UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Luiz Fernando Barbosa Rodrigues

Um Estudo de Caso sobre a Migração para Microfrontends Usando Bit e Module Federation

Recife

2024

Luiz Fernando Barbosa Rodrigues

Um Estudo de Caso sobre a Migração para Microfrontends Usando Bit e Module Federation

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Curso de Bacharelado em Ciência da Computação

Orientador: Vanilson André de Arruda Burégio

Recife

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

R696e Rodrigues, Luiz Fernando Barbosa
Um Estudo de Caso sobre a Migração para Microfrontends Usando Bit e Module Federation / Luiz Fernando Barbosa Rodrigues. - 2024.
36 f. : il.

Orientador: Vanilson Andre de Arruda Buregio.
Inclui referências.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal Rural de Pernambuco, Bacharelado em Ciência da Computação, Recife, 2024.

1. Microfrontends. 2. Arquitetura. 3. Bit. 4. Module federation. 5. Desenvolvimento Web. I. Buregio, Vanilson Andre de Arruda, orient. II. Título

CDD 004

MINISTÉRIO DA EDUCAÇÃO E DO DESPORTO
UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO (UFRPE)
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

http://www.bcc.ufrpe.br

FICHA DE APROVAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO

Trabalho defendido por Luiz Fernando Barbosa Rodrigues às 09 horas do dia 06 de março de 2024, no link https://meet.google.com/pmg-kqcx-qzx, como requisito para conclusão do curso de Bacharelado em Ciência da Computação da Universidade Federal Rural de Pernambuco, intitulado "Um Estudo de Caso sobre a Migração para Microfrontends Usando Bit e Module Federation", orientado por Vanilson André de Arruda Burégio e aprovado pela seguinte banca examinadora:

Vanilson André de Arruda Burégio
DC/UFRPE

Robson Wagner Albuquerque De Medeiros
DC/UFRPE

Agradecimentos

Agradeço de coração a minha família por ter me proporcionado uma educação sólida, oportunidades valiosas e por sempre terem me incentivado a correr atrás dos meus sonhos. Seu apoio inabalável foi fundamental em cada passo deste caminho.

Agradeço à Giulia por ser uma amiga indispensável, presente tanto em minhas conquistas quanto nos desafios enfrentados na faculdade e no desenvolvimento deste TCC. Sua dedicação, apoio e ânimo foram fundamentais especialmente nos momentos mais difíceis, impedindo-me de desistir e me encorajando a continuar, sou eternamente grato por todo o suporte e amizade que me ofereceu.

A todos os meus amigos que conheci na faculdade, meu sincero agradecimento por tornarem essa experiência verdadeiramente inesquecível, em especial a Lucas por sua amizade desde o início da faculdade e por sua constante disposição para ajudar sempre que necessário. Expresso também minha gratidão a Rodrigues por ter proporcionado o Tutoria, sendo fundamental para viabilizar o desenvolvimento deste TCC. Obrigado por todos os momentos compartilhados e por sempre estarem ao meu lado.

Por fim, agradeço ao meu orientador, Vanilson, por sua orientação, paciência e conhecimento, que foram essenciais para a realização deste trabalho.

“A persistência é o caminho do êxito.”
(Charles Chaplin)

Resumo

Na era da internet, o crescimento exponencial das aplicações web e a crescente necessidade de escalabilidade dos sites têm levado os desenvolvedores a buscar novos métodos para atender a esses requisitos. Nesse sentido, a arquitetura de microfrontends tem se tornado popular, pois oferece alternativas no desenvolvimento de frontends web. Uma das vantagens dessa abordagem é a independência das equipes de desenvolvimento. No entanto, conforme a complexidade das aplicações aumenta com o uso de microfrontends, as equipes de desenvolvimento precisam de ferramentas adequadas para gerenciar essa complexidade de forma eficaz. Este trabalho visa analisar as principais ferramentas para a construção de microfrontends, considerando a migração de um projeto monolítico para uma arquitetura de microfrontends. O objetivo é identificar como as ferramentas atuais lidam com os desafios enfrentados durante o desenvolvimento de microfrontends.

Palavras-chave: Microfrontends, Arquitetura, Bit, Module federation, Desenvolvimento Web.

Abstract

In the internet era, the exponential growth of web applications and the increasing need for site scalability have led developers to seek new methods to meet these requirements. In this sense, the microfrontends architecture has become popular as it offers alternatives in web frontend development. One of the advantages of this approach is the independence of development teams. However, as the complexity of applications increases with the use of microfrontends, development teams need suitable tools to manage this complexity effectively. This work aims to analyze the main tools for microfrontends construction, considering the migration from a monolithic project to a microfrontends architecture. The goal is to identify how current tools handle the challenges faced during microfrontends development.

Keywords: Microfrontends, Architecture, Bit, Module Federation, Web Development.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CDN	Content Delivery Network
CMS	Content Management System
CSS	Cascading Style Sheets
API	Application Programming Interface
URL	Uniform Resource Locator

Sumário

1 Introdução

Na era da internet, as aplicações *web* têm crescido consideravelmente e os sites têm cada vez mais complexidade e funcionalidades. Então, para conseguir entregar esses requisitos, os desenvolvedores têm que buscar novas formas de trabalhar e manter a aplicação com grande escalabilidade, performance e confiabilidade.

Com as arquiteturas monolíticas, todos os processos são altamente acoplados e executam como um único serviço. Isso significa que se um processo do aplicativo apresentar um pico de demanda, toda a arquitetura deverá ser escalada. A complexidade da adição ou do aprimoramento de recursos de aplicativos monolíticos aumenta com o crescimento da base de código. Essa complexidade limita a experimentação e dificulta a implementação de novas ideias.(AWS…,)

As arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo (AWS…,). No *backend*, a solução encontrada foi a de microsserviços, uma abordagem para a construção de sistemas distribuídos em que a aplicação é dividida em serviços menores e independentes, cada um executando uma única tarefa ou funcionalidade específica. Cada serviço pode ser desenvolvido, testado e implantado independentemente, o que facilita a escalabilidade e a manutenção da aplicação como um todo (DRAGONI et al., 2017).

A adoção de microsserviços tem sido algo recorrente entre as grandes empresas como Amazon, Netflix, LinkedIn e SoundCloud (TAIBI; LENARDUZZI; PAHL, 2017). Dado às histórias de sucesso ao utilizar microsserviços para resolver os problemas da arquitetura monolítica no *backend*, houve a ideia de trazer esse conceito também para o *frontend*, que enfrenta desafios em relação a escalabilidade e manutenção de código à medida que o projeto cresce e assim veio a ideia de microfrontend (PAVLENKO et al., 2020). Essa abordagem divide a interface do usuário em módulos independentes, cada um com sua própria responsabilidade. Cada um desses módulos podem ser desenvolvidos e hospedados por seus próprios times. Porém por ser um conceito relativamente novo, sendo introduzido em 2016 (TAIBI; MEZZALIRA, 2022), a aplicação dessa arquitetura traz desafios na implementação devido a sua complexidade trazendo novos problemas a serem resolvidos (PAVLENKO et al., 2020):

- Orquestração - como gerenciar o carregamento de aplicações somente quando necessário.
- Roteamento - como gerenciar rotas na aplicação e decidir qual aplicativo carre-

gar.

- Isolamento - como separar e limitar o escopo de cada microfrontend
- Comunicação - como lidar com a comunicação entre diferentes microfrontends
- Consistência de UI/UX - como gerenciar estilos comuns e garantir que a experiência do usuário seja consistente.
- Gerenciamento de dependências - como evitar que uma biblioteca seja carregada mais de uma vez.

Considerando o crescente desenvolvimento de novas soluções para simplificar a implementação de microfrontends, este trabalho tem como objetivo abordar as seguintes questões:

RQ1: Quais são as principais abordagens atuais para a construção de microfrontends?

RQ2: Qual das abordagens está em crescimento e sendo adotada pela comunidade?

RQ3: Como essa abordagem lida com os desafios enfrentados na implementação de microfrontends?

1.1 Justificativa

A arquitetura de microfrontends tem ganhado popularidade, principalmente porque empresas e profissionais estão buscando alternativas na implementação de *frontends web*. Essa abordagem visa escalar os processos de desenvolvimento e impulsionar a inovação em um campo de negócios que está se transformando rapidamente. Uma das vantagens dessa arquitetura é permitir que as equipes sejam autônomas e não dependentes de tecnologias específicas (PELTONEN; MEZZALIRA; TAIBI, 2021). No entanto, conforme a complexidade da aplicação aumenta com a adoção de microfrontends, as equipes precisam de ferramentas adequadas para gerenciar essa complexidade de forma eficaz. A falta dessas ferramentas pode resultar em desperdício de tempo para os desenvolvedores, que podem acabar dedicando mais tempo ao gerenciamento do projeto do que ao desenvolvimento real dos recursos (PELTONEN; MEZZALIRA; TAIBI, 2021). Para isso, torna-se essencial investigar as ferramentas atuais e como elas abordam as dificuldades encontradas na construção de microfrontends.

1.2 Objetivos

- Objetivo Geral:
  - Analisar e implementar uma arquitetura de microfrontends eficiente, baseada em um projeto monolítico, levando em consideração as principais abordagens, vantagens, desvantagens e cenários de utilização.
- Objetivos Específicos:
  - Mapeamento das principais abordagens para a construção de microfrontends.
  - Processo de migração de um projeto monolítico para microfrontends, utilizando a abordagem escolhida.
  - Levantamento das características, vantagens, desvantagens e cenários de utilização da abordagem escolhida.

2 Revisão da literatura

2.1 Conceitos Básicos

2.1.1 Microfrontend

Microfrontend é uma abordagem de arquitetura que consiste em dividir uma aplicação web em partes menores e independentes chamadas de microfrontends. Cada microfrontend é responsável por uma funcionalidade específica do sistema e pode ser desenvolvido, testado e implantado de forma independente. Isso permite que equipes diferentes trabalhem em diferentes partes da aplicação sem interferir no trabalho umas das outras (PELTONEN; MEZZALIRA; TAIBI, 2021).

Existem duas abordagens principais para dividir uma aplicação em microfrontends: divisão vertical e divisão horizontal (MEZZALIRA, 2021) (Figura 1).

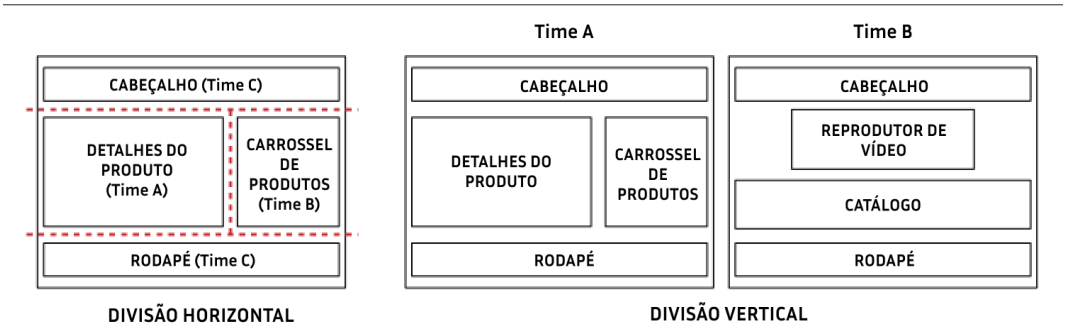

Figura 1 – Divisão horizontal x Divisão vertical, fonte: (MEZZALIRA, 2021)

Na divisão vertical, a aplicação é dividida em microfrontends de acordo com as funcionalidades ou recursos do sistema. Cada microfrontend vai ser uma página específica ou um conjunto de páginas que lidará com alguma funcionalidade do sistema, como por exemplo um fluxo de autenticação.

Na divisão horizontal, a aplicação é dividida em camadas, sendo cada camada representada por um microfrontend. Isso significa que haverá vários microfrontends sendo utilizados para compor uma única página.

Existem três abordagens distintas para a composição de aplicações usando microfrontends(MEZZALIRA, 2021) (Figura 2):

- Client Side Composition (Composição do Lado do Cliente): Nesse tipo de composição, a página é montada no lado do cliente por meio da combinação de dife-

rentes recursos, como HTML, CSS e JavaScript. O cliente carrega os recursos necessários e os combina para formar a página.

- Server Side Composition (Composição do Lado do Servidor): Nesse tipo de composição, a página é montada no lado do servidor antes de ser enviada para o cliente. O servidor combina os diferentes recursos para formar a página e a envia completa para o cliente.
- Edge Side Composition (Composição do Lado da Borda): Nesse tipo de composição, a página é montada a nível de CDN, recuperando os microfrontends da origem e entregando o resultado final ao cliente.

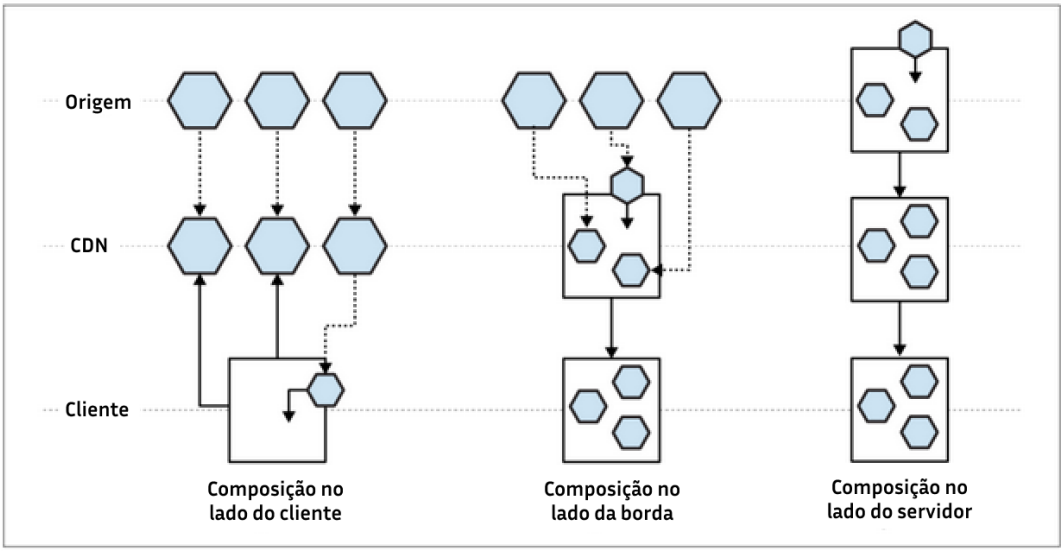

Figura 2 – Diferentes formas de compor microfrontends, fonte: (MEZZALIRA, 2021)

2.1.2 Webpack

O webpack é um agrupador de módulos estáticos para aplicações modernas em JavaScript. Quando o webpack processa a aplicação, ele constrói internamente um gráfico de dependências a partir de um ou mais pontos de entrada e depois combina todos os módulos necessários do projeto em um ou mais bundles, que são ativos estáticos para servir o conteúdo da sua aplicação (WEBPACK,).

2.2 Trabalhos relacionados

O trabalho de (TAIBI; MEZZALIRA, 2022) fornece uma introdução abrangente sobre microfrontends e apresenta várias abordagens para decompor uma aplicação web nessa arquitetura. Ele destaca a importância de decisões críticas, como a escolha

entre composição do lado do cliente ou do servidor, a aplicação de uma divisão vertical ou horizontal, entre outras. O trabalho proposto neste contexto tem como objetivo complementar a teoria com prática, fornecendo um guia detalhado para a migração de uma aplicação existente para a arquitetura de microfrontends. As decisões tomadas durante o processo de migração serão fundamentadas nas ideias apresentadas por (TAIBI; MEZZALIRA, 2022), visando garantir uma implementação eficaz e sustentável da arquitetura de microfrontends.

A pesquisa de (PELTONEN; MEZZALIRA; TAIBI, 2021) tem como foco entender o porquê das empresas adotarem a arquitetura de microfrontend e juntar conhecimento teórico sobre o assunto para discutir sobre os benefícios e desvantagens ao adotar tal método. Nesta pesquisa, foi concluído que o crescimento de uma aplicação monolítica e a necessidade de escalar o projeto para múltiplos times é o principal fator para as empresas adotarem a prática de microfrontend. Apesar deste estudo trazer dados importantes sobre o assunto, ele não discute sobre as ferramentas disponíveis e métodos para a construção de microfrontends.

(YANG; LIU; SU, 2019) apresenta uma abordagem alternativa ao desenvolver um sistema de gerenciamento de conteúdo (CMS) com microfrontends usando o *framework* Mooa. O autor observa que os projetos *frontend* monolíticos enfrentam desafios de escalabilidade e manutenção à medida que crescem. Ele conclui que um CMS baseado em microfrontends possibilita o desenvolvimento independente das equipes, implementação rápida e testes individuais, o que contribui para entregas ágeis de integração e implantação. Entretanto, (YANG; LIU; SU, 2019) destaca algumas limitações dos microfrontends. Primeiramente, o Mooa é restrito ao uso com Angular, não suportando diferentes frameworks. Além disso, a integração de múltiplos subprojetos se torna complexa, exigindo considerações como a segregação do JavaScript, a prevenção de conflitos de CSS e o carregamento de recursos sob demanda. A redundância de dependências entre os subprojetos aumenta a complexidade da gestão. O estudo utiliza o Mooa para a construção de microfrontends, uma framework que não recebe atualizações há 5 anos. Portanto, é importante investigar o estado da arte atual e como o desenvolvimento de microfrontends evoluiu.

Outro estudo relevante, realizado por (PAVLENKO et al., 2020), investigou a implementação de microsserviços no contexto de uma aplicação *frontend*. Neste trabalho, os autores construíram um microfrontend utilizando single-spa.js e discutiram as dificuldades encontradas durante a implementação, como roteamento e gerenciamento de repositórios. Eles também analisaram que o uso de microsserviços é mais adequado para projetos com alta complexidade lógica e que envolvem uma equipe grande de desenvolvedores. Isso ocorre porque o aumento da complexidade no desenvolvimento do projeto é compensado pela redução do esforço de manutenção e da

dependência entre módulos, já que o projeto é dividido em equipes separadas. Em contrapartida, para projetos menores ou com uma equipe reduzida de desenvolvedores, o uso de microsserviços pode não ser tão vantajoso, pois o tempo de desenvolvimento pode ser aumentado devido à complexidade adicional e à necessidade de suporte à arquitetura.

Assim como o trabalho de (PAVLENKO et al., 2020), o estudo de caso de (NASCIMENTO; SOTTO, 2020) também examina a implementação de microfrontends utilizando o single-spa.js, mas com a utilização de três *frameworks* diferentes para compor a aplicação (React, Vue e Angular). Da mesma forma que (PAVLENKO et al., 2020), ele concluiu que, embora a arquitetura de microfrontends traga benefícios como escalabilidade dos componentes e organização do código, em aplicações menores essa abordagem pode introduzir complexidade desnecessária.

Contrariando essas conclusões, o estudo de (BUI, 2021) sugere que a arquitetura de microfrontends pode ser recomendada mesmo no início do desenvolvimento, antes que o sistema cresça muito. Eles realizaram um trabalho de migração de um sistema monolítico para um baseado em microfrontends, em vez de criar um sistema do zero. Isso pode ter facilitado a separação da aplicação em partes menores. Além disso, eles adotaram uma abordagem de multi-framework, utilizando tanto React quanto Vue. O single-spa foi utilizado para a construção dos microfrontends. Por fim, o estudo conclui de forma otimista, destacando resultados promissores e incentivando a exploração e experimentação com outras ferramentas, como Bit ou Webpack Module Federation.

Os trabalhos relacionados que abordaram a construção prática de microfrontends concentraram-se principalmente na utilização do single-spa ou em uma framework baseada no single-spa, como o Mooa. Este trabalho visa explorar mais frameworks e ferramentas atuais para o desenvolvimento de microfrontends, avaliando se as soluções atuais abordam efetivamente os principais desafios enfrentados no desenvolvimento dessa arquitetura.

3 Estudo de Caso de Migração para Microfrontends

3.1 Metodologia

A metodologia utilizada consistiu em um estudo de caso, no qual se realizou uma investigação das principais abordagens para a construção de microfrontends. Além disso, foram descritas as primeiras decisões a serem tomadas ao decompor um projeto monolítico em microfrontends, bem como o processo de desenvolvimento envolvido.

3.2 Mapeamento das abordagens

A abordagem adotada para responder à primeira pergunta de pesquisa, ”Quais as principais abordagens atuais para construção de microfrontends?”, envolveu uma revisão da literatura. O ponto de partida foi a definição de palavras-chave relevantes, incluindo ”microfrontends”, ”arquitetura modular”e ”frameworks micro front-end”. Esta busca foi conduzida por meio de diversos recursos, tais como artigos científicos, ferramentas de busca, vídeos, sites, fóruns e postagens em blogs especializados.

O levantamento inicial permitiu identificar uma variedade de abordagens empregadas na construção de microfrontends. Para refinar e selecionar aquelas mais pertinentes ao escopo do estudo de caso, realizou-se uma análise. Os critérios considerados nesse processo incluíram a popularidade das soluções em termos de desenvolvimento, avaliada pelo número de estrelas no GitHub, e a manutenção, observando o período decorrido desde a última atualização desses frameworks no Github (Tabela 1).

Framework	Estrelas github	Última atualização
Single-spa	12.8k	3 semanas
Module Federation	63.9k	2 horas
Bit	17.3k	1 dia
MOOA	847	5 anos
Luigi	764	3 dias
Piral	1.6k	2 dias
Open components	41	5 dias
Mosaic Work	29	10 meses

Tabela 1 – Soluções Micro frontends, dados do dia 13 de janeiro de 2024

Na análise das estrelas no GitHub, destacam-se notáveis discrepâncias entre algumas das principais abordagens, como evidenciado pelos números impressionantes do Single-spa e Bit, registrando 12.8k e 17k estrelas, respectivamente. Esta disparidade sugere uma expressiva popularidade e reconhecimento da comunidade em torno dessas tecnologias específicas.

Em contraste, observa-se que a maioria das outras estratégias examinadas não ultrapassou a marca de 2k estrelas, indicando uma preferência significativa pela adoção do Single-spa e Bit no cenário de desenvolvimento de microfrontends.

Um caso peculiar que vale a pena destacar é o Module Federation, a qual se destaca por ser uma tecnologia intrinsecamente associada ao webpack se beneficiando de sua comunidade. Porem por não possui um repositório GitHub independente, optou-se por utilizar o GitHub do webpack como referência, resultando em uma contagem de 63.9k estrelas. No entanto, vale ressaltar que essa métrica, embora significativa, pode apresentar desafios ao tentar mensurar de maneira precisa a popularidade do Module Federation, uma vez que não dispõe de um repositório específico, dificultando uma comparação direta com as demais estratégias.

Essa análise quantitativa das estrelas no GitHub proporciona uma visão inicial valiosa sobre a popularidade relativa das abordagens para construção de microfrontends.

Além da análise da popularidade, também foi dada atenção à manutenção e atualização contínua dessas tecnologias. Para avaliar esse aspecto, examinou-se as datas das últimas atualizações nos repositórios correspondentes a cada abordagem. Notavelmente, o Bit destacou-se, evidenciando um padrão de atualização diária. As demais abordagens também apresentaram ciclos de atualização regulares, ocorrendo a cada poucos dias ou semanas.

Entretanto, observar-se que tanto o Mosaic quanto o MOAA não apresentaram desempenho tão satisfatório nesse critério. Suas últimas atualizações datam de 10 meses e 5 anos, respectivamente. Essa disparidade levanta preocupações sobre a manutenção contínua dessas tecnologias, especialmente considerando o ritmo acelerado das mudanças no cenário de desenvolvimento. A falta de atualizações recentes pode impactar negativamente a robustez, segurança e compatibilidade dessas abordagens em ambientes de desenvolvimento em constante evolução.

A aplicação dessas métricas permitiu identificar as abordagens mais robustas e amplamente aceitas, que são o Module Federation, o Bit e o Single-spa. No entanto, considerando que o Single-spa já possui um volume significativo de trabalhos acadêmicos investigando-o, para explorar soluções novas, optou-se por utilizar o Bit e o Module Federation para a construção dos microfrontends neste trabalho.

3.3 Desenvolvimento dos microfrontends

3.3.1 Arquitetura

Para este estudo de caso, optou-se por utilizar o Tutoria como projeto base para a migração para a arquitetura de microfrontend. O Tutoria é uma ferramenta de software projetada para auxiliar professores na criação de *feedback* escrito para atividades. Este sistema permite que os professores visualizem de forma simples e organizada as perguntas de cada atividade criada no Sistema de Gerenciamento de Aprendizagem, bem como as respostas dos alunos (FALCAO et al., 2022). Foi desenvolvido utilizando React em conjunto com a biblioteca de componentes Chakra UI. Já apresenta uma estrutura consolidada, com um número significativo de páginas, recursos de internacionalização e integrações essenciais com plataformas como Google Classroom, Moodle, além de uma API própria.

Para iniciar o desenvolvimento dos microfrontends, primeiramente foi necessário analisar o código do Tutoria e suas funcionalidades para determinar quais partes seriam transformadas em microfrontends.

As funcionalidades do Tutoria incluem:

- Uma Landing Page para explicar o que é o Tutoria e qual problema ele resolve.
- Um fluxo de autenticação com telas de login, cadastro, esqueci minha senha e redefinição de senha.
- Uma tela de revisão responsável pela integração com o Google Classroom e o Moodle, listando as turmas que os professores possuem nessas plataformas.
- O fluxo de correção, onde é realizado o processo de correção das atividades, possibilitando a criação de feedbacks personalizados para os alunos.
- Tela de ajuda, com tutoriais de como utilizar a plataforma.
- Tela de configuração
- Tela de política de privacidade

(TAIBI; MEZZALIRA, 2022) propõe quatro decisões chave que devem ser tomadas no início de um projeto de microfrontend:

1. Divisão horizontal ou vertical: A divisão horizontal é mais adequada para sites estáticos como catálogos ou e-commerces. Como o Tutoria é uma aplicação altamente interativa, optou-se pela divisão vertical. Além disso, a divisão vertical

se alinha melhor com o desenvolvimento de software tradicional, permitindo uma transição mais suave.

2. Composição: Foi escolhida a composição pelo lado do cliente, na qual uma aplicação *host* será utilizada para gerenciar o carregamento dos microfrontends.

3. Roteamento: O *host* será responsável pelo roteamento macro entre as aplicações, enquanto cada microfrontend cuidará de suas próprias rotas internas.

4. Comunicação dos microfrontends: Na comunicação entre os microfrontends, será empregado o *Local Storage*, o qual já é utilizado na aplicação original para armazenar o token e as informações do usuário. Assim, não ocorrerão mudanças abruptas na forma como esse processo já opera.

Com base nessas decisões e funcionalidades do Tutoria vão ser criados cinco microfrontends, com um deles atuando como o host principal responsável de carregar os demais microfrontends (Figura 3). Este host contará com um componente de *layout* que engloba todas as páginas, além de conter as telas de ajuda, configurações e política de privacidade. Optou-se por manter essas telas dentro do host, uma vez que são consideradas mais simples e não justificariam a criação de microfrontends separados.

Na tela inicial, acessível pela rota raiz "/", serão carregados os microfrontends da página de entrada (landing page) ou da página de revisão (review page), dependendo se o usuário estiver logado ou não. Para lidar com a autenticação do usuário, será utilizado um microfrontend dedicado, acessível pela rota "/auth", que incluirá as telas de login, cadastro, recuperação de senha e redefinição de senha.

Por fim, na rota "/question", teremos outro microfrontend responsável pelo fluxo de correção, englobando as telas e funcionalidades relacionadas a esse processo.

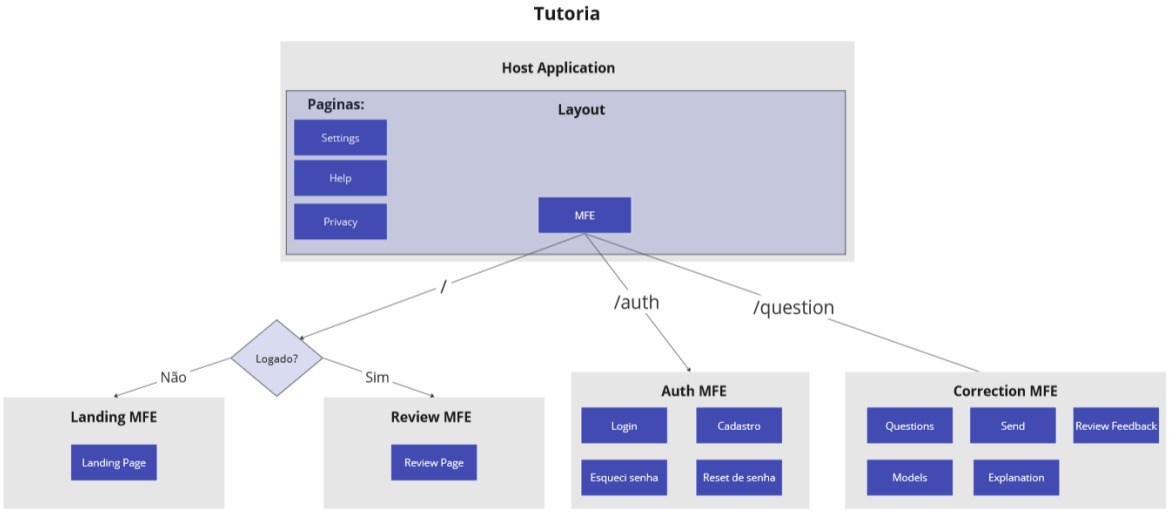

Figura 3 – Arquitetura de microfrontends do Tutoria

Componentes	Hooks
Import-Classrooms	use-user
search-bar	use-sync-host-router
change-lang	
student-tag	
activity-card	
onboard-video-modal	

Tabela 2 – Componentes e hooks criados utilizando Bit

3.3.2 Bit

Para a criação dos componentes e apps foi utilizado o Bit que é uma ferramenta que facilita a criação de software modular, permitindo que os desenvolvedores construam aplicativos combinando componentes independentes. Esses componentes podem ser desenvolvidos, testados e versionados separadamente, o que simplifica o processo de desenvolvimento e manutenção do software (BIT…,).

Por isso o Bit é especialmente adequado para a arquitetura de microfrontends devido à sua capacidade de criar e gerenciar componentes independentes que podem ser compostos para formar aplicativos maiores. A natureza modular do Bit permite que os desenvolvedores construam aplicativos flexíveis e escaláveis, combinando componentes de forma seletiva para atender às necessidades específicas de cada parte do aplicativo.

Para implementar essa arquitetura utilizando o Bit, foi configurado um espaço de trabalho (workspace) usando o comando ”bit init”. Esse espaço de trabalho foi utilizado para criar e manter um conjunto de componentes e aplicativos independentes, e integrá-los entre si.

Como o Tutoria foi desenvolvido em React, optamos por continuar utilizando essa tecnologia para os microfrontends. Para projetos React, o Bit oferece suporte para a criação de diversos tipos de componentes, incluindo Componentes de UI, React Hooks e até mesmo os próprios apps, que se tornarão os microfrontends.

Para evitar a duplicação de código entre as aplicações, foi identificado quais componentes e lógicas seriam compartilhados entre múltiplos microfrontends. Esses componentes e lógicas foram criados de forma independente e podem ser importados e utilizados em diferentes aplicações. Isso foi possível porque o Bit fornece uma interface para visualização dos componentes criados mesmo sem ter que usá-los em uma aplicação, facilitando o processo de extração desses componentes do projeto original, podendo desenvolvê-los e testá-los de maneira independente (Tabela 2).

Após a criação dos componentes necessários, deu-se início ao desenvolvimento das aplicações. Cada microfrontend foi desenvolvido de maneira independente, com

base nas funcionalidades específicas do Tutoria. Esses microfrontends podem ser executados individualmente usando o comando ”bit run app-name”permitindo que cada microfrontend seja desenvolvido e testado separadamente, proporcionando maior flexibilidade e agilidade no processo de desenvolvimento. Assim como os componentes, as aplicações também podem ser exibidas na interface de visualização do Bit, tornando-as acessíveis através do comando ”bit start”. Essa funcionalidade facilita a visualização de cada componente ou aplicativo criado no espaço de trabalho (Figura 4).

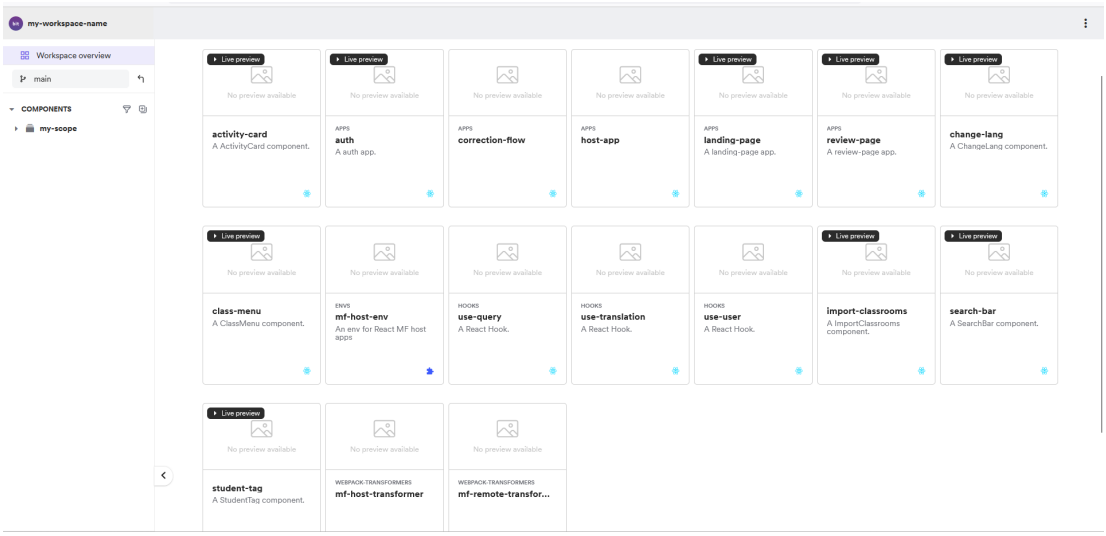

Figura 4 – Visualização dos componentes

Além disso, o Bit oferece a possibilidade de pré-visualização e documentação dos componentes, o que é extremamente útil para o trabalho em equipe, contribuindo para a manutenção e consistência do código e da interface de usuário (UI). Essa capacidade de visualização e documentação simplifica o processo de colaboração, garantindo que todos os membros da equipe tenham acesso às informações necessárias para desenvolver e manter o projeto de forma eficiente.

3.3.3 Module federation

O *module federation* é uma parte essencial do Webpack 5, que foi lançado em outubro de 2020. Ele introduz uma nova maneira de importar fragmentos, recursos ou dependências de outra aplicação implantada de forma independente para uma aplicação. Essa funcionalidade é exportada pelo empacotador Webpack na forma de um plugin chamado ModuleFederationPlugin (SOARES, 2023).

Para utilizar o *Module Federation* no Bit, é necessário um *host* e pelo menos um *remote*. O *host* é considerado a compilação webpack consumidora. Normalmente, é o primeiro que é inicializado durante o carregamento de uma aplicação. Já o *remote* se refere a uma compilação separada, onde parte dela está sendo consumida por um *host* (JACKSON...,).

O *host* foi inicializado a partir de um exemplo fornecido pelo Bit, utilizando o comando ”bit fork learnbit-react.module-federation/apps/host-app”. Este aplicativo *host* usará um transformador do Webpack para incorporar o ModuleFederationPlugin à configuração padrão do Webpack. Um transformador do Webpack é uma função que recebe uma configuração do Webpack e retorna uma versão modificada dela (WEBPACK|BIT…,) (Figura 5).

```
const {
  moduleFederationHostTransformer,
} = require('@my-scope/webpack-transformers.mf-host-transformer');

/** @type {import("@teambit/react.apps.react-app-types").ReactAppType} */
module.exports.default = {
  name: 'mf-host',
  entry: [require.resolve('./mf-host.app-root')],
  portRange: [3020, 3030],
  webpackTransformers: [moduleFederationHostTransformer],
};
```

Figura 5 – Configuração Webpack host

No plugin, serão definidos os aplicativos remotos que serão consumidos pelo *host*, além de especificar as dependências compartilhadas entre as aplicações para evitar a duplicação de dependências. Também será definido o publicPath no transformador, que determina o caminho base para todos os ativos dentro da aplicação *host* (Figura 6).

Para utilizar os microfrontends desenvolvidos, cada um foi configurado para chamar o transformador, permitindo que atuem como aplicações remotas. Um exemplo dessa configuração está no arquivo auth.react-18-app.cjs, apresentado na figura 7. Na primeira parte desse código, encontramos as seguintes definições:

- name: Define o nome da aplicação.
- portRange: Especifica a porta na qual a aplicação será executada.
- entry: Determina o arquivo principal a ser executado quando a aplicação é iniciada. Essas configurações são utilizadas quando a aplicação está rodando de forma independente.

Além disso, no campo webpackTranformers, há configurações específicas para quando o aplicativo é configurado como remoto:

- name: Define o nome da aplicação remota.

```
export const moduleFederationHostTransformer: WebpackConfigTransformer = (
  config
) => {
  config.raw.output.publicPath = 'http://localhost:3020/';
  config.addPlugin(
    new ModuleFederationPlugin({
      remotes: {
        landing_page: 'landing_page@http://localhost:60000/remoteEntry.js',
        auth: 'auth@http://localhost:3000/remoteEntry.js',
        review_page: 'review_page@http://localhost:3001/remoteEntry.js',
        correction_flow: 'correction_flow@//localhost:3002/remoteEntry.js',
      },
      shared: {
        react: {
          singleton: true,
          requiredVersion: '^18.2.0',
          eager: true,
        },
        'react-dom': {
          singleton: true,
          requiredVersion: '^18.2.0',
          eager: true,
        },
        'react-dom/client': {
          singleton: true,
          requiredVersion: '^18.2.0',
          eager: true,
        },
        '@chakra-ui/react': {
          singleton: true,
        },
      },
    })
  );
  return config;
};
```

Figura 6 – Transformer module fedaration

- filename: Determina o nome do arquivo de saída do pacote na Module Federation.
- exposePath: Especifica o arquivo a ser exposto na aplicação remota.
- publicPath: Define o caminho base para todos os ativos do aplicativo remoto.

Essa configuração é passada para ser utilizado pelo tranformador Remoto onde se adiciona o ModuleFederationPlugin (Figura 8).

Com a configuração dos aplicativos remotos finalizada, prosseguiu-se com a integração deles ao *host*. Eles foram importados usando a funcionalidade do ‘react.lazy‘, que é uma função que permite carregar componentes de forma assíncrona, ou seja,

```
const authApp = {
  name: 'auth',
  portRange: [3000, 3000],
  entry: [require.resolve('./auth.app-root')],
  webpackTransformers: [
    (config, ctx) => {
      const remoteConfig = {
        name: 'auth',
        filename: 'remoteEntry.js',
        exposePath: './auth',
        publicPath: 'http://localhost:3000/',
      };

      return moduleFederationRemoteTransformer(
        config,
        ctx,
        require.resolve('./auth'),
        remoteConfig
      );
    },
  ],
};
```

Figura 7 – Configuração Webpack remote

eles são carregados apenas quando são necessários. Isso pode melhorar significativamente o tempo de carregamento inicial do aplicativo, pois apenas os componentes que são necessários no momento são carregados, enquanto os demais são carregados sob demanda.

Em seguida, foi criado o BrowserRouter do react-router no *host*, e as aplicações remotas foram configuradas nas rotas determinadas pela arquitetura (Figura 3). No entanto, os microfrontends de autenticação e correção possuem mais de uma página, o que levou à necessidade de criar um BrowserRouter específico para eles, a fim de gerenciar suas rotas internas.

Essa abordagem enfrentou alguns desafios. Primeiramente, a navegação interna desses microfrontends não alterava a URL da página, pois o ‘BrowserRouter‘ do *host* controla a URL e não consegue detectar mudanças de rota dentro dos microfrontends. Além disso, ao tentar acessar uma rota interna de um microfrontend diretamente por meio de um link, a página correta não era encontrada, novamente devido à limitação do ‘BrowserRouter‘ do *host* em localizá-la.

Para superar esses desafios, foi necessário implementar uma solução que per-

my-scope > webpack-transformers > mf-remote-transformer > TS mf-remote-transformer.ts > ...

```typescript
import type {
  WebpackConfigMutator,
  WebpackConfigTransformContext,
} from '@teambit/webpack';
import { container } from 'webpack';

const { ModuleFederationPlugin } = container;

export const moduleFederationRemoteTransformer = (
  configMutator: WebpackConfigMutator,
  _context: WebpackConfigTransformContext,
  appEntry: string,
  remoteConfig: {
    name: string;
    filename: string;
    exposePath: string;
    productionPublicPath: string;
    publicPath: string;
    sharedPackages?: Record<
      string,
      { singleton: boolean; requiredVersion: string; eager: boolean }
    >;
  }
): WebpackConfigMutator => {
  configMutator.addPlugin(
    new ModuleFederationPlugin({
      name: remoteConfig.name,
      filename: remoteConfig.filename,
      exposes: {
        [remoteConfig.exposePath]: appEntry,
      },
      shared: {
        ...remoteConfig?.sharedPackages,
        react: { singleton: true, requiredVersion: '^18.2.0', eager: true },
        'react-dom': {
          singleton: true,
          requiredVersion: '^18.2.0',
          eager: true,
        },
      },
    })
  );

  configMutator.raw.output = configMutator.raw.output || {};
  configMutator.raw.output.publicPath = remoteConfig.publicPath || 'auto';
```

Figura 8 – Transformador remoto

mitisse atualizar a URL do *host* quando houvesse uma mudança de rota nos microfrontends para isso foi criado dois *hooks*. O primeiro chamado de ”useSyncHostRouter”que foi utilizado nos microfrontends (Figura 9). Essa função recebe como parâmetro um objeto com uma propriedade chamada *basename* que representa o caminho base para as rotas do microfrontend.

A função utiliza os *hooks useLocation* e *useNavigate* do react-router-dom para acessar a localização atual e a função de navegação, respectivamente. A partir da localização atual, é criado um novo caminho baseado no basename e no caminho atual da localização. Este novo caminho será usado para disparar um evento customizado para o *host*.

```tsx
type RouteEvent = CustomEvent<string>;

const useSyncHostRouter = ({ basename }: { basename: string }) => {
  const location = useLocation();
  const navigate = useNavigate();

  const newPath = `${basename}${
    location.pathname === '/' ? '' : location.pathname
  }`;

  useEffect(() => {
    window.dispatchEvent(new CustomEvent('app', { detail: newPath }));
    const appNavigated = ({ detail }: RouteEvent) => {
      if (detail === location.pathname) {
        return;
      }
      navigate(detail);
    };
    window.addEventListener('shell', appNavigated as EventListener);
    return () => {
      window.removeEventListener('shell', appNavigated as EventListener)
    };
  }, [location]);
};

export default useSyncHostRouter;
```

Figura 9 – UseSyncHostRouter baseado no código do vídeo Micro Frontends - Routing between remotes[1]

Em seguida, a função utiliza o *hook useEffect* para adicionar um ouvinte de evento para o evento customizado app que foi disparado pelo microfrontend. Quando o evento é disparado, a função *appNavigated* é chamada e verifica se o caminho recebido é diferente do caminho atual. Se for, ela navega para o novo caminho.

Para evitar duplicação de código esse *hook* foi criado utilizando o Bit com o comando ”bit create react-context context/sync-host-router”e assim ele pode ser usado como pacote em tempo de compilação em qualquer microfrontend.

Para utilizar esse hook no aplicativo remoto, foi criado um *wrapper* em torno do componente <Outlet/> do react-router-dom. Este *wrapper* chamará a função do *hook*, passando o caminho base daquele microfrontend. Esse *wrapper* será utilizado no *router* interno da aplicação (Figura 10).

O segundo *Hook* é o useSyncRemoteRouter que será utilizado no *host* (Figura 11) ele recebe um objeto com uma propriedade chamada *basename* que representa o

[1] https://www.youtube.com/watch?v=uRKUxZQ74os

Figura 10 – Utilização do useSyncHostRouter

caminho base para as rotas dos microfrontends.

No primeiro *useEffect*, ela adiciona um ouvinte de evento para o evento customizado app que é disparado pelo *host*. Quando o evento é disparado, a função *appNavigated* é chamada e verifica se o caminho recebido é diferente do caminho atual. Se for, ela navega para o novo caminho.

Em um segundo *useEffect*, a função verifica se o caminho atual começa com o *basename*. Se sim, ela dispara um evento customizado *shell* que é ouvido pelo *host*. O evento *shell* envia o novo caminho, removendo o *basename*, para que o *host* possa atualizar a URL.

Para utilizar esse *hook* no *host*, foi criado *wrappers* em torno dos apps remotos. Esses *wrappers* chamam a função do *hook*, passando o caminho base dos microfrontends. Esse *wrapper* será utilizado no *router* da aplicação (Figura 12).

Dessa forma, as funções useSyncHostRouter e useSyncRemoteRouter garantem que as rotas dos microfrontends e do *host* permaneçam sincronizadas, garantindo uma experiência de navegação consistente para o usuário. No entanto, identificou-se outro problema: a aplicação de autenticação não conseguia redirecionar para a tela inicial, que deveria ser o microfrontend de revisão. Isso ocorreu porque o router interno da aplicação de autenticação considerava a tela de login como a tela inicial. O microfrontend de revisão foi configurado como tela inicial no router do *host*, o que resultou no problema.

Para resolver essa questão, o hook *useNavigation* do react-router foi passado para os apps remotos via *props*. Isso permitiu lidar com a navegação entre microfrontends (Figura 12).

Com isso resolvido, todas as funcionalidades do Tutoria foram transferidas com

```
type RouteEvent = CustomEvent<string>;

const useSyncHostRouter = ({ basename }: { basename: string }) => {
  const location = useLocation();
  const navigate = useNavigate();

  const newPath = `${basename}${
    location.pathname === '/' ? '' : location.pathname
  }`;

  useEffect(() => {
    window.dispatchEvent(new CustomEvent('app', { detail: newPath }));
    const appNavigated = ({ detail }: RouteEvent) => {
      if (detail === location.pathname) {
        return;
      }
      navigate(detail);
    };
    window.addEventListener('shell', appNavigated as EventListener);
    return () => {
      window.removeEventListener('shell', appNavigated as EventListener)
    };
  }, [location]);
};

export default useSyncHostRouter;
```

Figura 11 – UseSyncRemoteRouter baseado no código do vídeo Micro Frontends - Routing between remotes[2]

sucesso para uma arquitetura de microfrontend, utilizando Bit e *Module Federation*. No próximo capítulo, será visto com mais detalhes os benefícios de ter utilizado essas abordagens.

[2] https://www.youtube.com/watch?v=uRKUxZQ74os

Figura 12 – Utilização do useSyncRemoteRouter

4 Resultados

A sinergia entre Bit e *Module Federation* revelou-se altamente frutífera. O Bit, encarregado da criação e gerenciamento de componentes e aplicações, trabalhou em conjunto com o *Module Federation*, que, por sua vez, foi responsável por compor esses aplicativos em tempo de execução. Isso permitiu a simplificação do desenvolvimento dos microfrontends.

Na implementação de microfrontends, onde cada equipe de desenvolvimento opera de forma independente, comumente enfrenta o desafio de manter uma consistência na interface do usuário (UI) entre as diversas aplicações. Nesse contexto, a capacidade de compartilhar componentes de maneira simples e eficiente torna-se uma necessidade crucial. Nesse sentido, Bit se destacou como uma solução para abordar essa questão, uma vez que permite a criação de componentes facilmente reutilizáveis, que são isolados em ”pacotes”autônomos, tornando sua reutilização em diferentes aplicações uma tarefa simples.

A utilização do Bit simplificou o desenvolvimento de um conjunto de componentes que podem ser facilmente mantidos e compartilhados, promovendo a consistência nos microfrontends. Isso se dá pela disponibilidade de uma interface de usuário fornecida pelo Bit, que permite a visualização individual de cada componente sem a necessidade de integrá-los em uma aplicação específica, utilizando o comando ”bit start”. Nessa interface, é possível acessar uma documentação sobre o uso do componente e uma pré-visualização do mesmo, além das propriedades que esse componente pede (Figura 13).

Essa visualização individual é viabilizada pelo fato de que, ao criar um componente utilizando o Bit, ele já vem com arquivos de pré-visualização denominados ”composições”e um arquivo de documentação no formato MDX para ser utilizada, Além disso, o Bit também gera arquivos .spec.tsx para a criação de testes, facilitando a manutenção futura do componente.

Adicionalmente, o Bit oferece recursos avançados de gerenciamento de versões, possibilitando o versionamento dos componentes de forma granular. Cada vez que se faz uma alteração em um componente, uma nova versão pode ser gerada e registrada no sistema. Isso permite o acompanhamento e a manutenção de um histórico detalhado de todas as mudanças feitas em um componente ao longo do tempo.

Dado que o Bit não executa a composição dos aplicativos em tempo de execução, o *Module Federation* é utilizado para realizar essa função. Ele permite que uma aplicação host consuma os aplicativos criados com Bit de forma dinâmica. Além disso,

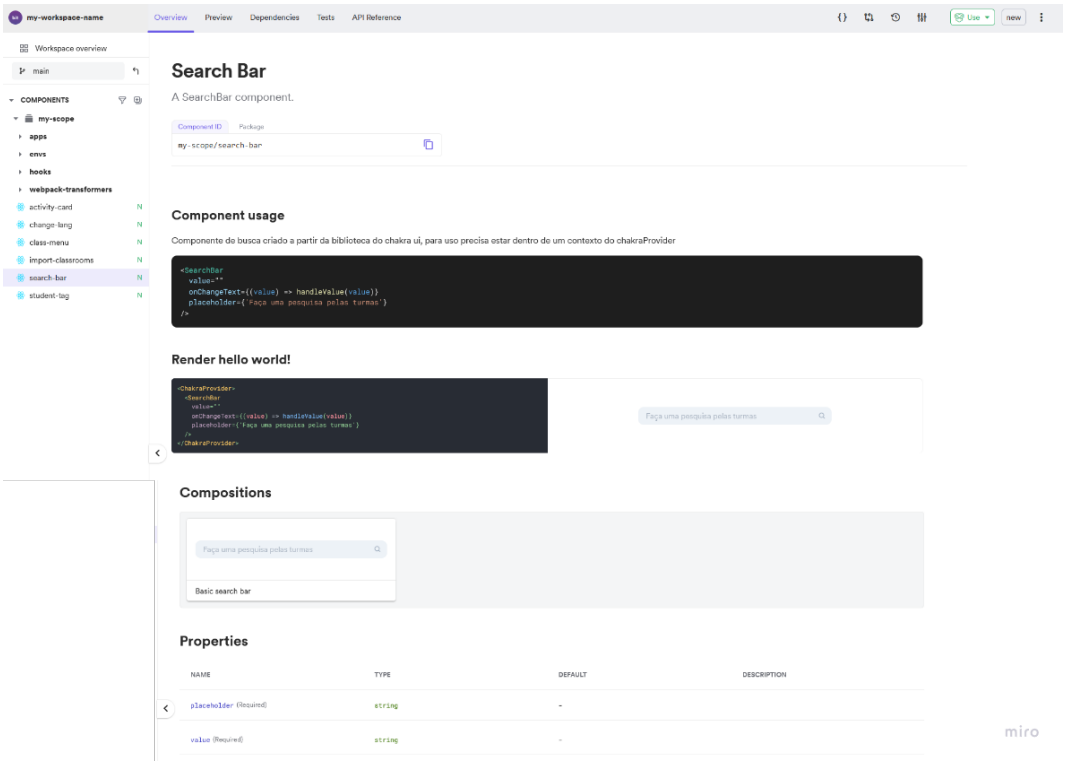

Figura 13 – Pré-visualização do componente

através do plugin do *Module Federation*, é possível compartilhar dependências entre as aplicações, evitando que uma mesma biblioteca seja carregada mais de uma vez. Isso não só melhora o desempenho da aplicação como um todo, mas também reduz o tamanho do *bundle*, o que é crucial para a otimização do tempo de carregamento da página e para a eficiência do aplicativo.

Quando se trata de roteamento, o carregamento dos microfrontends de acordo com as rotas pode ser simplificado através da importação dinâmica oferecida pelo *Module Federation*. Contudo, em cenários mais complexos onde os microfrontends precisam ter suas próprias rotas internas para carregar múltiplas páginas, identificou-se um problema: o host não consegue detectar as mudanças de rota dentro dos microfrontends. Isso torna a configuração mais complexa, pois nem o *Module Federation* nem o Bit oferecem ferramentas para lidar com esse problema.

Para resolver essa questão, foi necessário desenvolver uma solução personalizada para facilitar a comunicação entre o *host* e os aplicativos remotos. No entanto, graças à facilidade de compartilhamento de código proporcionada pelo Bit, foi possível exportar essa solução. Dessa forma, qualquer equipe poderia utilizar esse componente, importando-o em tempo de compilação sem causar impactos significativos no desenvolvimento de futuras aplicações. Essa abordagem promove a reutilização de soluções entre equipes, contribuindo para uma maior eficiência no desenvolvimento de software.

Em suma, a combinação do Bit e do *Module Federation* mostrou-se uma abordagem eficaz para enfrentar os desafios complexos da criação e manutenção de aplicações web modulares e escaláveis. O Bit oferece uma maneira intuitiva de criar e gerenciar componentes, promovendo a reutilização de código e a consistência na interface do usuário. Por outro lado, o *Module Federation* complementa essa abordagem, permitindo a composição dinâmica de aplicativos em tempo de execução e o compartilhamento eficiente de dependências, resultando em uma aplicação mais eficiente e de menor tamanho. Juntos, esses dois recursos oferecem uma solução abrangente para o desenvolvimento de aplicações web modernas e escaláveis, promovendo a colaboração entre equipes e a reutilização de código, ao mesmo tempo que mantêm a flexibilidade e a modularidade necessárias para lidar com as complexidades das aplicações web modernas.

5 Considerações finais

Conforme a internet e as demandas por escalabilidade das aplicações *web* continuam a crescer, a arquitetura de microfrontends emerge como uma solução promissora para enfrentar os desafios de desenvolvimento. Esta abordagem oferece independência às equipes de desenvolvimento, mas também traz consigo desafios de complexidade que precisam ser gerenciados de forma eficaz.

A análise das abordagens e ferramentas para a construção de microfrontends revelou que existem diferentes opções disponíveis no mercado, cada uma com suas próprias características e vantagens (Tabela 1). O Bit e o *Module Federation* surgiram como soluções promissoras, oferecendo uma abordagem modular e escalável para a implementação de microfrontends.

O Bit permite criar e gerenciar componentes independentes, que podem ser combinados para formar aplicativos maiores. Sua natureza modular facilita o desenvolvimento e a manutenção do código, além de permitir a reutilização de componentes em diferentes aplicações. Com o *Module Federation*, por sua vez, introduz a composição eficiente de microfrontends e o compartilhamento de dependências entre diferentes partes de uma aplicação.

Ao utilizar essas abordagens, foi possível migrar com sucesso um projeto monolítico para uma arquitetura de microfrontends. As funcionalidades do Tutoria, projeto utilizado como estudo de caso, foram divididas em cinco microfrontends, cada um responsável por uma parte específica da aplicação. A utilização do Bit e do *Module Federation* no desenvolvimento de microfrontends proporcionou uma série de benefícios, incluindo uma arquitetura modular e escalável, uma abordagem eficiente para compartilhar código entre diferentes aplicações e uma maneira colaborativa de trabalhar em equipe. Essas ferramentas se mostraram valiosas na migração de um projeto monolítico para uma arquitetura de microfrontends, e podem ser uma opção interessante para desenvolvedores que desejam adotar essa abordagem em seus projetos.

Como propostas para trabalhos futuros, é relevante aprofundar a análise comparativa entre distintas ferramentas, abordando aspectos como desempenho, curva de aprendizado, entre outros. Além disso, seria benéfico realizar mais experimentos abrangentes no desenvolvimento de microfrontends, incluindo o desenvolvimento de microsserviços, de modo a explorar o processo de ponta a ponta. Estas pesquisas adicionais têm o potencial de fomentar a contínua evolução das arquiteturas de microfrontends e contribuir para o refinamento das práticas de desenvolvimento de software em geral.

Referências

AWS, O que são microsserviços. <https://aws.amazon.com/pt/microservices/>. Accessado: 03/01/2024. Citado na página 10.

BIT documentation. <https://bit.dev/docs/intro/>. Accessado: 10/02/2024. Citado na página 21.

BUI, S. Micro frontend: Microservice implementation on web development. 2021. Citado na página 16.

DRAGONI, N. et al. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, Springer, p. 195–216, 2017. Citado na página 10.

FALCAO, T. P. et al. What did i get wrong? supporting the feedback process in computer science education. In: SBC. *Anais do XXX Workshop sobre Educaç ao em Computaç ao*. [S.l.], 2022. p. 239–250. Citado na página 19.

JACKSON, Z. <https://www.infoq.com/presentations/module-federation/>. Accessado: 14/02/2024. Citado na página 22.

MEZZALIRA, L. *Building Micro-Frontends*. [S.l.]: ”O’Reilly Media, Inc.”, 2021. Citado 3 vezes nas páginas 6, 13 e 14.

NASCIMENTO, C. P.; SOTTO, E. C. S. Microfrontend: um estudo sobre o conceito e aplicação no frontend. *Revista Interface Tecnológica*, v. 17, n. 1, p. 153–165, 2020. Citado na página 16.

PAVLENKO, A. et al. Micro-frontends: application of microservices to web front-ends. *J. Internet Serv. Inf. Secur.*, v. 10, n. 2, p. 49–66, 2020. Citado 3 vezes nas páginas 10, 15 e 16.

PELTONEN, S.; MEZZALIRA, L.; TAIBI, D. Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review. *Information and Software Technology*, Elsevier, v. 136, p. 106571, 2021. Citado 3 vezes nas páginas 11, 13 e 15.

SOARES, D. V. C. *Analysis of Module Federation Implementation in a Micro-Frontend Application*. Tese (Doutorado), 2023. Citado na página 22.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. *Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing 4, 5 (2017), 22–32*. 2017. Citado na página 10.

TAIBI, D.; MEZZALIRA, L. Micro-frontends: Principles, implementations, and pitfalls. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 47, n. 4, p. 25–29, 2022. Citado 4 vezes nas páginas 10, 14, 15 e 19.

WEBPACK. <https://webpack.js.org/concepts/>. Accessado: 14/02/2024. Citado na página 14.

WEBPACK|BIT.CLOUD. <https://bit.cloud/teambit/webpack>. Accessado: (26/02/2024). Citado na página 23.

YANG, C.; LIU, C.; SU, Z. Research and application of micro frontends. In: IOP PUBLISHING. *IOP conference series: materials science and engineering*. [S.l.], 2019. v. 490, p. 062082. Citado na página 15.