UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MARCELO COLLA

APLICAÇÃO DO ESTILO ARQUITETURAL MICRO FRONTEND: UM ESTUDO DE CASO

DOIS VIZINHOS

2022

MARCELO COLLA

APLICAÇÃO DO ESTILO ARQUITETURAL MICRO FRONTEND: UM ESTUDO DE CASO

Applying the Micro Frontend architectural pattern: A case study

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Gustavo Jansen de Souza Santos

DOIS VIZINHOS

2022

MARCELO COLLA

APLICAÇÃO DO ESTILO ARQUITETURAL MICRO FRONTEND: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Software do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 05/dezembro/2022

Gustavo Jansen de Souza Santos
doutorado
Universidade Tecnológica Federal do Paraná

André Roberto Ortoncelli
doutorado
Universidade Tecnológica Federal do Paraná

Francisco Carlos Monteiro Souza
doutorado
Universidade Tecnológica Federal do Paraná

DOIS VIZINHOS
2022

AGRADECIMENTOS

Sou grato a todos os professores que me influenciaram na minha trajetória, em especial, ao meu coordenador, o Prof. Dr. Gustavo Jansen de Souza Santos por ter aceitado me acompanhar neste projeto. O seu empenho foi essencial para a minha motivação à medida que as dificuldades iam surgindo ao longo do percurso.

Dedico este projeto de pesquisa à minha esposa Indiamara, cuja sua presença foi essencial para a conclusão deste trabalho. Grato pela sua compreensão durante as minhas horas de ausência. Te amo.

Dedico este projeto de pesquisa aos meus pais, vocês me ensinaram como se reerguer diante das adversidades da vida. Agradeço pelos seus ensinamentos e exemplos para ser humano íntegro e ético.

RESUMO

Nos últimos anos, o crescimento do desenvolvimento de software tem sido constante, em um contexto de mudanças frequentes, novas regras de negócio e comportamentos estão gerando aplicações cada vez maiores. Em sintonia com o desenvolvimento acelerado muitos padrões arquiteturais têm surgido, este trabalho explora o conceito de *Micro Frontend*. O objetivo central do trabalho é abordar e analisar a utilização desse estilo arquitetural no desenvolvimento de aplicações modernas, bem como identificar os impactos causados ao utilizar *Micro Frontend*. O trabalho inicia estabelecendo uma visão geral do desenvolvimento Web e as abordagens arquitetônicas mais utilizadas, além disso são evidenciadas as características e evoluções dessas arquiteturas. Em seguida são apresentadas as pesquisas relacionadas aos *Micro Frontends*, onde os experimentos e estudos de caso trazem reflexões importantes dessa nova arquitetura. Sob essa ótica, um estudo de caso com um projeto real construído pela Fábrica de Software da UTFPR. No decorrer do estudo de caso é demonstrado em detalhes os processos da construção utilizado-se de *Micro Frontends*. Ao aplicar as divisões por responsabilidade de negócio na construção do sistema é esperado que o resultado seja aplicações pequenas e modulares. Para avaliar a arquitetura usada neste estudo, as métricas de manutenibilidade são aplicadas para identificar se houve melhorias no estudo de caso. Foi concluído que ao utilizar *Micro Frontend* as métricas de manutenibilidade possuem médias e variâncias iguais ou aproximadas.

Palavras-chave: desenvolvimento frontend; arquitetura de software; micro frontend.

ABSTRACT

In recent years, there has been a constant growth software development, in a context of frequent changes, new business rules that are generating even larger applications. Several architectural patterns have emerged recently as well, and this work explores the concept of *Micro Frontend*. The main goal of the work is to analyse the use of this architectural pattern in the development of modern applications, and to identify the impacts caused by using *Micro Frontend*. We conducted a narrative review of the state of the art, in which we found a few studies concerning the usage, benefits and challenges of *Micro Frontends*. From this perspective, a case study with a real project built by the UTFPR Software Factory. During the case study, the processes of construction using *Micro Frontends* are demonstrated in detail. By applying the divisions by business responsibility in the construction of the system it is expected that the result will be small and modular applications. To evaluate the architecture used in this study, maintainability metrics are applied to identify if there were improvements in the case study. It was concluded that when using *Micro Frontend* the maintainability metrics have equal or approximate means and variances.

Keywords: frontend development; software architecture; micro frontend.

LISTA DE FIGURAS

Figura 1 – Arquitetura de três camadas. **13**
Figura 2 – Exemplo de uma arquitetura Cliente-Servidor. **14**
Figura 3 – Exemplo de uma arquitetura de página única. **16**
Figura 4 – Exemplo de uma arquitetura de microsserviços. **17**
Figura 5 – Exemplo de comunicação entre microsserviços utilizando mensageria. . **18**
Figura 6 – Implantação de um sistema em *Micro Frontend* **20**
Figura 7 – Integração *Micro Frontend* com Iframe's. **22**
Figura 8 – Aplicação de *Micro Frontend* em tempo de execução. **23**
Figura 9 – Composição de vários *Micro Frontend* na mesma estrutura da página. . **24**
Figura 10 – Casos de uso das funcionalidade do sistema-alvo. **33**
Figura 11 – Mapeamento da arquitetura do sistema alvo **35**
Figura 12 – Arquitetura proposta para construir os *Micro Frontend* **36**
Figura 13 – Configuração do Webpack para um projeto utilizando React **38**
Figura 14 – Configurações essenciais para os módulos federados **39**
Figura 15 – Configuração do módulo para um subdomínio de negócio **40**
Figura 16 – Configurar arquivo package.json para pacotes NPM **41**
Figura 17 – Configurar arquivo .npmcr para instalar pacotes **42**
Figura 18 – Fluxo de trabalho para publicar uma nova versão do pacote **42**
Figura 19 – Componentes que foram movidos para biblioteca **43**
Figura 20 – Configuração de pipeline executada a cada modificação **44**
Figura 21 – Organização para *Micro Frontend* de início rápido **46**
Figura 22 – Organização para *Micro Frontend* de dados **48**
Figura 23 – Acessar os dados do usuário usando *Micro Frontend* de dados **49**
Figura 24 – Estrutura aplicada ao *Micro Frontend* de perfil **50**
Figura 25 – Telas para aprovação de solicitação. **51**
Figura 26 – Interface para registro de acesso ao campus. **52**
Figura 27 – Processo de login usando *Micro Frontend* de dados **54**
Figura 28 – Registro de *Micro Frontends* remotos **54**
Figura 29 – Exemplo de configuração ferramenta *Plato* **60**
Figura 30 – Resultado do teste estatístico da métrica de Complexidade Ciclomática **64**

Figura 31 – Resultado do teste estatístico da métrica de Dificuldade Halstead 65
Figura 32 – Resultado do teste estatístico da métrica de Tempo 66
Figura 33 – Resultado do teste estatístico da métrica *LLOC* 67

LISTA DE TABELAS

Tabela 1 – Funcionalidades do sistema desenvolvido no projeto Fábrica de Software. 34

Tabela 2 – Análise comparativa do MI usando ferramenta *Plato* 63

SUMÁRIO

1 INTRODUÇÃO 10

2 ASPECTOS CONCEITUAIS 11

2.1 HTML Estático 11

2.2 Aplicações Monolíticas 12

2.3 Aplicações Cliente-Servidor 14

2.4 Aplicações de página única (SPA) 15

2.5 Desenvolvimento em Microsserviços 16

2.6 Desenvolvimento em *Micro Frontends* 19

2.6.1 Integração em tempo de execução com Iframe 22

2.6.2 Integração em tempo de execução usando CDN's 23

3 REVISÃO NARRATIVA DA LITERATURA 25

3.1 Motivações, benefícios e problemas de Micro Frontends 25

3.1.1 Problema apresentado 25

3.1.2 Procedimento utilizado 25

3.1.3 Resultados Observados 26

3.1.4 Como se aplica ao trabalho proposto 27

3.2 Aplicação de microsserviços para front-ends da Web 27

3.2.1 Problema apresentado 27

3.2.2 Procedimento utilizado 28

3.2.3 Resultados Observados 29

3.2.4 Como se aplica ao trabalho proposto 29

3.3 Um estudo exploratório de Micro Frontends 29

3.4 Considerações Finais 31

4 ESTUDO DE CASO 32

4.1 Sistema alvo utilizando SPA 32

4.2 Considerações sobre a arquitetura utilizando SPA 33

4.3 Arquitetura de *Micro Frontend* proposta 35

4.3.1 Webpack 37

4.3.2 Integração de módulos com Webpack 38

4.3.3 Registro de Pacotes do Gitlab 41

4.3.4 Integração Contínua com Netlify 44
4.4 *Micro Frontend* para início rápido 45
4.5 *Micro Frontend* de dados compartilhados 47
4.6 *Micro Frontend* de perfil 49
4.7 *Micro Frontend* de solicitações 50
4.8 *Micro Frontend* de liberações 52
4.9 *Micro Frontend* de visitas 53
4.10 *Micro Frontend* principal 53
5 AVALIAÇÃO DA ARQUITETURA PROPOSTA 56
5.1 Planejamento 56
5.1.1 Complexidade Ciclomática 57
5.1.2 Dificuldade Halstead 58
5.1.3 Tempo Halstead 58
5.1.4 Ferramenta Selecionada 59
5.2 Validação dos resultados 60
5.2.1 Índice de Manutenibilidade (MI) 63
5.2.2 Complexidade Ciclomática (R_1) 63
5.2.3 Dificuldade Halstead (R_2) 64
5.2.4 Tempo Halstead (R_3) 65
5.2.5 Linhas de Código-Fonte para Processamento Lógico (R_4) 67
5.3 Discussões 68
6 CONCLUSÕES 69
REFERÊNCIAS 70

1 INTRODUÇÃO

Nas últimas décadas, o desenvolvimento de software Web vem se tornando um padrão. O setor de tecnologia tem testemunhado mudanças rápidas em sistemas distribuídos e padrões arquiteturais. Aos poucos a instalação de software em computadores, está sendo modificada por sistemas que são acessados através de uma página Web. A maioria das soluções de software são divididas em três principais camadas: apresentação, regra de negócio e dados. As principais tecnologias utilizadas no *Frontend* (apresentação) desses sistemas são o HTML, CSS e Javascript. No *Backend* são utilizadas diferentes arquiteturas, entre as mais comuns são aplicações monolíticas e microsserviços.

Considerando um mercado cada vez mais emergente, o tempo necessário para a criação e manutenção dos projetos de software tem ficado menor. Com o passar dos dias, surgem novas necessidades, as mudanças se tornam mais frequentes e a velocidade com que novas funcionalidades são construídas segue esse ritmo. Porém, como as aplicações Web estão ficando cada vez maiores, os projetos foram se tornando difíceis de manter e modificar, tornando-se um grande problema.

No lado do servidor, a arquitetura de microsserviços se tornaram populares. Esta arquitetura traz consigo os benefícios de dividir grandes aplicações em pequenos serviços independentes que podem ser implantados separadamente. No lado do cliente, é comum observar grandes aplicações monolíticas em apenas uma base de código. Porém, em novembro de 2016, Cam Jackson (JACKSON, 2019) começou a discutir sobre um novo padrão arquitetural para decompor aplicações monolíticas de *Frontend* em partes menores. Essa técnica está sendo chamada de *Micro Frontends*.

Este trabalho propõe um estudo de caso para aplicação da arquitetura *Micro Frontends* no desenvolvimento *Frontend* de um sistema da Web. Assim sendo, o objetivo deste trabalho consiste utilizar um sistema já desenvolvido pela universidade e decompor ele em aplicações menores usando arquitetura de *Micro Frontend*. No Capítulo 2 são exploradas as evoluções da arquitetura no decorrer do tempo, e em seguida no Capítulo 3 são expostos os princípios e benefícios ao utilizar o conceito de *Micro Frontend*, por meio de uma revisão narrativa da literatura.

Durante a etapa prática do estudo de caso proposto, será construído um sistema-alvo utilizando o conceito de *Micro Frontend* para verificar como esse estilo pode ser utilizado e como ele afeta o software produzido. Os detalhes da metodologia e o sistema-alvo serão apresentados no Capítulo 4. Por fim, procura-se analisar se a utilização de *Micro Frontend* é adequada para construir aplicações menores; seguindo os princípios propostos pelos *Micro Frontends*: agnóstico de tecnologia, funcionalidades autônomas, testabilidade e modificabilidade. Nesse sentido, o Capítulo 5 apresenta os resultados das avaliações usando as métricas de manutenibilidade.

2 ASPECTOS CONCEITUAIS

Criar aplicações web é um desafio que envolve diversas atividades como permitir que diversos dispositivos, ferramentas e plataformas possam estar conectadas. Nos últimos anos, o setor de tecnologia tem testemunhado uma rápida mudança da arquitetura e práticas de sistemas distribuidos, o que tem levado os gigantes da indústria (como Netflix, Twitter, Amazon, eBay e Uber) a construir aplicações usando arquiteturas mais recentes (FOWLER, 2017).

O desenvolvimento web é baseado em padrões arquiteturais[1], e cada proposta de arquitetura visa facilitar o processo de criação do software. Consequentemente, com a diversidade de problemáticas encontradas na indústria de software, diferentes arquiteturas têm surgido com foco em resolver problemas frequentes. Os padrões arquiteturais mais utilizados ao longo dos últimos anos tem sido *Cliente-Servidor*, *Modelo-Visão-Controle* (MVC), Microsserviços e *Micro Frontends*. Deste modo, este capitulo está direcionado a apresentar um entendimento comum entre os diferentes padrões de arqueteturais utilizados atualmente.

2.1 HTML Estático

Os primeiros sites construidos foram desenvolvidos de forma monolitica[2], por exemplo, com HTML (*HyperText Markup Language*) estático e, a partir do aumento da complexidade das aplicações web foram surgindo outras alterantivas.

O HTML é uma linguagem de marcação utilizada para desenvolvimento de sites, criada pelo britânico Tim Berners-Lee em meados de 1990 (SILVA, 2019). Ela surgiu do desejo de criar a comunicação e a disseminação das pesquisas entre ele e seu grupo de colegas. O HTML foi inspirado na linguagem SGML (*Standard Generalized Markup Language*, ou linguagem de marcação generalizada padrão), tanto que herdou algumas *tags* importantes desta. A grande diferença proposta pelo o HTML foi a capacidade de ligar uma página à outra por meio de *hiperlinks* (ou simplesmente *link*).

As primeiras versões do HTML foram definidas com regras de sintaxe flexíveis, tornando-as úteis para os novatos na publicação na web (SILVA, 2019). Com o passar do tempo, o uso de ferramentas para criação de HTML aumentou e a sintaxe tendeu a se tornar cada vez mais rigorosa. Atualmente a W3C (World Wide Web Consortium) é a principal organização de padronização responsável por define as diretrizes da web.

A web que conhecemos hoje, não existe sem HTML, ao retirar o CSS (*Cascading Style Sheets*) e JavaScript ainda assim ficará o conteúdo e controles brutos, ou seja, o HTML é base para criar qualquer coisa na web (GODBOLT, 2016). Para isso as composições de textos, ima-

[1] Um padrão arquitetural expressa um esquema de organização estrutural fundamental para sistemas de software (BUSCHMANN, 2006)

[2] É um sistema que todo o código fonte é implatado com um único processo

gens, links, formulários e botões são usados para esse fim. Esses são elementos que estão presentes desde a concepção do HTML até as versões mais recentes.

Um site estático exibe sempre o mesmo conteúdo para todos os usuários, e geralmente esses projetos são desenvolvidos utilizando apenas HTML e CSS. Porém, sites estáticos não precisam necessariamente apresentar somente textos sem formatação, podendo conter vários elementos além de multimídia e vídeos. O site em si pode ter um desenho bem desenvolvido, mas o código fonte da página não muda: mesmo com as interações do usuário, o resultado mostrado em tela sempre será o mesmo.

Quando é demandada alguma alteração no site, será necessário um profissional com conhecimento na linguagem para realizar as modificações no código fonte. Ao utilizar esse formato, as páginas estáticas são, em geral, mais seguras e mais rápidas que páginas dinâmicas (JUNIOR, 2018). Nesse sentido, sites estáticos são ideais para empresas que desejam apenas realizar a apresentação/validação dos produtos ou serviços.

Nos últimos anos, muitos desenvolvedores estão usando os sites estáticos como uma opção para lidar com os novos desafios de garantir a segurança, confiabilidade e tempo de carregamento para os seus clientes e usuários (CUNHA, 2021). A seguir, a Seção de aplicações monolíticas apresenta as três camadas aplicadas a essa arquitetura.

2.2 Aplicações Monolíticas

Uma arquitetura monolítica é um sistema único, que é executado em um único processo, uma aplicação de software em que diferentes componentes estão ligados a um único programa dentro de uma única plataforma (PENNA, 2020). Diferente das aplicações de HTML estático na aplicação monolítica, o código em HTML é dinamico. Facilitando que lógicas de renderizações possam ser incluidas, resultando em páginas diferentes. Um exemplo é a página principal de uma website, ao ser acessasado sem realizar login, pode ter o seu conteúdo totalmente diferente para o mesmo usuário. Desta forma é possivel apresentar informações distintas aos usuários, seguindo as regras de cada projeto.

A evolução no desenvolvimento tem contribuído com o surgimento de padrões arquiteturais que auxiliam na construção de soluções de software mais robusto. Atualmente, quase todas as aplicações web podem ser divididas em três elementos distintos: um lado *Frontend* (ou lado do cliente), um lado de *Backend* e algum tipo de armazenamento de dados (FOWLER, 2017).

Assim como apresentado na Figura 1, a maioria das aplicações dispõe os dois primeiros elementos em uma base de código, onde são armazenados e executados ambos *Frontend* e *Backend* como um executável, com um banco de dados separado (FOWLER, 2017). Outras aplicações separam todo o código *Frontend* do código *Backend* e os armazena como executáveis lógicos separados (FOWLER, 2017). Independente da divisão ou da combinação, uma aplicação monolítica é composta por os esses três elementos.

Figura 1 – Arquitetura de três camadas.

Fonte: Autoria própria..

A divisão da arquitetura nas três camadas, mantem as diversas funcionalidades sendo executadas na mesma máquina e compartilham entre si os recursos de processamento, memória e banco de dados. Devido este compartilhamento, quando é necessário aumentar os recursos de uma parte da aplicação, só é possível ao escalar a arquitetura do projeto inteiro, gerando mais custos para manter o projeto. Essa arquitetura é viável ter várias instâncias desse processo por questões de escala, mas basicamente, todo o código estará contido em um único processo (NEWNAB, 2020).

Uma arquitetura monolítica pode ser considerada um boa escolha quando aplicada para validação um produto ou negócio utilizando uma prova de conceito (*Proof of Concept*, ou PoC) ou uma primeira versão (*Minimum Viable Product*, ou MVP). Os sistemas monolíticos podem simplificar a reutilização de código dentro do próprio sistema monolítico, pois basta fazer referência ao arquivo a ser usado (NEWNAB, 2020). Como o sistema inteiro está em uma única base de código, seu desenvolvimento é mais ágil, se comparado com outras arquiteturas.

Um sistema monolítico, seja um sistema com um único processo ou um sistema distribuído, é mais vulnerável aos perigos causados pelo acoplamento (NEWNAB, 2020). O acoplamento está relacionado a uma alteração sendo feita em um local exigindo mudanças em outro. À medida que o sistema cresce e novas pessoas estão trabalhando na mesma base de código, elas começaram a atrapalhar uma as outras (NEWNAB, 2020). Quando diferentes desenvolvedores alteram a mesma parte do código, haverá confusão sobre quem é dono de quê, e quem toma as decisões (NEWNAB, 2020). Em contraste com essa ideia, é fundamental uma sincronia entre as equipes, seguindo boas práticas de desenvolvimento e mantendo um baixo acoplamento entre as funcionalidades.

Uma boa escolha ao desenvolver um sistema utilizando a arquitetura monolítica é manter um baixo acoplamento e aplicar os princípios de responsabilidade única (FOWLER, 2017). Desta forma, o processo para refatorar e alterar responsabilidades serão mais compreensíveis quando se tem mais conhecimento das próximas funcionalidades e serviços do produto final (BERNARDES, 2021).

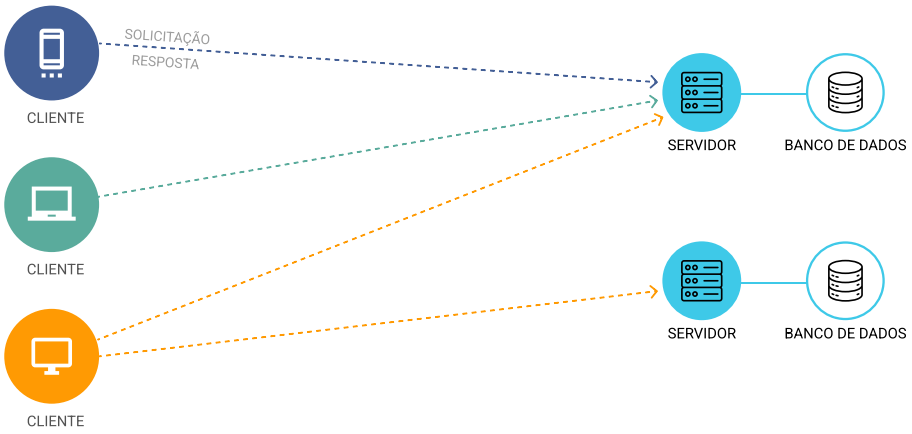
Os sistemas ainda são desenvolvidos como aplicações monolíticas, mas sua concepção está evoluindo para um enfoque centrado em serviços que se comunicam usando a infraestrutura da internet, em um cenário em que não importa a localização dos serviços e nem que aplicações são usadas para manipulá-los (LUNA, 2007).

2.3 Aplicações Cliente-Servidor

Uma arquitetura com o modelo cliente-servidor é uma estrutura distribuída que divide as execuções das tarefas entre o servidor e o cliente. Se comparado com aplicações monoliticas ela seria dividida em duas partes, sendo elas aplicações de *Frontend* (sendo o cliente) e *Backend*.

A comunicação entre o cliente e servidor se dá por meio de uma rede de computadores utilizando protocolo HTTP (Protocolo de Transferência de Hipertexto ou *Hypertext Transfer Protocol*). Um servidor pode se conectar com outros servidores e compartilhar um ou mais serviços com os clientes. Já o cliente se comunica com os servidores para solicitar ou enviar informações a serem gravadas. Ao receber uma solicitação, o servidor pode aceitar esses pedidos, processá-los e retornar o artefato para o cliente, como apresentado na Figura 2. Na coluna esquerda da figura os clientes estão se comunicando com os servidores localizados no lado direito da figura.

Figura 2 – Exemplo de uma arquitetura Cliente-Servidor.



Fonte: Autoria própria.

A utilização dessa arquitetura permite que os servidores sejam distribuidos entre vários computadores, e a responsabilidade de armazenar os dados é sempre de um servidor; ou seja, o cliente solicita as informações, os servidores processam e retornam para o cliente que então apresentam ao usuário na interface grafica.

Diferente de aplicações estáticas, a arquitetura cliente-servidor permite que sejam feitas alterações de conteúdo no cliente sem a necessidade de fazer alterações no código fonte. A utilização desse padrão permite a criação de sistemas mais complexos quando comparados com uso de aplicações estáticas. Os sistemas, tais como, portais de notícias, fóruns, blogs e lojas online podem ser desenvolvidos escolhendo uma linguagem de programação que seja mais adequada para a solução de software, facilitando a criação de diferentes regras de negócios. Por exemplo, pode-se construir funcionalidade para compor diferentes soluções, tais como, buscas personalizadas, autenticação de usuários, envio de novidades via e-mail, gerenciamento de produtos, pagamentos, entre outras funcionakidades.

Como todo o processamento de dados fica centralizado no servidor, ele pode ficar sobrecarregado caso receba muitas solicitações simultâneas dos clientes. Nessa arquitetura, se um servidor crítico falhar as solicitações feitas pelos clientes, estas não poderão ser processadas e uma resposta de erro será retornada ao cliente, necessitando que as chamadas sejam feitas novamente.

Ao construir aplicações cliente-servidor pode se utilizar de *framework* para construir aplicações de *Frontend* no lado do cliente, usando as aplicações de página única e *Micro Frontends*. Aplicações como foco *Backend* são apresendatadas em mais detalhes na Seção de microsserviços.

2.4 Aplicações de página única (SPA)

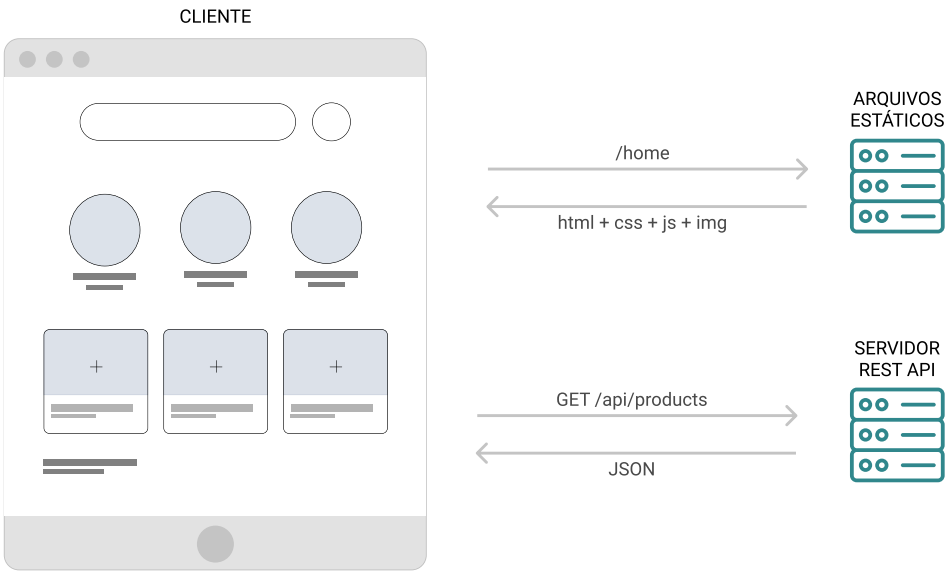
Um aplicativo de página única (SPA), é uma aplicação web que contém as funcionalidades concentradas em uma única página, fornecendo uma experiência de usuário similar a um aplicativo *desktop*. O que muda ao utilizar SPA em relação a páginas estáticas é o formato que os conteúdos são carregados: na primeira renderização será carregado todo conteúdo HTML, CSS e JavaScript. As aplicações de página única consistem em um único arquivo *JavaScript* ou alguns arquivos que são encapsulados (MEZZALIRA, 2021). A partir deste momento, quando o usuário navegar entre as diferentes partes da página, não será necessário mais fazer requisições para o servidor para o carregamento desses artefatos, uma vez que o conteúdo relacionado a eles já foi carregado no primeiro acesso (GUEDES, 2019).

Os SPA's geralmente se comunicam com APIs trocando dados com a camada de persistência do *Backend*. Ao usar aplicações SPA são evitadas várias idas e voltas ao servidor para carregar partes adicionais da aplicação e renderizar todas as visualizações instantaneamente durante o ciclo do sistema (MEZZALIRA, 2021), como apresentado na Figura 3. Os SPA's têm como alvo a experiência do usuário, entregando uma interface mais rápida e reativa. Além de otimizar o desempenho da aplicação, diminui o conteúdo a ser carregado ao navegar entre diferentes partes da aplicação.

Atualmente, existem diferentes bibliotecas que facilitam a criação de aplicações utilizando esse estilo. Aplicações SPA podem ser utilizadas em praticamente todas as situações, o que explica a popularização de *frameworks* que podem ser usados em projetos de SPA, os mais populares atualmente são *Angular*, *React* e *Vue.js* (GUEDES, 2019).

Uma das maiores vantagens ao usar essa arquitetura é o carregamento mais rápido (após sua carga inicial). O tráfego de informações transferidos entre o cliente e servidor é menor, fazendo com que a latência de renderização seja menor. Uma vez que as alterações visuais são processadas pelo cliente, e somente os elementos visuais afetados são atualizados; o restante da aplicação permanece sem sofrer nenhuma alteração. Ao usar SPA em um software, ganha-se na experiência do usuário e simula o que geralmente é encontrado ao interagir com um aplicativo nativo para dispositivos móveis ou *desktop* (MEZZALIRA, 2021).

Figura 3 – Exemplo de uma arquitetura de página única.



Fonte: Autoria própria.

Como desvantagens ao utilizar SPA, problemas de indexação podem ocorrer: pode ser mais difícil conseguir atingir um bom *ranking* nas pesquisas, uma vez que SPA's funcionam muito bem em aplicações que não estão relacionadas com a produção de conteúdos e não precisam ser achados. A carga inicial pode ser relativamente maior para aplicações maiores e a velocidade de carregamento inicial pode ser prejudicada. Outras preocupações estão relacionados à segurança; é necessário ter mais cuidado com os dados estão sendo compartilhados nas solicitações e respostas recebidas do servidor.

Ao construir soluções de software é fundamental estar atento para manter o código fonte de forma modular. O SPA permite a reutilização e a extensibilidade das partes da aplicação, sendo que a falta de código modular pode ser a diferença entre uma aplicação bem-sucedida e uma malsucedida (FINK; FLATOW, 2014). Com a implementação de novas funcionalidades contribui-se com aplicações ficando maiores.

Com o crescimento das aplicações SPA, a proposta de *Micro Frontends* se apresenta como uma solução para essa problemática; essa arquitetura é explorada em mais detalhes nas seções seguintes. No entanto, antes de abortar sobre as micro aplicações de *Frontend*, é fundamental entender sobre a arquitetura de microsserviços, pois é a partir dos aprendizados desse estilo, que os *Micro Frontends* foram propostos. A Seção a seguir tem como objetivo trazer uma breve apresentação da arquitetura de microsserviços.

2.5 Desenvolvimento em Microsserviços

O padrão da arquitetura de microsserviços se baseia em uma estrutura modular independente, mas interligada; ou seja, os microsserviços podem ser implantados de forma inde-

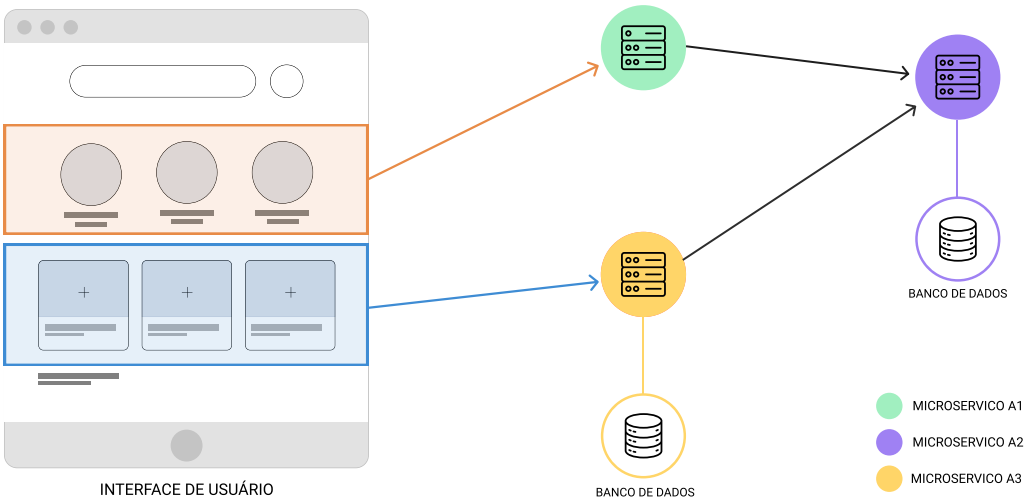
pendente, e são modelados em torno de um domínio de negócio, utilizando-se a comunicação entre si por meio de redes (NEWNAB, 2020). Em uma aplicação monolítica tanto o código *Frontend* quando o *Backend* estão combinados no mesmo repositório. A partir de um monolito, vários serviços podem ser separados e modularizados independentes um do outro, criando assim novos serviços em torno de um contexto de negócio.

Aplicações que são construídas com uma arquitetura monolítica são caracterizadas por possuírem diferentes funções de negócios implementadas sendo executadas em uma única unidade lógica em uma mesma máquina, compartilhando recursos como processamento e memória (SANTOS, apud KNOCHE; HASSELBRING, 2018). Todas as funcionalidades de um sistema precisam ser implantadas em conjunto (NEWNAB, 2020). Essas aplicações possuem módulos que não são executados independentemente (SANTOS, apud DRAGONI *et al.*, 2017). É comum as aplicações serem construídas com essa arquitetura pois possui uma maneira fácil de ser desenvolvida e implantada (SANTOS, apud SARITA; SEBASTIAN, 2018).

O que diferencia a arquitetura de microsserviços das abordagens monolíticas tradicionais é como ela decompõe a aplicação por funções básicas; cada função é denominada um serviço (REDHAT, 2018), conforme apresentado na Figura 4. Isso significa que cada serviço individual pode funcionar ou falhar sem comprometer os demais (REDHAT, 2018). Para garantir a possibilidade de implantações independentes, deve-se garantir que os serviços tenham baixo acoplamento (NEWNAB, 2020). Aplicações que utilizam a abordagem de microsserviços possuem componentes bem isolados que são organizados como um conjunto de recursos de negócios poucos acoplados (SANTOS, apud KNOCHE; HASSELBRING, 2018).

A possibilidade de implantações independentes consiste em realizar a alteração em produção em um microsserviço e implantá-lo em um ambiente de produção sem ter de utilizar outros serviços (NEWNAB, 2020). Com isso, ao isolar itens críticos em uma nova infraestrutura, é possível escalar ela de forma individual do restante da aplicação.

Figura 4 – Exemplo de uma arquitetura de microsserviços.

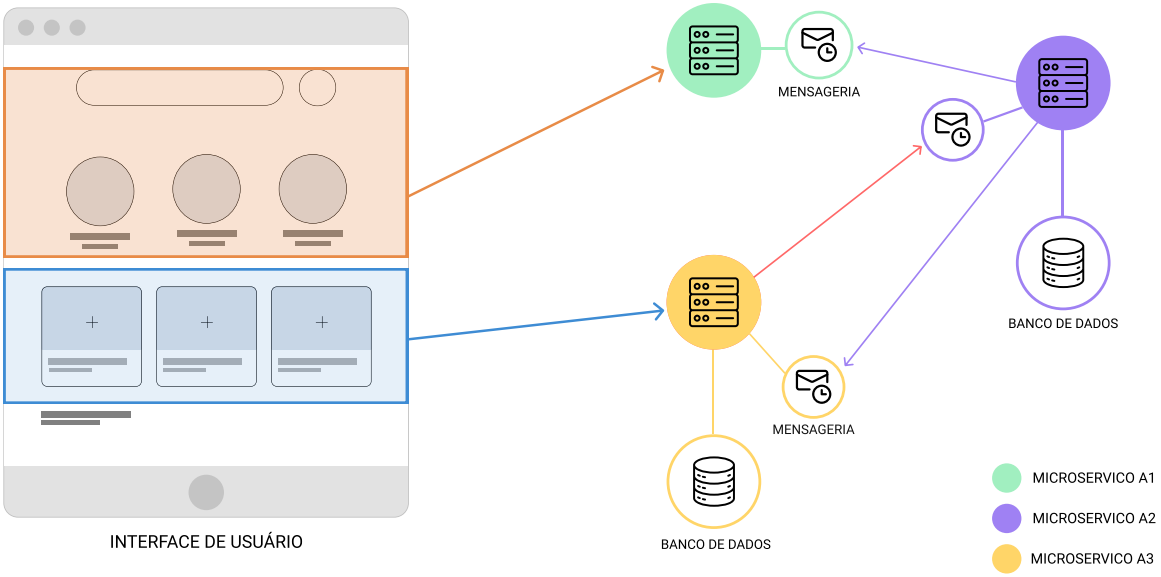


Fonte: Autoria própria.

Na Figura 4 apresenta a comunicação de uma SPA com três microsserviços, sendo que a interface de usuário fazer chamada para diferentes microsserviços. A interface destacada na cor laranja inicia a comunicação com o microsserviço A1, essa aplicação recebe a solicitação, processa os dados e se comunica com o microsserviço nomeado A2. A interface de usuário destacada na cor azul inicia a comunicação com o microsserviço A3, recebe a solicitação, processa os dados, armazena as informações em seu banco de dados e inicia uma comunicação o microsserviço A2.

Vale ressaltar que os microsserviços apresentados no exemplo acima são independentes e estão rodando em processos isolados e podem estar dentro do mesmo *cluster*[3] ou em diferentes infraestruturas de alocação.

Figura 5 – Exemplo de comunicação entre microsserviços utilizando mensageria.



Fonte: Autoria própria.

Um dos maiores desafios na transição de uma aplicação monolítica para uma arquitetura baseada em microsserviços é o mecanismo de comunicação entre as aplicações. Os microsserviços não devem compartilhar banco de dados, e se um serviço quiser acessar dados mantidos em outro serviço, ele deverá pedir os dados de que precisa para esse serviço (NEWNAB, 2020).

Em aplicações monolíticas, os componentes interagem entre si a nível de linguagem de programação, por meio de métodos e funções. Em aplicações de microsserviços, a comunicação entre dois ou mais processos é feita utilizando troca de mensagens (Filas de mensagens, ou *Queue message*). Esse padrão de comunicação assíncrona permite que aplicações consigam trocar informações entre si, conforme ilustrado na Figura 5.

[3] Um *cluster* corresponde a computadores conectados que operam em conjunto, de forma que, podendo ser considerados como um único sistema.

Na Figura 5, é apresentado um fluxograma visual de como ocorre a comunicação entre dois ou mais microsserviços por meio de troca de mensagens. Ao utilizar mensageria para comunicações externas entre os serviços, cada aplicação recebe uma solicitação para processar essa informação e adicionar uma mensagem em sua fila.

No exemplo apresentado entre a comunicação entre os microsserviços A3 e A2, o microsserviço A3 aguarda os eventos adicionados na fila do microsserviço A3. Ao receber uma mensagem disparada por o serviço A2, a aplicação A3 processa os dados, armazena em seu banco de dados e adiciona um novo evento em sua fila com os dados processados. Desta forma, o microsserviço A2 está aguardando as mensagens que estão sendo disparadas no microsserviço A3. Nesse fluxo, as duas aplicações distintas conseguem se comunicar com baixo nível de acoplamento e dependência.

As vantagens proporcionadas pelos microsserviços são variadas, pois a natureza independente das implantações dá abertura a novos modelos para aumentar a escalabilidade e a robustez dos sistemas, além de permitir que diferentes tecnologias sejam combinadas para compor uma solução de software (NEWNAB, 2020). A arquitetura de microsserviços é propícia para um desenvolvimento acelarado, pois a liberdade oferecida por esses serviços será sistema em um estado de mudança contínua, nunca estático, sempre evoluindo (FOWLER, 2017).

Ao planejar a implementação ou migração de uma aplicação para microsserviços é necessário lidar com uma complexidade intrínseca em diferentes níveis, como rede, camada de persistência, protocolos de comunicação, segurança e muitos outros (MEZZALIRA, 2021). Ou seja, a simplicidade de trabalhar com a lógica de negócio tem vários pontos de atenção a serem levados em consideração ao avaliar arquitetura (MEZZALIRA, 2021).

2.6 Desenvolvimento em *Micro Frontends*

Na maioria das aplicações, os microsserviços foram incorporados exclusivamente no lado do servidor, deixando a interface de usuário em um projeto único (NEWNAB, 2020). Com objetivo de ter uma arquitetura *Frontend* que facilite implantar novas funcionalidades com mais rapidez, deixar a UI como uma porção monolítica pode trazer alguns desafios, sendo necessário separar as interfaces de usuário também (NEWNAB, 2020).

Em novembro de 2016, começou a se discutir sobre um novo padrão arquitetural para decompor aplicações monolíticas de *Frontend* em partes menores e mais simples que podem ser desenvolvidas, testadas e implementadas independentemente, embora ainda pareçam para os usuários um único produto coeso (JACKSON, 2019). Esse estilo está sendo chamada de *Micro Frontends*. Trata-se de um estilo relativamente novo, e nesse momento ainda existem poucos artigos e exemplos práticos publicados.

Micro Frontends é um estilo emergente inspirado em microsserviços. A ideia principal por traz dele é dividir a base de código monolítico em partes menores, permitindo que uma organização espalhe o trabalho entre equipes autônomas, sejam elas agrupadas ou distribuídas,

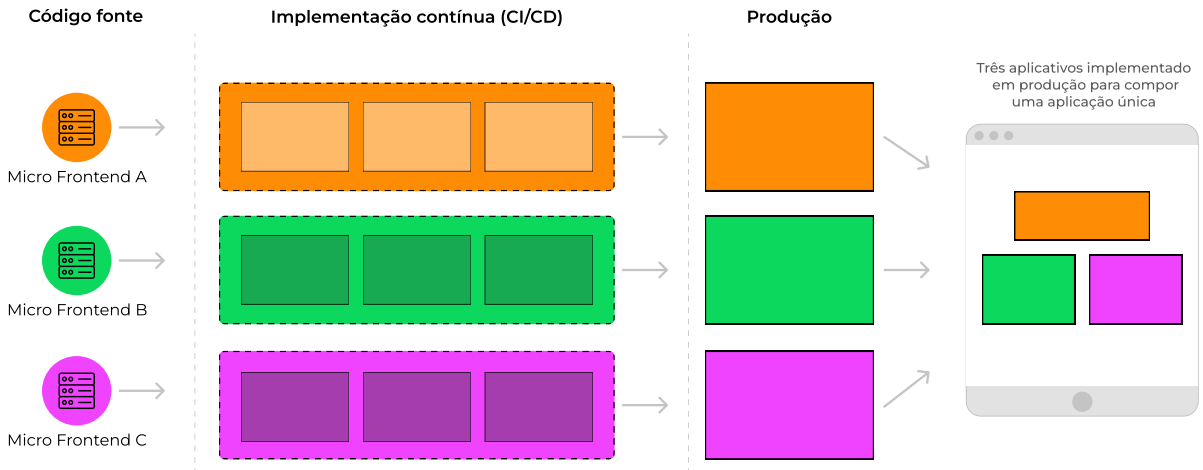
sem a necessidade de diminuir sua produção de entrega (MEZZALIRA, 2021). Na indústria de software, esse estilo tem despertado interesse dos profissionais e empresas que procuram analisar as vantagens e riscos ao utilizar essa arquitetura. Embora os *Micro Frontend* sejam uma abordagem bastante nova, ela têm sido usada por alguns anos em organizações médias e grandes (MEZZALIRA, 2021).

Há muita expectativa sobre a utilização desse estilo, muito devido aos casos de sucesso com a utilização de microsserviços. Acredita-se que o *Micro Frontend* seja o próximo passo para evolução do *Frontend*. Atualmente muitas empresas conhecidas fizeram do *Micro Frontend* seu principal sistema para escalar seus negócios para o o próximo nível, entre as empresas estão Zalando, HelloFresh, AllegroTech, Spotify, SAP, OpenTable e Dazn (MEZZALIRA, 2021). Os *Micro Frontend* tem se mostrado uma boa opção quando os projetos exigem que várias equipes estejam trabalhando na mesma aplicação ou tem a necessidade de substituir um projeto legado de forma iterativa (MEZZALIRA, 2021).

Hoje em dia, é comum construir o *Frontend* de um software com somente um *framework*, geralmente *React*, *Angular* ou *Vue*. Ao disponibilizar uma interface de usuário completa em uma única página, obviamente não é possível considerar uma composição baseada em páginas (NEWNAB, 2020). Portanto, é necessário considerar as composições de aplicações sendo mais flexiveis. Em contraste com essa ideia, sugere-se uma aplicação que é feita de pequenas aplicações independentes que se integram (HANASHIRO, 2019).

A ideia de *Micro Frontend* é exatamente permitir que cada parte independente seja agrupada para formar algo maior. Se cada pedaço da aplicação for independente, cada time pode escolher suas tecnologias, métodos de trabalho, horários de reunião, casos de teste, momentos para implantar (HANASHIRO, 2019), conforme apresentado na Figura 6.

Figura 6 – Implantação de um sistema em *Micro Frontend*



Fonte: Adaptado de Jackson (2019).

Na Figura 6 são apresentadas três micro aplicações, a primeira chamada de Micro Frontend A na cor laranja com base de código distintos, o processo de implementação contínua (onde são executado os testes e *build*) e publicação da nova versão. Esse ciclo, se repete para as aplicações nomeadas como Micro Frontend B e Micro Frontend C. No final do fluxo de implementação dos *Micro Frontends*, as três aplicações compõem uma única aplicação. Para fazer uma analogia sobre a responsabilidade de cada micro aplicação, assume-se que cada micro aplicação é uma parte pequena do sistema, desta forma o cabeçalho da página é uma aplicação, o rodapé poderia ser outra aplicação e a terceira aplicação poderia ser o conteúdo da página.

Ao planejar uma arquitetura de *Micro Frontends* é aplicado o aprendizado dos microsserviços. Porém a aplicação é destinada a construir a interfaces de usuário que vai ser executada diretamente no navegador do cliente. Ao aplicar esse estilo de arquitetura, deve-se ter bem claro quais as partes dessa aplicação serão divididas, definir cada contexto e quais as suas fronteiras. A ideia é que cada parte seja responsável por apenas uma necessidade de negócio. Seguindo esse pensamento, o próximo desafio é definir como as aplicações serão divididas e quais suas responsabilidades e, desta forma, será possível construir uma aplicação modular e flexível.

Ao arquitetar novos projetos utilizando *Micro Frontend*, seguem-se alguns princípios, sendo que cada aplicação deve ser executada separada das demais; ou seja, tudo o que essa aplicação precisa para ser executada está contida em seu código base. Os *Micro Frontend* podem exigir uma composição de uma interface de usuário em tempo de execução, o que pode resultar em falhas (MEZZALIRA, 2021). Nesse sentido, conteúdos alternativos devem ser fornecidos para que os usuários não sejam impactados quando parte do aplicativo falhar.

O uso de *Micro Frontend* não é apropriado a todos os casos, pois ao usar esse estilo arquitetural novas camadas adicionam maior complexibilidade a nivel técnico e organizacional (MEZZALIRA, 2021). Para diminuir esses riscos as equipes envolvidas necessitam ter uma cultura de automatizar as implementações em diferentes ambientes. Para isso é importante ter dominio com a implementação de regras *CI/CD* é uma combina as práticas de integração contínua e entrega contínua, aplicando monitoramento e automação continua a todo o ciclo de vida das aplicações, incluindo as etapas de teste e integração, além da entrega e implantação. Além disso, boas práticas de desenvolvimento devem ser seguidas e automatizar os processos de qualidade são fortes aliados para manter a saúde do software em harmonia.

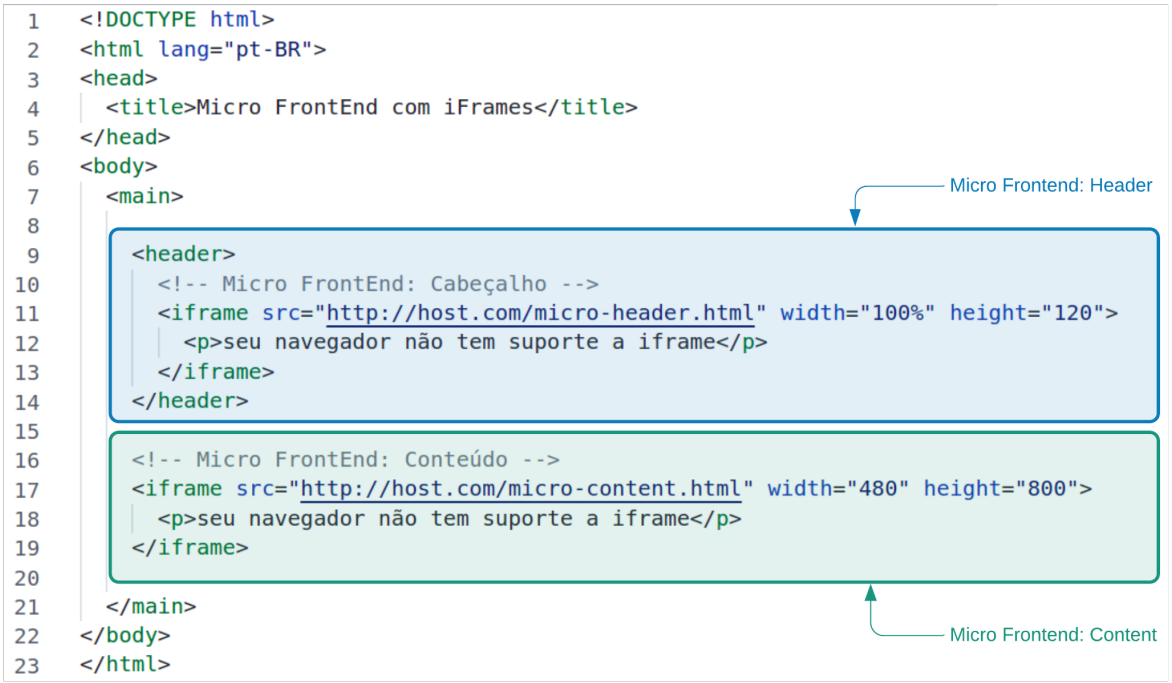
Nos próximos capítulos são exploradas algumas formas de construir *Micro Frontend*, o primeiro tópico aborda a construção utilizando a tag *Iframe's*, na sequência é colocada em enfoque a construção de soluções em tempo de execução fazendo uso de CDN's (*Content Delivery Network* ou Rede de Distribuição de Conteúdo).

2.6.1 Integração em tempo de execução com Iframe

Existem diferentes abordagens para a construção de um software que faz uso da arquitetura de *Micro Frontend*. O formato mais básico é usando tags *Iframe's* que podem ser incorporados a outras páginas HTML.

Por sua natureza, os *Iframe's* facilitam a construção de uma página a partir de subpáginas independentes (SARING, 2020). A utilização de tags *Iframe's* para construção de *Micro Frontend* é o formato mais primitivo e tradicional que pode ser aplicado para construir aplicações independentes e possuir um bom isolamento. Para construir um projeto utilizando esse formato basicamente são necessárias de algumas tags de *script* e *Iframe*, conforme apresentado na Figura 7.

Figura 7 – Integração *Micro Frontend* com Iframe's.



Fonte: Adaptado de Saring (2020).

Na Figura 7, pode-se visualizar nas linhas 11 e 17 as tags *Iframe's* aplicadas, com as URLs (*Uniform Resource Locator*) de aplicações disponíveis em seus respectivos domínios. Esse é o formato mais primitivo para implementar um software existente dentro de outra aplicação. Segundo esse formato, seria possível utilizar aplicações existentes e agrupá-las para construir uma nova solução. Se todas aplicações estiverem com o mesmo padrão visual aplicado, tem-se uma aplicação consistente. Um caso de sucesso aplicado pelo Spotify, pode ser consultado em (BIOMEE, 2020).

2.6.2 Integração em tempo de execução usando CDN's

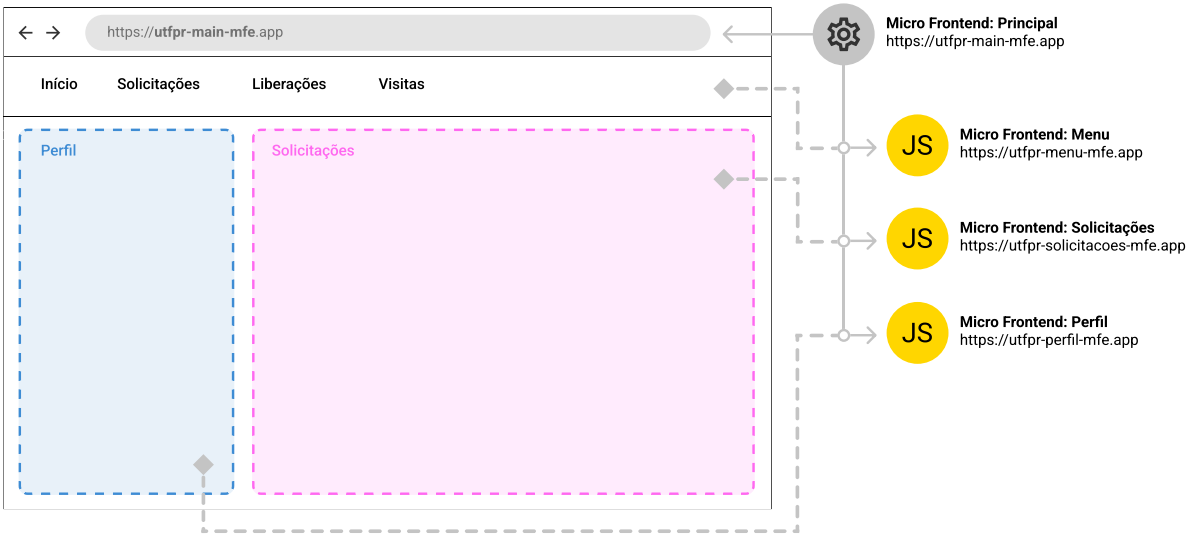
A composição de micro aplicações em tempo de execução, segue como base que cada aplicação tem um arquivo *JavaScript* estático, que pode ser acessado atráves de uma URL. Ao compor as aplicações para a integração de cada *Micro Frontend* em tempo de execução, usa-se uma aplicação raiz, que será responsável por orquestrar como as aplicações menores devem ser implementadas.

Assim sendo, cada módulo é executado de forma independente, e cada projeto não sabe da existência do outro. Para configurar uma aplicação nesse formato, é necessário definir quais as URL's de cada micro aplicação e em qual parte da página ela deve ser renderizada. Com essas informações, a aplicação será responsável por gerenciar a execução de cada micro aplicação.

As micro aplicações possuem dois estados essenciais que são: aplicações foram exibidas ao usuário ou serão renderizadas quando algum gatilho acontecer. Um exemplo de aplicação que está sempre montada é uma micro aplicação responsável por renderizar um menu: ela sempre está visível ao usuário e possui a responsabilidade de exibir os menus de navegação para outra parte da aplicação, apresentado na Figura 8 como MFE - MENU. O ciclo de vida de um *Micro Frontend* aguardando por um gatilho acontece quando mais de uma aplicação é renderizada no mesmo local da página.

Para permitir que a aplicação raiz consiga gerenciar e fazer um mapeamento de cada *Micro Frontend* construído, configura-se o processo de *build* das aplicações para disponibilizar todos os arquivos estáticos referentes a esse projeto em uma CDN. Com as regras de integração contínua configuradas, a cada nova alteração tem-se nova versão que estará disponível para ser utilizada. Assim, a única aplicação que precisa executar o processo de *deploy* será o *Micro Frontend* modificado. Os demais projetos não são impactados com as modificações.

Figura 8 – Aplicação de *Micro Frontend* em tempo de execução.

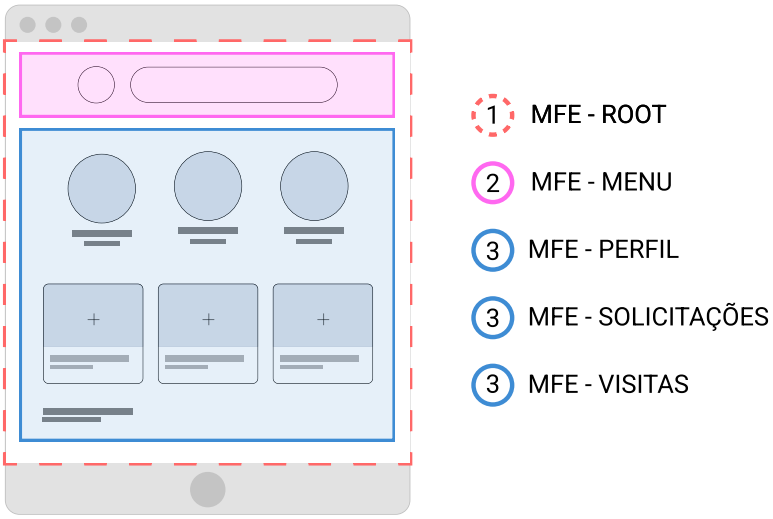


Fonte: Adaptado de HypeFlame (2021).

A Figura 8 traz uma ilustração para exemplificar essa dinâmica de renderização. Leva-se em consideração que renderização dos *Micro Frontends* na estrutura de uma página é responsabilidade da aplicação principal; é ela quem gerencia quando e aonde essas aplicações serão montadas. No exemplo, podem ser visualizadas três micro aplicações que juntas compoem uma aplicação única.

Ao realizar a implementação utilizando CDN's, o *deploy* dos arquivos estáticos (JavaScript) podem estar tanto dentro do mesmo dominio (URL) quanto em URL's distintas, e o único *Micro Frontend* que precisa ter essa informação é aplicação principal.

Figura 9 – Composição de vários *Micro Frontend* na mesma estrutura da página.



Fonte: Autoria própria.

A Figura 9 exibe um exemplo de composição de *Micro Frontend*. No exemplo, a interface do usuário apresenta o espaço na cor azul, onde diferentes *Micro Frontends* serão renderizados. Com isso ao navegar por diferentes rotas, um novo *Micro Frontend* será renderizado nesse espaço.

No exemplo, esse comportamento acontece na página inicial, que renderiza o *Micro Frontend* `MFE - PERFIL`. Mas quando o usuário acessar a página de Solicitações, a micro aplicação que será renderizada será `MFE - SOLICITAÇÕES`. Com isso, pode-se construir regras que servem como gatilhos para que em cada URL um *Micro Frontend* seja renderizado em determinada estrutura da página (elemento *HTML*). Com essa configuração, o restante da aplicação continua sem sofrer alterações.

Ao desenvolver o sistema com arquitetura de *Micro Frontend*, será usado o sistema SPA para construir aplicações menores. Nesse processo cada aplicação deve configurar e ter o seu processo de *deploy* independente. Os artefatos estáticos serão disponibilizados em uma URL; desse modo, a aplicação principal vai usar a implementação em tempo de execução para acessar os arquivos estáticos de cada aplicação para construir o projeto. Para ver em mais detalhes o processo de desenvolvimento, ele foi reportado no Capítulo 4.

3 REVISÃO NARRATIVA DA LITERATURA

O processo de avaliação da literatura foi baseado nos três artigos, sendo eles, (i) Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review (PELTONEN; MEZZALIRA; TAIBI, 2021), o qual traz as motivações e objetivos que levam a utilização de Micro Frontend; o artigo (ii) Micro-Frontends: application of microservices to Web front-ends (PAVLENKO *et al.*, 2020) apresenta as decisões arquiteturais na implementação desse estilo; e a tese de mestrado (iii) An Exploratory Study of Micro-Frontends (MONTELIUS, 2021), na qual a autora implementa duas soluções e realiza uma pesquisa com os desenvolvedores em relação a nova arquitetura com *Micro Frontends*, trazendo uma análise da implementação dessa arquitetura. Por ser um tema novo, existem poucas publicações relevantes, sendo estas recentes nos anos de 2020 e 2021.

3.1 Motivações, benefícios e problemas de Micro Frontends

3.1.1 Problema apresentado

O conceito de *Micro Frontend* foi introduzido em 2016 e vêm sendo incentivado por profissionais da área de desenvolvimento Web. Essencialmente, a arquitetura de *Micro Frontend* se assemelha com os microsserviços e, consequentemente, carrega suas vantagens e desvantagens (MEZZALIRA, 2021). No entanto, por ser uma arquitetura emergente, muitos profissionais têm receio de adotar essa arquitetura, uma vez que eles não estão completamente cientes dessas vantagens.

O aumento da complexidade do *Frontend* monolítico e a necessidade de ter diversas equipes atuando no desenvolvimento do mesmo projeto tem sido cada vez mais comum. Desta forma, o objetivo do trabalho proposto é mapear e compreender quais os benefícios e problemas encontrados ao utilizar esse estilo arquitetural. Por esse motivo, as questões de pesquisa propostas pelos escritores foram definidas para cobrir os tópicos mais básicos e trazer uma uma visão mais abrangente sobre o assunto, porém sem entrar em detalhes sobre as implementações.

3.1.2 Procedimento utilizado

Nesse sentido, os autores conduziram uma revisão multifocal da literatura, com o objetivo de identificar quais são as principais motivações para a utilização de *Micro Frontend*. A revisão contou com a participação de 43 participantes, a partir dos quais foram levantadas as seguintes questões:

- **Por que os profissionais estão adotando Micro Frontends**: Ao aplicar essa questão, procura-se compreender quais são as motivações que tem feito com que os profissionais adotem a arquitetura de *Micro Frontends*;
- **Quais são os benefícios obtidos com o uso de *Micro Frontends***: Procura-se compreender quais são os beneficios ao utilizar esse estilo arquetetural e a quais são os problemas que os *Micro Frontends* pretendem resolver;
- **Os *Micro Frontends* apresentam algum problema**: Procura-se compreender quais são os problemas que podem ser encontrados ao utilizar esse padrão, bem como propor soluções aos problemas identificados;

Em setembro de 2020, os autores realizaram uma busca na literatura, para coleta de trabalhos acadêmicos, relacionado aos termos de pesquisa *Micro-Frontend** e *Micro Front end**; o termo foi aplicado a todos os campos (título, resumo, palavra-chave, corpo e referências), e dessa forma foi possível coletar um maior número de trabalhos. Devido ao tema ser relativamente novo, os mesmos critérios de pesquisas foram realizadas para ferramentas de busca, para coletas de conteúdo publicados em podcasts, vídeos, sites, fóruns e postagens em blogs.

Ao realizar a classificação dos artigos encontrados na literatura, foi aplicada a revisão por pares. Para os conteúdos localizados na internet, a classificação foi feita utilizando a revisão formal, ao qual dois autores avaliaram a qualidade do conteúdo por meio de uma nota. Desta forma procurou-se avaliar a credibilidade e a qualidade das fontes. Após a avaliação, somente os conteúdos com nota relevante foram selecionados.

Com base nas fontes selecionadas, os dados foram extraídos para uma planilha de revisão, para identificar as motivações, benefícios ou problemas relatados pelas fontes. A análise qualitativa dos dados foi realizada por três autores. Caso houvessem diferentes interpretações entre os autores, uma discussão era iniciada para auxiliar na avaliação final. Após a coleta de dados, 172 fontes foram encontradas, sendo 129 excluídas no processo de classificação. Com isso, finalizou-se o processo com 43 fontes e apenas três (6,94%) foram artigos de conferência revisados por pares, enquanto os outros 40 foram originados de conteúdos coletados em podcasts, vídeos, sites, fóruns e postagens em blogs.

3.1.3 Resultados Observados

A utilização de *Micro Frontends* está se tornando cada vez mais popular, e o padrão arquitetural está sendo adotados por várias grandes empresas, o que permite a divisão de *Frontends* monolíticos em micro aplicações independentes e menores. No entanto, os motivos que estão levando à adoção ainda não são claros para a comunidade.

O aumento da complexidade do *Frontend* monolítico e a necessidade de ter diversas equipes de desenvolvimento no mesmo projeto, são os principais motivos para a utilização de

Micro Frontends. As fontes selecionadas mencionam, o problema para delegar responsabilidades a equipes independentes e a necessidade de facilitar o suporte para DevOps. Uma observação interessante é que vários profissionais relataram a adoção de arquiteturas baseadas em microsserviços e *Micro Frontends* porque muitas outras empresas estão adotando.

Os resultados mostraram que a utilização de *Micro Frontends* têm vários benefícios, como integração mais rápida para desenvolvedores, equipes multifuncionais e desempenho aprimorado de aplicativos da Web. Ao mesmo tempo, esse padrão compartilha vários benefícios com microsserviços, já que ambos são agnósticos em tecnologia, e ambos podem ser desenvolvidos, implantados e mantidos de forma independente diminuindo a dependência de *framework* e linguagem. Cada *Micro Frontend* é autônomo, o que proporciona uma entrega veloz, e equipes podem atuar em diferentes partes da aplicação sem influenciar umas nas outras. Dado que se trata de uma arquitetura fracamente acoplada com padrões bem definidos, torna mais transparente acrescentar novos recursos ou formar equipes quando necessário.

Os problemas encontrados ao utilizar o *Micro Frontend* estão relacionados ao uso de diferentes *frameworks*, resultar em projetos relativamente maiores, o que pode prejudicar o tempo de carregamento das aplicações. As dependências compartilhadas entre diferentes projetos podem gerar duplicação de código. A experiência de usuário pode se tornar um desafio com equipes autônomas, e são necessários processos bem definidos para garantir a consistência na UX entre diferentes *Micro Frontends*. Com base nos resultados, as empresas precisam levar em conta a maior complexidade introduzida pelos *Micro Frontends*, não só do ponto de vista técnico, mas também do ponto de vista de gestão e coordenação.

3.1.4 Como se aplica ao trabalho proposto

O trabalho proposto nessa monografia tem foco no desenvolvimento de uma aplicação seguindo as práticas de *Micro Frontend*. As vantagens e desvantagens encontradas pelos autores serão utilizadas para auxiliam na elaboração de métricas e procedimentos de avaliação da aplicação que serão desenvolvidas. Os questionarios utilizados e o formato como os resultados foram apresentados serão essenciais para direcionar o desenvolvimento utilizando os fundamentos de *Micro Frontend*.

3.2 Aplicação de microsserviços para front-ends da Web

3.2.1 Problema apresentado

Este artigo explora o conceito de desenvolvimento Web utilizando *Micro Frontends*, trazendo uma breve visão geral de como o desenvolvimento Web tem evoluído e como as abordagens de arquitetura têm surgido. A utilização de *Micro Frontends* foi considerada pelo o autor

uma evolução lógica para aplicações de *Frontend*. O conceito de arquitetura de micro-frontends é a abordagem de microsserviços adotada para o desenvolvimento *Frontend*, e é uma nova alternativa para o desenvolvimento de aplicativos *frontend* modulares.

O estudo de caso apresentado no estudo tem como objetivo examinar o uso de *Micro Frontends* aplicados em projetos desenvolvidos utilizando SPA. Durante o estudo de caso, os autores destacam que o uso da arquitetura de *Micro Frontend* não é indicada para projetos pequenos e com poucas pessoas envolvidas, pois a maioria do esforço seria gasto no suporte da arquitetura e não com o desenvolvimento de recursos. Em geral, ele se ajusta melhor se for óbvio que o sistema terá muita lógica de negócios no *Frontend* e será desenvolvido por uma grande equipe de desenvolvedores no longo prazo.

3.2.2 Procedimento utilizado

No estudo de caso, foi realizado o desenvolvimento de um sistema com foco em cursos online. Os serviços de *Backend* foram desenvolvidos utilizando a arquitetura de microsserviços e no desenvolvimento *Frontend* foi utilizado *Micro Frontends*. Em seguida, foi iniciada uma discussão sobre os benefícios e vantagens da utilização da arquitetura de *Micro Frontend*.

Os sistema proposto pelo autor trouxe o seguintes requisitos:

- Deve exibir uma lista de cursos e seus detalhes;
- Deve exibir feedback para os cursos;
- Deve exibir uma lista de universidades e seus detalhes;
- Deve exibir uma lista de eventos educacionais e seus detalhes
- Deve exibir os 5 melhores cursos;
- Deve realizar pesquisa parametrizada e textual em cursos;
- Deve permitir que um usuário se autentique usando OAuth
- Deve permitir que um usuário se autentique usando Login/Senha
- Deve permitir que um usuário deixe um feedback;
- Deve permitir que um usuário marque um curso, universidade ou evento como favorito;
- Deve permitir que o administrador se autentique usando login e senha;
- Deve permitir que o administrador execute operações CRUD em cursos, universidades e eventos;
- Deve permitir que o administrador modere o feedback;

O desenvolvimento do sistema utilizou o conceito arquitetural de divisão por camada, sendo estas apresentação (*Frontend*), lógica de negócio (*Backend*) e dados (banco de dados). A camada de apresentação foi separada em quatro *Micro Frontends*, que são desenvolvidos de forma independente, porém juntos compõem uma única solução para o cliente.

Todos os *Micro Frontends* se comunicaram com a camada de lógica de negócio utilizando apenas uma única entrada, nomeada de *API Gateway*. A aplicação de *gateway* recebe as solicitações do *Frontend* e direcionam as solicitações para o microsserviço correto, funcionando como uma ponte entre o *Frontend* e *Backend*.

3.2.3 Resultados Observados

Os autores coletaram relatos para utilizar a integração contínua e *pipeline* de entrega contínua para apoiar arquitetura escolhida. Ao adicionar essa automatização, os processos de desenvolvimento e implementação devem levar mais tempo para configuração inicial. Porém, com o crescimento do projetos, esses processos automatizados devem trazer mais consistência nas implementações. Algumas bibliotecas podem ajudar a construir um aplicativo baseado em arquitetura de *Micro Frontends*; no artigo, foi utilizada a biblioteca *Single SPA*[1] para orquestrar o gerenciamento de rotas, gerenciando quando uma micro aplicação seria renderizada na interface do usuário. Para que os micro-frontends sejam uma opção viável para pequenos projetos e equipes, é necessária uma ferramenta abrangente e um suporte de biblioteca que simplifique e agilize o processo de implementação da arquitetura.

3.2.4 Como se aplica ao trabalho proposto

O trabalho proposto na monografia dos autores tem foco no desenvolvimento de uma aplicação seguindo as práticas de*Micro Frontend*. As vantagens e desvantagens encontradas auxiliam na tomada de decisões ao construir a arquitetura de *Micro Frontend* para o sistema, algo que será desenvolvido no trabalho de estudo de caso.

3.3 Um estudo exploratório de Micro Frontends

A tese de mestrado descrita por Anna Montelius, tem como objetivo investigar como o desenvolvimento de software de aplicações Web é afetado pela adoção do conceito de Micro Frontends, tanto no aspecto arquitetônico quanto no organizacional. A pesquisa foi aplicada com natureza exploratória e realizou a coleta de informações para avaliar os pontos fortes e os quais são os desafios com a utilização de Micro Frontends. Foram aplicadas duas soluções Web, onde uma aplicação está utilizando o padrão SPA tradicional e outra implementando um

[1] Single SPA: https://single-spa.js.org/

padrão de *Micro Frontends*. O *framework* React foi escolhido por sua popularidade e estrutura de componentes.

O projeto SPA foi criado utilizando o template genérico do create-react-app[2], já o template para o projeto de MFA foi utilizado create-single-spa[3]. Ambos os projetos foram desenvolvidos com a mesma interface de usuário, para que a comparação entre os dois projetos fossem consistentes. Os projetos foram avaliados usando um modelo matemático de modificabilidade e com um método de avaliação baseado na experiência que consiste em entrevistas aos desenvolvedores perguntando suas considerações. As questões levantadas pela autora aos desenvolvedores foram:

- Quais são os efeitos da adoção do conceito de Micro Frontends em um projeto de aplicação Web;
- Como a modificabilidade de um aplicativo é afetada pela adoção do conceito de Micro Frontends;
- Quando o conceito de Micro Frontends deve ser usado;

Segundo os autores, durante as entrevistas, os desenvolvedores entrevistados relataram que ao utilizar *Micro Frontends*; as responsabilidades de cada equipe são claras, a comunicação entre os desenvolvedores *Frontend* e *Backend* é pequena, e as equipes se tornam especialistas em suas regras de negócios. Ao iniciar projeto utilizando *Micro Frontends*, alguns riscos devem ser considerados, como a qualidade do código que pode variar entre diferentes projetos, o conhecimento sobre uma aplicação pode desaparecer da empresa se não existir muito trabalho em um projeto por um tempo, e durante mudanças de equipes pode levar algum tempo para que os integrantes se tornem produtivos.

Ao realizar o experimento, foi possível identificar que o protótipo SPA é mais modificável. O uso dos *Micro Frontends* devem ser aplicados somente na criação de grandes projetos e com mais de uma equipe atuando no processo de criação. Como próximos passos, seria possível aplicar essa pesquisa para profissionais de diferentes empresas, para tornar os resultados mais abrangentes e independentes.

O experimento realizado por *Anna Montelius* é de fundamental importância para execução da análise da implantação *Micro Frontends*, como foco no sistema alvo desenvolvido em SPA. Ao aplicar as responsabilidades de cada *Micro Frontend*, será possível utilizar os aprendizados do experimento para construir uma solução sólida e fazer proveito de todos os benefícios relatados.

---

[2] create-react-app: https://create-react-app.dev/ é um template com configuração inicial para projeto React.

[3] create-single-spa: https://single-spa.js.org/docs/create-single-spa/ é um template com configuração inicial para projeto Single SPA.

3.4 Considerações Finais

Após a revisão da literatura, vale destacar alguns pontos importantes que são destacados pelos autores. Uma das principais vantagens da utilização do estilo arquitetural *Micro Frontend* é a divisão de código em pequenos projetos, trazendo maior agilidade com o processo de desenvolvimento e conhecimento de regras da aplicação, as responsabilidades bem definidas entre diferentes equipes permitem maior agilidade no processo de criação, gerenciamento na qualidade, entregas mais ageis e independentes.

A utilização da arquitetura de *Micro Frontends* é indicada para projetos com expectativas de crescimento e que possuem mais de uma equipe atuando em paralelo. Até o momento, a utilização para pequenas aplicações e com poucas regras de negócio envolvidas necessitam ser avaliadas, pois o processo de configuração e gerenciamento dessas aplicações propõem mais desafios em relação a aplicações convencionais.

Segundo os autores, atualmente muitas empresas e profissionais estão investigando esse padrão arquitetural que parece ser a solução adequada a diversos problemas em aplicações robustas no *Frontend*. Porém, alguns pontos de atenção precisam ser analisados como o tamanho da equipe, definições de padrões entre diferentes projetos, compartilhamento de componentes de interface de usuário e aplicações com dependências duplicadas que podem aumentar o tamanho da aplicação final.

4 ESTUDO DE CASO

O estudo de caso é um método de pesquisa extensa sobre um determinado tema, permitindo aprofundar o conhecimento sobre aquele tema e, assim, servir de base para estudos mais aprofundados sobre o mesmo. William Goode afirma que o estudo de caso é um meio de organizar os dados, preservando do objeto estudado o seu caráter único (BRUCHEZ *et al.*, 2016).

Para o desenvolvimento deste estudo, foi utilizado um comparativo arquitetural no *Frontend* de dois sistemas com as mesmas funcionalidades. O sistema se trata de uma aplicação Web utilizada por servidores do DESEG-DV (Departamento de Serviços Gerais Câmpus de Dois Vizinhos), docentes e vigilantes do câmpus Dois Vizinhos. O primeiro sistema já foi desenvolvido utilizando SPA e será apresentado na Seção 4.1. O segundo sistema foi desenvolvido utilizando as funcionalidades presentes no primeiro sistema, e para o desenvolvimento dessa aplicação foi utilizado o estilo arquitetural *Micro Frontend*, separando as aplicações por responsabilidade de negócio.

4.1 Sistema alvo utilizando SPA

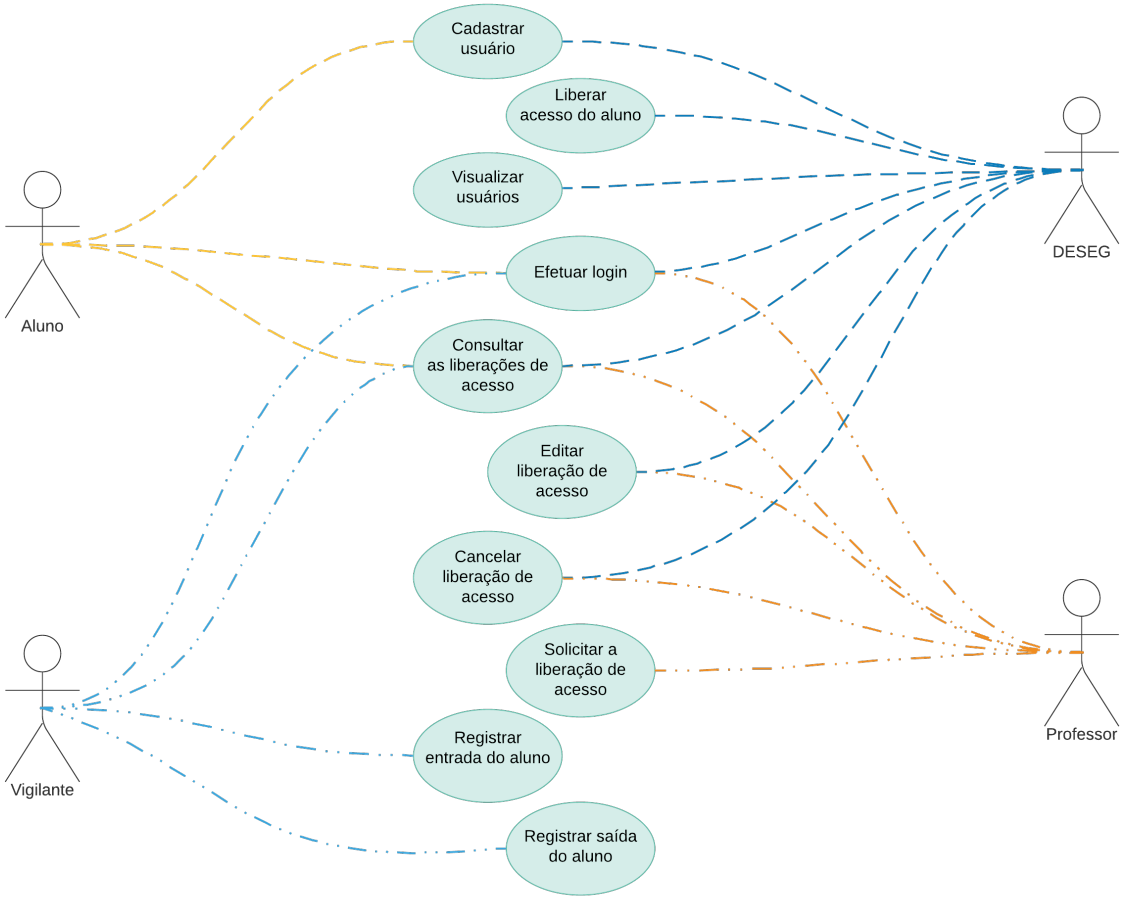
O sistema alvo escolhido foi o sistema desenvolvido pelo projeto Fabrica de Software da UTFPR (SANTOS *et al.*, 2021). O problema abordado por essa solução é facilitar o controle de entrada e saída de estudantes no câmpus, durante os finais de semana e feriados. Para que o aluno possa ter acesso ao câmpus, este necessita que o professor responsável autorize sua entrada, já os vigilantes por sua vez necessitam registrar a entrada e saída. Uma das dificuldades desse sistema é fornecer uma forma prática e automatizada para o processo de liberação para todos os perfis de usuário, e que esse processo consiga ser gerenciado de forma segura.

As funcionalidades presentes no projeto são controle de acesso por perfil de usuário, podendo ser DESEG, Vigilante e Professor; cada perfil de usuário possui suas permissões relacionadas, como mostrado em mais detalhes da Figura 10. Outras funcionalidades que compõem a solução são: realizar cadastro no sistema, acesso (*login*) utilizando as credenciais de acesso da UTFPR, cadastrar e editar a liberação do aluno, aprovar a entrada do aluno, e registrar entrada e saída do aluno. A Tabela 1 apresenta as estórias de usuário para o sistema alvo.

As tecnologias utilizadas para o desenvolvimento do *Frontend* foram o React com TypeScript, e para construir as interfaces de usuário foi aplicada a biblioteca MaterialUI[1]; para personalizações visuais adicionais foi usada a técnica CSS-in-JS[2] com a biblioteca *Styled Com-*

[1] https://mui.com/pt/material-ui/getting-started/overview/
[2] https://en.wikipedia.org/wiki/CSS-in-JS

Figura 10 – Casos de uso das funcionalidade do sistema-alvo.



Fonte: Autoria própria.

ponents[3]; e para realizar a comunicação com o servidor, foi utilizada chamada de API com auxílio da biblioteca *Axios*[4].

4.2 Considerações sobre a arquitetura utilizando SPA

Após investigação o sistema alvo, implementado de acordo com o estilo SPA, foi possível propor algumas atividades que servem de guia para o processo de criação, reestruturação e configuração dos *Micro Frontends*.

- Executar uma análise com objetivo de identificar quais as fronteiras de cada *Micro Frontend* e criar agrupamento por contexto de negócio; desta forma, será possível separar esse contexto em um novo projeto, que por sua vez resultará, para cada contexto separado, em um novo *Micro Frontend*;

[3] https://styled-components.com/
[4] https://axios-http.com/

Tabela 1 – Funcionalidades do sistema desenvolvido no projeto Fábrica de Software.

Func.	Estórias do Usuário	
	ID	**Descrição**
F1	EU01_T1	Eu como DESEG quero cadastrar usuários de perfil DESEG e Vigilante.
	EU01_T2	Eu como Professor quero ser cadastrado automaticamente no sistema, uma vez que realizar login com as credenciais de minha conta institucional.
F2	EU02_T1	Eu como usuário quero acessar o sistema, por meio de email e senha.
	EU01_T2	Eu como Professor realizo login por meio de credenciais de minha conta institucional.
F3	EU03_T1	Eu como Professor quero solicitar a liberação de acesso de um aluno ao câmpus.
	EU03_T2	Eu como DESEG quero liberar acesso de um aluno ao câmpus.
F4	EU04_T1	Eu como Professor/DESEG quero editar uma liberação.
	EU04_T2	Eu como Professor/DESEG quero cancelar uma liberação.
F5	EU05_T1	Eu como DESEG quero consultar todas as liberações cadastradas.
F6	EU06_T1	Eu como Vigilante quero consultar todas as liberações no dia corrente.
	EU06_T2	Eu como Vigilante quero registrar a entrada de um aluno liberado.
	EU06_T3	Eu como Vigilante quero registrar a saída de um aluno.
F7	EU07_T1	Eu como DESEG quero gerar um relatório com todas as entrdas e saídas em um determinado período.

Fonte: (SANTOS *et al.*, 2021).

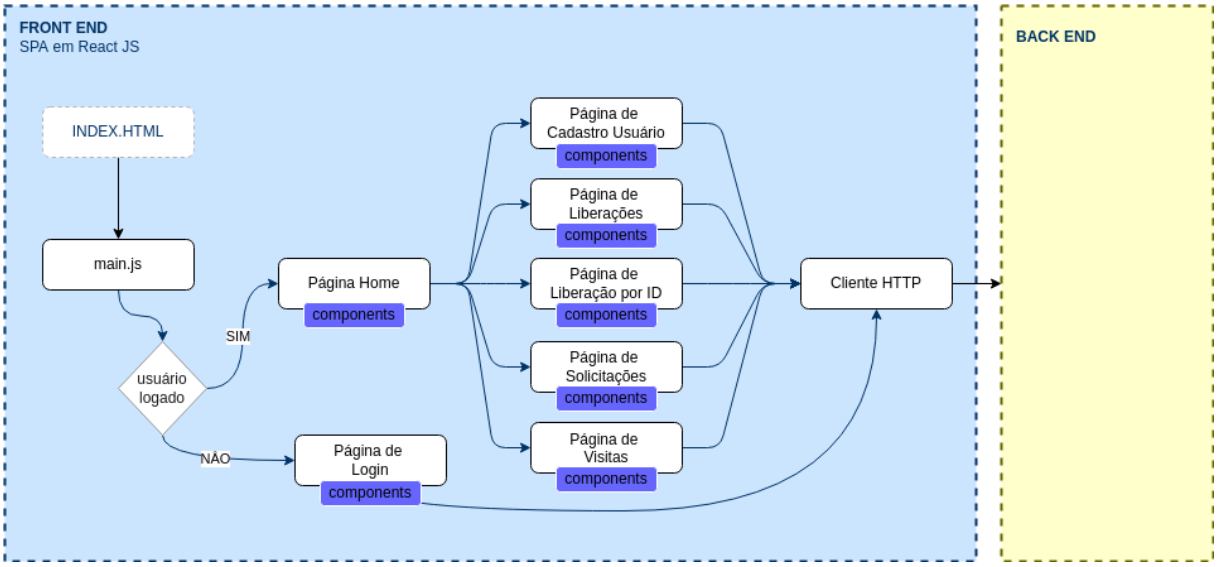
- Executar uma análise para identificar quais partes do projeto podem ser reutilizadas por diferentes aplicações; desta forma, os componentes e estruturas que são comuns podem ser separadas e usadas sem a necessidade de duplicação.
- Construir uma biblioteca de componentes *React*, com as estruturas comuns para reuso. A distribuição da biblioteca privada foi usado NPM[5];
- Configurar regras para integração contínua;
- Configurar regras para publicar as novas versões das bibliotecas;
- Construir template com arquitetura que será aplicada em cada micro aplicação;
- Construir biblioteca NPM para compartilhar regras e configurações que são comuns ao criar um novo *Micro Frontend*;
- Configurar regras para publicar uma nova versão do *Micro Frontend* quando as alterações são mescladas com a ramificação principal do repositório;

Como resultado dessa investigação, a Figura 11 apresenta como o projeto está organizado. A parte em azul da imagem apresenta um repositório com foco em construir a interface visual utilizada pelo usuário para executar as rotinas e tarefas. Em aplicações *Frontend*, é comum que elas tenham comunicação com o *Backend* para gravar ou buscar os dados que serão

[5] *Node Package Manager* (ou NPM): é o gerenciador de pacotes padrão para o ambiente de execução JavaScript Node.js.

apresentados; essa integração é representada pela cor amarela na Figura. Desta forma, são duas aplicações distintas se comunicando via API, para trocar informações.

Figura 11 – Mapeamento da arquitetura do sistema alvo



Fonte: Autoria própria.

Vale lembrar que quase todas as aplicações de software escritas atualmente são construídas nesse estilo; essas divisões seguem uma arquitetura de três camadas que podem ser compostas de diferentes formas (FOWLER, 2017). Em resumo, temos uma base de código responsável por armazenar e compartilhar diversas regras e comportamentos que serão usados pelo usuário desse sistema.

A arquitetura de *Micro Frontend* proposta será responsável por reestruturar o sistema alvo em novos repositórios com responsabilidades mais independentes e será apresentada em mais detalhes na sequência.

4.3 Arquitetura de *Micro Frontend* proposta

Considerando que o sistema alvo está em perfeito funcionamento, ao final do processo de migração para a arquitetura proposta, é imprescindível que toda implementação ocorra sem gerar atrito aos usuários. Do ponto de vista do usuário final, as funcionalidades permanecem as mesmas, assim como suas interações por meio da interface gráfica.

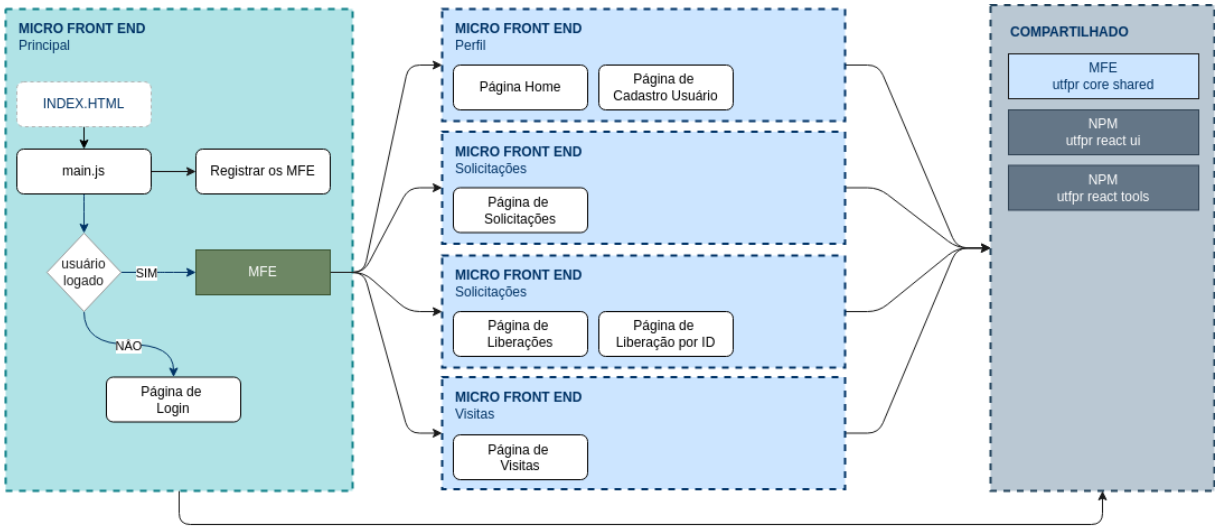
Nesse sentido, a arquitetura proposta visa criar divisões por subdominio de negócio. Essas divisões serão usadas para criar os novos *Micro Frontends*, os quais serão responsáveis por uma parte da jornada do usuário. Para realizar essas divisões, será necessario relembrar quais são esses contextos e como eles são utilizados pelo usuário dentro da aplicação:

- Usuário pode editar seus dados pessoais;

- Usuário do sistema necessita se identificar mediante *login* e senha;
- Usuário pode visualizar e criar novas solicitações;
- Usuário pode visualizar e registrar as liberações;
- Usuário pode visualizar as visitas ao campus;

Com essa compreensão, foi proposto o projeto da arquitetura com as divisões dos contextos, apresentado na Figura 12. Desta maneira, a separação dessa aplicação resultou em novos repositórios focados em resolver apenas uma pequena parte do negócio.

Figura 12 – Arquitetura proposta para construir os *Micro Frontend*



Fonte: Autoria própria.

A Figura 12 apresenta algumas divisões separadas por blocos e cores. A primeira coluna (da esquerda para direita) nomeada de Micro Front-end: Principal, será responsável por identificar usuário (*login*), registrar e orquestrar as demais micro aplicações, resultando na primeira base de código. Para a segunda coluna, são apresentadas as micro aplicações e suas responsabilidades, sendo que cada micro aplicação tem a sua base de código e não conhece ou troca informações entre outras aplicações da mesma camada. Para a terceira coluna, está a biblioteca *NPM* para os componentes, biblioteca de utilitários para construir *Micro Frontends* e uma aplicação de *Micro Frontend* usada para compartilhar os dados do usuário logado, nomeada de MFE: utfpr core shared.

A arquitetura proposta na Figura 12 também prevê dependências entre essas aplicações, sendo que a aplicação principal conhece todas as micro aplicações e os pacotes *NPM*. Os *Micro Frontends* não conhecem a aplicação principal, mas utilizam as bibliotecas e as informações do usuário logado.

No desenvolvimento dessa aplicação, foram utilizadas as bibliotecas/*frameworks* React com Typescript, Webpack, Material UI e Netlify. No decorrer desse capitulo será abordado

como essas ferramentas são utilizadas para construir a arquitetura de *Micro Frontend* proposta neste estudo. Nas seções a seguir, são apresentadas as ferramentas e tecnologias utilizadas para implementar a arquitetura proposta.

4.3.1 Webpack

O Webpack é um empacotador de arquivos (ou módulo) estático para aplicativos JavaScript (WEBPACK, 2022b). Ao configurar o Webpack, ele permite utilizar as funcionalidades mais recentes da linguagem e ao empacotar o projeto, ele converte o código para fornecer arquivos JavaScript que sejam compatíveis com diversos navegadores. A Figura 13 apresenta um exemplo de configuração do Webpack para um projeto utilizando React com TypeScript.

Para executar o processo de compilação, o Webpack utiliza o NodeJS para conversão de código. Durante o processo de empacotamento do projeto, o Webpack cria internamente um gráfico de dependência com os arquivos de entrada. A partir da árvore de dependências são combinados os módulos do projeto em vários pacotes, que são arquivos estáticos para servir o conteúdo (WEBPACK, 2022b). Essa abordagem permite que os arquivos JavaScript sejam carregados somente quando usados pela aplicação.

O arquivo de configuração para o Webpack é nomeado como *webpack.config.js*; sendo assim, ao fazer análise em um projeto e procurar essas configurações, é necessário buscar por nomes semelhantes. Ao utilizar o Webpack como empacotador, pode-se usar as funcionalidades mais recentes do *EcmaScript*[6].

A configuração mais trivial do Webpack é composta por um arquivo de entrada, uma pasta de saída, um template *HTML* e o modo de compilação, como desenvolvimento ou produção (MEZZALIRA, 2021). Na Figura 13 é apresentado como essas configurações essenciais são organizadas. Na linha 4, é definido qual será o arquivo de entrada para o projeto. Já nas linhas 13 até 20 são definidas as regras dos arquivos JavaScript, que são responsáveis por configurar como serão compilado esses arquivos. Na linha 25 é configurado qual será o modelo *HTML* utilizado pelo projeto.

Para permitir que uma aplicação desenvolvida utilizando Webpack possa ser incorporada para compor uma aplicação maior, *e.g.*, *Micro Frontend* que são integrados, é necessário utilizar um plugin desenvolvido pelo Webpack chamado de federação de módulo (ou *Module Federation*). Na Seção a seguir são apresentados quais problemas o plugin resolve e como será aplicado no desenvolvimento do projeto.

[6] ECMAScript é a especificação e o JavaScript é uma implementação por parte dos navegadores.

Figura 13 – Configuração do Webpack para um projeto utilizando React

```
import HtmlWebpackPlugin from 'html-webpack-plugin'

const webpackConfig = {
  entry: './src/index.tsx',
  mode: 'develop',
  output: {
    publicPath: 'auto',
    clean: true,
  },
  module: {
    rules: [
      {
        test: /\.(js|jsx|ts|tsx)$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env', '@babel/preset-react', '@babel/preset-typescript'],
          },
        },
      },
    ],
  },
  plugins: [
    new HtmlWebpackPlugin({ template: './public/index.html' })
  ],
}

export default webpackConfig
```

Fonte: Adaptado de Webpack (2022b).

4.3.2 Integração de módulos com Webpack

A federação de módulo é uma funcionalidade incluída a partir da versão 5 do Webpack, e tem como objetivo permitir o compartilhamento de dependências entre aplicações. A motivação da federação de módulos é permitir que várias compilações separadas formem um único aplicativo, mesmo que essas compilações sejam desenvolvidas e implantadas individualmente (WEBPACK, 2022a).

Os módulos do Webpack são divididos em dois tipos, módulos locais e remotos. Os módulos locais fazem parte da compilação do projeto. Módulos remotos não fazem parte da compilação do projeto e são carregados a partir de um *Micro Frontend* chamado contêiner em tempo de execução (WEBPACK, 2022a). Ao implementar a federação de módulo, a aplicação vai carregar outros módulos em tempo de execução. Geralmente esses módulos são carregados através da URL do arquivo JavaScript compilado.

Dentre as configurações mais relevantes para a federação de módulo se destaca a configuração de quais bibliotecas serão compartilhadas entre aplicações. Com esta configuração as dependências que são utilizada por diferentes *Micro Frontends* serão carregados apenas uma única vez, ou seja, quando uma aplicação for carregar uma dependência caso ela já tenha sido

carregada por outra aplicação o *Webpack* deverá utilizar essa referência do carregamento evitando que o mesmo pacote seja baixado várias vezes (WEBPACK, 2022a). Por exemplo, se cada micro aplicação tem como dependência a biblioteca *React*, apenas uma compilação dessa biblioteca será carregada. Desta forma quando esse pacote é carregado por qualquer *Micro Frontend* as demais aplicações irão usar essa referência, deixando assim o compartilhamento e o carregamento de desses pacotes mais eficientes.

Figura 14 – Configurações essenciais para os módulos federados

```
import { container } from 'webpack'

function moduleFederation() {
  return new container.ModuleFederationPlugin({
    name: 'AppShell',
    filename: 'remoteEntry.js',
    remotes: {
      AppPerfil: 'AppPerfil@http://localhost:3001/remoteEntry.js',
      AppVisita: 'AppVisita@http://localhost:3002/remoteEntry.js',
    },
    shared: {
      react: { singleton: true },
      axios: { singleton: true },
    },
  })
}

export default {
  plugins: [moduleFederation()],
}
```

Fonte: Adaptado de Mezzalira (2021).

Uma das partes mais importantes do estilo *Micro Frontend* é configurar a federação de modulos para carregar os *Micro Frontend* remotos (MEZZALIRA, 2021). Como a aplicação principal é responsável por orquestrar outras aplicações, é necessário que cada micro aplicação remota seja registrada. Dessa forma, a orquestração dos *Micro Frontend* será feita em tempo de execução. A Figura 14 apresenta um exemplo dessa configuração, que por sua vez utiliza alguns atributos, descritos mais a seguir:

* **name**: responsável por armazenar o nome da aplicação federada.

* **filename**: responsável por definir o nome do arquivo que será exportado ao módulo federado durante o processo de compilação.

* **remotes**: responsável por definir qual o identificador e URL do arquivo JavaScript que foi exportado durante o processo de compilação, esses dois valores são separados pelo símbolo

@. Na linha 8, está sendo apresentado o exemplo de uma aplicação AppPerfil e será acessada por meio da URL http://localhost:3001/remoteEntry.js.

* **shared**: responsável por definir quais as bibliotecas são compartilhadas com outros *Micro Frontend*. Na linha 12, está sendo configurado que o biblioteca *React* será compartilhada e terá apenas uma instancia de carregamento, ou seja, essa dependência será compartilhada entre todas as micro aplicações.

Ao construir soluções de software usando a arquitetura de *Micro Frontend* será necessário ter uma aplicação principal (ou *Shell Application*). Essa aplicação está presente durante toda a sessão do usuário e tem a responsabilidade de orquestrar os *Micro Frontends* vinculados ao subdomínio de negócio (MEZZALIRA, 2021). A configuração de aplicações para um subdomínio de negócio é bem semelhante à configuração da aplicação principal, a Figura 15 apresenta um exemplo dessa configuração.

Figura 15 – Configuração do módulo para um subdomínio de negócio

```
1  import { container } from 'webpack'
2
3  function moduleFederation() {
4    return new container.ModuleFederationPlugin({
5      name: 'AppPerfil',
6      filename: 'remoteEntry.js',
7      exposes: {
8        './UserRoutes': './src/app/RoutesApp.tsx',
9      },
10     shared: {
11       react: { singleton: true },
12       'react-dom': { singleton: true },
13     },
14   })
15 }
16
17 export default {
18   plugins: [moduleFederation()],
19 }
```

Fonte: Adaptado de Webpack (2022a).

Outra configuração essencial para os módulos federados no contexto de negócio, apresentado na Figura 15, consiste no atributo a seguir:

* **exposes**: Atributo responsável por definir quais partes do projeto serão exportados na federação e indica como eles serão acessados. Na linha 8, está sendo definido que ao carregar ./UserRoutes o arquivo que será carregado está no diretório ./src/app/RoutesApp.tsx, para esse exemplo esse arquivo contém as configurações de rotas do *React*.

4.3.3 Registro de Pacotes do Gitlab

Do mesmo modo que o NPM, o registro de pacotes (ou *Package Registry*) do Gitlab possibilita registrar as bibliotecas de pacotes, que podem ser utilizadas como dependências em outros projetos. Com objetivo de compartilhar componentes e lógicas de código comum entre diferentes micro aplicação, foram criadas duas novas bibliotecas *NPM* que foram publicadas na propria infraestrutura de pacotes do *GitLab*.[7]

O valor usado para o *@scope* é a raiz do projeto que vai acabar hospedando os pacotes e não a raiz do projeto com o código fonte do próprio pacote (GITLAB, 2022). Para publicar os pacotes, é necessário configurar o arquivo *package.json*. É essencial que o nome do projeto tenha o escopo correto e seja incluido o campo de *publishConfig*, apresentado como exemplo na Figura 16.

Figura 16 – Configurar arquivo package.json para pacotes NPM

```
{
  "name": "@utfpr/utfpr-lib-ui-react",
  // ...
  "repository": {
    "type": "git",
    "url": "git+ssh://git@gitlab.com/utfpr/utfpr-lib-ui-react.git"
  },
  "publishConfig": {
    "@utfpr:registry": "https://gitlab.com/api/v4/projects/1234/packages/npm/"
  }
}
```

Fonte: Adaptado de GitLab (2022).

Os escopos são uma maneira de agrupar pacotes relacionados e também afetam a maneira como o NPM trata o pacote. Cada usuário/organização NPM tem seu próprio escopo e somente esse usuário pode adicionar pacotes ao seu escopo (NPM, 2022a). Os escopos dentro do GitLab são nomeados quando um grupo é criado. Dessa forma ao criar um grupo temos o nome do escopo definido, com isso basta incluir as configurações no arquivo *package.json*.

Para instalar os pacotes privados é necessário realizar autenticação. Com base na autenticação, apenas os pacotes que o usuário possui acesso poderão ser instalados, seguindo o formato *@scope/nome-pacote*. Para auxiliar a autenticação usa-se o arquivo chamado *.npmrc*, localizado na raiz do projeto, apresentado na Figura 17. Vale resaltar que o *token* não deve ser adicionado ao controle de versão nem armazenado de forma insegura. Como o *token* tem permissão para

[7] GitLab: é uma empresa de núcleo aberto que desenvolve software para auxiliar o ciclo de vida de desenvolvimento de software.

ler pacotes privados, publicar novos pacotes em nome do usuário, ou alterar configurações de usuário, é importante manter essa informação protegida (NPM, 2022b). A Figura 17 apresenta um exemplo dessa configuração, onde o trecho `CI_JOB_TOKEN` é uma variável de ambiente ao qual foi definido o *token* de autenticação.

Figura 17 – Configurar arquivo .npmcr para instalar pacotes

```
@utfpr:registry=https://gitlab.com/api/v4/packages/npm/
registry="https://gitlab.com/api/v4/packages/npm"

//gitlab.com/api/v4/packages/npm/:_authToken=${CI_JOB_TOKEN}
```

Fonte: Adaptado de GitLab (2022).

Para permitir que novas versões do pacote sejam implantadas de forma transparente e o processo de publicação de novas versões ocorram de forma contínua, a cada modificação é necessário a configuração de fluxos de trabalhos. Os ambientes de implantação contínua geralmente envolvem a criação de um artefato de implantação. Portanto, convém criar um *token* somente de leitura para uso em um ambiente de integração contínua (NPM, 2022b), como apresentado na Figura 18.

Figura 18 – Fluxo de trabalho para publicar uma nova versão do pacote

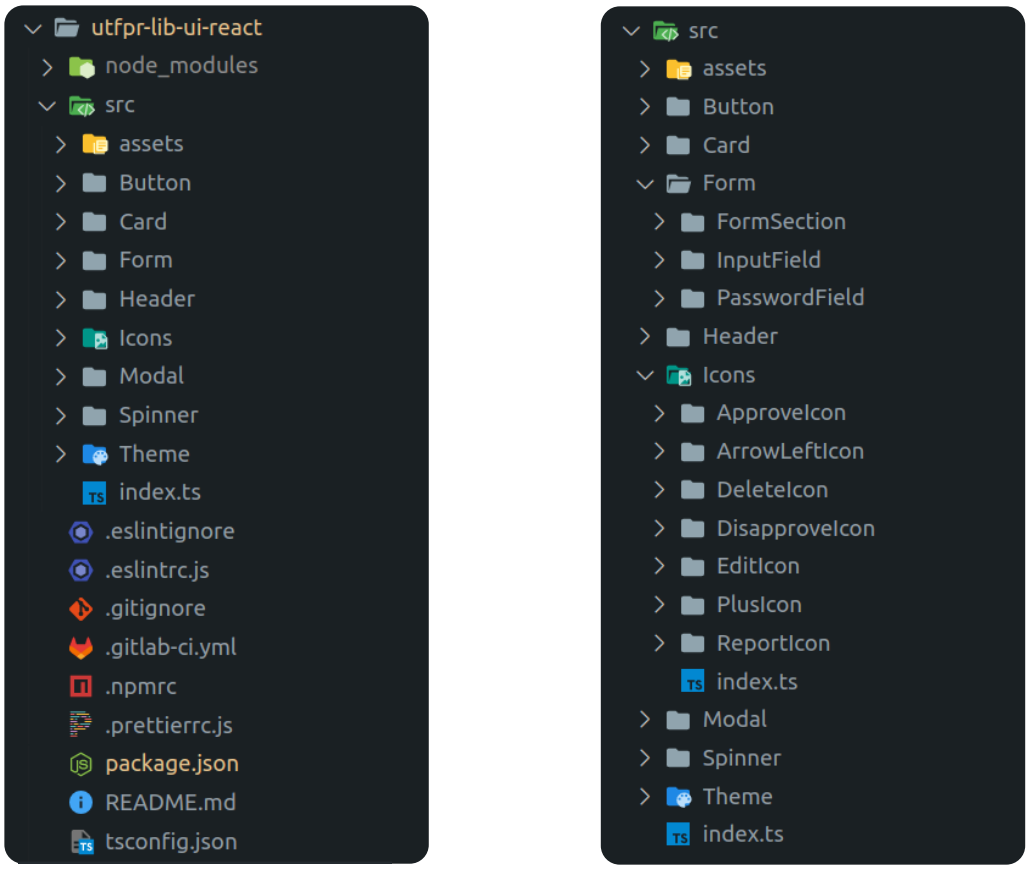
```
image: node:latest

install-job:
  stage: build
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"
    - npm install
  artifacts:
    paths:
      - node_modules/

npm-publish:
  stage: deploy
  script:
    - CI=false npm run build
    - echo "//${CI_SERVER_HOST}/api/v4/projects/${CI_PROJECT_ID}/packages/npm/:_authToken=${CI_JOB_TOKEN}">.npmrc
    - npm publish
    - echo "-- publicação concluída com sucesso"
  dependencies:
    - install-job
  only:
    - main
```

Fonte: Adaptado de GitLab (2022).

Durante o processo de implementação da arquitetura de *Micro Frontend* foi necessária a criação de novas bibliotecas para dar suporte ao desenvolvimento das aplicações. Essas bibliotecas estão publicadas no escopo *@utfprfabricadesoftware*. Para instalar as bibliotecas é necessário solicitar acesso a universidade, e após o acesso ser concedido pode ser instalada usando os comandos diretamente no terminal:

- **Biblioteca de componentes**: *npm install @utfprfabricadesoftware/utfpr-lib-ui-react*;
- **Biblioteca de utilitários**: *npm install @utfprfabricadesoftware/utfpr-tools-react*.

Figura 19 – Componentes que foram movidos para biblioteca



Fonte: Autoria própria.

A Figura 19, apresenta os componentes que foram movidos do projeto SPA para biblioteca de componentes, e que estão sendo instaladas pelas aplicações de *Micro Frontend* detalhados nas seções seguintes. Neste pacote estão armazenados componentes simples, que são usados como base para comportar a interface do usuário. Os componentes não possuem lógicas ou regras de integração de serviços, são estruturas focadas em estilização.

No proximo capitulo, é discutido como as configurações de implementação contínua foram aplicadas no desenvolvimento de cada micro aplicação.

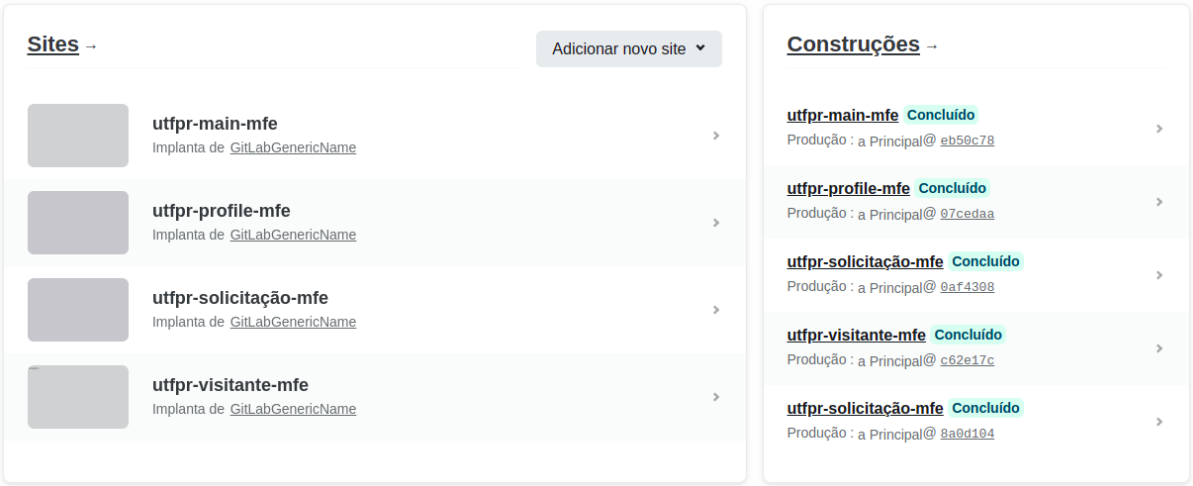
4.3.4 Integração Contínua com Netlify

A partir da configuração dos diversos *Micro Frontends*, é proposta a integração contínua desses componentes. Ou seja, a partir de uma nova alteração no código fonte, realizado na ramificação principal, seja iniciado o processo de empacotamento de uma nova versão, e por fim disponibilizada essa versão ao cliente.

O serviço usado para realizar a integração dos *Micro Frontends* foi o Netlify[8], plataforma que entrega um conjunto bom de configurações mesmo no plano gratuito. Tem integração com os repositórios *Git* mais utilizados do mercado e permite a personalização do subdomio para cada novo projeto.

A cada alteração é necessario verificar se a instalação de dependências e o *build* do projeto aconteceram sem nenhuma falha para garantir que a versão publicada não contém falhas. As validações são configuradas usando os próprios recursos do *GitLab* onde o repositório do projeto é armazenado. Caso as validações sejam bem sucedida, uma nova versão pode ser disponibilizada ao usuário. Neste caso, a plataforma Netlify vai fazer uma cópia do repositório e implantará aplicação automaticamente. A cada modificação feita no código fonte, uma nova implementação será executada automaticamente.

Figura 20 – Configuração de pipeline executada a cada modificação



Fonte: Imagem extraída da plataforma netlify.com.

Na plataforma, é possível definir qual será a URL para acessar o projeto, bem como as variáveis de ambiente usadas ao construir novas versões. Todas as configurações podem ser feitas através da interface gráfica da ferramenta, e cada *Micro Frontend* desenvolvido pelo estudo de caso será implementado utilizando a URL correspondente:

- ***Micro Frontend* de utilitários**: *https://**utfpr-core-shared-mfe**.netlify.app/*;

[8] https://www.netlify.com

- ***Micro Frontend* de Perfil**: *https://**utfpr-profile-mfe**.netlify.app/*;
- ***Micro Frontend* de Solicitações**: *https://**utfpr-solicitation-mfe**.netlify.app/*;
- ***Micro Frontend* de Liberações**: *https://**utfpr-admit-mfe**.netlify.app/*;
- ***Micro Frontend* de Visitas**: *https://**utfpr-visitor-mfe**.netlify.app/*;
- ***Micro Frontend* Principal**: *https://**utfpr-main-mfe**.netlify.app/*;

Para cada *Micro Frontend* é necessário definir as variaveis de ambiente que serão aplicadas durante o processo de compilação, sendo elas:

- `CI_JOB_TOKEN`: *token* de autorização para permitir instalar as bibliotecas registradas no GitLab;
- `REACT_APP_API_URL`: URL utilizada para realizar as chamadas de API;
- `REACT_APP_SHARED_DOMAIN`: Dominio que será usado para carregar os *Micro Frontend*;

O atributo `REACT_APP_API_URL` é configurado com o endereço para acessar o *Backend* da aplicação, a saber *https://utf-io-staging.herokuapp.com*. Já o atributo chamado `REACT_APP_SHARED_DOMAIN` está sendo definido com o valor *netlify.app* pois todos os *Micro Frontend* estão no mesmo domínio.

O atributo `CI_JOB_TOKEN` pode ser criado utilizado o passo a passo definido na documentação do GitLab.[9] Ao usar NPM com GitLab CI/CD, um *token* de trabalho de integração contínua poderá ser usado em vez de um *token* de acesso pessoal ou *token* de implantação (GITLAB, 2022).

Essas são as configurações necessárias para que toda a implementação de cada micro aplicação aconteça de forma independente, e sempre que uma alteração é integrada à ramificação principal. Os *Micro Frontend* que foram criados para a implementação do estudo de caso, são apresentados nas próximas seções e estão utilizando as configurações do Netlify para disponibilizar os arquivos JavaScript estáticos.

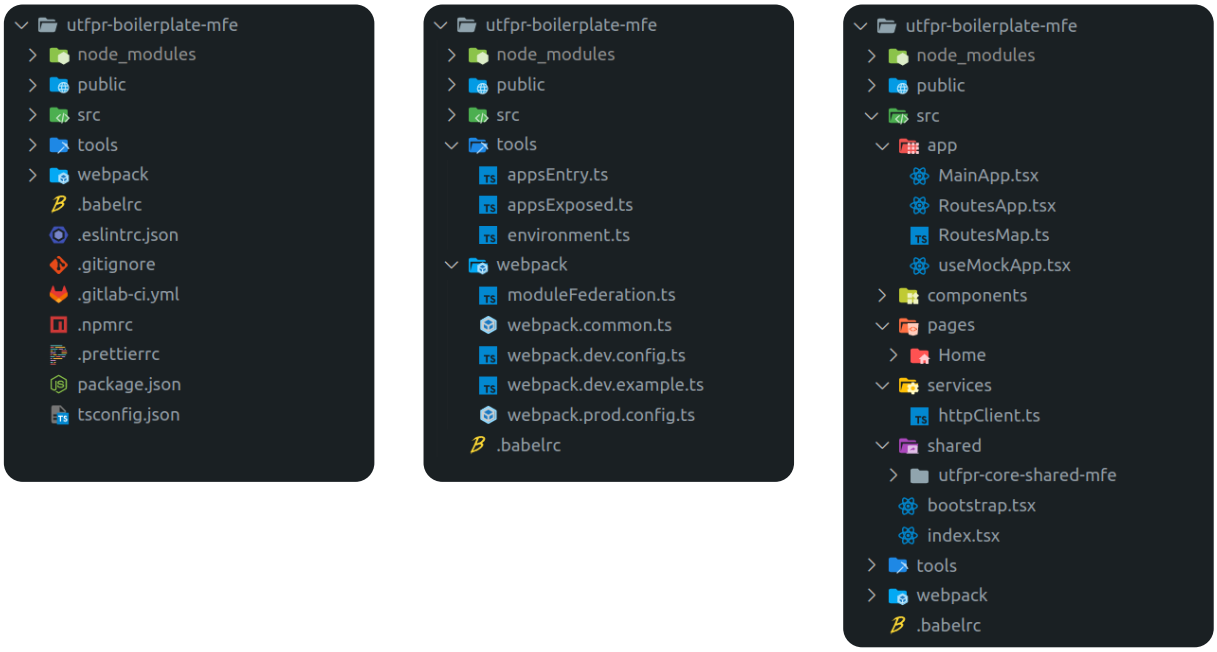
4.4 *Micro Frontend* para início rápido

Com propósito de facilitar a criação de novas aplicações de *Micro Frontend*, foi criado um novo repositório para ser seguido como base de início rápido ao começar um novo projeto, e está disponível no GitLab[10]. Essa Seção apresenta a organização desse projeto. O desenvolvimento dos *Micro Frontends* criados no processo de desenvolvimento desse estudo de caso, utilizando esse repositório como guia de início.

[9] https://docs.gitlab.com/ee/user/packages/npm_registry/#authenticate-with-a-ci-job-token
[10] https://gitlab.com/utfprfabricadesoftware/utfpr-boilerplate-mfe

Ao usar esse projeto como base, as configurações essenciais do projeto já estão definidas. Para ter um entendimento melhor de como esse projeto está estruturado, a Figura 21 apresenta três organizações do projeto. A primeira coluna da esquerda apresenta uma visualização quais as pastas e arquivos que estão localizados na raiz do diretório, que são eles:

Figura 21 – Organização para *Micro Frontend* de início rápido



Fonte: Autoria própria.

- `node_modules`: Essa pasta estão as dependências instaladas pelo projeto, essa pasta pode ser excluída, pois o seu conteúdo é sempre recriado usando as informações presentes no arquivo `package.json`;
- `public`: Nessa pasta está localizado o arquivo index.html que será usado como modelo de renderização ao iniciar o projeto;
- `src`: Essa pasta é onde todo o código fonte do projeto é armazenado;
- `tools`: Essa pasta está sendo usada como diretório de apoio para as configurações do Webpack;

Na segunda coluna (no centro), na pasta `tools` estão sendo definidas as variáveis de ambiente através do arquivo environment.ts, e essas definições são usadas apenas no ambiente local do programador. O arquivo appsEntry.ts está sendo usado para definir quais *Micro Frontends* remotos serão usados no projeto. Já o arquivo appsExposed.ts é usado para definir quais arquivos poderão ser acessados e qual o seu identificador.

Na pasta `webpack` estão os arquivos de configurações usadas pelo Webpack, no arquivo webpack.common.ts estão as configurações que são usadas pelos ambientes de desenvolvimento

e produção. Assim sendo, o arquivo webpack.dev.config.ts é apenas as configurações usadas no desenvolvimento local e o arquivo webpack.prod.config.ts estão as configurações usadas durante o processo de *build* para publicar novas versões. Já o arquivo moduleFederation.ts é responsável por definir as configurações que serão usadas para construir a federação de módulo.

Na terceira coluna a direita está sendo apresentado o directório `src` onde está centralizado o código fonte do projeto. O arquivo de entrada para aplicação é o arquivo index.tsx que por sua vez vai carregar a configurações iniciais definidas no arquivo bootstrap.tsx. A pasta `app` é destinada a armezenar as configuração de tema, rotas e configurações necessárias para que o projeto seja executado em ambiente local, como é exemplo do arquivo useMockApp.tsx que simula as informações de login para um usuário hipotético.

A pasta `services` é usada como ponto de acesso para realizar integrações com serviços. Na Figura 21, o exemplo proposto é de realização de chamadas *API* com o *Backend*. A pasta `shared` tem como intuito configurar a integração com os módulos remotos que serão carregados em tempo de execução.

Vale destacar que essa estrutura é uma proposta sugerida e ao ser implementada deve ser avaliada quais itens são necessários para compor o projeto de *Micro Frontend*. Nas próximas seções será demonstrado o desenvolvimento de cada *Micro Frontend* que foi proposto para esse estudo de caso, alguns desses projetos estão aplicando toda a estrutura apresentada neste capítulo, outros estão utilizando as configurações de forma parcial, de acordo com a organização proposta.

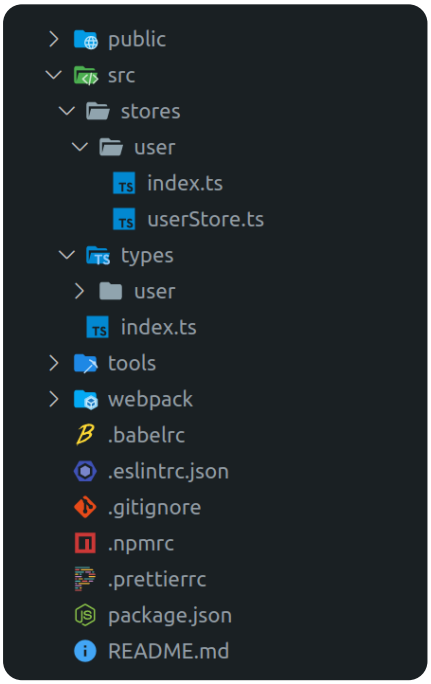
4.5 *Micro Frontend* de dados compartilhados

Ao implementar a arquitetura de *Micro Frontend*, cada aplicação possui os seus dados isolados no seu contexto e geralmente esses dados não são compartilhados com outras aplicações. Uma exceção a essa ideia são as informações do usuário logado, que devem ser usadas por outras aplicações. A escolha usada para compartilhar os dados do usuário, foi criar um *Micro Frontend* para que os dados sejam acessados e modificados. Com isso, qualquer outro *Micro Frontend* pode ter acesso as informações do usuário que estão sendo compartilhadas ao fazer login no sistema.

A Figura 22 apresenta a estrutura dessa aplicação, uma organização muito semelhante aos demais *Micro Frontend* que foram desenvolvidos durante a implementação da aplicação. A maior diferença desse projeto está relacionado ao diretório `src` que possui apenas informações de tipagem e armazenamento de informações.

Essa aplicação está configurada com o nome `utfpr_core_shared_mfe`. Nesse primeiro momento a única informação que pode ser acessada desse *Micro Frontend* está nomeada como `UserStore`. A publicação está configurada no Netlify com a URL https://utfpr-core-shared-mfe.netlify.app. Sendo assim, para acessar os dados de usuário a configuração a ser

Figura 22 – Organização para *Micro Frontend* de dados



Fonte: Autoria própria.

incluida no Webpack como módulo remoto (ou *remotes*) deve ter a URL https://utfpr-core-shared-mfe.netlify.app/remoteEntry.js.

As informações armazenadas por essa aplicação são os dados do aluno, professor, vigilante, DESEG e o *token* de acesso. Esses dados são retornados ao fazer uma chamada ao *Backend* quando o login e senha são informados corretamente. Para permitir que esse dados sejam modificados, os seguintes modificadores podem ser usados:

- `updateUser`: função responsável por atualizar os dados do usuário. Será usada geralmente após o usuário fazer login, a função recebe por parâmetros as informações do usuário;
- `resetUser`: função responsável por limpar os dados do usuário logado. Será usada geralmente após a sessão do usuário expirar ou quando usuário selecionar a opção desconectar na interface gráfica;
- `getRegistrationNumber`: função responsável por acessar o número do registro do usuário logado. Essa informação é exibida na página de perfil de cada usuário.

A Figura 23 apresenta o exemplo de como usar os dados armazenados em outras aplicações de *Micro Frontend*. Na linha 5 é apresentado um exemplo da configuração para módulos remotos usando o Webpack, nessa configuração está sendo definido o identificador `uttpr-core-shared-mfe` que será usado para acessar os dados que foram expostos por esse módulo. Na linha 10, os dados do usuário que foram expostos pelo *Micro Frontend* são

Figura 23 – Acessar os dados do usuário usando *Micro Frontend* de dados

```
1  /**
2   * Configuração de módulos Webpack:
3   * ModuleFederationPlugin({
4   *  remotes: {
5   *    'utfpr-core-shared-mfe': 'utfpr_core_shared_mfe@https://utfpr-core-shared-mfe.netlify.app/remoteEntry.js',
6   *  },
7   * }),
8   */
9
10 import useUserStore from 'utfpr-core-shared-mfe/UserStore'
11
12 export function useUserStoreExample() {
13   const {
14     token, pessoa, professor, vigilante, deseg, redirectAuth, getRegistrationNumber, updateUser, resetUser,
15   } = useUserStore()
16 }
```

Fonte: Autoria própria.

obtido pelo identificador `UserStore`. A linha 14, apresenta as informações que podem ser usadas ao executar o método importado (na linha 10).

A Seção a seguir apresenta o processo de desenvolvimento para o *Micro Frontend* de perfil de usuário e as responsabilidades aplicadas ao projeto.

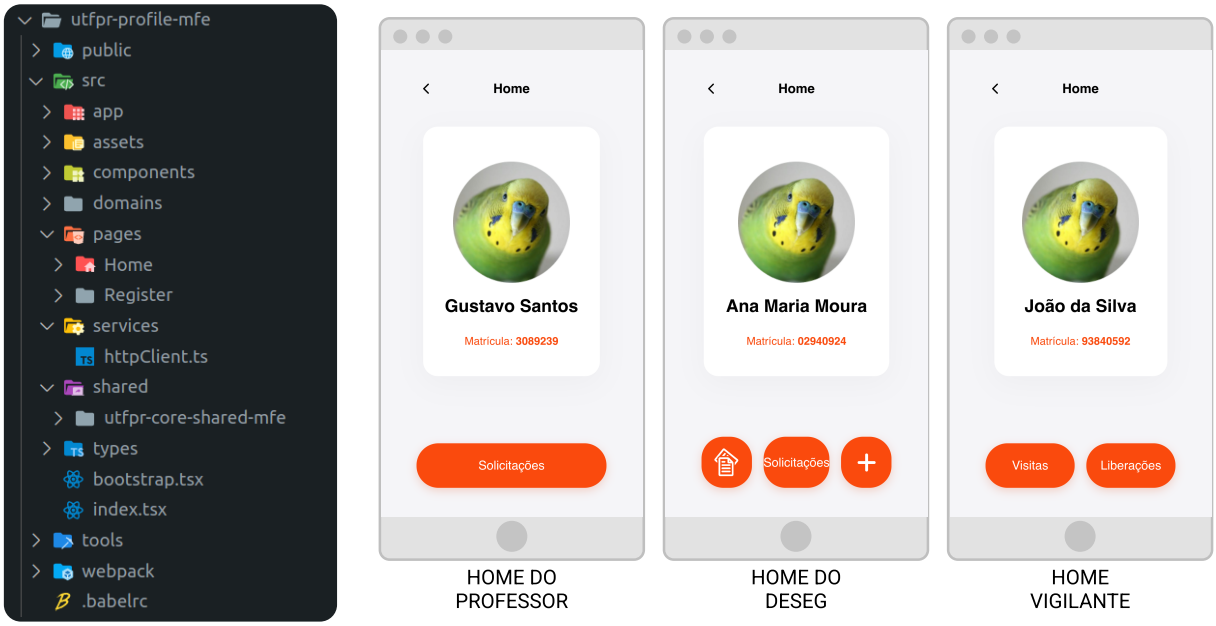
4.6 *Micro Frontend* de perfil

Com o intuito de apresentar as informações do usuário logado, foi criado um novo repositório no GitLab com a responsabilidade de exibir as informações do usuário logado e os *links* de navegação que levam para outros *Micro Frontends*. Desta forma, cada perfil de usuário tem opções de navegações distintas, desta forma, cada usuário pode acessar apenas as informações que são definidas para o seu perfil. Esse *Micro Frontend* está configurado para ser executado na rota */usuario* por padrão, sendo que o valor da rota pode ser alterado em tempo de execução enquanto a aplicação é utilizada.

Nessa funcionalidade (perfil), o propósito é apresentar as informações do usuário e solicitar relatório. A funcionalidade de cadastrar um novo usuário está disponível desde que o perfil de acesso tenha essa permissão. Na regra atual apenas os usuários DESEG tem esse recurso disponível. Para que essa aplicação seja executada, ela tem como dependência o *Micro Frontend* `utfpr_core_shared_mfe` e o pacote NPM `utfpr-lib-ui-react`. A Figura 24 apresenta a estrutura aplicada ao *Micro Frontend* de perfil. Para compor a interface visual desse *Micro Frontend* foram necessários os seguintes componentes:

- Um novo componente foi adicionado para **estrutura da página**, sendo responsável por definir como a página é apresentada ao usuário;

Figura 24 – Estrutura aplicada ao *Micro Frontend* de perfil



Fonte: Adaptado de Santos *et al.* (2021).

- Para permitir que os novos usuários sejam adicionados, o **formulário de cadastro** foi copiado do projeto SPA para esse *Micro Frontend*;
- Para permitir que seja solicitado um novo relatório, o **formulário de solicitação** foi copiado do projeto SPA, pois são usados apenas nessa aplicação;

As modificações realizadas no código fonte estão configuradas para serem publicadas usando a ferramenta Netlify. O *Micro Frontend* foi implementado usando o identificador `utfpr_profile_mfe` e o arquivo estático publicado na URL.[11] A comunicação com o *Backend* para buscar e gravar os dados foi mantido via chamadas de *API*. Na Seção a seguir são exibidos os detalhes do processo de desenvolvimento para o *Micro Frontend* de solicitações e as responsabilidades aplicadas ao projeto.

4.7 *Micro Frontend* de solicitações

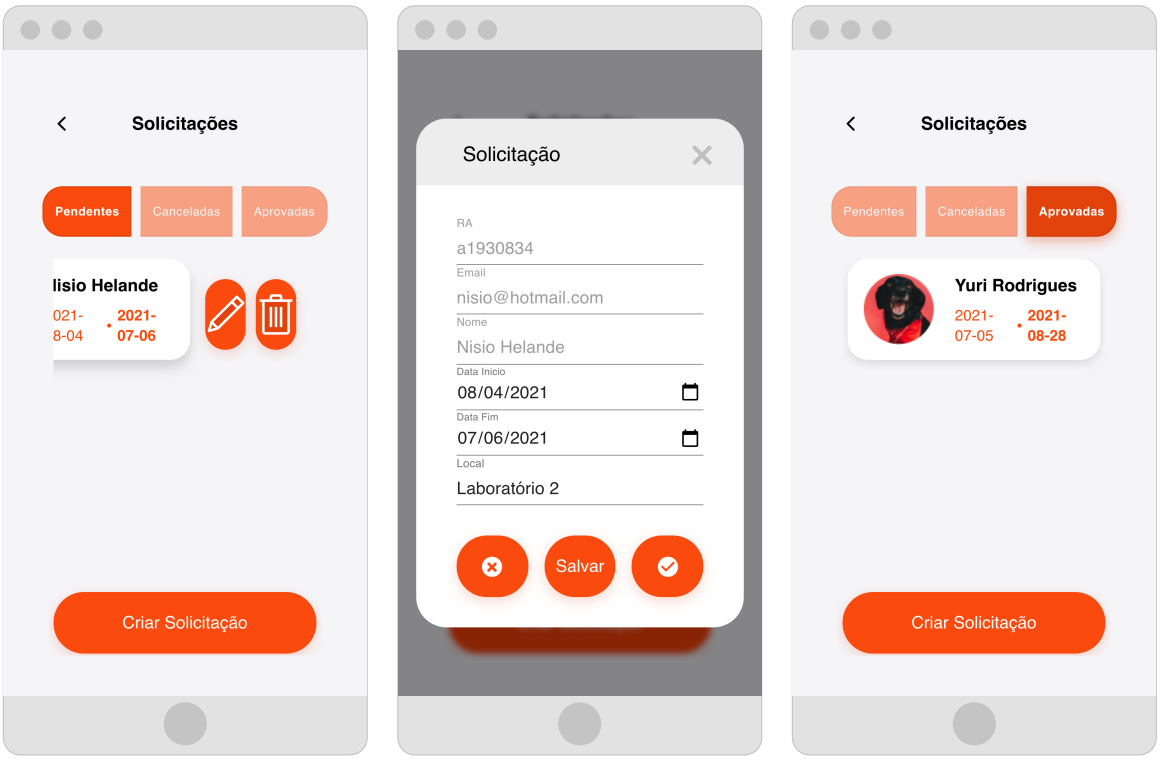
O *Micro Frontend* de solicitações tem como propósito mostrar as solicitações de acesso ao campus, dessa forma, foi fundamental criar um novo repositório no GitLab com o nome `@utfprfabricadesoftware/utfpr-solicitation-mfe`. Ao interagir com a lista de solicitações, é possível consultar os detalhes dessa solicitação. Na interface gráfica tem disponível a opção para adicionar uma nova solicitação usando o formulário de cadastro, os dados a serem incluídos são do aluno. Esse *Micro Frontend* está configurado para ser executado na rota */solicitacoes*.

[11] https://**utfpr-profile-mfe**.netlify.app/remoteEntry.js

A Figura 25 apresenta a interface visual usada pelo usuário vigilante ao consultar e registrar a entrada de um estudantes no câmpus. O *Micro Frontend* de liberações para ser executado tem como dependência o *Micro Frontend* `utfpr_core_shared_mfe` e o pacote NPM `utfpr-lib-ui-react`. Para compor a interface visual desse *Micro Frontend* foram necessários os seguintes componentes:

- Componente para **estrutura da página**, sendo responsável por definir como a página é apresentada ao usuário;
- **Filtro de status** para exibir apenas as solicitações do tipo selecionado. Esse componente foi copiado do projeto SPA para esse *Micro Frontend*;
- Formulário para **cadastro de solicitação**, para permitir que as novas solicitações sejam adicionadas. Esse componente foi copiado do projeto SPA para esse *Micro Frontend*;

Figura 25 – Telas para aprovação de solicitação.



Fonte: Adaptado de Santos *et al.* (2021).

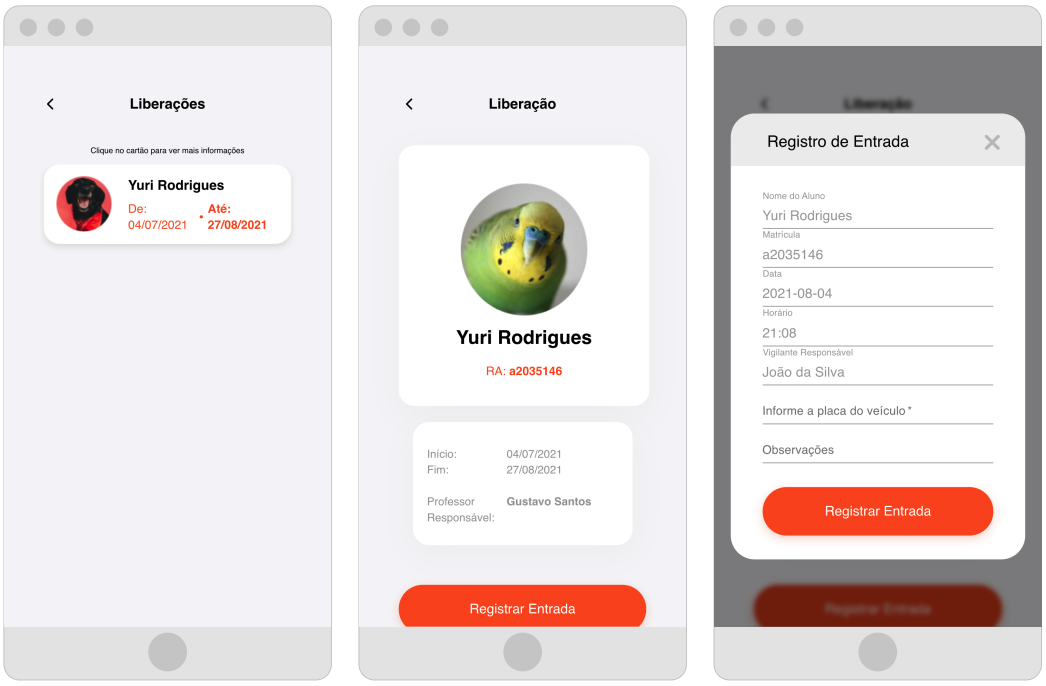
O *Micro Frontend* foi implementado pela aplicação principal usando o identificador `utfpr_solicitation_mfe` e o arquivo estático publicado na URL [12]. A Seção a seguir exibe os detalhes do processo de desenvolvimento para o *Micro Frontend* de liberações e as responsabilidades aplicadas ao projeto.

[12] https://**utfpr-solicitation-mfe**.netlify.app/remoteEntry.js

4.8 *Micro Frontend* de liberações

Com o objetivo de mostrar as liberações de acesso ao campus, foi criado um novo repositório no GitLab com o nome `@utfprfabricadesoftware/utfpr-admit-mfe`. Essa aplicação é responsabilidade de exibir as liberações em forma de lista. Interagindo com a lista, é possível ver as especificidades da liberação e, caso o registro esteja pendente de aprovação, é apresentada a opção para registrar a entrada ao campus. Esse *Micro Frontend* está configurado para ser executado na rota */liberacoes*. A interface visual dessa aplicação é apresentada na Figura 26.

Figura 26 – Interface para registro de acesso ao campus.



Fonte: Adaptado de Santos *et al.* (2021).

A estrutura de componentes que foram incluídas nesse *Micro Frontend* foi o formulário para registrar entrada, e exibir os dados do aluno ao visualizar a liberação. O *Micro Frontend* foi implementado pela aplicação principal usando o identificador `utfpr_admit_mfe` e o arquivo estático publicado na URL [13]. A Seção a seguir apresenta o desenvolvimento do *Micro Frontend* de visitas e as responsabilidades aplicadas ao projeto.

[13] https://**utfpr-admit-mfe**.netlify.app/remoteEntry.js

4.9 *Micro Frontend* de visitas

As visitas de acesso ao campus foram separadas em um nova aplicação, e o repositório no GitLab ficou com o nome `@utfprfabricadesoftware/utfpr-visitor-mfe`. Essa aplicação tem responsabilidade de exibir as visitas em forma de lista. Interagindo com a lista, é possível ver as especificidades da visita e registrar os detalhes do visitante como placa do veículo, data de entrada e saída. Esse *Micro Frontend* está configurado para ser executado na rota */visitas*.

A estrutura de componentes que foram incluídas nesse *Micro Frontend* foi a lista de visitas e o formulário para registrar visita no formato de modal. O *Micro Frontend* foi implementado pela aplicação principal usando o identificador `utfpr_admit_mfe` e o arquivo estático publicado na URL [14]. A Seção a seguir apresenta a implantação dos *Micro Frontends* na aplicação principal.

4.10 *Micro Frontend* principal

A aplicação principal, tem um novo repositório no GitLab e foi nomeada como `@utfprfabricadesoftware/utfpr-main-mfe`. Essa aplicação tem como responsabilidade exibir interface gráfica para realizar login e orquestrar a renderização dos *Micro Frontends*. Nesse momento, começa-se a juntar os diferentes *Micro Frontend* que foram desenvolvidos durante todo o processo de criação. Nessa etapa, a principal meta é fazer a composição de aplicações menores que juntas fornecem uma experiência de sistema único ao usuário. Com essa compreensão, o passo a passo desse processo pode parecer complexo, mas é relativamente simples, como as etapas anteriores.

Para construir a interface de login, pode-se usar a biblioteca com os componentes de base. O pacote NPM `@utfprfabricadesoftware/utfpr-lib-ui-react` será incluído no projeto como dependência. A integração com os serviços para validar as credenciais continua a usar as chamadas *API*. Ao efetuar login com as credenciais válidas, o *Backend* deve retornar as informações do usuário como: *token*, dados do aluno, professor, vigilante e DESEG. Essas informações que foram retornadas precisam ser armazenadas no *Micro Frontend* de dados; e esses dados são armazenados usando o método `updateUser`, disponível quando usado importação `shared/utfpr-core-shared-mfe/UserStore`. A Figura 27 apresenta o processo de login. Após esse processo de autenticação do usuário, os demais *Micro Frontend* podem ser apresentados ao usuário com segurança.

Para registrar os *Micro Frontends* é necessário o identificador da aplicação e a URL do arquivo *JavaScript*. Mais detalhes da configuração foram discutidos na Seção 4.3.2. A Figura 28 apresenta essa configuração: na linha 7, o comentário `ID_INTERNO` está se referindo ao

[14] https://**utfpr-visitor-mfe**.netlify.app/remoteEntry.js

Figura 27 – Processo de login usando *Micro Frontend* de dados

```
1  import useUserStore from 'shared/utfpr-core-shared-mfe/UserStore'
2
3  import { signIn } from 'services/signIn'
4
5  async function handleSubmit({ email, password }) {
6    const { updateUser } = useUserStore()
7    const result = await signIn({ email, password })
8
9    updateUser(result.data)
10 }
```

Fonte: Autoria própria.

identificador que será usado para usar acessar o *Micro Frontend*, o comentário `ID_MFE` está referenciando o identificador usado durante o processo de desenvolvimento, já o comentário `URL_MFE` está se referindo à URL do arquivo estático criado durante o processo de *build* do *Micro Frontend*.

Figura 28 – Registro de *Micro Frontends* remotos

```
1  import { container } from 'webpack'
2
3  function moduleFederation() {
4    return new container.ModuleFederationPlugin({
5      name: 'utfpr_main_mfe',
6      remotes: {
7        // ID_INTERNO          : ID_MFE@URL_MFE
8        'utfpr-core-shared-mfe': 'utfpr_core_shared_mfe@https://utfpr-core-shared-mfe.netlify.app/remoteEntry.js',
9        'utfpr-profile-mfe': 'utfpr_core_shared_mfe@https://utfpr-profile-mfe.netlify.app/remoteEntry.js',
10       'utfpr-solicitation-mfe': 'utfpr_core_shared_mfe@https://utfpr-solicitation-mfe.netlify.app/remoteEntry.js',
11       'utfpr-admit-mfe': 'utfpr_core_shared_mfe@https://utfpr-admit-mfe.netlify.app/remoteEntry.js',
12       'utfpr-visitor-mfe': 'utfpr_core_shared_mfe@https://utfpr-visitor-mfe.netlify.app/remoteEntry.js',
13     },
14   })
15 }
```

Fonte: Autoria própria.

Executando essas configurações, ao fazer importações de arquivos usando os identificadores que foram definidos na coluna de `ID_INTERNO`, os *Micro Frontends* já estão sendo acessados e podem ser usados em qualquer parte do projeto principal.

Durante o processo de desenvolvimento, a escolha feita para gerenciamento de renderizações foi usar as próprias rotas fornecidas pela biblioteca *React Router Dom*. Sendo assim,

quando usuário está navegando entre diferentes páginas, ao mudar de URL o *Micro Frontend* que é renderizado será o configurado para aquele endereço.

Após concluir o desenvolvimento usando arquitetura de *Micro Frontend*, pode ser compreendido que grande parte do esforço para a construção, não está centralizada na escrita de código fonte e sim nos processos de apoio ao desenvolvimento. No próximo capítulo serão apurados os resultados preliminares dessa nova aplicação.

5 AVALIAÇÃO DA ARQUITETURA PROPOSTA

A manutenibilidade é um atributo de qualidade que define a facilidade com que um sistema pode melhorar seus componentes ou como ele se adapta os requisitos em constante mudança.[1] A utilização das métricas tem como objetivo realizar análise comparativa entre o sistema SPA e *Micro Frontend*. Portanto, procura-se usar análise comparativa para identificar se, ao utilizar da arquitetura de *Micro Frontend*, houve um aumento na manutenibilidade do sistema quando comparado ao estilo SPA.

5.1 Planejamento

Em 1991, Oman e Hagemeister da Idaho University projetaram o índice de manutenibilidade para determinar objetivamente a manutenibilidade de um produto com base no código-fonte correspondente (MOUSAVI, 2017). O índice de manutenibilidade é um valor entre 0 a 100 que representa a relativa facilidade de manutenção do código. Um valor alto significa melhor capacidade de manutenção.

O índice de manutenibilidade ou MI (*Maintainability Index*) combina diversas métricas já existentes em uma única métrica, com a finalidade de representar o esforço sobre a manutenção em um único valor (FERREIRA; ARAKAKI, apud SJøBERG *et al.*, 2013). O MI é calculado como uma fórmula fatorada que consiste numa combinação de métricas de número de linhas de código ou SLOC (*Source Lines Of Code*), Complexidade Ciclomática (*Cyclomatic Complexity*) e volume Halstead.[2]

As variáveis elencadas para este experimento estão dispostas a seguir:

C: Complexidade ciclomática média do projeto;
C(MFE): Complexidade ciclomática do código no sistema MFE;
C(SPA): Complexidade ciclomática do código no sistema SPA;

[1] https://www.iso.org/standard/35733.html

[2] https://radon.readthedocs.io/en/latest/intro.html#maintainability-index

D: Dificuldade média para interpretação do código-fonte;
D(MFE): Dificuldade para interpretação do código no sistema MFE;
D(SPA): Dificuldade para interpretação do código no sistema SPA;
T: Tempo médio despendido para a compreensão do código-fonte;
T(MFE): Tempo para a compreensão do código do sistema MFE;
T(SPA): Tempo para a compreensão do código do sistema SPA;
L: Quantidade média de linhas de código para processamento lógico;
L(MFE): Quantidade de linhas de código no sistema MFE;
L(SPA): Quantidade de linhas de código no sistema SPA;

5.1.1 Complexidade Ciclomática

A complexidade ciclomática (CC) é uma métrica de software usada para descrever a complexidade do código. Quanto maior o código-fonte, maior a dificuldade de se entender, modificar e, consequentemente, testar o código-fonte (BERLEZI, 2017 apud MCCABE, 1976). Para efeitos de visualização, *McCabe* utilizou o conceito de grafos para representar os caminhos linearmente independentes, onde cada nó representa um bloco de comando do programa, e as arestas que os conectam representam os caminhos que os fluxos de execução seguem (JUNIOR ALISSON FERNANDES DO PRADO, 2016).

Matematicamente, esse cálculo pode ser expresso como:

$$CC = A - V + 2 \tag{1}$$

CC**:** Complexidade ciclomática;
A**:** Número de arestas;
V**:** Número vértices do grafo;

A métrica mede a quantidade de caminhos linearmente independentes no código. Se a estrutura do código não possui condicionais, então a CC é 1; Se o código apresenta uma condição, então a CC é 2, pois existem dois caminhos possíveis (CARDOSO, 2021). Dessa forma, quanto mais caminhos o trecho de código possui, por meio de estruturas de controle e repetição, mais complexo esse código será.

A complexidade ciclomática pode ser aplicada a funções de programa, módulos, métodos ou classes individuais para identificar sua complexidade. Essa métrica deve especificar a dificuldade de compreensão em um trecho de código.

5.1.2 Dificuldade Halstead

As métricas complexidade de Halstead foram publicadas por Maurice Howard Halstead em 1977 e tem como intuito distinguir as propriedades mensuráveis do código-fonte e as relações entre elas (HALSTEAD, 1977). Quanto menor o valor, maior é a facilidade de compreensão do código; consequentemente, quanto maior o valor, menor é a facilidade (CARDOSO, 2021).

Matematicamente, esse cálculo pode ser expresso como:

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2} \tag{2}$$

η_1**:** Número de operadores diferentes;
η_2**:** Número de operandos diferentes;
N_2**:** Número de total de operandos;
D**:** Dificuldade Halstead de um código-fonte;

Portanto, quanto mais operandos e operadores um trecho de código tiver, como variáveis, parâmetros e operações lógicas, mais difícil será entender o código. Ao diminuir o valor da métrica de dificuldade pode-se, portanto, indicar uma melhoria no código-fonte. Um exemplo de uma expressão contendo operandos e operadores é 3 + 1, onde os números 3 e 1 são operandos, e o sinal de mais é o operador.

5.1.3 Tempo Halstead

Essa métrica foi produzida por *Maurice Halstead* para estimar uma medida quantitativa de complexidade para desenvolvedores (HARIPRASAD *et al.*, 2017). Essa métrica também está diretamente relacionada à dificuldade de leitura e interpretação do código-fonte, e procuram estimar em segundos, quanto tempo um desenvolvedor levaria para analisar um determinado trecho de código. As medições de Halstead dependem da execução do programa e suas medidas, que são analisadas especificamente a partir dos operadores e operandos do código-fonte (HARIPRASAD *et al.*, 2017).

Matematicamente, esse cálculo pode ser expresso como:

$$
\begin{aligned}
N &= N_1 + N_2 \\
\eta &= \eta_1 + \eta_2 \\
V &= N \times \log_2 \eta \\
E &= D \times V \\
T &= \tfrac{E}{18} segundos
\end{aligned}
\tag{3}
$$

N_1**:** Número de total de operadores;
N_2**:** Número de total de operandos;
N**:** Tamanho do código-fonte;
η_1**:** Número de operadores distintos;
η_2**:** Número de operandos distintos;
η**:** Total do vocabulário do programa;
V**:** Volume do código-fonte;
D**:** Dificuldade Halstead de um código-fonte;
E**:** Esforço para compreensão de um código-fonte;
T**:** Tempo Halstead em segundos;

Quanto mais operandos e operadores o código-fonte tiver, a exemplo de variáveis, atribuições, declarações e operadores matemáticos, maior será o tempo para compreender o código. Portanto, ao reduzir o valor da métrica de tempo pode-se, portanto, indicar uma melhoria no código-fonte.

5.1.4 Ferramenta Selecionada

Para realizar análise de manutenibilidade foi escolhida a ferramenta *Plato*.[3] A ferramenta faz análise estática em código-fonte *JavaScript* e tem o seu código aberto. *Plato* é uma ferramennta popular, possuindo 4.5 mil estrelas e em média de 14 mil *downloads* por semana segundo o NPM.[4] Além disso, o *Plato* consegue medir o MI de acordo com a equação que foi proposta nas seções anteriores.

A ferramenta *Plato* é executada via terminal, usando códigos escritos em *Node.js*. Ao final da execução, a ferramenta cria arquivos com o resultado da análise. Esses arquivos podem ser abertos diretamente no navegador, pois o resultado do processamento são arquivos em geral JS, CSS e HTML.

[3] https://github.com/es-analysis/plato
[4] https://www.npmjs.com/package/plato

Figura 29 – Exemplo de configuração ferramenta *Plato*

```
1  const plato = require('plato')
2  const pkg = require('../package.json')
3  const reportToCsv = require('./reportToCsv')
4
5  const src = ['./dist/**/*.js']
6  const outputDir = './metrics/artifacts'
7  const platoArgs = {
8    title: pkg.name,
9    exclude: /(styles.js|types.js)$/,
10   eslint: {},
11 }
12
13 function callback(reports) {
14   const overview = plato.getOverviewReport(reports)
15
16   reportToCsv(overview.reports)
17 }
18
19 plato.inspect(src, outputDir, platoArgs, callback)
```

Fonte: Adaptado de GitHub[5].

Na Figura 29 é apresentado um exemplo da configuração utilizada ao realizar análise de cada projeto. Nas linhas 1 até 3, é realizada a importação de arquivos externos; na linha 5, são selecionados os arquivos que serão analisados; na linha 9, são selecionados os arquivos que serão excluídos da análise; Nas linhas 13 até 17, é definido o método responsável por receber o resultado da análise. Por fim, na linha 19, é executada a ferramenta passando via parâmetros as configurações de inicialização.

A análise feita pela ferramente só pode ser realizada em arquivos com a extensão JS. Como os projetos foram inicialmente desenvolvidos usando *TypeScript*, foi necessário fazer uma conversão desses arquivos usando o formato *EcmaScript5* (ou ES5).

5.2 Validação dos resultados

Para realizar a mensuração da arquitetura proposta neste estudo, foram extraidas as métricas de manutenibilidade para o projeto SPA e comparadas com as métricas dos novos projetos de *Micro Frontends*. Nesse estudo, procura-se comparar as arquiteturas usando as métricas de manutenibilidade para avaliar se houve melhoria no código-fonte.

Para a execução da coleta de métricas usando a ferramenta *Plato*, os arquivos que foram analisados estavam no formato *JS*. Outras extensões de arquivos como JSON, CSS e HTML não foram consideradas pela análise.

As questões a serem respondidas são identificadas como R_1, R_2, R_3 e R_4, bem como suas respectivas métricas e hipóteses:

- R_1) A utilização da arquitetura de *Micro Frontend* reduziu a complexidade do código-fonte produzido?

  Métrica: Uma diminuição no índice de complexidade ciclomática representa uma diminuição na complexidade do código-fonte.

  - **Hipótese nula** (H_0): A aplicação de *Micro Frontend* não reduziu a complexidade ciclomática do sistema.

$$C(MFE) \geq C(SPA) \tag{4}$$

  - **Hipótese alternativa** (H_1): A aplicação de *Micro Frontend* reduziu a complexidade ciclomática do sistema.

$$C(MFE) < C(SPA) \tag{5}$$

- R_2) A utilização de *Micro Frontend* reduziu a dificuldade para compreensão do código-fonte produzido?

  Métrica: A diminuição no índice de Dificuldade de Halstead representa uma diminuição na dificuldade de compreender o código-fonte.

  - **Hipótese nula** (H_0): A utilização da arquitetura de *Micro Frontend* não reduziu a dificuldade de compreensão de código.

$$D(MFE) \geq D(SPA) \tag{6}$$

  - **Hipótese alternativa** (H_1): A utilização da arquitetura de *Micro Frontend* reduziu a dificuldade de compreensão de código.

$$D(MFE) < D(SPA) \tag{7}$$

- R_3) A utilização de *Micro Frontend* reduziu o tempo para compreensão do código?

  Métrica: Diminuição da métrica de Tempo Halstead representa uma melhoria no tempo de compreensão do código pelo desenvolvedor.

  - **Hipótese nula (H_0)**: A utilização de *Micro Frontend* aumentou o tempo de compreensão do código.

$$T(MFE) \geq T(SPA) \tag{8}$$

  - **Hipótese alternativa (H_1)**: A utilização de *Micro Frontend* diminuiu o tempo de compreensão do código.

$$T(MFE) < T(SPA) \tag{9}$$

- R_4) A utilização de *Micro Frontend* reduziu o número de linhas de código-fonte para operações lógicas?

  Métrica: Diminuição da métrica *LLOC* representa uma redução no número de linhas de código para operações lógicas.

  - **Hipótese nula (H_0)**: A utilização de *Micro Frontend* não diminuiu a quantidade de linhas de código para operações lógicas.

$$L(MFE) \geq L(SPA) \tag{10}$$

  - **Hipótese alternativa (H_1)**: A utilização de *Micro Frontend* diminuiu a quantidade de linhas de código para operações lógicas.

$$L(MFE) < L(SPA) \tag{11}$$

Para melhorar conclusão dos resultados, foi aplicado o teste estatístico **Teste T**. O objetivo do teste T de *Wilcoxon* é comparar o desempenho entre duas amostras no sentido de verificar se existem diferenças significativas entre os seus resultados.[6]

Para realizar o **Teste T**, foram utilizadas as métricas extraidas da ferramenta *Plato*. O teste ajuda a determinar se uma amostra da métrica antes das modificações é diferente da amostra depois da utilização da arquitetura de *Micro Frontend*. Portanto, uma amostra consistiu nas medidas coletadas das funções do sistema SPA, e a segunda amostra consistiu nas medidas coletadas de funções do sistema MFE.

Pode-se concluir que duas amostras são estatisticamente diferentes se as amostras tiverem distribuições, variações e médias diferentes. Portanto, ao comparar as amostras elencadas, será viável avaliar se houve uma melhoria com a utilização da arquitetura *Micro Frontend*, de acordo com cada métrica selecionada. Caso o teste não seja conclusivo, então não há evidências que houve melhora ou piora na aplicação de MFE, de acordo com a métrica sob análise.

[6] https://data.library.virginia.edu/the-wilcoxon-rank-sum-test/

5.2.1 Índice de Manutenibilidade (MI)

Na Tabela 2, são apresentados os valores de MI média e SLOC do sistema SPA e cada *Micro Frontend* desenvolvido nesse trabalho. Os valores foram calculados usando a ferramenta *Plato*.

Tabela 2 – Análise comparativa do MI usando ferramenta *Plato*

Arquitetura	Projeto	MI	SLOC
SPA	Sistema Alvo	78,49	58
MFE	Principal com Login	89,75	11
MFE	Perfil	88,93	18
MFE	Solicitações	82,77	26
MFE	Liberações	87,36	16
MFE	Visitas	86,54	16

Fonte: Autoria própria.

Com base nos dados extraídos da ferramenta *Plato*, é possível constatar que a manutenibilidade média dos *Micro Frontend* separadamente é maior em comparação com o projeto SPA. Contudo, a média não é uma medida significativa quando aplicada a métricas de código-fonte. A seguir, as diferenças das distribuições nas métricas de cada projeto serão analisadas em detalhes na próxima seção, de modo a identificar as distinções em cada métrica que compõe o cálculo do MI.

5.2.2 Complexidade Ciclomática (R_1)

Com a finalidade de validar a variância entre as amostras das métricas de Complexidade Ciclomática entre as arquiteturas, foram extraídos os valores das métricas nas principais funções modificados que representam as mesmas funcionalidades em ambos os projetos SPA e MFE.

Após aplicar o Teste de **Wilcoxon** nas amostras coletadas, foi obtido o resultado mostrado no Código 5.1. O teste resultou em um *p-value* maior que 5%; portanto não há evidências para descartar a H_0, ou seja, as distribuições das amostras de CC para os sistemas SPA e MFE possuem médias e variâncias iguais ou aproximadas.

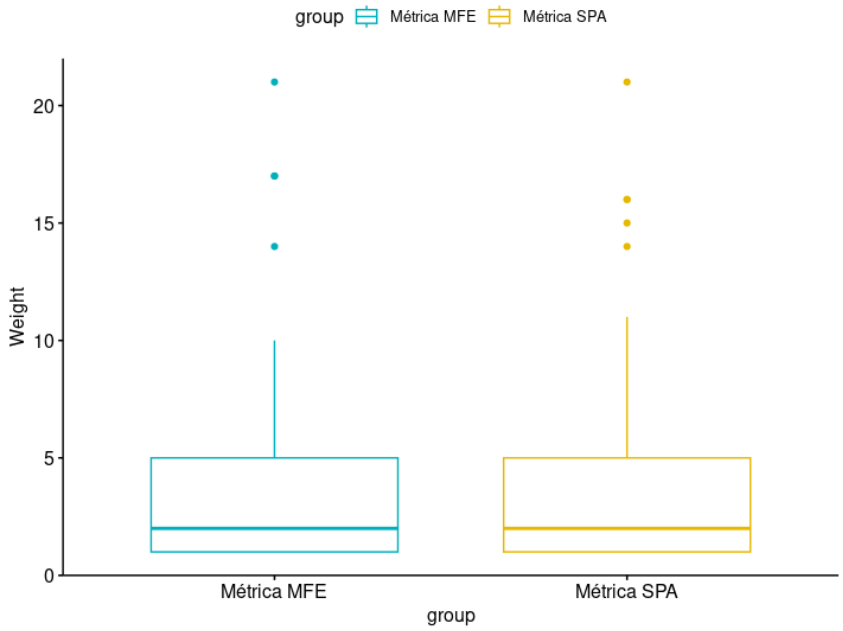
Listing 5.1 – Resultado do teste estatístico da métrica de Complexidade Ciclomática

```
1 Wilcoxon rank sum test with continuity correction
2
3 data:  metricaSpa and metricaMfe
4 W = 511, p-value = 0.6591
5 alternative hypothesis: true location shift is not equal to 0
```

A Figura 30 apresenta o comparativo entre as arquiteturas usando a métrica de Complexidade Ciclomática. Ao analisar a média de CC para o projeto SPA, o valor médio é 4,85; já a

média para arquitetura de *Micro Frontend* foi de 4,64. Nota-se que as médias dessas distribuições ficaram próximas e a variância entre elas é pequena.

Figura 30 – Resultado do teste estatístico da métrica de Complexidade Ciclomática



Fonte: Autoria própria.

Portanto, como p-valor é maior que 5%, não há evidências de que a aplicação da arquitetura de *Micro Frontend* melhorou a complexidade do sistema.

5.2.3 Dificuldade Halstead (R_2)

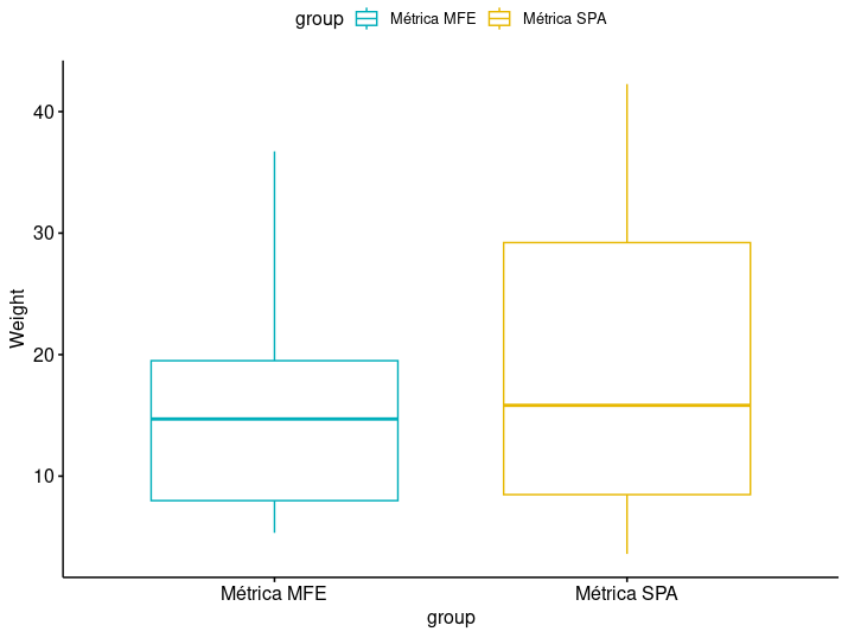
Com a finalidade de validar a variância entre as amostras das métricas de Dificuldade Halstead, após aplicar o Teste de **Wilcoxon** nas amostras coletadas, foi obtido o resultado mostrado no Código 5.2. O teste resultou em um *p-value* maior que 5%; portanto não há evidências para descartar a H_0, ou seja, as distribuições das amostras de Dificuldade Halstead para os sistemas SPA e MFE possuem médias e variâncias iguais ou aproximadas.

Listing 5.2 – Resultado do teste estatístico da métrica de Dificuldade Halstead

```
1 Wilcoxon rank sum test with continuity correction
2
3 data: metricaSpa and metricaMfe
4 W = 627.5, p-value = 0.29
5 alternative hypothesis: true location shift is not equal to 0
```

A Figura 31 apresenta o comparativo entre as arquiteturas usando a métrica de Dificuldade Halstead. Ao analisar a média de dificuldade para o projeto SPA, foi calculado o valor médio de 19; a média para arquitetura de *Micro Frontend* foi de 15,6. Portanto, como p-valor é

Figura 31 – Resultado do teste estatístico da métrica de Dificuldade Halstead



Fonte: Autoria própria.

maior que 5%, não há evidências que o sistema utilizando a arquitetura de *Micro Frontend* teve um índice melhor de Dificuldade Halstead.

5.2.4 Tempo Halstead (R_3)

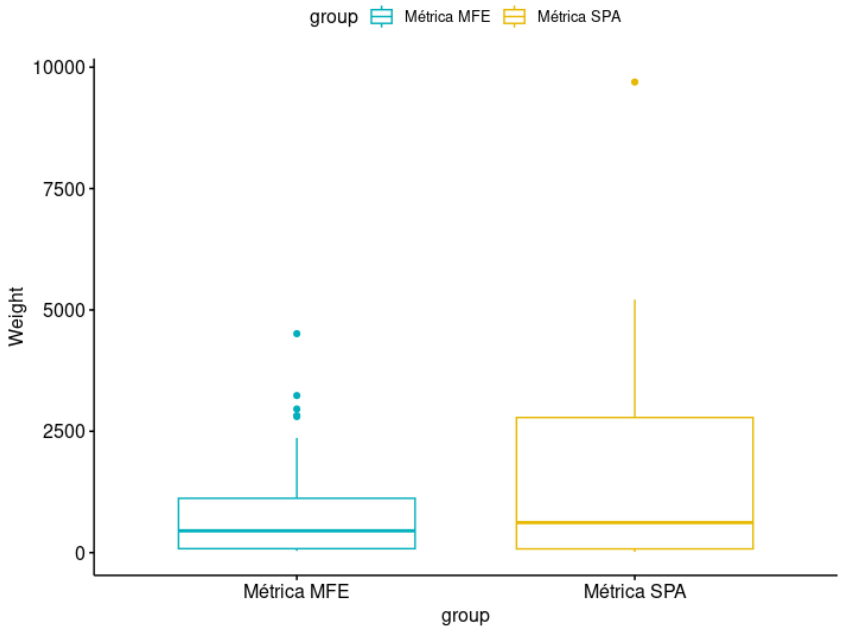
Com a finalidade de validar a variância entre as amostras das métricas de Tempo Halstead, após aplicar o Teste de **Wilcoxon** nas amostras coletadas, foi obtido o resultado mostrado no Código 5.3. O teste resultou em um *p-value* maior que 5%; portanto não há evidências para descartar a H_0, ou seja, as distribuições das amostras de Tempo Halstead para os sistemas SPA e MFE possuem médias e variâncias iguais ou aproximadas.

Listing 5.3 – Resultado do teste estatístico da métrica de Tempo Halstead

```
1 Wilcoxon rank sum test with continuity correction
2
3 data:  metricaSpa and metricaMfe
4 W = 588, p-value = 0.5813
5 alternative hypothesis: true location shift is not equal to 0
```

A Figura 32 apresenta o comparativo entre as arquiteturas usando a métrica de Tempo Halstead. Ao analisar a média de tempo para o projeto SPA, foi calculado o valor médio de 1586; a média para arquitetura de *Micro Frontend* foi de 952. Portanto, como p-valor é maior que 5%, não há evidências que o sistema utilizando a arquitetura de *Micro Frontend*, teve um índice de Tempo Halstead melhor.

Figura 32 – Resultado do teste estatístico da métrica de Tempo



Fonte: Autoria própria.

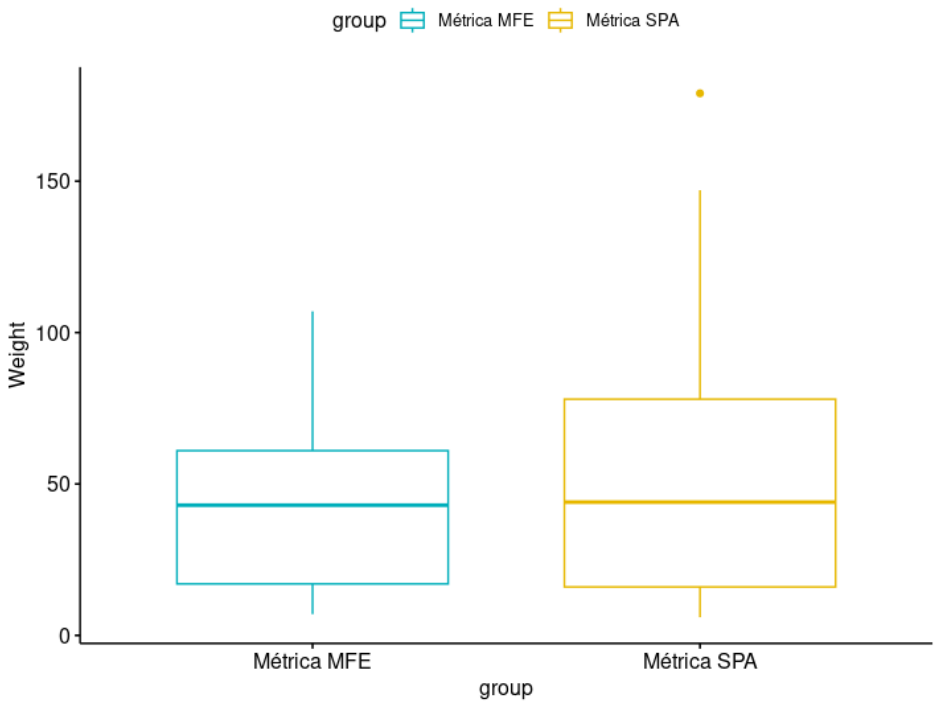
5.2.5 Linhas de Código-Fonte para Processamento Lógico (R_4)

Com a finalidade de examinar a variância entre as amostras das métricas de *LLOC*, após aplicar o Teste de **Wilcoxon** nas amostras coletadas, foi obtido o resultado mostrado no Código 5.4. O teste resultou em um *p-value* maior que 5%; portanto não há evidências para descartar a H_0, ou seja, as distribuições das amostras de LLOC para os sistemas SPA e MFE possuem médias e variâncias iguais ou aproximadas.

Listing 5.4 – Resultado do teste estatístico da métrica *LLOC*

```
1 Wilcoxon rank sum test with continuity correction
2
3 data: metricaSpa and metricaMfe
4 W = 578.5, p-value = 0.6674
5 alternative hypothesis: true location shift is not equal to 0
```

Figura 33 – Resultado do teste estatístico da métrica *LLOC*



Fonte: Autoria própria.

A Figura 32 apresenta o comparativo entre as arquiteturas usando a métrica *LLOC*. Ao analisar a média para métrica de *LLOC* para o projeto SPA, o valor médio calculado é 54; a média para arquitetura de *Micro Frontend* foi de 45,4. Portanto, como p-valor é maior que 5%, não há evidências que o sistema, ao utilizar a arquitetura de *Micro Frontend*, diminuiu o valor de LLOC.

5.3 Discussões

Durante o processo de construção deste estudo de caso foi identificado que grande parte do esforço para utilizar arquitetura de MFE foi para organizar e configurar o processo de publicação dessas micro aplicações. O desenvolvimento utilizando essa arquitetura quando comparada com projeto SPA são muitos similares, gerando uma curva de aprendizado bem pequena.

Portanto, ao arquitetar as configurações da primeira micro aplicações é essencial documentar o processo. Desta forma, é compreensível que as demais aplicações sejam configuradas rapidamente, pois as regras aplicadas serão replicadas. Ao combinar esse processo com a criação de artefatos que automatizam a construção, com o passar do tempo esse custo inicial com as configurações tende a ser reduzido.

Ao avaliar as métricas do estudo de caso usando a arquitetura de *Micro Frontend*, não foi possível identificar uma melhora significativa nas aplicações construídas. As métricas coletadas evidenciaram que houve melhorias pequenas ao fazer a comparação entre as arquiteturas. Porém, por ser uma aplicação pequena as amostras de métricas coletadas não são grandes o suficiente para mensurar essa melhoria expressiva.

O fato da ferramenta *Plato* não ter suporte às novas funcionalidades do *JavaScript*, pode ter contribuído para que os resultados não fossem claros. Para realizar a análise o código-fonte é necessário passar por um processo de compilação, que ao utilizar as funcionalidades mais recentes da linguagem, podem ter resultado em código mais complexos. Assim, o código analisado pela ferramenta sofre várias modificações durante o processo de compilação.

Em alguns experimentos feitos ao usar as funcionalidade mais recentes do *EcmaScript* o impacto na análise era maior, com isso quanto menos funcionalidades recentes eram usadas a métricas eram melhores. Acredito que essa descoberta no final do processo pode gerar resultados com margens de erros maiores.

Desta forma, a coleta de métricas pode ter sido impactada por falso positivos ou falsos negativos. Por ser um campo de pesquisa relativamente novo e que até então eu não havia explorado, acredito que aplicar esse estudo em projetos maiores podem trazer mais evidencias sobre os resultados.

Como trabalho futuros acredito ser apropriado revisar novamente as coletas das métricas para identificar possíveis falhas que podem ter acontecido durante o processo de extração. Portanto, seria ideal pesquisar por outras ferramentas que podem surgir nesse meio tempo, com suporte às novas funcionalidades da linguagem. Assim, ao extrair as métricas dos arquivos escritos em *TypeScript*, sem passar por um processo de *build* podem gerar resultados mais precisos.

Com objetivo de trazer credibilidade na avaliação das arquiteturas, pode ser realizada uma análise avaliativa com outros profissionais trazendo assim mais contexto aos resultados. Desta forma, seria mais adequado argumentar se os sistemas avaliados possuem diferenças distintas no ponto de vista do engenheiro de software.

6 CONCLUSÕES

Por se tratar de padrão arquitetural que está sendo difundindo recentemente, existem poucas referências na literatura sobre *Micro Frontends*, muitas vezes sendo necessário realizar pesquisa sobre o tema em veículos como blogs, podcasts e fóruns. Após revisão da literatura, os autores encontrados apresentaram pensamentos próximos; em geral, os estudos confirmam que a modularização do código em pequenos projetos, responsabilidades bem definidas e entregas mais ágeis são benefícios encontrados ao aplicar o padrão *Micro Frontends*.

Dessa forma, muitos dos benefícios propostos pela arquitetura foram observados em entrevistas e estudos de caso. As motivações e pontos fortes estão alinhados com os princípios do padrão arquitetural. No estado da prática, é possível encontrar diversos exemplos de soluções que foram aplicadas por diferentes organizações e profissionais.

Observa-se que o padrão arquitetural consegue aderir-se a diferentes projetos e soluções. No entanto, existem poucos relatos de como aplicar os conceitos de *Micro Frontends*. Desse modo, publicações em blogs foram utilizadas como base para construir o planejamento do sistema-alvo. Como as apresentações são relacionadas às necessidades de negócio propostas, foi necessário analisar diferentes abordagens para identificar qual seria o formato ideal para ser utilizado no sistema-alvo.

Durante a leitura sobre o tema, foi possível identificar que existem muitas dúvidas e inseguranças de como usar *Micro Frontend* ao desenvolver novos projetos, na maioria das vezes devido a existirem poucos exemplos práticos. Um ponto que chama atenção, é a existência de alguns relatos das organizações e profissionais acreditando que a utilização dessa arquitetura pode ser a solução para os seus problemas. Assim sendo, acredita-se que essa nova arquitetura deve ser adotada por mais profissionais e o compartilhamento desses casos de uso irão auxiliar a comunidade a fazer o bom uso desse estilo, bem como propor novas soluções e ferramentas para os problemas encontrados.

Neste trabalho foi realizada uma comparação entre arquitetura de *Micro Frontend* e SPA. Ao realizar uma análise comparativa entre elas para as métricas de manutenibilidade, a utilização de *Micro Frontend* se mostrou mais eficiente e com um ganho pequeno, em relação ao SPA. O sistema alvo utilizado para implementar essa arquitetura é relativamente pequeno. Porém, ao implementar essa arquitetura em projetos maiores é esperado que as métricas analisadas mantenham-se coerente e os resultados sejam mais expressivos. Portanto, ao utilizar essa arquitetura nos próximos projetos é importante avaliar os princípios que a norteiam, com isso ao escolhê-la deve ser uma escolha consciente.

Espera-se que esse estudo de caso possa contribuir para o desenvolvimento de novas pesquisas e possam ajudar a trazer qualidade ao código-fonte produzido no dia a dia dos desenvolvedores. Melhorar a qualidade interna do software produzido e ajudar a quebrar aplicações *Frontend* monolíticas em partes menores. Contribuir assim para a utilização de *Micro Frontends* por mais profissionais.

REFERÊNCIAS

BERLEZI, R. Análise de métricas de manutenibilidade de um sistema de software de integração: Mulesoft. **V SFCT**, v. 14, 2017.

BERNARDES, H. de F. Monolith first, sorry martin fowler. **LinkedIn Corporação: Rede social de negócios**, 2021. Disponível em: https://pt.linkedin.com/pulse/monolith-first-sorry-martin-fowler-heitor-de-freitas-bernardes.

BIOMEE, M. Micro frontends — extend your microservices backend. **Medium Corporação: Plataforma de publicação online americana**, 2020. Disponível em: https://medium.com/@m.biomee/micro-fronend-extend-your-microservices-backend-part-1-e2d829d5c210.

BRUCHEZ, A. *et al.* Análise da utilização do estudo de caso qualitativo e triangulação na brazilian business review. **Revista ESPACIOS| Vol. 37 (Nº 05) Año 2016**, 2016.

BUSCHMANN, F. **Pattern-Oriented Software Architecture-A System of Patterns**. UFPE: Universidade Federal de Pernambuco, 2006. Disponível em: https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/termdefinitions/architectural_pattern_E2E8EB79.html.

CARDOSO, L. F. A. G. N. Análise da influência de padroes de projeto do gang of four na manutenibilidade de software por meio de experimentos controlados. PUC Minas, p. 7–8, 2021.

CUNHA, G. J. C. d. O uso de sites estáticos como uma opção aos sites dinâmicos para a melhoria no desempenho. 2021.

DRAGONI, N. *et al.* Microservices: yesterday, today, and tomorrow. **Present and ulterior software engineering**, Springer, p. 195–216, 2017.

FERREIRA, B. N.; ARAKAKI, J. Aplicação de métricas de manutebilidade na re-fatoração de softwares.

FINK, G.; FLATOW, I. **Modular JavaScript Development**. Apress, 2014. 35–48 p. Disponível em: https://doi.org/10.1007/978-1-4302-6674-7_3.

FOWLER, S. J. **Microsserviços prontos para a produção**. [*S.l.*]: Novatec, 2017. v. 1.

GITLAB. **Npm packages in the Package Registry**. 2022. Disponível em: https://docs.gitlab.com/ee/user/packages/npm_registry/.

GODBOLT, M. **Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable websites**. [*S.l.*]: O'Reilly Media, 2016.

GUEDES, M. **O que são aplicações SPA**. 2019. Disponível em: https://www.treinaweb.com.br/blog/o-que-sao-aplicacoes-spa.

HALSTEAD, M. **Elements of Software Science**. [*S.l.*: *s.n.*], 1977. 128 p.

HANASHIRO, A. Micro front-end - microserviços no seu navegador. **Treinaweb: Escola online para desenvolvedores**, 2019. Disponível em: https://www.treinaweb.com.br/blog/micro-front-end-microservicos-no-seu-navegador.

HARIPRASAD, T. *et al.* Software complexity analysis using halstead metrics. *In*: **2017 International Conference on Trends in Electronics and Informatics (ICEI)**. [*S.l.*: *s.n.*], 2017. p. 1109–1113.

HYPEFLAME. **Prós e contras na adoção de Microfrontends**. 2021. Disponível em: https://hypeflame.blog/2021/01/06/pros-e-contras-na-adocao-de-microfrontends.

JACKSON, C. **Micro frontends**. 2019. Disponível em: https://microfrontends.com/. Acesso em: 04 de outubro de 2021.

JUNIOR ALISSON FERNANDES DO PRADO, M. A. P. A. Heleno de S. C. Complexity tool: Uma ferramenta para medir complexidade ciclomática de métodos java-complexity tool: a tool for measuring cyclomatic complexity in java methods. **Multiverso: Revista Eletrônica do Campus Juiz de Fora-IF Sudeste MG**, v. 1, n. 1, p. 66–76, 2016.

JUNIOR, J. L. D. Implementando uma página dinâmica com um gerador de sites estáticos. 2018.

KNOCHE; HASSELBRING. **Using Microservices for Legacy Software Modernization**. [*S.l.*]: IEEE Software, 2018. 44–49 p.

LUNA, N. S. A. M. D. Anotações semânticas em serviços web para aprendizagem colaborativa. Série Informática, p. 21–30, 2007.

MCCABE, T. J. **A complexity measure. Em: IEEE Trans. Software Eng.** [*S.l.*: *s.n.*], 1976. 44–49 p.

MEZZALIRA, L. **Building Micro-Frontends: Scaling Teams and Projects, Empowering Developers**. [*S.l.*]: O'Reilly Media, 2021. v. 1.

MONTELIUS, A. **An Exploratory Study of Micro Frontends**. 2021. Dissertação (Mestrado) — Linköping University, 2021.

MOUSAVI, S. **Maintainability evaluation of single page application frameworks: Angular2 vs. react**. 2017.

NEWNAB, S. **Migrando sistemas monolíticos para microsserviços**. [*S.l.*]: Novatec, 2020. v. 1.

NPM. **Instalando pacotes com escopo**. 2022. Disponível em: https://docs.npmjs.com/cli/v8/using-npm/scope.

NPM. **Using private packages in a CI/CD**. 2022. Disponível em: https://docs.npmjs.com/using-private-packages-in-a-ci-cd-workflow.

PAVLENKO, A. *et al.* Micro-frontends: application of microservices to web front-ends. **Journal of Internet Services and Information Security**, v. 10, n. 2, p. 49–66, 2020.

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

PENNA, W. **Arquitetura Monolítica e Microsserviços**. 2020. Disponível em: https://www.zappts.com/blog/arquitetura-monolitica-e-microsservicos/. Acesso em: 13 de Maio de 2020.

REDHAT. **ARQUITETURA DE MICROSSERVIÇOS**. 2018. Disponível em: https://www.redhat.com/pt-br/topics/microservices/what-are-microservices. Acesso em: 13 de Abril de 2018.

SANTOS, A. S. dos *et al.* Experiência do projeto Fábrica de Software em um curso de engenharia de software. *In*: **Anais da 5ª Escola Regional Engenharia de Software (ERES 2021)**. [*S.l.*: *s.n.*], 2021. p. 105–110.

SANTOS, H. B. D. P. Ana Paula dos. **Implementação e Avaliação de uma Ferramenta de Sugestões para Decomposição de Aplicação Monolítica em Microsserviços**. Belo Horizonte: PUC Minas. Disponível em: http://bib.pucminas.br:8080/pergamumweb/vinculos/00008f/00008ffb.pdf.

SARING, J. **4 maneiras práticas de construir micro front-ends**. 2020. Disponível em: https://codeburst.io/4-practical-ways-to-build-micro-frontends-4dc4f0b8a921.

SARITA; SEBASTIAN. **Transform monolith into microservices using docker**. [*S.l.*]: International Conference on Computing, Communication, Control and Automation, ICCUBEA 2017, 2018. 1–5 p.

SILVA, M. S. **HTML5: a linguagem de marcação que revolucionou a web**. [*S.l.*]: Novatec Editora, 2019.

SJøBERG, D. I. *et al.* Quantifying the effect of code smells on maintenance effort. **IEEE Transactions on Software Engineering**, v. 39, n. 8, p. 1144–1156, 2013.

WEBPACK. **Module Federation**. 2022. Disponível em: https://webpack.js.org/concepts/module-federation/.

WEBPACK. **Webpack**. 2022. Disponível em: https://webpack.js.org/concepts/.