Fragmentando a Interface: O Impacto e os Desafios da Migração para Micro Frontends no Desenvolvimento Web

Lucas Gabriel Rodrigues de Melo[1], **Breno Alexandro Ferreira de Miranda**[1]

[1] Informatics Center – Federal University of Pernambuco (UFPE)
Recife – PE – Brazil

{lgrm,bafm}@cin.ufpe.br

Resumo. *Este trabalho de conclusão de curso investiga a migração de uma aplicação web monolítica para diferentes arquiteturas de micro frontends, utilizando frameworks como Single-SPA, Turborepo e Lerna. A motivação para estudar microfrontends vem da necessidade crescente de melhorar a escalabilidade, modularidade e manutenção de grandes aplicações web, que se tornam cada vez mais complexas à medida que evoluem. Adotar microfrontends permite que equipes de desenvolvimento trabalhem de maneira mais independente e eficiente, integrando diferentes tecnologias e frameworks dentro de uma única aplicação. O objetivo é avaliar o desempenho, escalabilidade e facilidade de manutenção de cada abordagem. A aplicação base foi desenvolvida em Next.js e posteriormente migrada para as três diferentes arquiteturas de micro frontends. A metodologia envolveu testes de desempenho com a ferramenta Lighthouse, que indicaram que Single-SPA apresentou a melhor performance, seguido por Turborepo e Lerna. Testes de escalabilidade com o Apache JMeter mostraram que Lerna teve maior consistência nos tempos de resposta. A análise do tamanho do bundle, realizada com o Webpack Bundle Analyzer, revelou que Single-SPA teve o menor tamanho de bundle. Este estudo fornece uma análise detalhada das vantagens e desafios das arquiteturas de micro frontends, oferecendo recomendações práticas para a implementação e contribuindo para o avanço das práticas de desenvolvimento de software.*

Palavras-chave: micro frontends; Single-SPA; Turborepo; Lerna; desempenho; escalabilidade; manutenção.

1. Introdução e Caracterização do Problema

No atual ecossistema de desenvolvimento de software, a transição de arquiteturas monolíticas para microserviços representou um avanço significativo para aplicações backend, melhorando a escalabilidade, flexibilidade e agilidade dos processos de desenvolvimento. Analogamente, este trabalho de graduação propõe a exploração dos micro frontends, uma inovação que promete revolucionar o desenvolvimento frontend ao fragmentar aplicações web monolíticas em componentes mais granulares e independentes. A evolução das arquiteturas de front-end desempenha um papel crucial na capacidade das empresas de se adaptarem rapidamente às mudanças e demandas do mercado. Esta pesquisa não só aborda as limitações de escalabilidade e flexibilidade das interfaces de usuário monolíticas, mas também introduz uma nova dimensão de modularidade e independência de desenvolvimento web, alinhando-se com as demandas crescentes por aplicativos web mais dinâmicos e personalizáveis. Ao abordar especificamente os desafios enfrentados pelas arquiteturas monolíticas, como ciclos de desenvolvimento lentos e dificuldades de

manutenção, os micro frontends oferecem uma solução modular que permite atualizações mais rápidas e menos disruptivas, melhorando significativamente a eficiência do desenvolvimento.

Ao migrar uma aplicação Next.js existente para diferentes arquiteturas de micro frontends, este estudo visa analisar os potenciais benefícios e desafios dessa transição e contribuir para a definição de melhores práticas no desenvolvimento de interfaces de usuário modernas, gerando insumos sobre a aplicabilidade e a implementação eficaz de micro frontends em projetos reais. A migração de um projeto existente oferece um cenário real para a experimentação prática com micro frontends, permitindo uma avaliação detalhada de várias estratégias e tecnologias. Neste contexto, serão exploradas abordagens como o Webpack Module Federation, Single SPA, Turborepo e Lerna, cada uma oferecendo perspectivas únicas sobre a modularidade, compartilhamento de código e independência operacional. Através de uma metodologia comparativa, o estudo analisa o impacto dessas tecnologias na performance, na experiência do desenvolvedor, na escalabilidade e na manutenção das aplicações, com o intuito de identificar as melhores práticas e possíveis desafios.

O trabalho apresentará uma implementação detalhada da arquitetura selecionada, justificando a escolha com base nos critérios estabelecidos durante a fase de experimentação. Serão destacadas as lições aprendidas, as soluções adotadas para superar os desafios encontrados e os benefícios específicos da arquitetura de micro frontends para o projeto em questão. Espera-se que este estudo forneça uma base teórica e prática sólida para o entendimento e aplicação dos micro frontends, além de insipirar estudos e desenvolvimentos na área, contribuindo assim para o avanço contínuo das práticas de desenvolvimento de software frente às demandas de um mercado global cada vez mais dinâmico e interconectado.

2. Objetivos

O objetivo geral deste trabalho é explorar a transição de aplicações web monolíticas para arquiteturas de micro frontends, avaliando os potenciais benefícios e desafios dessa mudança, encontrando a melhor alternativa para situações distintas. Além disso, o trabalho visa desenvolver um conjunto de melhores práticas para a implementação eficaz de micro frontends em projetos reais. Especificamente, o projeto pretende alcançar os seguintes objetivos:

1) Realizar uma revisão da literatura existente para analisar e sintetizar os conceitos, vantagens, desvantagens e casos de uso dos micro frontends, a fim de aplicar os conceitos adquiridos no desenvolvimento do estudo. 2) Migrar uma aplicação existente, construída inicialmente com Next.js, para diferentes arquiteturas de micro frontends, utilizando as tecnologias **Single SPA**, **Turborepo** e **Lerna**, para exemplificar as possíveis abordagens e suas peculiaridades. 3) Avaliar cada arquitetura de micro frontend implementada com base em critérios predefinidos, como **performance e desempenho, escalabilidade, tamanho do bundle e tempo de build**, identificando os benefícios e desafios de cada abordagem. 4) Desenvolver e documentar um conjunto de melhores práticas e diretrizes para a implementação de micro frontends, incluindo recomendações para a escolha de ferramentas, tecnologias, facilitando a adoção dessa arquitetura por desenvolvedores e organizações.

Ao atingir esses objetivos, este trabalho contribuirá para o avanço das metodologias de desenvolvimento de software, oferecendo insights valiosos sobre a aplicabilidade e implementação de micro frontends, e promovendo a sua adoção como uma solução inovadora para os desafios enfrentados pelo desenvolvimento front-end moderno.

3. Background

A emergência da abordagem de micro front-ends no cenário do desenvolvimento de software moderno representa um avanço significativo na maneira como as aplicações web são concebidas, desenvolvidas e mantidas. Analogamente aos princípios arquiteturais subjacentes aos micro serviços no desenvolvimento de backend, os micro frontends buscam decompor interfaces de usuário complexas em componentes menores e funcionalmente independentes. Este paradigma surgiu como uma resposta às limitações e desafios associados ao modelo monolítico tradicional, onde grandes bases de código e equipes crescentes resultavam em ciclos de desenvolvimento lentos, dificuldades na manutenção e na implementação de novas funcionalidades.

Um exemplo notável de uma migração bem-sucedida para micro frontends é o da Dream11, uma plataforma líder de fantasy sports na Índia. A empresa enfrentava desafios com sua aplicação monolítica conforme sua base de usuários e funcionalidades expandiam rapidamente. A migração para micro frontends permitiu que a Dream11 segmentasse sua interface de usuário em componentes menores, gerenciáveis e independentes, melhorando a escalabilidade e a capacidade de manutenção. Além disso, essa mudança facilitou o desenvolvimento rápido e a implantação independente de novas funcionalidades, contribuindo significativamente para a inovação contínua dentro das equipes. A utilização de técnicas como o Webpack Module Federation - sem frameworks atrelados - foi fundamental neste processo, permitindo o compartilhamento eficiente de código e a integração dinâmica de diferentes tecnologias de frontend [Dream11 2021].

O conceito de micro front-ends começou a ganhar destaque no radar tecnológico da ThoughtWorks no final de 2016, destacando-se como uma tendência emergente no desenvolvimento front-end. A proposta central dessa abordagem é a segmentação da interface de usuário em múltiplos, pequenos, e autônomos front-ends, cada um representando diferentes partes ou funcionalidades da aplicação web. Cada micro front-end é desenvolvido, testado e implantado por uma equipe dedicada, que tem total controle sobre sua pilha tecnológica, design e ciclo de vida. Esta modularização facilita o gerenciamento do código e a colaboração entre equipes e permite atualizações mais rápidas e específicas, melhorando significativamente a eficiência e a agilidade do processo de desenvolvimento [Geers 2023].

Além de abordar questões de escalabilidade e manutenibilidade, a adoção de micro front-ends permite uma maior flexibilidade na escolha de tecnologias, potencializando a inovação e a experimentação dentro das equipes. Nesse sentido, diferentes equipes podem escolher frameworks ou bibliotecas que melhor atendam às suas necessidades específicas ou preferências, sem afetar o desenvolvimento ou a operação dos demais componentes da aplicação. Esta independência tecnológica é crucial para manter a aplicação atualizada com as últimas tendências e avanços tecnológicos, sem necessidade de reescrever ou refatorar grandes partes do sistema [Geers 2020].

Um dos principais facilitadores tecnológicos para a implementação de micro front-ends é o Module Federation, introduzido com o Webpack 5. Este mecanismo permite a integração dinâmica de diferentes componentes de front-end em tempo de execução, oferecendo uma solução eficaz para compartilhamento de código, gerenciamento de dependências e carga sob demanda de funcionalidades. Com o Module Federation, aplicações podem compartilhar bibliotecas e módulos entre si de maneira eficiente, reduzindo a redundância e otimizando o desempenho da aplicação como um todo [web 2024].

Os principais benefícios dessa arquitetura incluem melhor escalabilidade da aplicação e eficiência das equipes, flexibilidade tecnológica, produtividade aprimorada dos desenvolvedores e possibilidade de implantações independentes. Essas vantagens permitem que aplicações complexas evoluam mais rapidamente e de maneira mais eficiente, suportando uma experiência de usuário coesa [Severi Peltonen 2020].

No entanto, os micro frontends também apresentam desafios, como a complexidade na integração das diversas micro aplicações, especialmente em termos de experiência do usuário e dependências compartilhadas. A gestão de múltiplos repositórios, pipelines de build e implantações pode se tornar mais complicada do que em aplicações monolíticas. Questões de desempenho, como o aumento do tempo de carregamento inicial, e a manutenção da consistência na UI/UX entre os micro frontends são preocupações que necessitam ser cuidadosamente gerenciadas.

3.1. Integração em Execution-time

A integração em tempo de execução para micro frontends oferece uma abordagem altamente dinâmica e flexível, permitindo que componentes individuais de uma aplicação sejam desenvolvidos, construídos e atualizados de maneira independente. Este tipo de integração é ideal para cenários onde a modularidade e a capacidade de resposta rápida a mudanças são críticas.

Exemplos de Frameworks e Tecnologias:

Webpack Module Federation: Permite que diferentes builds de JavaScript compartilhem dinamicamente dependências em tempo de execução, sem necessidade de recompilar ou deployar novamente a aplicação completa.
Single SPA: Um framework para orquestrar múltiplos micro frontends como se fossem uma única aplicação. Single SPA pode carregar micro frontends baseados em Angular, React, Vue, etc., dependendo da rota ou ação do usuário.

A integração em tempo de execução traz uma flexibilidade significativa para atualizar e modificar componentes individualmente, reduzindo a latência de inicialização, pois apenas os componentes necessários são carregados no início. Isso também permite a experimentação e inovação rápida com impacto reduzido nos outros componentes da aplicação. No entanto, essa abordagem enfrenta desafios como a complexidade na gestão de dependências dinâmicas e a necessidade de um mecanismo robusto para lidar com a comunicação entre componentes independentes, o que pode complicar a arquitetura geral.

3.2. Integração em Build-time

A integração em tempo de construção é uma abordagem mais tradicional e controlada, onde todas as dependências e componentes são definidos e integrados durante o processo de build da aplicação. Isso é geralmente gerenciado através de sistemas de módulos e gerenciadores de pacotes.

Exemplos de Ferramentas e Práticas:

NPM ou Yarn: Gerenciadores de pacotes que gerenciam as dependências e permitem a integração de módulos externos durante o build.
Lerna: Uma ferramenta que otimiza a gestão de múltiplos pacotes em monorepos, permitindo a integração de várias bibliotecas e aplicativos em um único repositório. Será utilizado no desenvolvimento.

Turborepo: Um sistema de build de alto desempenho para código JavaScript e TypeScript, projetado para gerenciar grandes monorepos. Facilita a execução paralela de tarefas, caching de builds e pipelines incrementais, melhorando significativamente a eficiência do desenvolvimento.

A integração em tempo de construção oferece controle completo sobre as versões das dependências, reduzindo conflitos e incompatibilidades e promovendo estabilidade e segurança, pois todos os componentes são testados e integrados em um ambiente controlado. No entanto, essa abordagem tem desvantagens, como o aumento significativo do tamanho do aplicativo com cada nova dependência, o que pode resultar em tempos de carregamento prolongados. Além disso, há uma menor flexibilidade para atualizações rápidas, pois qualquer mudança exige um novo processo de build e deploy, o que pode atrasar a entrega de novas funcionalidades [Bhadresh 2024].

3.3. Descrição da Aplicação Inicial

A aplicação base se chama Where Can I Watch (WCIW) e é um site que permite aos usuários pesquisar em quais provedores eles podem assistir a um filme específico, dependendo da região em que estão localizados. A aplicação utiliza a API do TheMovieDB para obter informações detalhadas sobre filmes e suas disponibilidades nos diferentes serviços de streaming. A aplicação consiste em três telas principais:

Tela Inicial: onde os usuários podem iniciar suas buscas e ver filmes que estão em destaque.
Tela de Pesquisa: seção dedicada para navegação da pesquisa feita pelo usuário.
Tela de Detalhamento do Filme: que apresenta informações básicas sobre o filme e opções de onde assistir, variando conforme a região selecionada.

O site também inclui uma topbar que lida com a seleção de regiões, permitindo aos usuários ajustar o conteúdo mostrado com base em sua localização geográfica.

3.4. Frameworks para migrações

Para o desenvolvimento dos micro frontends, foram selecionadas ferramentas e frameworks específicos que se alinham com as necessidades de diferentes fases do projeto, cada uma trazendo características únicas que beneficiam o processo de desenvolvimento:

3.4.1. Single SPA

Single SPA é um framework projetado para facilitar o desenvolvimento de micro frontends, permitindo a integração de múltiplas aplicações JavaScript sob uma única página web. Operando em tempo de execução, Single SPA carrega dinamicamente os micro frontends necessários sem recarregar toda a página, melhorando assim a performance e a experiência do usuário. Este framework é compatível com os principais frameworks de JavaScript, como React, Angular e Vue, oferecendo uma flexibilidade essencial para projetos que requerem uma arquitetura robusta e modular [sin 2023].

3.4.2. Lerna

Lerna é uma ferramenta de gerenciamento para monorepos que facilita o versionamento e a gestão de múltiplos pacotes dentro de um único repositório. Utilizada principalmente em tempo

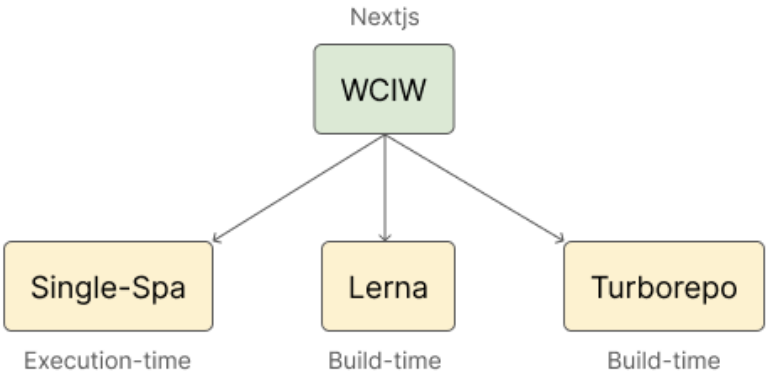

Figura 1. Fluxo de Migrações

de construção, Lerna automatiza tarefas de linkagem de dependências e execução de scripts NPM em paralelo, simplificando significativamente o processo de desenvolvimento em projetos grandes. Lerna é escolhida por sua capacidade de melhorar a eficiência operacional, reduzindo a complexidade no gerenciamento de projetos com múltiplos pacotes [ler 2024].

3.4.3. Turborepo

Turborepo é uma ferramenta moderna de build para monorepos que oferece uma execução de builds de alta performance, especialmente através de seu suporte para caching e builds incrementais. Apenas as partes do projeto que sofreram alterações são reconstruídas, o que economiza tempo significativo em comparação com os métodos tradicionais. Selecionada por sua eficácia na redução dos tempos de build, Turborepo é uma escolha valiosa para equipes que buscam maximizar a produtividade e agilizar os processos de integração contínua em ambientes de desenvolvimento dinâmicos [Vercel 2024].

Estas ferramentas foram escolhidas por suas capacidades individuais e por como complementam o ciclo de desenvolvimento dos micro frontends, desde a fase de construção até a execução, proporcionando uma abordagem compreensiva e eficiente para a implementação da arquitetura proposta.

3.5. Ferramentas para análise comparativa

Para realizar uma análise comparativa eficaz das diferentes arquiteturas de micro frontends, serão utilizadas várias ferramentas especializadas. Estas ferramentas permitem medir e avaliar o desempenho, a escalabilidade, e a eficiência dos bundles gerados por cada abordagem.

3.5.1. Lighthouse

O Lighthouse é uma ferramenta automatizada de código aberto desenvolvida pelo Google que permite avaliar a qualidade das páginas web. Ele gera relatórios sobre o desempenho, a acessibilidade, as melhores práticas de SEO, e a conformidade com padrões da web. Para este trabalho, o Lighthouse será utilizado para medir o desempenho das diferentes arquiteturas de

micro frontends, fornecendo insights detalhados sobre tempos de carregamento, renderização, e eficiência geral das páginas [Developers 2024].

3.5.2. JMeter

O Apache JMeter é uma ferramenta de código aberto projetada para testar a performance de aplicações web sob carga. Ela permite a simulação de múltiplos usuários acessando a aplicação simultaneamente, fornecendo métricas detalhadas sobre tempos de resposta, throughput, e estabilidade sob diferentes condições de carga. O JMeter será utilizado para realizar testes de carga, pico e resistência, avaliando como cada arquitetura de micro frontends lida com aumentos de tráfego e cargas pesadas [Foundation 2024].

3.5.3. Webpack Bundle Analyzer

O Webpack Bundle Analyzer é uma ferramenta que ajuda a visualizar o tamanho dos módulos incluídos no bundle gerado pelo Webpack. Ele gera um gráfico interativo que mostra a composição do bundle, permitindo identificar quais módulos contribuem significativamente para o seu tamanho. Esta ferramenta será usada para avaliar o tamanho do bundle de cada micro frontend, identificando áreas para otimização e fornecendo uma visão clara sobre a eficiência da composição dos módulos. No caso do framework Nextjs e Vite, ambos têm uma ferramenta nativa para analisar esses aspectos.

4. Metodologia

Este trabalho adota uma abordagem sistemática para explorar a migração de uma aplicação web monolítica para arquiteturas de micro frontends, avaliando os benefícios e desafios associados. A metodologia é dividida em quatro fases principais: revisão da literatura, migração prática, avaliação comparativa e documentação de melhores práticas.

4.1. Revisão da Literatura

Para fundamentar teoricamente o estudo, será realizada uma revisão da literatura baseada. Esta revisão ajudará a identificar, analisar e sintetizar as publicações existentes sobre micro frontends, focando em seus conceitos, vantagens, desvantagens e casos de uso práticos. Serão consultadas bases de dados acadêmicas e fontes da indústria para obter uma visão abrangente do estado atual da tecnologia. Com o conteúdo aprendido, o conhecimento será aplicado no desenvolvimento dos microfrontends.

4.2. Arquitetura

Embora micro frontends sejam tipicamente mais vantajosos em aplicações complexas devido à sua capacidade de decompor grandes bases de código em componentes mais gerenciáveis, este estudo opta por replicar a arquitetura em um sistema mais simples. Isso permite uma análise detalhada dos processos de implementação e integração em um contexto controlado, facilitando a compreensão e a documentação das etapas envolvidas e dos desafios enfrentados.

Nas migrações práticas, a aplicação será dividida em três micro frontends principais:

- **Topbar**: Um micro frontend responsável pela gestão das informações de região, permitindo aos usuários selecionar a localidade desejada.

- **Home**: Inclui a tela inicial e a tela de pesquisa, funcionando como o ponto de entrada para as consultas dos usuários.
- **Filme**: Micro frontend que apresenta os detalhes do filme, incluindo onde o filme pode ser assistido, com base na região selecionada.

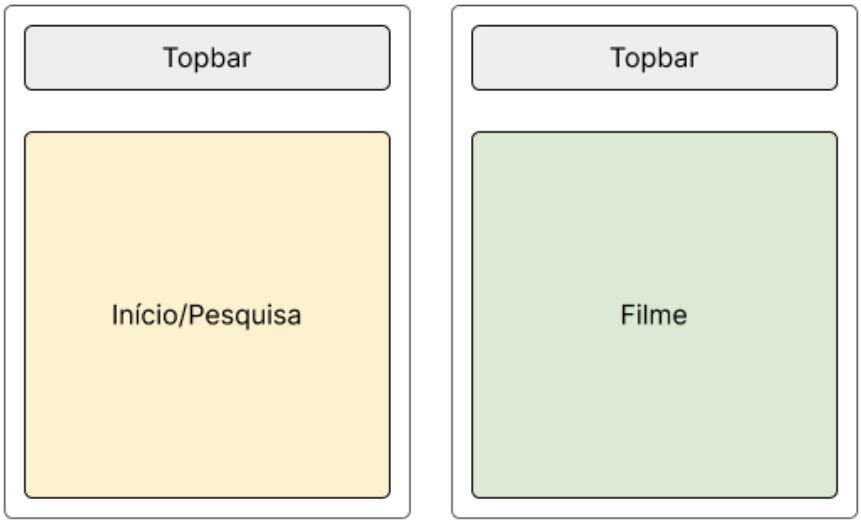

Figura 2. Arquitetura

4.3. Avaliação Comparativa e Melhores Práticas

Após a implementação, as arquiteturas de micro frontends serão submetidas a uma série de testes para avaliar seu desempenho, escalabilidade, tamanho do bundle e tempo de build. Serão utilizadas métricas quantitativas para realizar esta avaliação, e os resultados serão comparados para determinar as vantagens e desvantagens de cada arquitetura implementada.

4.3.1. Avaliação de Performance com Lighthouse

Lighthouse é uma ferramenta automatizada de código aberto desenvolvida pelo Google para melhorar a qualidade das páginas web. O objetivo da avaliação de performance é entender o qualidade de cada arquitetura em termos de performance. Para este estudo, a avaliação de performance será realizada com foco nas seguintes métricas (dado que são utlizadas para formar a média, disponibilizada pela própria ferramenta):

First Contentful Paint (FCP): Mede o tempo até que o primeiro elemento de conteúdo (texto ou imagem) seja renderizado.
Largest Contentful Paint (LCP): Mede o tempo até que o maior elemento de conteúdo visível na viewport seja renderizado.
Total Blocking Time (TBT): Mede o tempo total durante o qual a thread principal ficou bloqueada, impedindo a interação do usuário.
Cumulative Layout Shift (CLS): Mede a estabilidade visual ao quantificar quanto o layout da página muda inesperadamente durante o carregamento.
Speed Index: Mede a rapidez com que o conteúdo é visivelmente preenchido durante o carregamento da página.

Em relação a condições de estrutura do browser, o Lighthouse aplica CPU throttling para emular dispositivos com diferentes capacidades de memória, fazendo com que os testes levem em conta cenários distintos. Além disso, a ferramenta carrega a aplicação múltiplas vezes, testando diferentes condições (velocidade de internet, carregamento de bibliotecas, código minificado...) [Developers 2024].

Para cada tela das aplicações micro frontend, será coletada uma média (fornecida pelo próprio Lighthouse), e uma média aritmética será calculada para proporcionar uma visão geral da arquitetura.

4.3.2. Avaliação de Estabilidade e Escalabilidade com JMeter

Para este estudo, foram configurados dois cenários distintos para avaliar a estabilidade e escalabilidade dos micro frontends. Testes com JMeter em frontends são importantes para analisar como a aplicação e a infraestrutura lidam dado um número de usuários [Clutch 2024].

> **Load Testing**: Fundamental para avaliar como uma aplicação se comporta sob uma carga constante de usuários. Neste teste, simulamos usuários fixos enviando requisições continuamente por 10 minutos. A motivação para realizar este teste é verificar a estabilidade e a capacidade de resposta da aplicação durante um período de uso regular, sem variações abruptas na quantidade de usuários. Os ganhos deste teste incluem a identificação de gargalos de performance, a verificação da consistência dos tempos de resposta e a garantia de que a aplicação pode suportar uma carga contínua sem degradação significativa de desempenho.
> **Spike Testing**: O teste de picos é utilizado para avaliar a capacidade da aplicação de lidar com aumentos súbitos e rápidos na carga de usuários. Este teste começa com 10 usuários e aumenta gradualmente até 50, simulando um cenário onde a aplicação deve lidar com um pico repentino de acessos. A motivação para realizar este teste é garantir que a aplicação possa escalar eficientemente e manter a performance aceitável mesmo em situações de alta demanda. Os ganhos deste teste incluem a identificação de possíveis falhas ou quedas de desempenho sob carga extrema, a avaliação da capacidade de escalabilidade da aplicação e a determinação dos limites de performance antes que a aplicação se torne instável ou ineficiente.

Esses métodos de teste escolhidos são fundamentais para garantir a robustez e a capacidade de resposta das aplicações sob diversas condições de uso. [Quinn 2021].

4.3.3. Avaliação de Renderização e Deploy

Para medir a facilidade de manutenção e os benefícios de cada abordagem de micro frontend. Todas as aplicações serão deployadas no mesmo provedor, para não haver diferença por causa da infraestrutura ser diferente. O provedor escolhido foi a Netlify - empresa de computação em nuvem remota que oferece o serviço implantação de websites. Serão utilizadas as seguintes métricas quantitativas:

> **Análise de Renderização (Tamanho do Bundle)**: Aplicação do Webpack Bundle Analyzer, ou ferramenta nativa para Next - para visualizar a composição do bundle e identificar os módulos que contribuem significativamente para o seu tamanho [Webpack 2024].

Complexidade do Build e Deploy: Serão considerados dois fatores principais: o tempo de build e a simplicidade do deploy. O tempo de build será medido em termos do tempo médio necessário para compilar e empacotar a aplicação, proporcionando uma visão clara sobre a eficiência do processo. A simplicidade do deploy será analisada com base na quantidade de passos necessários para realizar o deploy de cada micro frontend.

Com base nos resultados obtidos e nas lições aprendidas durante a fase de migração e avaliação, será desenvolvido um conjunto de melhores práticas para a implementação de micro frontends. Este trabalho de graduação incluirá recomendações práticas sobre a escolha de ferramentas, a configuração de ambientes de desenvolvimento, a gestão de equipes e a manutenção de aplicações micro frontend.

Esta metodologia permite uma abordagem estruturada e detalhada para investigar a aplicabilidade e os benefícios de micro frontends em um contexto real de desenvolvimento de software, ajudando a comunidade acadêmica e profissional.

5. Resultados e Discussões

5.1. Single-SPA (WCIW-SPA)

A migração da aplicação *Where Can I Watch* para a arquitetura de micro frontends foi orientada por uma análise detalhada da documentação do Single SPA, um framework robusto que suporta a integração de múltiplos frameworks de JavaScript como Vue.js, Angular React. Para a arquitetura, a escolha de Vue.js para a Nav e React para os outros componentes foi motivada pela necessidade de explorar a flexibilidade do Single SPA em unir tecnologias distintas sob uma mesma plataforma [sin 2023].

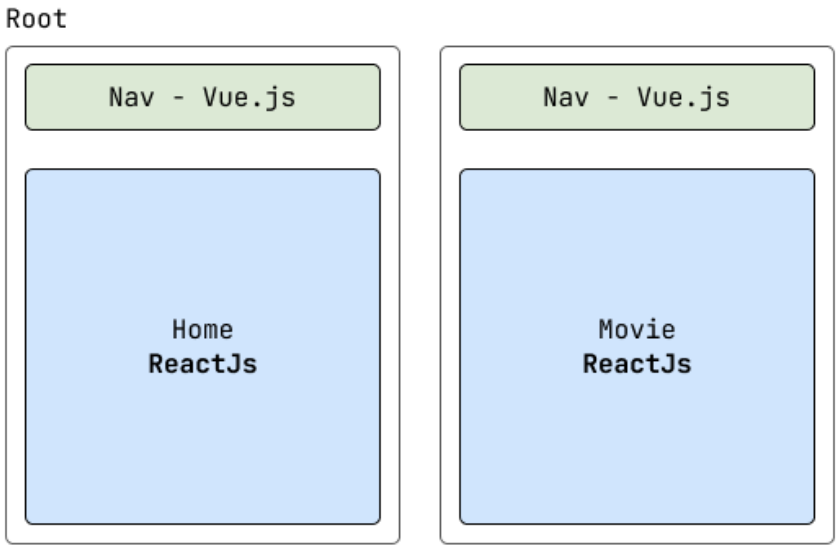

Figura 3. Arquitetura do Single Spa

5.1.1. Estruturação e Desenvolvimento dos Micro Frontends

Cada micro frontend (MFE) foi projetado para ter um fluxo de desenvolvimento independente, garantindo que as atualizações em um componente não afetassem os outros. Especificamente,

a Nav foi desenvolvida em Vue.js, enquanto as demais funcionalidades da aplicação foram implementadas em React, a tecnologia original da aplicação. Esta abordagem permitiu não apenas a utilização ótima de cada framework de acordo com suas forças, mas também uma integração harmoniosa dentro do ecossistema do Single SPA.

5.1.2. Adaptações e Integração

Para a Nav em Vue.js, foram necessárias adaptações específicas para alinhar o framework com o ambiente de execução do Single SPA. Apesar dos desafios, a integração foi bem-sucedida, demonstrando a compatibilidade do Single SPA com múltiplos frameworks. Os micro frontends são construídos individualmente e a integração no Root é feita a partir do deploy dos arquivos buildados e minificados [sin 2023]. O deploy pode ser feito em qualquer provedor, como o S3 da *AWS*. A visualização dessas rotas pode ser restringida para que apenas o package *root* tenha acesso, priorizando a segurança da aplicação. Com a importação, a integração é feita em tempo de execução e reduzindo a complexidade do deployment.

5.1.3. Gerenciamento de Estilos e Convenções

Um desafio notável foi o gerenciamento de estilos globais, onde estilos definidos em um micro frontend poderiam inadvertidamente influenciar a apresentação de outros. Para mitigar esse problema, adotou-se convenções estritas de nomenclatura e escopo de estilos, garantindo que cada micro frontend mantivesse sua independência estilística e funcional.

5.1.4. Roteamento e Comunicação Inter-Micro Frontends

O roteamento é orquestrado pelo root e gerenciado por um layout específico do Single SPA, que direciona as rotas para o micro frontend correspondente, proporcionando uma navegação fluida e coesa. Para a comunicação entre os micro frontends, especialmente para compartilhar informações de região que está presente na nav para outros componentes, utilizou-se o sistema de Events do navegador. Este método permitiu a troca de dados entre Vue.js e React sem complicações, exemplificando a eficácia do Single SPA em facilitar a comunicação entre diferentes tecnologias.

5.1.5. Deploy

O Single-SPA não possui um modelo de deploy usual devido à necessidade de fazer o deploy de configurações e import maps que são consumidos pelo root do aplicativo. Este arquivo, um arquivo javascript, precisa ser deployado de forma separada da aplicação principal. Após o deploy deste arquivo de configuração, a aplicação precisa consumir essas configurações para estar corretamente deployada. Isso significa que, além do deploy tradicional da aplicação, é necessário garantir que o import map esteja acessível e atualizado, permitindo que o root do Single-SPA carregue corretamente os micro frontends configurados.

5.1.6. Conclusão da Migração

A migração para micro frontends com o Single SPA demonstrou ser uma estratégia eficiente para modernizar a aplicação *Where Can I Watch*, aumentando sua modularidade e flexibilidade. Os insights obtidos reforçam a viabilidade de micro frontends como uma solução escalável para aplicações complexas, promovendo melhor isolamento de código, facilidade de manutenção e aprimoramento contínuo.

5.2. Lerna (WCIW-Lerna)

A migração da aplicação *Where Can I Watch* para uma arquitetura de monorepos gerenciada pelo Lerna representou uma mudança significativa na maneira como os pacotes são integrados e gerenciados durante o build time.

5.2.1. Configuração e Integração com Lerna

A utilização do Lerna facilitou a integração dos diversos micro frontends que compõem a aplicação. Como todos os pacotes foram desenvolvidos usando o framework React e o empacotador moderno Vite, a migração e gerenciamento tornaram-se mais coesos e eficientes. O Lerna permitiu que todos os micro frontends fossem construídos e renderizados simultaneamente, garantindo uma experiência de usuário uniforme e uma integração visual coesa entre os componentes [ler 2024].

5.2.2. Biblioteca de componentes: *common*

Durante a migração, foi criado um pacote chamado *common*, que se tornou fundamental para a reutilização de componentes e códigos através dos micro frontends. Esse pacote permitiu a centralização de componentes essenciais, como botões, inputs e elementos de interface do usuário, que são frequentemente utilizados em várias partes da aplicação. Esta prática não apenas garantiu a consistência na identidade visual, como também aumentou significativamente a produtividade e reduziu o tempo de desenvolvimento, ao evitar a duplicação de código e esforços.

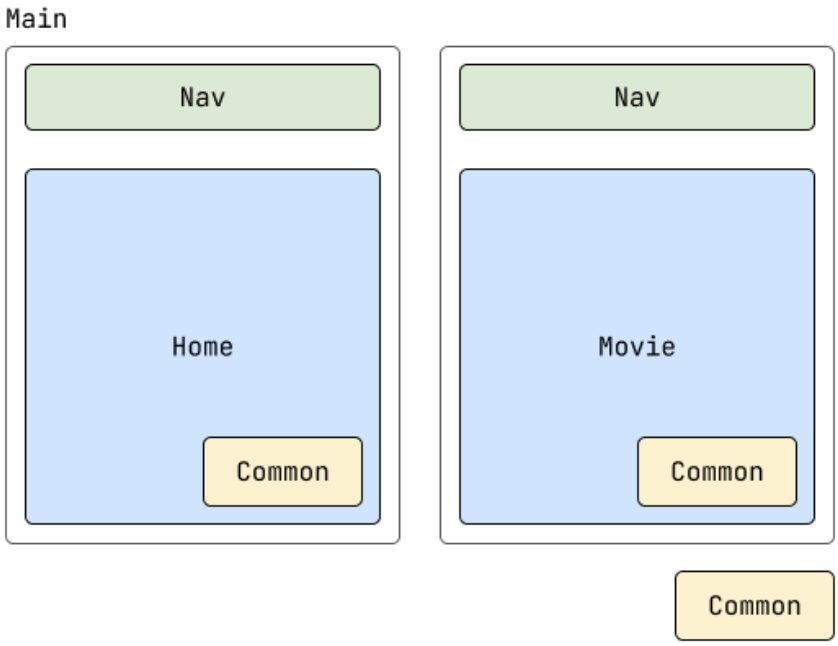

Figura 4. Arquitetura do Lerna

Criar uma biblioteca de interface do usuário (UI Library) personalizada oferece vantagens significativas em termos de consistência, eficiência e identidade de marca. Uma UI Library bem projetada encapsula todos os elementos visuais e componentes de interação que são comuns em diversas aplicações de um projeto, empresa ou organização, assegurando uma consistência visual e funcional que reflete a identidade visual, conforme destacado pelo [Salesforce 2024], que é um case de ótimo Design System. Além disso, uma UI Library centralizada reduz o tempo de desenvolvimento e os custos associados, permitindo que os desenvolvedores reutilizem componentes existentes em vez de criar novos do zero para cada projeto, garantindo a manutenção das práticas recomendadas de design e acessibilidade em toda a organização. Essa eficiência não só acelera o processo de desenvolvimento, mas também minimiza erros e inconsistências, permitindo uma melhor alocação de recursos e um produto final mais robusto e atraente.

5.2.3. Simplificação da Integração e Manutenção

A integração dos pacotes foi simplificada pela capacidade do Lerna de gerenciar dependências de maneira centralizada. Comandos como ‘npm install movie -w main‘ permitiram adicionar diretamente pacotes específicos ao aplicativo principal, facilitando a gestão de dependências e garantindo que todos os micro frontends estejam sempre sincronizados. Além disso, a utilização do TypeScript melhorou o gerenciamento de tipos entre os pacotes, permitindo uma integração mais segura e previsível, especialmente em termos de manutenção e escalabilidade da aplicação.

5.2.4. Impacto da Migração

Em suma, a transição para o Lerna não apenas racionalizou o processo de build e integração dos micro frontends, mas também estabeleceu um padrão de desenvolvimento sustentável e escalável para a aplicação. Esta abordagem fortaleceu a arquitetura de desenvolvimento, oferecendo uma base sólida para futuras expansões e melhorias contínuas.

5.3. Turborepo (WCIW-Turbo)

A migração do WCIW para uma estrutura de micro frontends utilizando o Turborepo, framework em Nextjs, em conjunto com as facilidades providas pela Vercel, ilustra um avanço significativo no processo de desenvolvimento e implantação. Turborepo oferece suporte para a gestão eficaz de monorepos, similar ao Lerna, permitindo uma centralização eficiente de componentes de UI, configurações de TypeScript e ESLint, o que melhora a consistência do código e facilita a manutenção ao mesmo tempo que promove a reutilização do código [Vercel 2024].

5.3.1. Desenvolvimento e Estruturação dos Pacotes

Durante a migração, também foi criado um pacote adicional chamado utils (que contém constantes, funções e tipos), demonstrando a flexibilidade do Turborepo em adaptar-se a diversas necessidades de desenvolvimento. Esses pacotes foram projetados para serem consumidos por uma única instância web, simplificando a arquitetura geral e reduzindo a complexidade de gerenciamento.

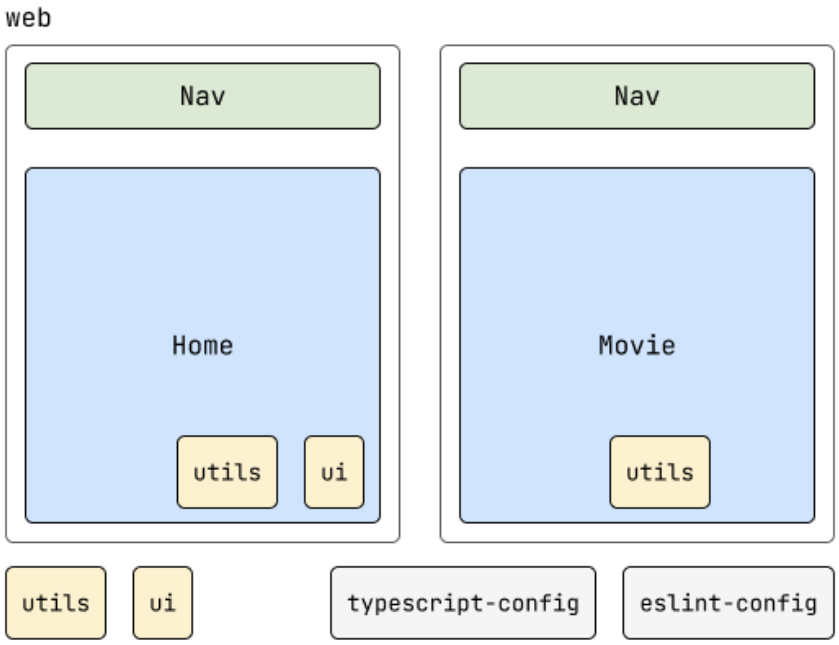

Figura 5. Arquitetura do Turborepo

5.3.2. Deploy e Integração com a Vercel

O deploy dos micro frontends foi notavelmente facilitado pelo uso da Vercel, uma plataforma que oferece suporte nativo para Next.js e integra-se de forma transparente com o Turborepo. Isso resultou em uma configuração de deploy simplificada e acelerada, permitindo atualizações ágeis e contínuas da aplicação em produção.

5.3.3. Vantagens e Impacto da Migração

A documentação do Turborepo ressalta as vantagens de escalabilidade e eficiência que esta ferramenta proporciona, o que é corroborado pela experiência prática obtida durante a migração. A escolha do Turborepo e da Vercel reflete uma estratégia bem fundamentada para o desenvolvimento moderno de aplicações web, estabelecendo uma base sólida para a evolução contínua da aplicação e uma experiência de usuário otimizada.

A integração do Turborepo com a Vercel representa um modelo exemplar de como ferramentas modernas podem ser utilizadas para maximizar a eficiência no desenvolvimento e implantação de software, fazendo desta combinação uma solução estratégica para qualquer equipe de desenvolvimento que busca excelência e inovação em suas práticas.

5.4. Performance

A otimização da performance de uma aplicação web é crucial para garantir uma experiência de usuário agradável e eficiente. Neste contexto, a ferramenta Lighthouse, desenvolvida pelo Google, emerge como uma solução robusta para a avaliação de performance, acessibilidade, práticas recomendadas e SEO de aplicações web. Lighthouse é amplamente reconhecida por sua capacidade de fornecer insights detalhados e acionáveis, o que permite aos desenvolvedores identificar e mitigar gargalos de performance de forma efetiva.

Lighthouse realiza uma série de auditorias automatizadas contra uma página web, simulando diversas condições de uso e coletando métricas sobre o desempenho da aplicação. As auditorias abrangem aspectos críticos como o tempo de carregamento, a interatividade e a estabilidade visual da página, oferecendo uma visão compreensiva sobre o comportamento da aplicação sob diferentes perspectivas de usuário.

A escolha do Lighthouse para a análise de performance neste projeto se justifica pela precisão e pela profundidade das informações que a ferramenta oferece. Com relatórios detalhados e recomendações específicas, o Lighthouse auxilia não apenas na identificação de problemas, mas também na implementação de soluções eficazes para melhorar a qualidade geral das aplicações. A capacidade da ferramenta de simular diferentes condições de rede e dispositivos torna-a particularmente valiosa para testar a aplicação em cenários que refletem o uso real pelos usuários finais.

Esta seção detalhará os resultados obtidos através da utilização do Lighthouse, discutindo as principais métricas coletadas, os desafios identificados e as melhorias implementadas como resposta aos insights fornecidos pela ferramenta. A análise visa não apenas avaliar o desempenho atual dos micro frontends, mas também orientar as futuras otimizações e desenvolvimentos dentro do projeto.

As Tabelas 1 a 4 apresentam as métricas de desempenho analisadas com o suporte do Lighthouse para o projeto base, Single SPA, Lerna e Turborepo, respectivamente.

Tabela 1. Métricas de performance para aplicação base

Metric	Search	Movie	Home
Performance	84	81	78
First Contentful Paint	0.2 s	0.3 s	0.3 s
Largest Contentful Paint	2.4 s	2.6 s	3.1 s
Total Blocking Time	150 ms	180 ms	180 ms
Cumulative Layout Shift	0	0.005	0
Speed Index	0.2 s	0.4 s	0.3 s

Tabela 2. Métricas de Performance para Single SPA

Metric	Search	Movie	Home
Performance	90	93	91
First Contentful Paint	0.6 s	0.6 s	0.6 s
Largest Contentful Paint	2.0 s	1.7 s	1.9 s
Total Blocking Time	0 ms	30 ms	0 ms
Cumulative Layout Shift	0	0.013	0
Speed Index	0.7 s	0.9 s	1.0 s

Tabela 3. Métricas de Performance para Lerna

Metric	Search	Movie	Home
Performance	60	60	60
First Contentful Paint	2.8 s	2.8 s	2.9 s
Largest Contentful Paint	5.1 s	5.1 s	4.9 s
Total Blocking Time	0 ms	0 ms	0 ms
Cumulative Layout Shift	0	0	0
Speed Index	3.3 s	3.2 s	3.2 s

Tabela 4. Métricas de Performance para Turborepo

Metric	Search	Movie	Home
Performance	77	80	77
First Contentful Paint	0.3 s	0.3 s	0.2 s
Largest Contentful Paint	3.2 s	2.0 s	2.8 s
Total Blocking Time	180 ms	200 ms	170 ms
Cumulative Layout Shift	0	0.005	0
Speed Index	0.4 s	0.4 s	0.4 s

5.4.1. Conclusões de Performance

A análise comparativa das médias de desempenho das ferramentas de gerenciamento de micro frontends e do projeto base desenvolvido em Next.js fornece insights cruciais sobre a eficácia de cada abordagem no contexto de diferentes arquiteturas de desenvolvimento. O projeto base, utilizando Next.js sem micro frontends, apresentou um desempenho médio de 81, indicando uma configuração robusta e bem otimizada que beneficia diretamente da simplicidade e eficiência de um monolito moderno.

Single SPA lidera o ranking com uma média de 90, destacando-se pela sua habilidade em integrar múltiplos frameworks de maneira eficiente em Execution-time, proporcionando um desempenho superior. Turborepo, com uma média de 78, e Lerna, com 60, mostram diferentes níveis de eficácia, com o Turborepo beneficiando-se de suas otimizações de build e capacidades de cache que melhoram o desempenho de desenvolvimento e deploy.

Tabela 5. Performance Ranking of Management Tools and Base Project

Management Tool	Average Performance	Rank
Single SPA	90	1
Base Project (Next.js)	81	2
Turborepo	78	3
Lerna	60	4

Este ranking não apenas ressalta a superioridade técnica do Single SPA e a eficiência do projeto base em Next.js, mas também enfatiza a importância de uma escolha criteriosa de ferramentas e arquiteturas, considerando tanto a eficiência do build quanto a performance em runtime para maximizar a experiência do usuário final e a eficácia operacional.

5.5. Análise de Escalabilidade e Estabilidade

A análise de escalabilidade e performance é crucial para entender como diferentes arquiteturas de micro frontends se comportam sob condições de carga variáveis. Para isso, utilizamos o Apache JMeter, uma ferramenta poderosa para testes de carga e performance. O JMeter nos permitiu simular múltiplos usuários acessando a aplicação simultaneamente, fornecendo dados detalhados sobre o tempo de resposta, vazão, e estabilidade sob diferentes cenários de teste. Testes de spike, carga e stress foram realizados para avaliar a robustez e a capacidade de resposta de cada arquitetura.

Os dados coletados dos testes foram organizados em tabelas que incluem várias métricas essenciais para a análise de performance. A coluna ”Média”representa o tempo médio de resposta das requisições, calculado ao somar todos os tempos de resposta e dividir pelo número de amostras. A ”Mediana”mostra o valor central dos tempos de resposta, onde 50% das amostras têm tempos inferiores e 50% têm tempos superiores a este valor. As colunas ”90% Line”, ”95% Line”e ”99% Line”indicam os tempos de resposta abaixo dos quais 90%, 95% e 99% das amostras foram completadas, respectivamente, proporcionando uma visão das variações de desempenho em condições de carga. As colunas ”Mín.”e ”Máx.”registram os menores e maiores tempos de resposta observados durante os testes, ajudando a identificar picos de performance ou possíveis gargalos. Essas métricas combinadas nos permitem avaliar a eficácia e a eficiência de cada arquitetura de micro frontend sob diferentes cargas.

5.5.1. Load Testing

Para avaliar a estabilidade e a escalabilidade das diferentes arquiteturas de micro frontends, foi realizado um load testing utilizando Apache JMeter. Foram configurados três cenários distintos para simular picos súbitos de carga, cada um apontando para diferentes implementações de micro frontends: Turborepo, Single-SPA e Lerna. Abaixo estão os resultados obtidos:

Tabela 6. Load Testing - HTTP Requests

Rótulo	Média	Mediana	90% Line	95% Line	99% Line	Mín.	Máx.
Single-SPA	471	363	556	773	1487	313	1487
Lerna	431	330	665	697	985	297	985
Turborepo	500	430	695	737	796	382	796

Os resultados indicam que a arquitetura Single-SPA apresentou a maior variabilidade nos tempos de resposta, com uma média de 471 e mediana de 363. A arquitetura Lerna teve uma performance intermediária, com uma média de 431 e mediana de 330, mas menor variabilidade comparada a Single-SPA. Já a arquitetura Turborepo apresentou a maior média (500) mas com menor variabilidade, sugerindo uma maior consistência sob carga de pico. Em termos de vazão, todas as arquiteturas mostraram valores similares, mas a Turborepo demonstrou ser a mais estável, sugerindo uma experiência de usuário mais consistente e responsiva sob carga.

5.5.2. Spike Testing

Para avaliar a estabilidade e a escalabilidade das diferentes arquiteturas de micro frontends, foi realizado um spike testing utilizando Apache JMeter. Foram configurados três cenários distintos para simular picos súbitos de carga, cada um apontando para diferentes implementações de micro frontends: Turborepo, Single-SPA e Lerna. Abaixo estão os resultados obtidos:

Tabela 7. Spike Testing - HTTP Requests

Rótulo	Média	Mediana	90% Line	95% Line	99% Line	Mín.	Máx.
Single-SPA	328	284	325	537	1007	266	1007
Lerna	287	276	296	301	586	258	586
Turborepo	397	391	427	444	678	321	678

Os resultados indicam que a arquitetura Lerna apresentou a melhor performance e consistência nos tempos de resposta, com uma média de 287 ms e mediana de 276 ms. A arquitetura Single-SPA apresentou uma variabilidade significativa, com uma média de 328 ms e mediana de 284 ms, mas com picos que atingiram até 1007 ms. Já a arquitetura Turborepo apresentou a maior média (397 ms) e uma variabilidade intermediária, sugerindo uma maior consistência sob carga de pico em comparação com Single-SPA, mas inferior a Lerna.

5.5.3. Conclusão Geral

Os resultados dos testes de desempenho indicam que a arquitetura Lerna apresentou a melhor performance e consistência nos tempos de resposta tanto nos testes de carga quanto nos testes de pico.

Desempenho Geral: A arquitetura **Lerna** demonstrou ter o melhor desempenho em termos de consistência e menor variabilidade nos tempos de resposta tanto nos testes de carga quanto nos testes de pico. Sua média e mediana foram as mais baixas, e a variabilidade foi

a menor. A arquitetura **Single-SPA** apresentou uma variabilidade significativa nos tempos de resposta nos testes de carga e pico, indicando que pode enfrentar problemas de desempenho sob carga extrema. A arquitetura **Turborepo** teve a maior média de tempo de resposta nos testes de carga e pico, indicando uma maior latência em comparação com as outras duas arquiteturas.

Estabilidade e Escalabilidade: Lerna demonstrou ser a mais estável e escalável, com tempos de resposta consistentes mesmo sob condições de pico. Single-SPA teve uma boa performance em condições normais de carga, mas apresentou picos de latência sob condições de pico. Turborepo mostrou-se menos eficiente em termos de tempo de resposta, mas ainda manteve uma variabilidade intermediária, o que sugere que é relativamente estável, mas pode não ser a melhor escolha para aplicações que exigem baixa latência.

Recomendações: Para aplicações que exigem alta estabilidade e baixa latência, **Lerna** seria a escolha recomendada devido à sua consistência de desempenho. **Single-SPA** pode ser uma boa escolha para aplicações que podem tolerar alguma variabilidade nos tempos de resposta, especialmente se a modularidade e a flexibilidade forem prioritárias. **Turborepo** pode ser considerado para aplicações onde a facilidade de desenvolvimento e a integração contínua são mais importantes do que a latência.

5.6. Análise de Tamanho do Bundle

O tamanho do bundle refere-se à quantidade de código (JavaScript, CSS, etc.) que precisa ser baixado pelo navegador para carregar uma aplicação web. Um bundle menor geralmente resulta em tempos de carregamento mais rápidos, melhor desempenho e melhor experiência do usuário. Analisar o tamanho do bundle é crucial para identificar possíveis áreas de otimização, reduzir a latência e melhorar a eficiência da aplicação.

Tabela 8. Comparação de Tamanhos de Bundle

Métrica	Single-SPA	Turborepo	Lerna
Stat Size	19.92 kB	13.34 kB	708.43 kB
Parsed Size	17.36 kB	598.70 kB	204.05 kB
Rendered	-	-	175.08 kB
Total	37.28 kB	612.04 kB	1087.56 kB

Stat Size: Representa o tamanho total dos arquivos necessários para a renderização inicial da aplicação.
Parsed Size: Representa o tamanho do JavaScript necessário para carregar a aplicação na primeira vez que um usuário acessa o site.
Rendered: Para Lerna (Vite), isso representa o tamanho do bundle após renderização, enquanto os valores Gzip e Brotli mostram a eficiência das técnicas de compressão.

Os resultados indicam que cada arquitetura de micro frontends tem diferentes vantagens e desvantagens em termos de tamanho de bundle e performance:

1 - Single-SPA: Apresenta o menor tamanho de bundle entre as arquiteturas, sugerindo uma performance inicial rápida, dado que a integração é em execution-time.
2 - Turborepo: Possui um tamanho estático pequeno, devido às features do Nextjs, mas o total de primeiro carregamento é relativamente grande, o que pode impactar o tempo de carregamento inicial.

3 - Lerna: Apresenta o maior tamanho de bundle, mas as técnicas de compressão (Gzip e Brotli) são eficazes em reduzir o tamanho do payload entregue ao navegador, mas mesmo com isso permanece o maior.

Essas tabelas e explicações ajudam a visualizar e entender as diferenças no tamanho dos bundles entre as arquiteturas de micro frontends, fornecendo insights sobre a performance e a eficiência de cada abordagem.

5.7. Análise do Tempo de Build

O tempo de build refere-se ao tempo necessário para compilar e empacotar a aplicação antes de ser implantada em um ambiente de produção. Um tempo de build menor geralmente resulta em um ciclo de desenvolvimento mais rápido, permitindo que os desenvolvedores iterem e implantem mudanças com maior eficiência. É importante notar que o tempo pode ser influenciado por diversos fatores adicionais, como a inclusão de processos de CI/CD (Continuous Integration/Continuous Deployment) que podem envolver testes automatizados, linting, e outras verificações de qualidade.

Single-SPA Para a arquitetura Single-SPA, o tempo de build foi de 3.42 segundos. Este tempo reflete a eficiência do processo de build para esta arquitetura, permitindo uma rápida compilação e empacotamento da aplicação.

Lerna Para a arquitetura Lerna, o tempo de build foi de 4.57 segundos. Este tempo de build é ligeiramente maior do que o do Single-SPA, mas ainda assim demonstra uma eficiência considerável no processo de compilação e empacotamento.

Turborepo Para a arquitetura Turborepo, o tempo de build foi significativamente maior, totalizando 13.03 segundos. Este tempo inclui vários passos adicionais que contribuem para a robustez e qualidade do build, como:

- **Linting e verificação de validade de tipos**
- **Coleta de dados da página**
- **Geração de páginas estáticas (6/6)**
- **Coleta de rastreamentos de build**
- **Finalização da otimização de páginas**

Esses passos adicionais garantem que a aplicação seja bem testada, validada e otimizada antes de ser implantada, embora isso resulte em um tempo de build mais longo.

Tabela 9. Comparação do Tempo de Build

Arquitetura	Tempo de Build
Single-SPA	3.42 segundos
Lerna	4.57 segundos
Turborepo	13.03 segundos

Explicação:

- **Single-SPA**: O tempo de build mais rápido entre as três arquiteturas, permitindo ciclos de desenvolvimento e deploy mais rápidos.
- **Lerna**: Tempo de build ligeiramente maior, mas ainda eficiente para desenvolvimento ágil.
- **Turborepo**: Tempo de build mais longo devido a etapas adicionais de verificação e otimização, garantindo uma maior robustez e qualidade na aplicação final.

Conclusão Geral:

Os resultados indicam que a escolha da arquitetura pode ter um impacto significativo no tempo de build. Single-SPA oferece o menor tempo de build, tornando-se uma excelente opção para projetos onde a rapidez no ciclo de desenvolvimento é crítica. Lerna apresenta um bom equilíbrio entre tempo de build e eficiência. Turborepo, embora tenha o maior tempo de build, proporciona uma maior qualidade e robustez na aplicação final, sendo ideal para projetos onde a qualidade e a verificação são prioritárias.

Essas observações fornecem insights importantes sobre como diferentes arquiteturas de micro frontends podem influenciar o tempo de build e, por extensão, o ciclo de desenvolvimento e deploy.

6. Trabalhos Relacionados

O trabalho de [Bui 2021], ”Migrating a Monolithic Application to Micro Frontends Using Single-SPA and Module Federation”aborda a comparação de performance e escalabilidade entre a arquitetura de micro frontends e a monolítica. Os objetivos do trabalho incluem a análise detalhada das etapas de migração, a identificação dos desafios enfrentados durante a transição e a avaliação dos benefícios e dificuldades da arquitetura de micro frontends. A parte mais relevante para meu estudo é a utilização do Single-SPA, que permite uma comparação direta com a abordagem de Lerna e Turborepo no meu trabalho. O diferencial do trabalho de Son Bui está no foco específico na combinação de Single-SPA e Module Federation, enquanto meu estudo abrange uma análise mais ampla de diferentes arquiteturas. As conclusões de Son Bui destacam a melhoria na modularidade e escalabilidade, assim como os desafios técnicos e a complexidade adicional.

No estudo ”Energy Consumption of Micro Frontends: A Comparison of Micro Frontends and Single-Page Applications”, [Harandi and Mirzaei 2023] comparam o consumo de energia entre micro frontends e SPAs. Enquanto meu trabalho foca na comparação do processo de migração de diferentes arquiteturas de micro frontends, este estudo se aprofunda no impacto energético dessas arquiteturas. Ambos os estudos destacam a modularidade e a escalabilidade como vantagens significativas dos micro frontends, apesar de diferentes enfoques. As conclusões de Harandi e Mirzaei sobre o maior consumo energético dos micro frontends oferecem uma perspectiva adicional que complementa a avaliação de performance e escalabilidade no meu trabalho.

O trabalho de [Peltonen et al. 2021] fornece uma análise abrangente das motivações, benefícios e problemas associados à adoção de Micro-Frontends, com base em uma revisão de 173 fontes da literatura. Este estudo é particularmente relevante para o meu trabalho, pois ambos compartilham o objetivo de entender os impactos das arquiteturas de Micro-Frontends. Peltonen et al. destacam que as principais motivações para a adoção incluem a escalabilidade das equipes de desenvolvimento e a flexibilidade tecnológica, fatores que também são centrais ao meu estudo. No entanto, meu trabalho se diferencia ao focar na implementação prática e

na avaliação comparativa de diferentes frameworks de Micro-Frontends, incluindo a análise de desempenho e manutenção. As conclusões de Peltonen et al. sobre a complexidade adicional e os desafios de monitoramento fornecem insights valiosos que complementam a minha investigação, reforçando a importância de um planejamento cuidadoso e de estratégias eficazes de gestão de dependências em projetos de Micro-Frontends.

O estudo de [Capdepon et al. 2023] apresenta uma abordagem inovadora para a migração de arquiteturas monolíticas mobile para Micro Frontends, utilizando um modelo dirigido por engenharia. Os objetivos do estudo incluem a modularização de aplicações móveis monolíticas para melhorar a manutenibilidade e escalabilidade, utilizando Flutter e um meta-modelo Dart. A identificação de Micro Frontends é realizada através da análise de dependências e métricas de coesão, com visualizações que auxiliam na decisão de agrupamento. Em comparação ao meu trabalho, que foca em aplicações web e a avaliação de diferentes frameworks de Micro Frontends, o estudo fornece insights valiosos sobre a modularização e os desafios específicos na migração de arquiteturas móveis. As conclusões ressaltam a importância da modularidade, reusabilidade e independência de desenvolvimento para alcançar uma arquitetura de Micro Frontends eficaz, destacando os desafios de manutenção da compatibilidade.

7. Conclusão e Trabalhos Futuros

Este trabalho de conclusão de curso explorou a migração de uma aplicação monolítica para diferentes arquiteturas de micro frontends, utilizando frameworks como Single-SPA, Turborepo e Lerna. Através de uma série de testes de desempenho, escalabilidade e análise de tempo de build e deploy, foi possível identificar os benefícios e desafios de cada abordagem.

Os testes de desempenho realizados com o Lighthouse indicaram que a arquitetura de Single-SPA teve a melhor performance, com uma média de 91.3, seguida por Turborepo com 78 e Lerna com 60. A aplicação base em Next.js obteve uma média de 81, destacando que a escolha da arquitetura de micro frontends pode oferecer vantagens significativas em termos de modularidade e flexibilidade.

Os testes de escalabilidade e estabilidade, realizados com o Apache JMeter, mostraram que Lerna apresentou a melhor consistência nos tempos de resposta, tanto nos testes de carga quanto nos testes de pico. Single-SPA apresentou maior variabilidade nos tempos de resposta sob carga extrema, enquanto Turborepo mostrou-se menos eficiente em termos de tempo de resposta, mas ainda manteve uma variabilidade intermediária.

A análise de tamanho de bundle, realizada com o Webpack Bundle Analyzer, revelou que Single-SPA apresentou o menor tamanho de bundle, sugerindo uma performance inicial rápida. Turborepo teve um tamanho estático pequeno, mas o total de primeiro carregamento foi relativamente grande, enquanto Lerna apresentou o maior tamanho de bundle, mesmo com técnicas de compressão eficazes.

Há cenários ideais para cada tipo dos frameworks utilizados. **Single-SPA:** Ideal para projetos que exigem alta modularidade e flexibilidade, integrando múltiplos frameworks JavaScript em uma única aplicação. É especialmente útil quando há necessidade de atualização independente de componentes e uma rápida integração de novas tecnologias. Também demonstrou mais agilidade, por não precisar ser buildada com todo o código do projeto, precisando apenas apontar para o código gerado pelo build. Por exemplo, uma aplicação legada pode ser integrada mais facilmente nessa abordagem, porque a evolução da aplicação pode ser feita em outros frameworks,

sem afetar o que já está desenvolvido. **Turborepo:** Adequado para equipes que buscam maximizar a produtividade e agilizar os processos de integração contínua, por causa da arquitetura feita em Next.js. Também ideal para times mais modernos e que possuem conhecimento de ferramentas da Vercel. Turborepo é vantajoso em ambientes onde a qualidade do build e a verificação contínua são prioritárias, oferecendo robustez e estabilidade. **Lerna:** Recomendado para projetos que exigem alta estabilidade e baixa latência, com um gerenciamento eficiente de monorepos e uma experiência de usuário consistente. Lerna se destaca em cenários onde a consistência do desempenho e a simplicidade na gestão de pacotes são cruciais. Ideal para migração com foco em build-time de forma simplificada e fácil, sem depender de muitos processos.

O trabalho contribuiu para a identificação das melhores práticas para a implementação de micro frontends, considerando a modularidade, independência de deploy e flexibilidade tecnológica. Contribuiu para a avaliação detalhada do desempenho, escalabilidade, e tempo de build e deploy, fornecendo uma base sólida para decisões informadas em projetos futuros. E, por fim, contribuiu para o desenvolvimento de recomendações práticas para a escolha de ferramentas e estratégias de implementação, auxiliando desenvolvedores e equipes na adoção de micro frontends.

Embora este estudo tenha fornecido insights valiosos, algumas questões interessantes ainda podem ser exploradas. Impacto da modularização no consumo energético: Estudos futuros podem investigar como diferentes arquiteturas de micro frontends afetam o consumo de energia das aplicações, contribuindo para a sustentabilidade do desenvolvimento de software. Integração com serviços legados: A avaliação da compatibilidade e desempenho de micro frontends em ambientes com sistemas legados complexos é crucial para a adoção desta arquitetura em empresas que possuem infraestruturas estabelecidas. Aprimoramento das estratégias de monitoramento e gerenciamento de dependências: Desenvolvimento de novas ferramentas e técnicas para melhorar a gestão de dependências e monitoramento de micro frontends em produção, garantindo uma operação contínua e eficiente.

Em resumo, este trabalho demonstrou que a escolha da arquitetura de micro frontends deve ser feita com base nas necessidades específicas do projeto, considerando fatores como modularidade, robustez e estabilidade. As contribuições deste estudo fornecem uma base sólida para futuras pesquisas e implementações, promovendo o avanço contínuo das práticas de desenvolvimento de software.

Referências

(2023). Single-spa: a javascript router for front-end microservices. `https://single-spa.js.org/`. Acessado em: 06 de junho de 2024.

(2024). Lerna: A tool for managing javascript projects with multiple packages. `https://lerna.js.org/`. Acessado em: 06 de junho de 2024.

(2024). Webpack: A static module bundler for modern javascript applications. `https://webpack.js.org/`. Acessado em: 25 de junho de 2024.

Bhadresh, P. (2024). Micro frontend architecture: Benefits, implementation, and best practices. Acessado em: 06 de junho de 2024.

Bui, S. (2021). Migrating a monolithic application to micro frontends using single-spa and module federation. Master's thesis, Metropolia University of Applied Sciences.

Capdepon, Q., Hlad, N., Seriai, A.-d., and Derras, M. (2023). Migration process from monolithic to micro frontend architecture in mobile applications. In *IWST 2023: International Workshop on Smalltalk Technologies*. CEUR Workshop Proceedings.

Clutch (2024). How to use jmeter to test your web application. `https://clutch.co/resources/how-to-use-jmeter-to-test-your-web-application`. Acessado em: 25 de junho de 2024.

Developers, C. (2024). Lighthouse. `https://developer.chrome.com/docs/lighthouse/`. Acessado em: 07 de junho de 2024.

Dream11 (2021). How to convert a huge frontend monolith to a micro frontend. Acessado em: 06 de junho de 2024.

Foundation, A. S. (2024). Apache jmeter user's manual. `https://jmeter.apache.org/usermanual/`. Acessado em: 06 de junho de 2024.

Geers, M. (2020). *Micro Frontends in Action*. Manning Publications. ISBN: 978-1617296871.

Geers, M. (2023). Micro frontends - extending the microservice idea to frontend development. `https://micro-frontends.org/`. [Acessado em: 07/04/2024].

Harandi, G. and Mirzaei, R. (2023). Energy consumption of micro frontends: A comparison of micro frontends and single-page applications. Master's thesis, University of Gothenburg.

Peltonen, S., Mezzalira, L., and Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136:106571.

Quinn, B. (2021). Performance testing with jmeter. *Scott Logic Blog*. Acessado em: 7 de junho de 2024.

Salesforce (2024). Best practices for using salesforce lightning design system. `https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/vf_dev_best_practices_slds_intro.htm`. Acessado em: 06 de junho de 2024.

Severi Peltonen, Luca Mezzalira, D. T. (2020). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 3.

Vercel (2024). Turborepo: The high-performance build system for javascript and typescript codebases. `https://turborepo.org/`. Acessado em: 06 de junho de 2024.

Webpack (2024). Webpack bundle analyzer. `https://www.npmjs.com/package/webpack-bundle-analyzer`. Acessado em: 07 de junho de 2024.