalog-mf pode ser encontrada no repositório: https://github.com/giovanniamorim/exp-catalog-mf

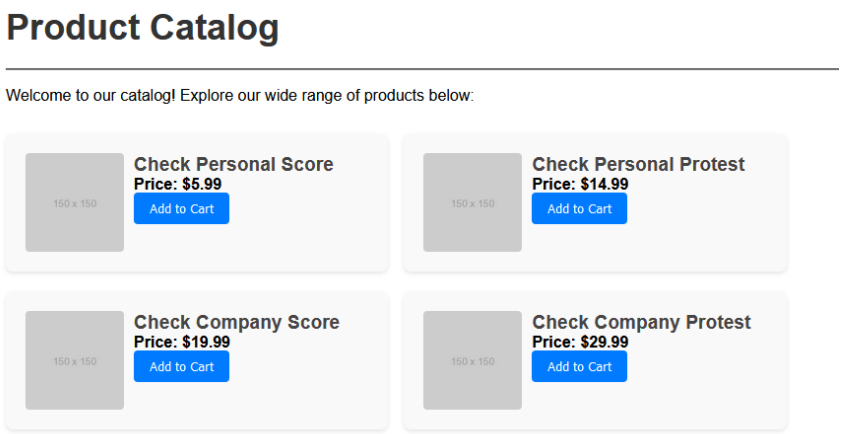


Figura 5.3: Micro-frontend Catalog

Implementação do Cart (exp-cart-mf)

Para criar e instalar o micro-frontend exp-cart-mf em React, foram seguidos os mesmos passos da implementação do Navbar alterando apenas o nome do componente. A Figura 5.5 apresenta o resultado da renderização do componente no navegador:

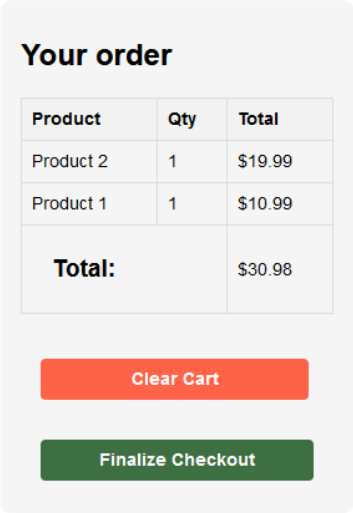


Figura 5.4: Micro-frontend Cart

Importante: O código completo da aplicação micro-frontend exp-cart-mf pode ser encontrada no repositório: https://github.com/giovanniamorim/exp-cart-mf

Implementação do Checkout (exp-checkout-mf)

Para criar e instalar o micro-frontend exp-checkout-mf em React, foram seguidos os mesmos passos da implementação do Navbar alterando apenas o nome do componente. A Figura 5.5 apresenta o resultado da renderização do componente no navegador:

Importante: O código completo da aplicação micro-frontend exp-checkout-mf pode ser encontrada no repositório: https://github.com/giovanniamorim/exp-checkout-mf

Implementação do Footer (exp-footer-mf)

Para criar e instalar o micro-frontend exp-footer-mf em Angular, foram seguidos os mesmos passos da implementação do Navbar alterando apenas a opção do framework para Angular e o nome do componente para exp-footer-mf. A Figura 5.5 apresenta o resultado da renderização do componente no navegador:

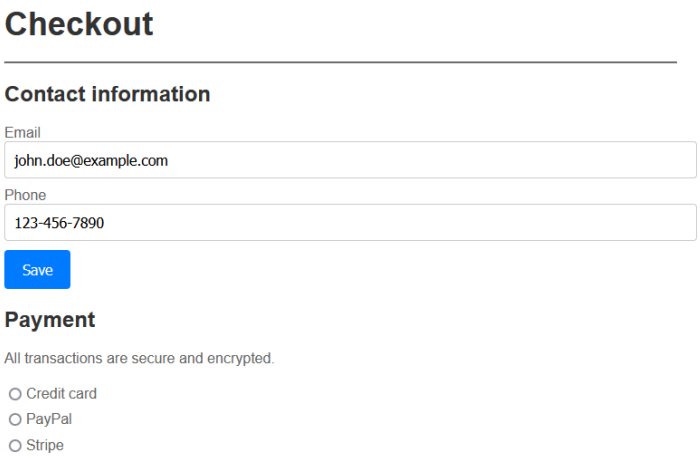


Figura 5.5: Micro-frontend Cart

Figura 5.6: Micro-frontend Footer

Importante: O código completo da aplicação micro-frontend exp-footer-mf pode ser encontrado no repositório: https://github.com/giovanniamorim/exp-footer-mf

Implementação do Home (exp-home-mf)

Assim como o Footer, para criar e instalar o micro-frontend exp-home-mf em Angular, foram seguidos os mesmos passos da implementação do Navbar alterando apenas a opção do framework para Angular e o nome do componente para exp-home-mf. A Figura 5.7 apresenta o resultado da renderização do componente no navegador:

Importante: O código completo da aplicação micro-frontend exp-home-mf pode ser encontrado no repositório: https://github.com/giovanniamorim/exp-home-mf

5.3 Coleta de Dados

5.3.1 Análise Documental

Durante a coleta de dados para o estudo de caso, foram analisados os documentos relacionados a arquitetura micro-frontend antes da sua adoção. Esses documentos incluíam:

- **Guideline micro-frontend:** Este documento apresenta a arquitetura micro-frontend, seus conceitos e aplicabilidades e também um *hands on* onde mostra passo-a-passo

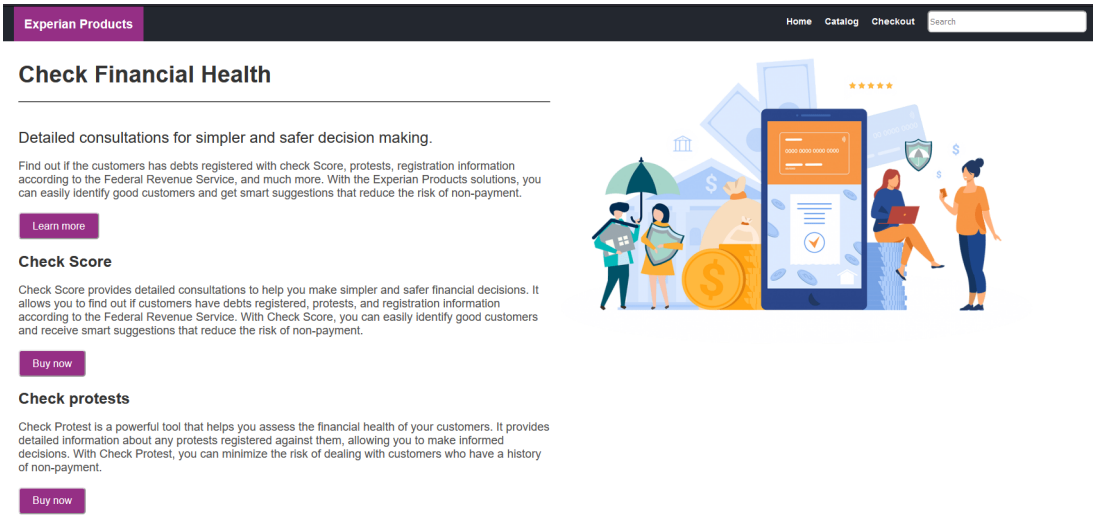


Figura 5.7: Micro-frontend Home

como desenvolver um sistema baseado na arquitetura micro-frontend usando o framework react e suas bibliotecas react dom e react testing library. Além disso, outras bibliotecas foram utilizadas, tais como: webpack, Eslint, Design System (Atomic Design [3]).

- **Rollback em micro-frontend:** Este documento ensina como realizar um rollback manual em um micro-frontend nos ambientes de desenvolvimento e produção
- **Introdução ao micro-frontend:** Apresenta conceitos básicos sobre a arquitetura micro-frontend ensinando como criar um novo projeto e novas aplicações utilizando o framework react.
- **Integrando Micro-frontend derivado a Micro-frontend Orquestrador:** Mostra como integrar um micro-frontend derivado a um orquestrador, conectando um micro-frontend autônomo a um sistema de gerenciamento central. Apresenta as configurações do arquivo package.json, index.ejs e configurações de rota.
- **Ambiente AWS de Desenvolvimento:** documentação que apresenta a automação do micro-frontend no ambiente a Amazon Web Service, contém parâmetros de configurações e ambientes para testes.

5.3.2 Survey B

Foi conduzido um segundo survey com os membros da equipe do da companhia Serasa envolvidos no desenvolvimento e implementação do projeto piloto, que incluiu desenvolvedores, arquitetos de software, líderes técnicos e gerentes de projeto. O survey foi

[3] https://atomicdesign.bradfrost.com/

composto de 16 questões, conforme apresentado na Tabela 5.1. As questões abordam diferentes aspectos do projeto, tais como: objetivos, desafios enfrentados, lições aprendidas e percepções da equipe sobre a transição para a nova arquitetura.

ID	Pergunta
Q1	Qual seu nível educacional
Q2	Há quanto tempo trabalha na área de tecnologia/engenharia de software?
Q3	Qual é o seu cargo/função atual na sua Organização Serasa Experian?
Q4	Há quanto tempo trabalha na Organização Serasa Experian?
Q5	Quais Linguagens de Programação você tem experiência?
Q6	Quais dos seguintes frameworks possui experiência?
Q7	Qual é o seu nível de envolvimento com as decisões arquiteturais do frontend na Organização Serasa Experian?
Q8	Como você descreveria seu nível de familiaridade com a arquitetura de micro-frontends?
Q9	Sobre o tamanho do projeto em que atua, considerando sua complexidade e escopo em termos de funcionalidades e requisitos, pode-se afirmar que:
Q10	Com relação ao número de pessoas que trabalham no seu time:
Q11	Como você descreveria o nível das expectativas sobre o micro-frontend antes da adoção e implementação da arquitetura?
Q12	Qual o nível de importância das Provas de Conceitos (POCs) ou Projetos Pilotos na adoção do micro-frontend antes da adotar a arquitetura?
Q13	Indique quais os desafios foram identificados durante adoção e implementação da arquitetura micro-frontend nos projetos que você atuou:
Q14	Sobre os impactos positivos da adoção da arquitetura micro-frontend, pode-se afirmar que:
Q15	Sobre os impactos negativos da adoção da arquitetura micro-frontend, pode-se afirmar que:
Q16	Alguma questão não abordada neste survey que gostaria de comentar a respeito?

Tabela 5.1: Questões do Survey 2

5.4 Análise de Dados

O survey B foi aplicado à equipe da Companhia Serasa Experian que desenvolveu a Prova de Conceito (POC) para analisar os impactos e desafios de se adotar a arquitetura micro-frontend. O survey englobou perguntas que exploraram as percepções e experiências iniciais dos desenvolvedores relacionadas ao micro-frontend. Em seguida, foi discutido com os participantes as abordagens abordadas na arquitetura empregada, os desafios enfrentados durante o ciclo de desenvolvimento e os documentos técnicos do projeto.

Foram analisadas as respostas do survey sobre a arquitetura de micro-frontends na Serasa Experian, identificando padrões, tendências e insights relevantes. A Tabela 5.2 apresenta os dados demográficos

dos participantes da pesquisa. No total, obteve-se 8 respostas de colaboradores (P1 a P8) da Serasa Experian, com diferentes cargos, níveis de experiência e envolvimento com micro frontends.

	P1	P2	P3	P4	P5	P6	P7	P8
Nível Educacional	Graduado	Especializado	Mestre	Graduado	Especializado	Especializado	Graduado	Especializado
Tempo de Trabalho na TI	De 7 a 9 anos	De 4 a 6 anos	Mais de 10 anos	De 4 a 6 anos	De 4 a 6 anos	De 4 a 6 anos	Mais de 10 anos	De 7 a 9 anos
Cargo	Desenvolvedor de Software	Desenvolvedor de Software	Líder Técnico	Desenvolvedor de Software	Líder Técnico	Product Owner	Desenvolvedor de Software	Scrum Master
Tempo de trabalho na Serasa	De 1 a 3 anos	De 1 a 3 anos	De 1 a 3 anos	De 1 a 3 anos	De 4 a 6 anos	De 1 a 3 anos	De 1 a 3 anos	Menos de 1 anos
Linguagens de Programação	Java, Typescript	JavaScript, Typescript	JavaScript, Java, Typescript	JavaScript, Typescript	JavaScript, Java, Typescript	JavaScript	JavaScript, Java, Typescript	N/A
Frameworks	Angular, Spring Boot	Angular, React, Vue	Angular, Vue, Spring Boot	Angular	Angular, React, Spring Boot	Angular	Angular, React, Spring Boot	N/A
Envolvimento no frontend	Moderado	Alto	Moderado	Moderado	Alto	Moderado	Nenhum	Nenhum
Familiaridade com o micro-frontend	Alto	Moderado	Moderado	Moderado	Alto	Baixo	Baixo	Nenhum
Tamanho do Projeto	Médio	Médio	Médio	Médio	Grande	Médio	Grande	Médio
Tamanho do Time	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas	De 4 a 8 pessoas
Expectativas da Adoção	Baixa	Moderada	Moderada	Alta	Alta	Alta	Baixa	Moderada
Importância do Projeto Piloto	Alta	Alta	Alta	Alta	Alta	Alta	Alta	Alta
Desafios Identificados	Integração	Integração, Desempenho	Integração, Escopo, Roteamento	Integração, Desempenho	Integração, Escopo	Escopo	Desempenho, Escopo	N/A
Impactos positivos	Codificação Paralela	Manutenção	Desempenho, Colaboração Equipe, Manutenção	Modularidade e Escalabilidade	Modularidade e Escalabilidade, Desempenho, Equipe, Manutenção	Manutenção	Colaboração Equipe, Manutenção	N/A
Impactos negativos	Rotas	Curva de Aprendizado	Rotas	Integração	Curva de Aprendizado	Integração	Curva de Aprendizado	N/A

Tabela 5.2: Dados Demográficos dos Participantes - Survey B

37.5% dos participantes do survey possuem curso de graduação na área de TI e 50% deles possuem curso de especialização na área de TI. Apenas 12,5% dos participantes possuem mestrado, conforme apresentado na Figura 5.8.

No que se refere ao tempo de trabalho e experiência que os participantes possuem na área de tecnologia ou em engenharia de software, o survey mostrou que 50% dos participantes possuem entre 4 e 6 anos de tempo de trabalho, 25% dos participantes possuem entre 7 e 9 anos, enquanto que, aqueles que possuem mais de 10 anos de tempo de trabalho somam 25% dos participantes, conforme apresentado na Figura 5.9.

A experiência e os papéis dos participantes na Serasa é um fator relevante, as demais questões são direcionadas a esse contexto. Quando indagado sobre a função ou cargo que os participantes atuam na organização o survey mostrou que 50% dos participantes são desenvolvedores de software, 25% dos participantes são líderes técnicos, 12,5% são scrum master e, por fim, 12,5% dos participantes possuem o

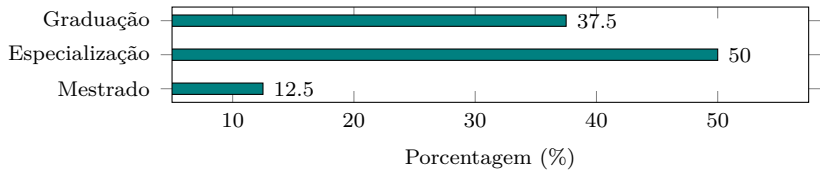


Figura 5.8: Nível de formação dos participantes.

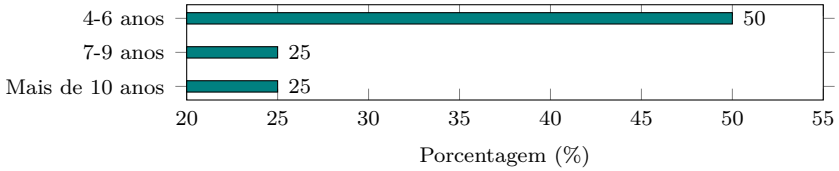


Figura 5.9: Tempo de experiência dos participantes

papel de agilista. A Figura 5.10, mostra como ficou essa distribuição de cargos ou funções. Essa diversidade de funções sugere uma abordagem multidisciplinar, com diferentes habilidades e responsabilidades contribuindo para o sucesso geral da implementação do micro-frontend.

Figura 5.10: Cargo/Função dos participantes

Seguindo com a importância dos papéis dos participantes na Companhia, perguntamos sobre o tempo de trabalho na Serasa Experian. A Figura 5.11 apresenta o resultado do survey com a seguinte distribuição: 12,5% dos participantes possuem menos de 1 ano, 12,5% de participantes possuem entre 4 a 6 anos, e a maioria, 75% dos participantes possuem entre 1 a 3 anos na empresa. Assim, podemos considerar que os participantes possuem experiência com as atividades desenvolvida pela organização do estudo de caso.

Também perguntamos sobre as linguagens de programação nas quais os participantes possuíam experiência. Conforme ilustrado na Figura 5.12, 75% dos participantes afirmaram ter experiência em Javascript, 50% em Java e 75% em Typescript. Além disso, 12,5% dos participantes não possuem experiência nessas linguagens de programação. É relevante observar que este último percentual corresponde aos agilistas, corroborando a observação de que este perfil geralmente não se destaca por habilidades em programação, mas possui importância no desbloqueio de atividaDesenvolvedor

Os frameworks são ferramentas indispensáveis para a adoção da arquitetura micro-frontend. Neste contexto, perguntamos aos participantes sobre suas experiências com os principais frameworks utilizados nesta abordagem. A Figura 5.13 mostra que: 87,5% dos participantes possuem experiência no framework Angular, 37,5% dos participantes possuem experiência em React, 25% possuem experiencia em Vue, e 12,5% não possuem experiência nessas tecnologias.

Perguntamos aos participantes sobre o nível de envolvimento com as decisões arquiteturais do frontend na organização Serasa Experian. Esta questão visa compreender o grau de participação e influência dos colaboradores nas escolhas relacionadas à arquitetura do frontend, o que permite avaliar o alinhamento entre a equipe de desenvolvimento e as estratégias arquiteturais da empresa. 50% dos participantes

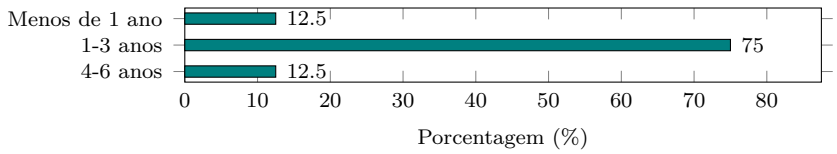


Figura 5.11: Tempo de experiência dos participantes no Serasa Experian.

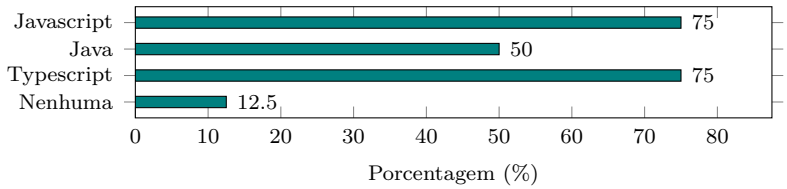


Figura 5.12: Experiência dos participantes em linguagens de programação

afirmaram que o nível de envolvimento é moderado, 25% escolheu a opção onde o nível de envolvimento é baixo e os 25% restantes dos participantes afirmaram que possuíam um envolvimento alto nas decisões arquiteturais na camada frontend das aplicações. Seguindo a mesma abordagem também perguntamos sobre a familiaridade do participante quanto a arquitetura micro-frontend, o resultado foi que 12.5% dos participantes não possuem familiaridade, 25% dos participantes se declararam com baixa familiaridade, 37.5% afirmaram que possuem familiaridade moderada e, por fim, 25% possuem alta familiaridade com a arquitetura micro-frontend. A Figura 5.14 mostra o resultado do survey comparando os aspectos dessas duas questões (Q7 e Q8 da Tabela 5.1).

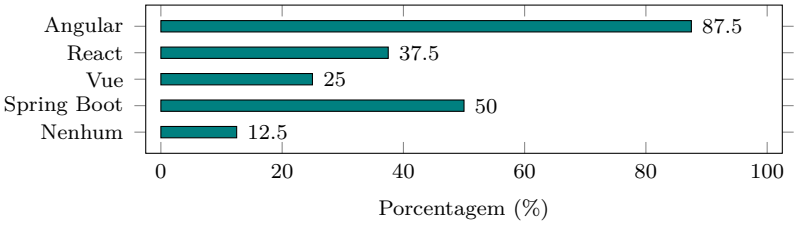


Figura 5.13: Experiência dos participantes em Frameworks

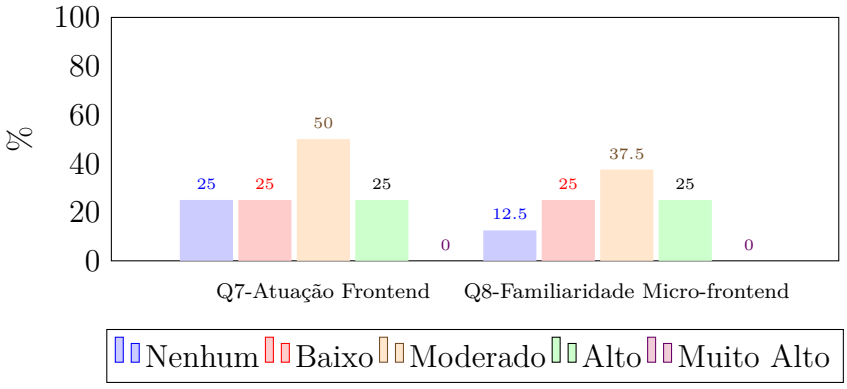


Figura 5.14: Nível de atuação no desenvolvimento frontend e nível de conhecimento na arquitetura micro-frontend

As questões Q9 e Q10 do survey (Tabela 5.1) foram definidas para compreender a relação da arquitetura micro-frontend entre o tamanho do projeto e o número de integrantes do time, respectivamente.

Com relação a Q9, que trata do tamanho do projeto, entende-se por projetos pequenos aqueles com poucas funcionalidades, requisitos e baixa complexidade. Os projetos médios possuem um número moderado de funcionalidades e requisitos com complexidade intermediária. Por fim, projetos grandes são aqueles que possuem muitas funcionalidades e requisitos com alta complexidade. A Figura **??** apresenta o resultado do survey para essa questão, onde 25% dos participantes informaram que atuam em projetos grandes e 75% atuam em projetos médios.

Por sua vez, os resultados da questão Q10 apresenta dados que veem de encontro ao que é sugerido pelos times que adotam a metodologia ágil, onde 100% dos participantes escolheram a opção "de 4 a 8 pessoas".

Sobre a adoção e implementação da arquitetura micro-frontend no Serasa, nas questões Q11 e Q12 respectivamente, foi questionado aos participantes sobre as suas expectativas e percepções sobre a importância das POCs e projetos pilotos como ferramenta para avaliar a adoção da arquitetura micro-frontend. Os resultados do survey mostraram que 25% dos participantes tinham expectativas baixas, enquanto 37.5% atribuíram expectativas moderadas. Por fim, a mesma porcentagem de participantes, 37.5% afirmaram possuir expectativas altas em relação a arquitetura de micro-frontend. Em relação a importância das POCs (Q12), todos os participantes foram unânimes quanto a sua alta importância, conforme apresentado na Figura 5.15.

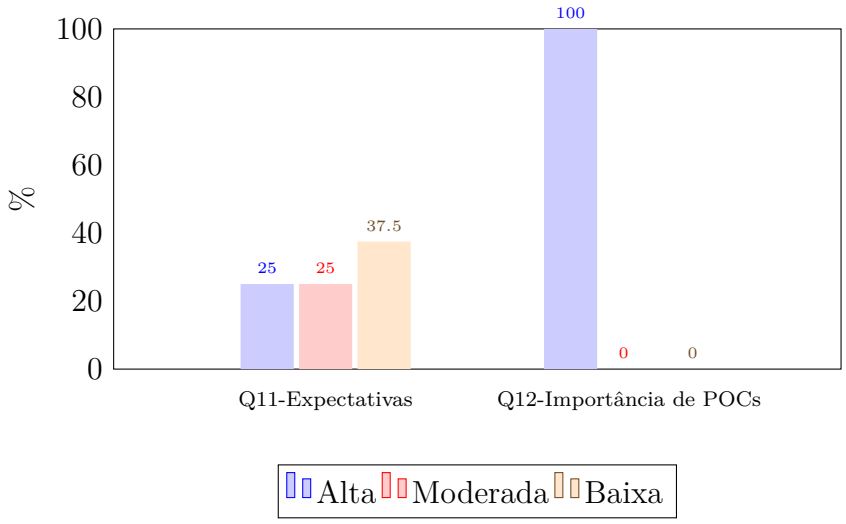


Figura 5.15: Expectativas sobre o micro-frontend e importância das POCs

Na questão Q13 (Tabela 5.1), foi solicitado que os participantes identificassem, em uma lista, os desafios enfrentados pela equipe durante a adoção e implementação da arquitetura de micro-frontends. Conforme apresentado na Figura 5.16, 62,5% dos participantes selecionaram os desafios relacionados à integração de diferentes tecnologias e frameworks. Em seguida, 37,7% dos participantes indicaram problemas de desempenho devido à demanda de recursos no cliente, enquanto 50% dos participantes indicaram a complexidade na definição de responsabilidades entre os micro-frontends. Além disso, 25% dos participantes mencionaram os obstáculos relacionados ao roteamento e navegação na aplicação, enquanto apenas 12,5% deles afirmaram que a sua equipe não enfrentava nenhum dos desafios mencionados.

A questão Q14 da Tabela 5.1, teve como objetivo identificar os principais impactos positivos percebidos pelos participantes, durante a adoção e implementação da arquitetura de micro-frontends. 25% dos

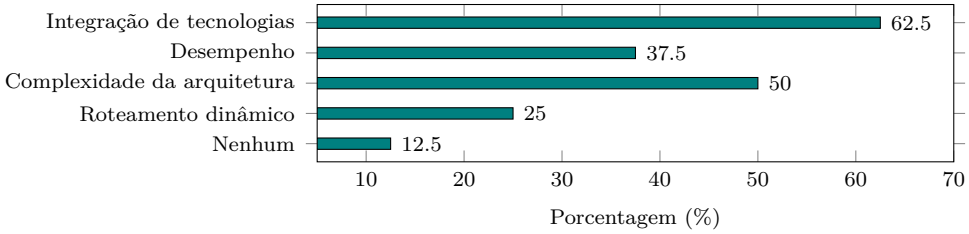


Figura 5.16: Desafios enfrentados durante adoção e implementação

participantes mencionaram como impacto positivo a Modularidade e a melhoria no desempenho da aplicação, respectivamente. A colaboração da equipe foi considerada positiva por 37.5% dos participantes, enquanto a manutenção do código foi selecionada por 62.5% dos participantes. No entanto, apenas 12.5% dos participantes mencionaram benefícios relacionados ao desenvolvimento paralelo, conforme apresentado na Figura 5.17.

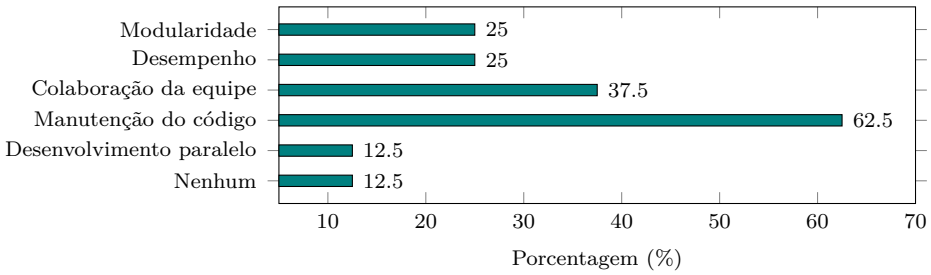


Figura 5.17: Impactos positivos da adoção e implementação da arquitetura micro-frontend

Quanto aos impactos negativos da adoção e implementação da arquitetura de micro-frontend (questão Q15 da Tabela 5.1), 37.5% dos participantes afirmaram que a curva de aprendizado da arquitetura foi o maior desafio enfrentado pela equipe. 12.5% dos participantes afirmaram ser a integração entre os micro-frontends, a complexidade no roteamento e na navegação, respectivamente. Por fim, 25% dos participantes afirmaram não ter tido nenhum impacto negativo na sua equipe, conforme apresentado na Figura 5.18.

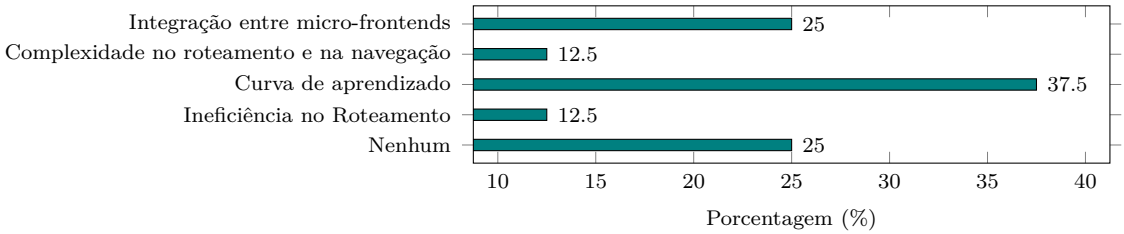


Figura 5.18: Impactos negativos durante adoção e implementação

Por fim, o survey possuía uma questão aberta para que os participantes pudessem fazer comentários que julgassem relevantes. No entanto, nenhum dos 8 profissionais fizeram nenhum comentário.

Análise de Documentos Técnicos

Os documentos analisados são usados pelos desenvolvedores da empresa como guia para a implementação da arquitetura de micro-frontend, eles cobrem uma variedade de aspectos desde os conceitos básicos até procedimentos práticos e integração com ferramentas e serviços específicos. O "Guideline micro-frontend" é um recurso que oferece uma compreensão detalhada da arquitetura, suas aplicações e um tutorial prático para desenvolver sistemas baseados nesse modelo utilizando o framework React e outras bibliotecas relevantes. Isso não apenas fornece uma base para as equipes de desenvolvimento, mas também ajuda a garantir a consistência e a qualidade na implementação dos micro-frontends.

Além disso, o documento sobre "Rollback em micro-frontend" apresenta informações importantes para o desenvolvedor garantir a estabilidade do sistema, oferecendo orientações sobre como reverter implementações de micro-frontends em ambientes de desenvolvimento e produção. Isso é essencial para lidar com possíveis problemas ou falhas durante o processo de implementação e deploy, minimizando o impacto nos usuários e permitindo uma rápida recuperação.

A "Introdução ao micro-frontend" serve como um ponto de partida para equipes que estão iniciando sua jornada com essa arquitetura, fornecendo uma visão geral dos conceitos básicos e orientações sobre como começar novos projetos e aplicações usando o framework React. Isso ajuda a nivelar o conhecimento dentro da equipe e a garantir uma compreensão uniforme dos princípios subjacentes do micro-frontend.

O documento sobre "Integrando Micro-frontend derivado a Micro-frontend Orquestrador" aborda o aspecto da implementação de micro-frontends em escala, fornecendo instruções detalhadas sobre como integrar micro-frontends derivados a um orquestrador central. Isso é essencial para garantir a coesão e a interoperabilidade do sistema, especialmente em ambientes complexos e distribuídos.

Por fim, a documentação sobre o "Ambiente AWS de Desenvolvimento" oferece uma visão prática da automação do processo de implantação e teste de micro-frontends na infraestrutura da *Amazon Web Services*. Isso permite que as equipes desenvolvam e iterem rapidamente em um ambiente controlado e escalável, facilitando o desenvolvimento e a entrega contínuos de novas funcionalidades e atualizações.

5.5 Discussão dos Resultados

O projeto piloto demonstrou que, ao adotarem a arquitetura de micro-frontend, as equipes de desenvolvimento de software se deparam com alguns impactos positivos e negativos. Entre os impactos positivos, destacou-se a capacidade de desenvolvimento paralelo. Com o micro-frontend, as equipes podem trabalhar em diferentes partes do sistema de forma independente, acelerando o tempo de entrega e permitindo uma resposta mais rápida às demandas do mercado.

Além disso, a modularidade inerente ao micro-frontend simplifica a manutenção do sistema. As atualizações e correções podem ser feitas em micro-frontends individuais sem afetar o restante do sistema, tornando o processo de manutenção mais ágil e eficiente. Isso é especialmente importante em ambientes de desenvolvimento ágil, onde a capacidade de resposta rápida às mudanças é fundamental.

Outro impacto positivo foi a promoção da colaboração em equipe. Com o micro-frontend, equipes de desenvolvimento com diferentes especialidades podem trabalhar juntas de forma mais integrada. Por exemplo, na Figura 5.1 é apresentada uma visão geral da aplicação onde três equipes de diferentes especialidades trabalham de forma coesa. Isso permite uma troca mais fluída de conhecimento e habilidades, resultando em uma maior eficiência e qualidade do trabalho produzido.

No entanto, a adoção do micro-frontend também apresenta desafios significativos. Um dos principais desafios é garantir a integração adequada entre os diferentes micro-frontends e o sistema como um todo.

Isso requer uma coordenação cuidadosa e uma comunicação eficaz entre as equipes de desenvolvimento, a fim de evitar inconsistências e conflitos no sistema final. Além disso, o conflito de escopo pode surgir à medida que o sistema cresce e se torna mais complexo. Diferentes equipes podem ter visões diferentes sobre o escopo de seus micro-frontends, o que pode levar a dificuldades na manutenção e evolução do sistema.

Outros desafios incluem questões de desempenho, como sobrecarga de rede e tempo de carregamento da página, e a curva de aprendizado associada à transição para uma arquitetura de micro-frontend. As equipes de desenvolvimento podem precisar adquirir novas habilidades e conhecimentos para trabalhar efetivamente com essa arquitetura, o que pode levar tempo e recursos significativos, podendo impactar o planejamento do projeto.

Portanto, ao adotar a arquitetura de micro-frontend, as equipes de desenvolvimento podem esperar experimentar uma série de impactos positivos, mas também enfrentarão desafios significativos. Gerenciar esses desafios de forma eficaz é essencial para garantir o sucesso da implementação do micro-frontend e aproveitar ao máximo os potenciais benefícios da sua adoção.

5.6 Validade e Confiabilidade do Estudo de Caso

A validação do estudo de caso foi baseada no projeto piloto e nos resultados da análise das respostas do participantes do survey B, conduzida de maneira rigorosa para assegurar a credibilidade, validade e relevância das conclusões obtidas. Primeiramente, uma revisão da literatura foi realizada para comparar detalhadamente os resultados do estudo com pesquisas anteriores. Esse processo permitiu identificar consistências e discrepâncias, destacando a contribuição do presente estudo para o corpo de conhecimento existente sobre a arquitetura micro-frontend.

Para garantir a validade dos dados coletados, utilizou-se a triangulação, que envolveu múltiplas fontes de dados, métodos de coleta e perspectivas diversas. As fontes de dados incluíram observações diretas do desenvolvimento do projeto e análise de documentos como especificações técnicas. Além disso, diferentes funções e níveis hierárquicos dentro da equipe foram considerados, proporcionando uma visão abrangente das experiências e opiniões sobre o projeto.

Uma descrição detalhada do contexto do projeto piloto foi fornecida, incluindo código que foram disponibilizados publicamente. Não foram incluídas informações sobre o ambiente organizacional da Serasa e tampouco de seus projetos. A transparência metodológica foi mantida ao fornecer uma descrição detalhada da metodologia adotada, incluindo a coleta e análise de dados, e ao justificar todas as escolhas metodológicas, desde a seleção dos participantes do survey até os métodos de análise. Esses esforços buscaram garantir que os resultados apresentados fossem robustos, confiáveis e contribuissem significativamente para o entendimento da implementação da arquitetura micro-frontend em diversos cenários.

Capítulo 6

Discussão

Neste capítulo interpretaremos e contextualizaremos os resultados obtidos deste trabalho durante suas fases, examinanado os achados à luz da literatura existente, discute as implicações teóricas e práticas, identifica as limitações do estudo e sugere direções para pesquisas futuras.

Resumo dos Principais Achados

Os principais achados deste estudo incluem a eficácia da arquitetura micro-frontend na modularidade e escalabilidade do projeto, a variação nas expectativas de adoção entre diferentes participantes e os desafios enfrentados, como integração e curva de aprendizado. Além disso, a familiaridade dos participantes com a arquitetura micro-frontend influenciou significativamente seu envolvimento e percepção do projeto.

Interpretação de Resultados

Os resultados do estudo indicam que a arquitetura micro-frontend proporciona vantagens significativas em termos de modularidade e escalabilidade, alinhando-se com estudos anteriores que destacam esses benefícios [2]. No entanto, alguns desafios específicos, como a integração e a curva de aprendizado, foram mais pronunciados neste estudo, possivelmente devido ao contexto organizacional específico da Serasa e ao nível de experiência variado dos participantes.

A variação nas expectativas de adoção entre os participantes pode ser explicada pelas diferenças em seus níveis de experiência e familiaridade com a arquitetura micro-frontend. Participantes mais experientes mostraram expectativas mais altas, refletindo confiança na tecnologia, enquanto aqueles menos familiarizados apresentaram expectativas mais moderadas ou baixas.

Implicações Teóricas e Práticas

Este estudo contribui para a teoria da arquitetura de software ao fornecer evidências empíricas sobre os benefícios e desafios da implementação de micro-frontends. Ele apoia a literatura existente que defende a modularidade e escalabilidade como principais vantagens [1], mas também destaca a necessidade de considerar fatores como a integração e a curva de aprendizado, que podem ser subestimados em análises teóricas. Além disso, os achados reforçam a importância do contexto organizacional na implementação de novas tecnologias, conforme sugerido por Orlikowski [64]. Quanto as implicações Práticas, os resultados deste estudo fornecem insights valiosos para empresas considerando a adoção da arquitetura micro-frontend. A modularidade e escalabilidade desta abordagem podem facilitar a gestão de projetos

complexos, permitindo o desenvolvimento paralelo e a manutenção eficiente. No entanto, as empresas devem estar preparadas para enfrentar desafios de integração e apoiar a curva de aprendizado dos desenvolvedores. Oferecer treinamento adequado e criar uma estratégia de integração robusta são medidas práticas recomendadas.

Limitações do Estudo

Embora este estudo tenha proporcionado insights significativos, ele possui algumas limitações. A amostra de participantes do survey B foi limitada ao contexto da Serasa, o que pode restringir a generalização dos resultados para outras organizações. Além disso, a coleta de dados foi baseada em um único projeto piloto, o que pode não capturar todas as variáveis relevantes associadas à implementação de micro-frontends. Finalmente, a dependência de auto-relatos dos participantes pode introduzir vieses subjetivos.

Trabalhos Futuros

Considerando os desafios identificados na integração e na gestão de estado em arquiteturas de micro-frontends, há diversas oportunidades para pesquisas futuras que possam aprofundar nosso entendimento e oferecer soluções mais eficazes. Uma direção promissora é a investigação de padrões de integração mais sofisticados. Por exemplo, estudos podem se concentrar em explorar a eficácia de gateways de integração ou contêineres centralizados para gerenciar a comunicação entre micro-frontends. Além disso, a análise comparativa dessas abordagens em termos de complexidade, escalabilidade e facilidade de manutenção pode oferecer insights valiosos para os desenvolvedores.

Capítulo 7

Conclusão

Neste estudo, exploramos estratégias fundamentais e diretrizes para enfrentar os desafios e maximizar os benefícios da adoção de micro-frontends. Ao longo desta pesquisa, aprofundamo-nos nas implicações práticas e teóricas dessa arquitetura emergente, considerando uma variedade de fontes de evidência, incluindo surveys, revisão sistemática da literatura (SLR) e estudos de caso. Inicialmente, examinamos as motivações por trás da adoção de micro-frontends e os principais desafios enfrentados pelas organizações durante esse processo. Identificamos uma série de benefícios potenciais, como aumento da modularidade, independência das equipes e melhoria na escalabilidade, bem como desafios, como a complexidade da gestão e impactos na experiência do usuário.

Em seguida, discutimos diretrizes para enfrentar esses desafios e maximizar os benefícios da adoção de micro-frontends. Isso incluiu considerações sobre arquitetura, gestão de dependências, comunicação entre micro-frontends e integração contínua. Além disso, enfatizamos a importância de aspectos como desempenho e escalabilidade ao planejar e implementar sistemas baseados em micro-frontends. Com base em nossos achados, consolidamos diretrizes práticas no GAM, visando fornecer orientação útil e prática para organizações que buscam implementar essa arquitetura.

Com base nos achados desta pesquisa e nas lacunas identificadas, uma área promissora que merece exploração envolve a realização de estudos mais aprofundados sobre aspectos específicos da arquitetura de micro-frontends. Isso inclui investigar estratégias de comunicação entre micro-frontends, padrões de integração e técnicas de gestão de estado, visando uma compreensão mais profunda de suas implicações práticas e possíveis melhorias.

Além disso, é importante expandir a pesquisa por meio de estudos empíricos que validem e aprofundem as conclusões obtidas neste trabalho. Coletar dados quantitativos e qualitativos em ambientes reais de desenvolvimento de software pode fornecer insights valiosos sobre o desempenho, escalabilidade e mantenibilidade das arquiteturas de micro-frontends em diferentes contextos. Outro aspecto relevante é a comparação de diferentes abordagens utilizadas em arquiteturas semelhantes, como microservices. Investigar e comparar as vantagens e desvantagens dessas estratégias pode fornecer orientações importantes para decisões específicas sobre a arquitetura de micro-frontends.

Referências

[1] Mezzalira, L.: *Building Micro-Frontends*. O'Reilly Media, 2021, ISBN 9781492082941. https://books.google.com.br/books?id=MjpPEAAAQBAJ. x, 2, 3, 4, 5, 15, 16, 17, 18, 23, 28, 29, 46, 58, 66, 69, 109

[2] Geers, M.: *Micro Frontends in Action*. Manning Publications, 2020, ISBN 9781617296871. https://books.google.com.br/books?id=FFD9DwAAQBAJ. x, 2, 3, 4, 5, 19, 20, 23, 29, 36, 50, 58, 59, 109

[3] Sommerville, I.: *Engenharia de Software*. Pearson Universidades, 2019, ISBN 9788543024974. https://books.google.com.br/books?id=0pwB0AEACAAJ. 1

[4] Bosch, Jan: *Speed, Data and Ecosystems: The Future of Software Engineering*. Em Jürjens, Jan e Kurt Schneider (editores): *Software Engineering 2017, Fachtagung des GI-Fachbereichs Softwaretechnik, 21.-24. Februar 2017, Hannover, Deutschland*, volume P-267 de *LNI*, páginas 23–24. GI, 2017. https://dl.gi.de/handle/20.500.12116/1282. 1

[5] Richardson, Chris: *Microservices Patterns: With examples in Java*. Manning Publications, 2018. 1, 14, 15

[6] Tilak, P Yedhu, Vaibhav Yadav, Shah Dhruv Dharmendra e Narasimha Bolloju: *A platform for enhancing application developer productivity using microservices and micro-frontends*. Em *2020 IEEE-HYDCON*, páginas 1–4, 2020. 1, 3

[7] Newman, Sam: *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. 1, 14, 15

[8] Yang, Caifang, Chuanchang Liu e Zhiyuan Su: *Research and Application of Micro Frontends*. IOP Conference Series: Materials Science and Engineering, 490(6):062082, apr 2019. https://dx.doi.org/10.1088/1757-899X/490/6/062082. 1, 2

[9] Development, IBM Market e Insights: *Microservices in the Enterprise, 2021: Real Benefits, Worth the Challenges - How Organizations Are Finding Speed, Agility, and Resiliency through Microservices*. IBM, 2021. 1

[10] Technology, ThoughtWorks: *Nossas ideias sobre tecnologias e tendências que estão moldando o futuro*. Relatório Técnico, ThoughtWorks, 2016. URLdodocumento,sedisponÃŋvel. 2, 37

[11] Taibi, Davide e Luca Mezzalira: *Micro-Frontends: Principles, Implementations, and Pitfalls*. ACM SIGSOFT Softw. Eng. Notes, 47(4):25–29, 2022. https://doi.org/10.1145/3561846.3561853. 2, 4, 5

[12] Simões, Bruno, María del Puy Carretero, Jorge Martínez Santiago, Sebastián Muñnz Segovia e Nieves Alcaín: *TwinArk: A Unified Framework for Digital Twins based on Micro-frontends, Micro-Services, and Web 3D*. Em Posada, Jorge, Aitor Moreno, Alberto Jaspe, Imanol Muñoz-Pandiella, Didier Stricker, Christophe Mouton, Ayat Mohammed e Ane Elizalde (editores): *The 28th International ACM Conference on 3D Web Technology, Web3D 2023, San Sebastian, Spain, October 9-11, 2023*, páginas 9:1–9:10. ACM, 2023. https://doi.org/10.1145/3611314.3615915. 2, 3

[13] Simeunovic, Andrej and Tripunovic, Uros: *Managing Micro Frontends Across Multiple Tech Stacks - Sharing, Finding & Publishing*, 2023. ISSN 1650-2884. Student Paper. 3

[14] Lewis, James e Martin Fowler: *Microservices*, 2014. https://www.martinfowler.com/articles/microservices.html, Acesso em 06 de janeiro de 2024. 3, 4, 14, 15

[15] Rappl, F. e L. Schöttner: *The Art of Micro Frontends: Build websites using compositional UIs that grow naturally as your application scales.* Packt Publishing, 2021, ISBN 9781800565609. https://books.google.com.br/books?id=lGI0EAAAQBAJ. 3, 4, 6, 17, 20

[16] Evans, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2004. 3, 17, 24, 70

[17] SPA, Single: *Single SPA - Getting Started Overview*, 2022. https://single-spa.js.org/docs/getting-started-overview, Acesso em: 10 Janeiro de 2024. 3, 4, 25

[18] *Serasa Experian*, acessado em 2023. https://www.serasaexperian.com.br/sobre-nos/o-que-fazemos/. 4

[19] Curtis, Nathan: *Defining Design Systems*, 2017. https://medium.com/eightshapes-llc/defining-design-systems-6dd4b03e0ff6, Acesso em 06 de janeiro de 2024. 5, 25

[20] *Web Components.* https://www.w3.org/standards/techs/components, Acesso em: 06 de janeiro de 2024. 5, 26

[21] Bourque, Pierre, Richard E. Fairley e IEEE Computer Society: *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* IEEE Computer Society Press, Washington, DC, USA, 3rd edição, 2014, ISBN 0769551661. 8, 58

[22] Fowler, Martin: *MonolithFirst*, 2018. https://martinfowler.com/bliki/MonolithFirst.html, Acesso em 06 de janeiro de 2024. 13

[23] Rousos, M., B. Wagner e C. Torre: *The API Gateway pattern versus the Direct client-to-microservice communication.* Microsoft Developer Division, .NET and Visual Studio product teams, 2021. https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern, [Accessed 04 Jan. 2024]. 27

[24] Newman, S.: *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, Incorporated, 2019, ISBN 9781492047841. https://books.google.com.br/books?id=iul3wQEACAAJ. 28

[25] Fowler, Martin: *Micro Frontends*, ano de publicação. https://martinfowler.com/articles/micro-frontends.html, [Accessed 04 Jan. 2024]. 28

[26] Chacon, Scott e Ben Straub: *Pro Git - Everything you need to know about Git.* Apress, 2nd edição, 2023. 30

[27] Hat, Red: *O que é ci/cd?* https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd. 30

[28] Mezzalira, Luca: *Orchestrating micro-frontends.* DAZN Engineering, abril 2019. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html. 31

[29] Barbara, Kitchenham e Stuart Charters: *Guidelines for performing systematic literature reviews in software engineering.* Keele University, UK, 9:1–65, 2007. 33, 36, 38

[30] Kitchenham, Barbara: *Procedures for performing systematic reviews.* Keele, UK, Keele University, 33(2004):1–26, 2004. 33, 56

[31] Wohlin, Claes, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell e Anders Wesslén: *Systematic Literature Reviews*, páginas 45–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, ISBN 978-3-642-29044-2. https://doi.org/10.1007/978-3-642-29044-2_4. 34

[32] Brereton, Pearl, Barbara A. Kitchenham, David Budgen, Mark Turner e Mohamed Khalil: *Lessons from applying the systematic literature review process within the software engineering domain.* J. Syst. Softw., 80(4):571–583, 2007. https://doi.org/10.1016/j.jss.2006.07.009. 35

[33] Costal, Dolors, Carles Farré, Xavier Franch e Carme Quer: *Inclusion and Exclusion Criteria in Software Engineering Tertiary Studies: A Systematic Mapping and Emerging Framework.* Em Lanubile, Filippo, Marcos Kalinowski e Maria Teresa Baldassarre (editores): *ESEM '21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, páginas 30:1–30:6, Bari, Italy, 2021. ACM. https://doi.org/10.1145/3475716.3484190. 35, 36

[34] Petersen, K., S. Vakkalanka e L. Kuzniarz: *Guidelines for conducting systematic mapping studies in software engineering: An update.* Information and Software Technology, 64:1–18, 2015. 37

[35] Peltonen, Severi, Luca Mezzalira e Davide Taibi: *Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review.* Inf. Softw. Technol., 136:106571, 2021. https://doi.org/10.1016/j.infsof.2021.106571. 40

[36] Pavlenko, Andrey, Nursultan Askarbekuly, Swati Megha e Manuel Mazzara: *Micro-frontends: application of microservices to web front-ends.* J. Internet Serv. Inf. Secur., 10(2):49–66, 2020. https://doi.org/10.22667/JISIS.2020.05.31.049. 40

[37] Mohammed, Sabah, Jinan Fiaidhi, Darien Sawyer e Mehdi Lamouchie: *Developing a GraphQL SOAP Conversational Micro Frontends for the Problem Oriented Medical Record (QL4POMR).* Em *Proceedings of the 6th International Conference on Medical and Health Informatics, ICMHI 2022, Virtual Event, Japan, May 13-15, 2022*, páginas 52–60. ACM, 2022. https://doi.org/10.1145/3545729.3545738. 40

[38] Abdelfattah, Amr S. e Tomás Cerný: *Filling The Gaps in Microservice Frontend Communication: Case for New Frontend Patterns.* Em Steen, Maarten van e Claus Pahl (editores): *Proceedings of the 13th International Conference on Cloud Computing and Services Science, CLOSER 2023, Prague, Czech Republic, April 26-28, 2023*, páginas 184–193. SCITEPRESS, 2023. https://doi.org/10.5220/0011812500003488. 40

[39] Yang, Caifang, Chuanchang Liu e Zhiyuan Su: *Research and Application of Micro Frontends.* IOP Conference Series: Materials Science and Engineering, 490(6):062082, apr 2019. https://dx.doi.org/10.1088/1757-899X/490/6/062082. 40

[40] Petcu, A., M. Frunzete e D.A. Stoichescu: *BENEFITS, CHALLENGES, AND PERFORMANCE ANALYSIS OF A SCALABLE WEB ARCHITECTURE BASED ON MICRO-FRONTENDS.* UPB Scientific Bulletin, Series C: Electrical Engineering and Computer Science, 85(3):319–334, 2023. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85169813523&partnerID=40&md5=f812d625e1144766be2156970c61590e, cited By 0. 40

[41] Shakil, Muddasir e Alois Zoitl: *Towards a Modular Architecture for Industrial HMIs.* Em *25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020, Vienna, Austria, September 8-11, 2020*, páginas 1267–1270. IEEE, 2020. https://doi.org/10.1109/ETFA46521.2020.9212011. 40

[42] Lorenz, Julius, Candy Lohse e Leon Urbas: *MicroFrontends as Opportunity for Process Orchestration Layer Architecture in Modular Process Plants.* Em *26th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2021, Vasteras, Sweden, September 7-10, 2021*, páginas 1–4. IEEE, 2021. https://doi.org/10.1109/ETFA45728.2021.9613474. 40

[43] Wanjala, Sylvester Timona: *A framework for implementing micro frontend architecture.* Int. J. Web Eng. Technol., 17(4):337–352, 2022. https://doi.org/10.1504/IJWET.2022.10054340. 40

[44] Nishizu, Yuma e Tetsuo Kamina: *Implementing Micro Frontends Using Signal-based Web Components.* J. Inf. Process., 30:505–512, 2022. https://doi.org/10.2197/ipsjjip.30.505. 40

[45] Pölöskei, István e Udo Bub: *Enterprise-level migration to micro frontends in a multi-vendor environment.* Acta Polytechnica Hungarica, 18(8):7 – 25, 2021. http://dx.doi.org/10.12700/APH.18.8.2021.8.1, Cited by: 4; All Open Access, Bronze Open Access. 40

[46] Stefanovska, Emilija e Vladimir Trajkovik: *Evaluating Micro Frontend Approaches for Code Reusability.* Em Zdravkova, Katerina e Lasko Basnarkov (editores): *ICT Innovations 2022. Reshaping the Future Towards a New Normal - 14th International Conference, ICT Innovations 2022,*

Skopje, Macedonia, September 29 - October 1, 2022, Proceedings, volume 1740 de *Communications in Computer and Information Science*, páginas 93–106. Springer, 2022. https://doi.org/10.1007/978-3-031-22792-9_8. 40

[47] Noppadol, Nattaporn e Yachai Limpiyakorn: *Application of Micro-frontends to Legal Search Engine Web Development*. Em Kim, Hyuncheol e Kuinam J. Kim (editores): *IT Convergence and Security*, páginas 165–173, Singapore, 2021. Springer Singapore, ISBN 978-981-16-4118-3. 40

[48] Bühler, Fabian, Johanna Barzen, Lukas Harzenetter, Frank Leymann e Philipp Wundrack: *Combining the Best of Two Worlds: Microservices and Micro Frontends as Basis for a New Plugin Architecture*. Em Barzen, Johanna, Frank Leymann e Schahram Dustdar (editores): *Service-Oriented Computing - 16th Symposium and Summer School, SummerSOC 2022, Hersonissos, Crete, Greece, July 3-9, 2022, Revised Selected Papers*, volume 1603 de *Communications in Computer and Information Science*, páginas 3–23. Springer, 2022. https://doi.org/10.1007/978-3-031-18304-1_1. 40

[49] Sorgalla, Jonas, Philip Wizenty, Florian Rademacher, Sabine Sachweh e Albert Zündorf: *Applying Model-Driven Engineering to Stimulate the Adoption of DevOps Processes in Small and Medium-Sized Development Organizations*. SN Comput. Sci., 2(6):459, 2021. https://doi.org/10.1007/s42979-021-00825-z. 40

[50] Tilak, P Yedhu, Vaibhav Yadav, Shah Dhruv Dharmendra e Narasimha Bolloju: *A platform for enhancing application developer productivity using microservices and micro-frontends*. Em *2020 IEEE-HYDCON*, páginas 1–4, 2020. 40

[51] Männistö, Tomi, Antti-Pekka Tuovinen e Mikko Raatikainen: *Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization*. Em *20th International Conference on Software Architecture, ICSA 2023 - Companion, L'Aquila, Italy, March 13-17, 2023*, páginas 61–67. IEEE, 2023. https://doi.org/10.1109/ICSA-C57050.2023.00025. 40

[52] Perlin, Rodrigo, Denilson Ebling, Vinícius Maran, Glenio Descovi e Alencar Machado: *An Approach to Follow Microservices Principles in Frontend*. Em *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*, páginas 1–6, 2023. 40

[53] Mena, Manel, Antonio Corral, Luis Iribarne e Javier Criado: *A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things*. IEEE Access, 7:104577–104590, 2019. 40

[54] Taibi, Davide e Luca Mezzalira: *Micro-Frontends: Principles, Implementations, and Pitfalls*. ACM SIGSOFT Softw. Eng. Notes, 47(4):25–29, 2022. https://doi.org/10.1145/3561846.3561853. 40

[55] Simões, Bruno, María del Puy Carretero, Jorge Martínez Santiago, Sebastián Muñnz Segovia e Nieves Alcaín: *TwinArk: A Unified Framework for Digital Twins based on Micro-frontends, Micro-Services, and Web 3D*. Em Posada, Jorge, Aitor Moreno, Alberto Jaspe, Imanol Muñoz-Pandiella, Didier Stricker, Christophe Mouton, Ayat Mohammed e Ane Elizalde (editores): *The 28th International ACM Conference on 3D Web Technology, Web3D 2023, San Sebastian, Spain, October 9-11, 2023*, páginas 9:1–9:10. ACM, 2023. https://doi.org/10.1145/3611314.3615915. 40

[56] Wang, Daojiang, Dongming Yang, Huan Zhou, Ye Wang, Daocheng Hong, Qiwen Dong e Shubing Song: *A Novel Application of Educational Management Information System based on Micro Frontends*. Em Cristani, Matteo, Carlos Toro, Cecilia Zanni-Merk, Robert J. Howlett e Lakhmi C. Jain (editores): *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES-2020, Virtual Event, 16-18 September 2020*, volume 176 de *Procedia Computer Science*, páginas 1567–1576. Elsevier, 2020. https://doi.org/10.1016/j.procs.2020.09.168. 40

[57] Zhang, C., D. Zhang, H. Zhou, X. Wang, L. Lv, R. Zhang, Y. Xie, H. Liu, Y. Li, D. Jia, Y. Huang e T. Jiang: *Application business information interaction bus based on micro frontend framework*. Em *2023 7th International Symposium on Computer Science and Intelligent Control (ISCSIC)*, páginas 306–310, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society. https://doi.ieeecomputersociety.org/10.1109/ISCSIC60498.2023.00070. 40

[58] Gashi, Elvisa, Dhuratë Hyseni, Isak Shabani e Betim Çiço: *The advantages of micro-frontend architecture for developing web application.* Em *2024 13th Mediterranean Conference on Embedded Computing (MECO)*, páginas 1–5, 2024. 40

[59] Kuhrmann, M., D. M. Fernández e M. Daneva: *On the pragmatic design of literature studies in software engineering: an experience-based guideline.* Empirical Software Engineering, 22:2852–2891, 2017. https://doi.org/10.1007/s10664-016-9492-y. 40

[60] Gamma, Erich, Richard Helm, Ralph Johnson e John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman Publishing Co., Inc., USA, 1995, ISBN 0201633612. 42

[61] Wohlin, Claes, Per Runeson, Martin Host, MC Ohlsson, Bjorn Regnell e Anders Wesslen: *Experimentation in Software Engineering.* Springer Science & Business Media, 2012. 51

[62] ISO/IEC 25010: *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.* ISO, 2011. 77

[63] Yin, Robert K.: *Case Study Research and Applications: Design and Methods.* SAGE Publications, 2018. 90

[64] Friesen, M.E., Sloan School of Management Center Fo e W.J. Orlikowski: *Assimilating CASE Tools in Organizations: An Empirical Study of the Process and Context of CASE Tools.* Creative Media Partners, LLC, 2023, ISBN 9781019951088. https://books.google.com.br/books?id=QRxW0AEACAAJ. 109

Apêndice A

Survey A

TÍTULO: EXPERIÊNCIA E AVALIAÇÃO NA ADOÇÃO E IMPLEMENTAÇÃO DA CAMADA FRONTEND

Obrigado por participar do nosso estudo sobre impactos e desafios na adoção da arquitetura de micro-frontends. O objetivo deste estudo é investigar os impactos e desafios enfrentados pelas companhias e equipes de desenvolvimento quando surge a necessidade de adotar a abordagem Micro-frontend. A pesquisa levará cerca de 15 minutos. Ao prosseguir, você concorda com essas condições:

DECLARAÇÃO DE CONSENTIMENTO INFORMADO

Ao responder esta pesquisa você permite que os pesquisadores obtenham, usem e divulguem as informações anônimas fornecidas conforme descrito abaixo:

CONDIÇÕES E ESTIPULAÇÕES

1. Entendo que todas as informações fornecidas são confidenciais, e minha identidade não será revelada. Concordo em preencher a pesquisa para fins de estudo, e compreendo que os dados resultantes desta pesquisa anônima podem ser divulgados em periódicos, conferências e postagens em blogs.
2. Reconheço que minha participação nesta pesquisa é totalmente voluntária, e a recusa em participar não acarretará penalidades ou perda de benefícios. Posso retirar minha participação a qualquer momento, e compreendo que, ao decidir participar, tenho o direito de recusar responder qualquer pergunta que me deixe desconfortável.
3. Estou ciente de que posso entrar em contato com os pesquisadores caso tenha dúvidas sobre a pesquisa. Entendo que meu consentimento não gera benefícios diretos para mim, e reconheço que o autor manterá os dados coletados em perpetuidade, podendo utiliza-los em trabalhos acadêmicos futuros.
4. Ao concordar em participar, forneço livremente meu consentimento e reconheço meus direitos como participante voluntário da pesquisa, conforme descrito anteriormente. Autorizo os pesquisadores a utilizarem minhas informações na condução de pesquisas nas áreas mencionadas acima.

Sobre sua Experiência e Papeis na sua Organização:

- **Qual seu nível educacional?**

  [ ] Ensino médio - completo

[ ] Graduação - cursando

[ ] Graduação - completa

[ ] Pós-graduação(especialização) -cursando

[ ] Pós-graduação(especialização) - completa

[ ] Mestrado - cursando

[ ] Mestrado - cursando

[ ] Mestrado - completo

[ ] Doutorado - cursando

[ ] Doutorado - completo

- **Qual a natureza do seu trabalho e/ou organização?**

  [ ] Administração pública federal

  [ ] Administração pública estadual ou municipal

  [ ] Empresa pública

  [ ] Empresa privada

  [ ] Entidade sem fins lucrativos

  [ ] Microempreendedor individual (MEI) / Autônomo

  [ ] Projeto de pesquisa

- **Há quanto tempo trabalha na área de tecnologia/engenharia de software?**

  [ ] Menos de 1 ano

  [ ] De 1 a 3 anos

  [ ] De 4 a 6 anos

  [ ] De 7 a 9 anos

  [ ] Mais de 10 anos

- **Qual é o seu cargo/função atual na sua Organização Serasa Experian?**

  [ ] Desenvolvedor(a) de Software

  [ ] Líder Técnico

  [ ] Arquiteto(a) de Software

  [ ] Gerente de Projetos

- **Como você descreveria o seu nível de atuação no desenvolvimento frontend em sua organização?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

- **Como você classificaria seu nível de atuação nas decisões arquiteturais do frontend em sua organização?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

- **Com relação ao número de pessoas que trabalham no seu time, pode-se afirmar que há:**

  [ ] De 1 a 3 pessoas

  [ ] De 3 a 5 pessoas

  [ ] De 5 a 10 pessoas

  [ ] Mais de 10 pessoas

- **Qual o seu nível de conhecimento sobre a arquitetura monolítica?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

- **Qual seu nível de conhecimento sobre a arquitetura de micro-frontend?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

- **Indique quais linguagens de programação você possui experiência?**

  [ ] Javascript

  [ ] Python

  [ ] Java

  [ ] Typescript

  [ ] C#

  [ ] C++

  [ ] C

  [ ] PHP

  [ ] Outro: _____

- **Indique quais frameworks com os quais você possui experiência?**

  [ ] Angular

  [ ] React

  [ ] Vue

  [ ] Spring Boot

  [ ] Outro: _____

- **Qual sua percepção sobre soluções da arquitetura de micro-frontends?**

  [ ] Melhor escalabilidade

  [ ] Facilita a manutenção e implementação de novas funcionalidades

  [ ] Maior flexibilidade na escolha de tecnologias

  [ ] Não tenho conhecimento suficiente para opinar

- **Sobre tamanho do projeto onde trabalho, você pode afirmar que:**

  [ ] É um projeto pequeno, com apenas uma equipe de desenvolvimento

  [ ] É um projeto médio, com até 5 equipes trabalhando em partes diferentes

  [ ] É um projeto grande, com mais de 5 equipes trabalhando em partes diferentes

  [ ] Não tenho conhecimento suficiente para opinar

- **Já teve a oportunidade de participação em estudos de viabilidade para a adoção de alguma tecnologia**

  [ ] Sim, participei ativamente

  [ ] Sim, mas em um papel observador

  [ ] Não participei, mas tenho interesse

  [ ] Não tenho experiência nesse tipo de atividade

- **Como avalia sua experiência com Web Components?**

  [ ] Positiva, trouxe modularidade e reusabilidade à aplicação

  [ ] Neutra, não percebi grande impacto

  [ ] Negativa, enfrentei desafios significativos

  [ ] Não tenho experiência com Web Components

- **Como você avalia sua experiência com Frameworks JavaScript para Micro-frontend?**

  [ ] Preferência por um framework específico

  [ ] Utilizei mais de um framework, conforme necessidade

  [ ] Sem preferência, adotei conforme equipe

  [ ] Não tenho experiência com micro-frontend

- **Como você avalia a sua experiência com a integração de APIs e Microserviços no contexto de micro-frontends?**

[ ] Facilitou a integração e comunicação entre os componentes

[ ] Apresentou desafios, mas foram superados

[ ] Enfrentei dificuldades significativas na integração

[ ] Não tenho experiência com integração de APIs em micro-frontends

- **Alguma consideração sobre alguma coisa não foi abordada neste questionário?**

_____

_____

_____

_____

Apêndice B

Survey B

Obrigado por participar do nosso estudo sobre impactos e desafios na adoção da arquitetura de micro-frontends. O objetivo deste estudo é investigar os impactos e desafios enfrentados pelas companhias e equipes de desenvolvimento quando surge a necessidade de adotar a abordagem Micro-frontend. A pesquisa levará cerca de 15 minutos. Ao prosseguir, você concorda com essas condições:

DECLARAÇÃO DE CONSENTIMENTO INFORMADO

Ao responder esta pesquisa você permite que os pesquisadores obtenham, usem e divulguem as informações anônimas fornecidas conforme descrito abaixo:

CONDIÇÕES E ESTIPULAÇÕES

1. Entendo que todas as informações fornecidas são confidenciais, e minha identidade não será revelada. Concordo em preencher a pesquisa para fins de estudo, e compreendo que os dados resultantes desta pesquisa anônima podem ser divulgados em periódicos, conferências e postagens em blogs.
2. Reconheço que minha participação nesta pesquisa é totalmente voluntária, e a recusa em participar não acarretará penalidades ou perda de benefícios. Posso retirar minha participação a qualquer momento, e compreendo que, ao decidir participar, tenho o direito de recusar responder qualquer pergunta que me deixe desconfortável.
3. Estou ciente de que posso entrar em contato com os pesquisadores caso tenha dúvidas sobre a pesquisa. Entendo que meu consentimento não gera benefícios diretos para mim, e reconheço que o autor manterá os dados coletados em perpetuidade, podendo utiliza-los em trabalhos acadêmicos futuros.
4. Ao concordar em participar, forneço livremente meu consentimento e reconheço meus direitos como participante voluntário da pesquisa, conforme descrito anteriormente. Autorizo os pesquisadores a utilizarem minhas informações na condução de pesquisas nas áreas mencionadas acima.

Sobre sua Experiência e Papeis na Organização Serasa Experian:

- **Qual seu nível educacional?**

  [ ] Ensino médio - completo

  [ ] Graduação - cursando

  [ ] Graduação - completa

[ ] Pós-graduação(especialização) -cursando

[ ] Pós-graduação(especialização) - completa

[ ] Mestrado - cursando

[ ] Mestrado - cursando

[ ] Mestrado - completo

[ ] Doutorado - cursando

[ ] Doutorado - completo

- **Há quanto tempo trabalha na área de tecnologia/engenharia de software?**

  [ ] Menos de 1 ano

  [ ] De 1 a 3 anos

  [ ] De 4 a 6 anos

  [ ] De 7 a 9 anos

  [ ] Mais de 10 anos

- **Qual é o seu cargo/função atual na sua Organização Serasa Experian?**

  [ ] Desenvolvedor(a) de Software

  [ ] Líder Técnico

  [ ] Arquiteto(a) de Software

  [ ] Product Owner (PO)

  [ ] Outro: _____

- **Há quanto tempo trabalha na Organização Serasa Experian?**

  [ ] Menos de 1 ano

  [ ] De 1 a 3 anos

  [ ] De 4 a 6 anos

  [ ] De 7 a 9 anos

  [ ] Mais de 10 anos

- **Quais Linguagens de Programação você tem experiência?**

  [ ] Javascript

  [ ] Java

  [ ] Typescript

  [ ] Outro: _____

- **Quais dos seguintes frameworks possui experiência?**

  [ ] Angular

  [ ] React

  [ ] Vue

[ ] Spring Boot

[ ] Outro: _____

- **Qual é o seu nível de envolvimento com as decisões arquiteturais do frontend na Organização Serasa Experian?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

- **Como você descreveria seu nível de familiaridade com a arquitetura de micro-frontends?**

  [ ] Nenhum

  [ ] Baixo

  [ ] Moderado

  [ ] Alto

  [ ] Muito Alto

Sobre os projetos em que atua ou atuou na Organização Serasa Experian

- **Sobre o tamanho do projeto em que atua, considerando sua complexidade e escopo em termos de funcionalidades e requisitos, pode-se afirmar que:**

  [ ] Pequeno - Poucas funcionalidades e requisitos, baixa complexidade

  [ ] Médio - Um número moderado de funcionalidades e requisitos, complexidade intermediária

  [ ] Grande - Muitas funcionalidades e requisitos, alta complexidade

- **Com relação ao número de pessoas que trabalham no seu time:**

  [ ] De 1 a 3 pessoas

  [ ] De 4 a 8 pessoas

  [ ] De 9 a 15 pessoas

  [ ] Mais de 15 pessoas

Sobre a adoção/implementação da arquitetura micro-frontend no Serasa Experian

- **Como você descreveria o nível das expectativas sobre o micro-frontend antes da adoção e implementação da arquitetura?**

  [ ] Alta

  [ ] Moderada

  [ ] Baixa

- **Qual o nível de importância das Provas de Conceitos (POCs) ou Projetos Pilotos na adoção do micro-frontend antes da adotar a arquitetura?**

  [ ] Alta

  [ ] Moderada

  [ ] Baixa

- **Indique quais os desafios foram identificados durante adoção e implementação da arquitetura micro-frontend nos projetos que você atuou:**

  [ ] Desafios na integração de diferentes tecnologias e frameworks utilizados em cada micro-frontend

  [ ] Problemas de desempenho devido à necessidade de carregamento excessivo de recursos no cliente

  [ ] Complexidade na definição de limites de responsabilidade entre os micro-frontends, levando a conflitos de escopo e funcionalidades

  [ ] Obstáculos relacionados ao roteamento dinâmico e navegação entre os micro-frontends

  [ ] Outro: _____

- **Sobre os impactos positivos da adoção da arquitetura micro-frontend, pode-se afirmar que:**

  [ ] Houve uma melhoria significativa na modularidade e na capacidade de escalabilidade dos sistemas

  [ ] Observou-se uma redução notável nos tempos de carregamento de página e melhoria na experiência do usuário

  [ ] A colaboração entre equipes de desenvolvimento foi facilitada, resultando em maior eficiência e produtividade

  [ ] A manutenção do código foi simplificada devido à clara separação de responsabilidades entre os micro-frontends

  [ ] Outro: _____

- **Alguma questão não abordada neste survey que gostaria de comentar a respeito?**

  _____

  _____

  _____

  _____