PROPOSTA DE ARQUITETURA BASEADA EM MICRO-FRONTENDS PARA MULTI APLICAÇÕES WEB[1]

Kimberly Geremia[2]

Roberto Wiest[3]

Jorge Luis Boeira Bavaresco[4]

RESUMO

Este artigo propõe uma arquitetura de aplicativo web baseada em micro-frontends. A abordagem de micro-frontend surgiu como uma alternativa flexível e escalável para a construção de sistemas complexos. Ao decompor a interface do usuário em módulos independentes, a arquitetura proposta permite a integração de diferentes tecnologias e equipes de desenvolvimento. Cada módulo pode ser desenvolvido e implantado individualmente, utilizando a tecnologia que melhor se adapta às necessidades de cada equipe. A arquitetura promove a reutilização de componentes e simplifica a manutenção, pois cada módulo é independente e pode ser atualizado sem afetar o restante do sistema. Este artigo fornece uma visão geral da abordagem de micro-frontends, destacando suas vantagens e desafios. Também descreve a arquitetura proposta, fornecendo diretrizes para a implementação e integração de micro-frontends em sistemas multiaplicativos. Um estudo de caso prático é apresentado para demonstrar a viabilidade e eficácia da arquitetura proposta. Os resultados mostram que a abordagem de micro-frontend fornece uma maneira flexível e modular de construir aplicativos web com escalabilidade, reusabilidade e independência tecnológica. No entanto, desafios como coordenação entre micro-frontends e gerenciamento de estado compartilhado são reconhecidos. Por fim, áreas para futuras pesquisas e melhorias na arquitetura proposta são sugeridas.

Palavras-chave: microsserviços; arquitetura de software; aplicações web; single-spa.

---

[1] Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense, Campus Passo Fundo, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação, na cidade de Passo Fundo, em 2023.

[2] kimberlygeremia@gmail.com.

[3] Orientador do trabalho (robertowiest@ifsul.edu.br).

[4] Coorientador do trabalho (jorgebavaresco@ifsul.edu.br).

1 INTRODUÇÃO

Na última década, organizações estão focando nos serviços oferecidos aos usuários finais, pois provedores de nuvem são utilizados para auxiliar em tarefas que estão tornando-se *commodities*. Escalar um monolito, sistema em que suas funcionalidades operam como um só, se não modularizado ou escrito com padrões altos, pode causar problemas como quebra de APIs em produção (MEZZALIRA, 2021). Assim, urge a necessidade da reestruturação e migração dessas aplicações monolíticas para arquiteturas que possibilitem maior independência entre times, deploys automatizados, alta disponibilidade e escalabilidade, de modo a dinamizar a crescente exigência por novas implementações de software (FOWLER, 2017).

Neste contexto, devido às entregas imediatas de recursos de software, causadas pelas novas aspirações em relação ao ritmo de lançamento de aplicações e funcionalidades inovadoras no mercado, muitas equipes trabalham constantemente para suprir a alta demanda por gestão de incrementações em sistemas monolíticos e legados. Com base nisso, surge a necessidade de reorganização entre times, de maneira a permitir entregas eficientes (NEWMAN, 2020).

A decomposição do *front-end* em Micro-frontends é uma agregação valiosa à arquitetura de microsserviços, visando a desacoplação de sistemas e independência entre times. Os microsserviços isoladamente contribuem para a ampliação da capacidade de escalabilidade, eficiência e consequentemente aumentam a velocidade de desenvolvimento (FOWLER, 2017). Outrossim, a abordagem de Micro-Frontends integra conceitos como usabilidade, componentização, performance Web e segurança Web. Nesse contexto, a autenticação entre as múltiplas aplicações pode se tornar um desafio, devido à existência de diversas abordagens de implementação (KARIM et al., 2020).

A autenticação entre os Micro-Frontends pode ser realizada por meio de *JSON Web Tokens* (JWT), onde um fragmento ou página disponibiliza uma tela de login e comunica o status para as demais aplicações (GEERS, 2020). Em vista disso, é indispensável o aperfeiçoamento das técnicas de segurança no que

refere-se ao gerenciamento da autenticação de usuários em múltiplas aplicações, optando pelo melhor método de comunicação para a abordagem escolhida.

O objetivo do trabalho é propor uma arquitetura baseada em Micro-frontends com foco em aplicações web, composta por múltiplas funcionalidades integradas, aprimorando as fases de desenvolvimento e gerenciamento dessas aplicações, bem como garantir a autenticação compartilhada, e por fim, avaliar a performance dos Micro-frontends em relação à aplicações SPA convencionais. Além da caracterização da proposta de arquitetura, constituída de aplicações Micro-frontend que utilizam a biblioteca ReactJS e microsserviços construídos com o framework Java Spring-boot, será apresentada a estrutura da aplicação escolhida para o caso de uso, padrões de design utilizados na implementação e métodos de teste e avaliação.

Portanto, este artigo está organizado da seguinte forma: na Seção 2 é detalhada a fundamentação teórica a respeito das arquiteturas de Microsserviços e Micro-frontends; a Seção 3 mostra a análise do sistema de estudo de caso, bem como a arquitetura proposta e implementação; por fim, a Seção 4 apresenta a avaliação a respeito do estudo e implementação.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção são apresentadas as arquiteturas de Microsserviços e Micro-frontends, abordando os problemas que buscam solucionar, além de detalhar princípios, estruturas e padrões a serem seguidos.

2.1 ARQUITETURA DE MICROSSERVIÇOS

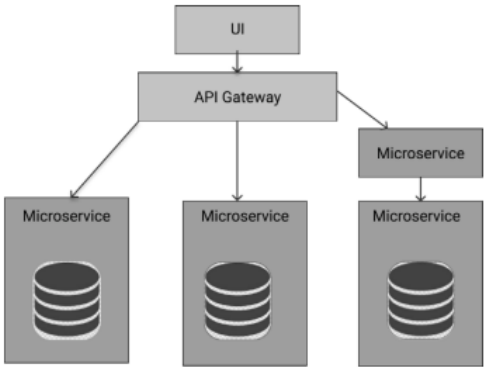
Um microsserviço é uma pequena aplicação com funcionalidade única e eficiente, onde o conjunto de tais micro aplicações autossuficientes formam uma arquitetura de microsserviços (FOWLER, 2017). O termo tem origem de um padrão emergente em 2011 denominado “*Microapps*”, que diz respeito a aplicações de escopo limitado e que poderiam ser reescritas (NEWMAN, 2020).

Essas pequenas aplicações são desenvolvidas e executadas em ambientes chamados de “ecossistema de microsserviços”, uma analogia às comunidades ecológicas. Os microsserviços devem ser construídos com o intuito de serem simples e, de acordo com a carga de trabalho requerida para o desenvolvimento do mesmo, as equipes devem ser reestruturadas de maneira a explorar eficientemente as vantagens do desenvolvimento independente (FOWLER, 2017).

Uma opção para realizar a modelagem de microsserviços é a metodologia de *Domain-Driven Design* (DDD), utilizando domínios (*core domain* e demais funcionalidades complementares ou de suporte). Essa abordagem permite estruturar o código de maneira a definir contextos delimitados, onde cada um possui suas responsabilidades e fornece suas funcionalidades, ocultando a complexidade do sistema. Desta forma, a implementação das funcionalidades e a combinação de microsserviços é facilitada. (NEWMAN, 2015).

A comunicação entre tais microsserviços efetiva-se por meio das redes de computadores podendo sofrer com problemas de latência e, consequentemente, perda de pacotes. A Figura 1 demonstra a comunicação entre Microsserviços e Interfaces de Usuário por meio de uma *API Gateway*[5].

Figura 1 - Arquitetura de uma microsserviço



Fonte: Peltonen; Mezzalira; Taibi (2020, p.6).

Essa arquitetura é viável quando pretende-se aumentar a autonomia das equipes, utilizar novas tecnologias, reduzir o tempo de entrega do sistema, escalar com um custo procedente (tanto sistemas quanto desenvolvedores) e aumentar a

[5] API Gateway é um serviço que aborda o problema de clientes de diferentes naturezas (MONTESI, WEBER, 2016).

robustez das aplicações. Porém, para aplicar alguns desses conceitos, não é necessária a implementação de microsserviços.

> A autonomia [...] pode estar presente de várias maneiras. Descobrir maneiras de atribuir mais responsabilidades a uma equipe não exige uma mudança de arquitetura (NEWMAN, 2020).

2.2 ARQUITETURA DE MICRO-FRONTENDS

A arquitetura de Micro-Frontends oferece uma solução para mitigar os desafios enfrentados pelas equipes em aplicações complexas, permitindo uma melhor distribuição de trabalho e facilitando a introdução de novas implementações em sistemas amplos. Isso ocorre devido ao tamanho do time que aumenta para suprir a alta demanda por entregas de novas funcionalidades e, consequentemente, o conhecimento dessas novas entregas ou de outras já implementadas começa a ficar concentrado em pessoas específicas, fazendo com que nem todos os desenvolvedores conheçam todas as funcionalidades presentes no sistema (GEERS, 2020).

O padrão arquitetural de Micro-Frontends trabalha com o conceito de divisão da aplicação em partes verticais, ou seja, com interfaces e banco de dados autônomos, de forma que vários times independentes possam atuar nessas pequenas aplicações que compõem o todo, que será integrado no *browser* e visto como único pelo usuário. A proposta emergente de Micro-Frontends contrasta com as demais abordagens monolíticas adotadas no desenvolvimento web por definirem entregas de software em concordância com fronteiras de negócio e responsabilidade (MEZZALIRA, 2021).

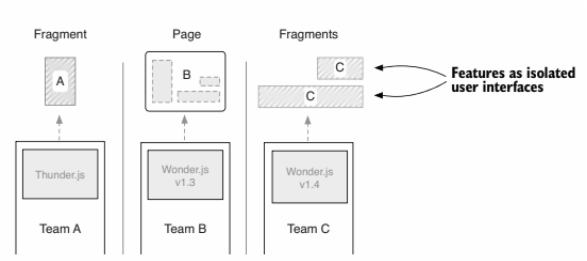
De acordo com Peltonen et al. (2020), a abordagem de micro-frontends é uma decisão de arquitetura e compartilha dos mesmos princípios que a arquitetura de microsserviços, no qual detalhes de implementação são ocultados entre as aplicações, utilizando a modelagem de domínio de negócio (DDD), além da descentralização de *deploys*. Com a abordagem de Micro-frontends, não é necessária a coordenação entre times de *front-end* e *back-end* pois o time deve conter todas as competências básicas para desenvolver funcionalidades que serão

entregues diretamente ao cliente, além da possibilidade de escolha da tecnologia que será aplicada no projeto (GEERS, 2020).

O estilo de atuação para o time deve ser multidisciplinar onde todos os membros são envolvidos com o desenvolvimento das funcionalidades contribuindo de maneira ativa. Um Micro-frontend, pensando como funcionalidade (Figura 2), pode ser uma página ou fragmentos de uma página, tal qual cada equipe possui responsabilidade fim-a-fim no desenvolvimento de cada item que será integrado apenas ao final do processo (GEERS, 2020).

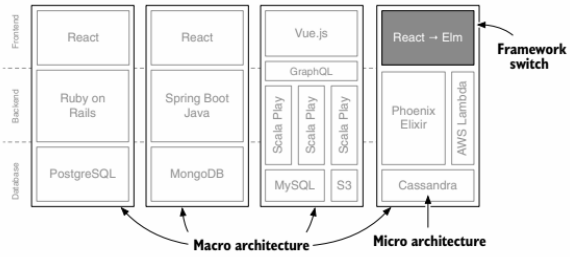
Há ainda a possibilidade de aplicar atualizações de ferramentas e *frameworks* a nível de equipe (arquitetura micro), pois cada time é responsável pela tecnologia aplicada em seu Micro-frontend, desde que esteja de acordo com convenções estabelecidas pela arquitetura macro, ou seja, entre todas as equipes (Figura 3). Desta forma, um dos grandes benefícios gerados pelos microsserviços e aplicados pelos Micro-frontends é a independência. No caso dos Micro-frontends, seja de estilos, *scripts* ou dependências da infraestrutura responsável pela execução da aplicação (GEERS, 2020).

Figura 2 - Recursos como interfaces de usuário isoladas.



Fonte: Geers (2020, p.8).

Figura 3 - Arquitetura macro



Fonte: Geers (2020, p.15).

A arquitetura de Micro-Frontends é uma abordagem apropriada para projetos de tamanho médio a grande (quando equipes ultrapassam 10 pessoas), pois a técnica de divisão das aplicações facilita escalar os projetos e desacoplar times. Outrossim, a abordagem de Micro-Frontends corresponde melhor ao desenvolvimento Web, ao contrário de plataformas para aplicativos nativos que requerem a construção de um único pacote que será enviado para o processo de avaliação de lojas de aplicativos e exige outras técnicas para sua composição (como

a exposição das funcionalidades por meio de uma API REST - *Representational State Transfer*) (GEERS, 2020).

A integração entre os Micro-Frontends, caso sejam construídos a nível de páginas, chamada de divisão vertical (utilizando a abordagem DDD), pode ser realizada optando pelas técnicas de composição *client-side*, *server-side* e *edge-side*. Caso seja empregada a abordagem de fragmentos (divisão horizontal), podem ser integrados por meio de composição *server-side* e *client-side* (GEERS, 2020).

Desta forma, a composição *client-side* utiliza principalmente um *Application Shell* como o disponibilizado pelo *meta-framework* Single-Spa[6]. Um *Application Shell* pode ser considerado similar a um *proxy front-end*, encapsulando e gerenciando a chamada de aplicações. Desta forma, cada Micro-Frontend pode ser uma pequena aplicação SPA[7], contendo toda lógica para sua execução, isoladamente (MEZZALIRA, 2021). Também, podem ser utilizados *iframes*, integrando páginas por meio de URLs, fornecendo alto isolamento. Ademais, há possibilidade da utilização de *Web Components* (*Custom Elements*[8], *Shadow DOM*[9] e HTML *Templates*).

Já na abordagem “lado da borda” (*edge-side*), a composição é realizada via CDN[10]. Outra possibilidade é realizar a formação via lado do servidor (*server-side*), que é obtida em tempo de execução ou de compilação, onde o servidor compõe a página final com base em uma visualização de todos os Micro-Frontends (MEZZALIRA, 2021). Ainda, para executar o roteamento do lado do servidor (*server-side*), pode-se utilizar um *web server* como Ngnix[11] (*proxy front-end*) ou outras ferramentas como Traefik[12] ou Varnish[13] (GEERS, 2020).

---

[6] Framework SPA que fornece um application shell com carregamento sob demanda, além de uma estrutura completa que facilita a unificação entre as aplicações e suporte para bibliotecas como React, Vue.js, Angular, Svelte ou Cycle.js (GEERS, 2020).

[7] Single-Page Application (SPA) é o processo no qual a aplicação carrega todos seus recursos na primeira requisição e substitui seus componentes por outros com informações atualizadas sem recarregar a página inteira a cada ação do usuário (JADHAD et al., 2015).

[8] Permite construir elementos personalizados que estão faltando na especificação HTML (GEERS, 2020).

[9] Possibilita isolar uma parte do DOM do restante da página (GEERS, 2020).

[10] Uma CDN (content delivery network) é uma rede de servidores que armazenam em cache ou guardam conteúdo da Web e entregam de maneira inteligente aos usuários com base em localização geográfica (HOSANAGAR et al., 2008).

[11] https://www.nginx.com/

[12] https://doc.traefik.io/traefik/

[13] https://varnish-cache.org/

Em cada abordagem de composição dos Micro-Frontends existe uma forma de se realizar o roteamento, que não é exclusivo, podendo ser combinado. Para o *server-side*, como toda a lógica concentra-se no servidor, deve-se rotear as solicitações na origem, podendo utilizar uma CDN para acompanhar as solicitações e escalar horizontalmente. Já para a composição *edge-side*, o roteamento é realizado por meio das URLs e a CDN atende à página requisitada para montar os Micro-Frontends. O roteamento utilizado no *client-side* é por estados do usuário, como renderizar uma área da aplicação para usuários autenticados ou uma *landing page* caso o acesso seja realizado pela primeira vez, podendo utilizar um *Application Shell* para carregar os Micro-Frontends como SPA, deixando o roteamento a cargo do mesmo (MEZZALIRA, 2021).

Mezzalira (2021) disponibiliza um método para escolher a melhor abordagem na construção de Micro-Frontends, de acordo com características próprias de cada projeto. O *Micro-Frontends Decisions Framework* (Figura 5) permite visualizar de maneira coerente qual é a abordagem de Micro-Frontend ideal, seja na identificação da necessidade, composição, no roteamento ou na comunicação.

Figura 5 - Framework de decisões para Micro-frontends

Micro-frontends definition	Composition	Routing	Communication
Horizontal	Client side Server side Edge side	Client side Server side Edge side	Event emitter Custom events Web storage Query strings
Vertical	Client side Server side	Client side Server side Edge side	Web storage Query strings

Fonte: Mezzalira (2021, p.35).

Segundo Peltonen et al. (2020), a comunicação entre os Micro-Frontends pode ser realizada por meio de injeções de eventos, no caso da divisão horizontal, onde cada Micro-Frontend dispara eventos para o objeto *window* de maneira que apenas os subscritos possam reagir. Já no caso de divisão vertical, a comunicação pode ser efetuada pela anexação de informações nas URLs, strings de consulta, ou até mesmo o *web storage* (*cookies, session storage* e *local storage*).

Caso seja utilizada a abordagem vertical aliada à comunicação via *web storage*, o time que detém o processo de login deve prover a autenticação e o status para os demais por meio de OAuth ou *JSON Web Tokens* (JWT), que é um meio *URL-safe* para troca de informações codificadas como objetos JSON criptografados entre duas partes (GEERS, 2020). Desta forma, o token é salvo no local ou *session storage* e é recuperado por cada aplicação para autenticar requisições web.

2.3 TRABALHOS RELACIONADOS

Nesta seção, é apresentada uma revisão da literatura existente, destacando estudos e descobertas importantes que contribuem para a compreensão do problema em discussão. Ao explorar os trabalhos relacionados, é possível identificar as lacunas de conhecimento, as abordagens utilizadas por pesquisadores anteriores e as contribuições mais significativas na área, o que é essencial para fundamentar o presente estudo.

Wang et al. (2020) apresenta uma solução baseada em Micro-Frontends, combinada à ideia da arquitetura orientada à serviços, para solucionar questões relacionadas ao alto acoplamento entre os negócios educacionais dos sistemas informacionais referidos no estudo, baseados em estruturas MVC tradicionais. É utilizado o método *Domain-Driven Design* (DDD) para análise de regras de negócio complexas e então possibilitar a divisão do sistema já existente.

Duarte et al. (2021), em sua proposta de arquitetura para realizar a gestão de identidade de usuários que utilizam vários sistemas independentes, faz uso da abordagem de Micro-Frontends, aliada à arquitetura de Microsserviços, para permitir entregas independentes e a autonomia total dos times. Ademais, é realizada a autenticação das aplicações por meio do protocolo *OpenID Connect* aliado à utilização de um repositório de tokens armazenados em banco de dados.

Pavlenko et al. (2020) explora os conceitos acerca da utilização de Micro-frontends no desenvolvimento Web aliado à implementação de Microsserviços e suas vantagens, como otimização no tempo de desenvolvimento e padrões de escalabilidade. Ademais, utiliza uma aplicação para estudo de caso, onde são discutidos os problemas identificados durante o desenvolvimento (roteamento entre

Micro-frontends e armazenado do estado da aplicação), bem como o aumento da complexidade da estrutura.

Peltonen, et al. (2021) detalha em seu trabalho as questões principais que levaram companhias de software a utilizarem a abordagem de Micro-Frontends, como independência de desenvolvimento, entrega à produção e gerenciamento, equipes autônomas e o processo de integração de novos desenvolvedores é simplificado. Também, apresentam possíveis problemas na adoção da tecnologia, como dependências compartilhadas, duplicação de código, monitoramento e acessibilidade afetados.

3 PROVA DE CONCEITO

Com o intuito de validar a proposta e alcançar o objetivo do trabalho, foi conduzida uma prova de conceito (POC), onde, com base nas informações obtidas durante a pesquisa e análise do sistema de estudo de caso proposto, constitui-se uma arquitetura de Micro-frontends, implementada e avaliada com base nos trabalhos disponíveis na literatura.

3.1 TIPO DE PESQUISA

Realizou-se uma pesquisa exploratória a respeito dos conceitos da arquitetura baseada em Micro-frontends, onde tornou-se evidente as intercorrências e desafios causados por sistemas amplos ou legados, como escalabilidade reduzida, alto acoplamento de implementação e implantação, defasagem técnica e conflito de código durante o versionamento.

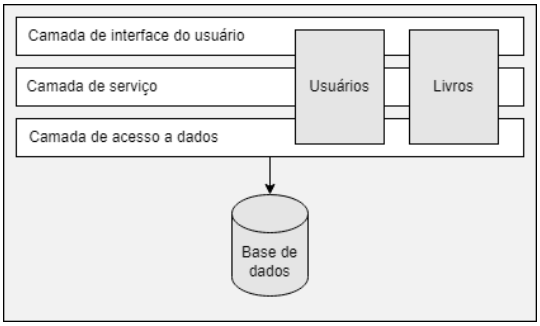
Portanto, com base em tais desafios identificados e com o propósito de fornecer uma arquitetura flexível e aplicável em cenários distintos (criação de novos projetos ou migração de sistemas legados), constituiu-se uma proposta aplicando a abordagem *Domain Driven Design* a fim de dividir o *front-end* e *back-end* em componentes independentes e autônomos, permitindo que diferentes equipes de desenvolvimento possam atuar de forma paralela e eficiente.

Ademais, após identificar a escassez de resultados referentes à avaliação de desempenho entre Micro-frontends e aplicações construídas utilizando a abordagem de página única, como as *Single Page Applications (SPA)*, foi realizada uma comparação entre a aplicação construída com base na arquitetura de Micro-frontends e uma aplicação SPA em ReactJS, contendo as mesmas funcionalidades.

3.2 ANÁLISE

O sistema de estudo de caso foi concebido em três camadas: interface do usuário, camada de negócios e de dados. Seu propósito é servir como uma biblioteca online. Possui tela de login, criação de novos usuários, um menu, telas para cadastro, listagem e locação de livros. Por ser um sistema monolítico, apresenta acoplamento de implementação e implantação e escalabilidade afetada.

Figura 6 - Arquitetura do sistema de estudo de caso



Fonte: Da autora.

Desta forma, foram delimitados os requisitos funcionais para a aplicação: deve permitir a criação de novos usuários; o acesso de usuários já cadastrados ao sistema; listagem de livros já cadastrados; o usuário pode cadastrar novos livros; retirar livros e devolver livros retirados anteriormente.

3.3 ARQUITETURA

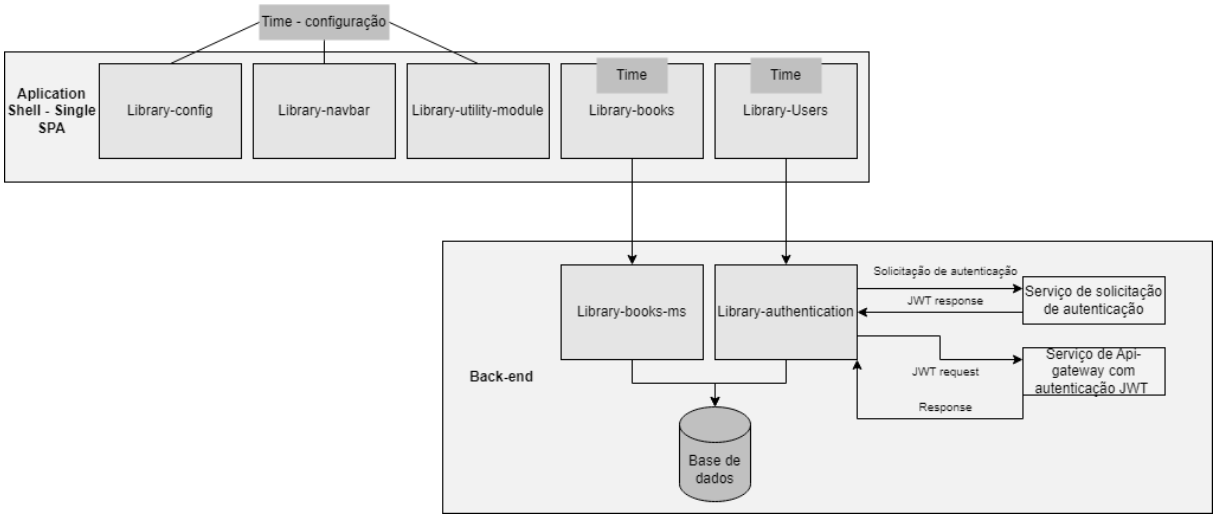
Foi escolhida a estratégia de divisão vertical, utilizando os princípios da abordagem *Domain Driven Design*, após análise do sistema proposto e de acordo com a literatura apresentada. Ademais, seguindo a lógica do padrão DDD, foi idealizado um Micro-frontend para cada funcionalidade presente na plataforma: cadastro e locação de livros estão concentrados na aplicação denominada “library-books”; as operações de *signin* e *signup* estão localizadas na aplicação “library-users”.

A aplicação “library-navbar” contém o roteamento da aplicação; o módulo “library-utility-module” possui todas as funções de manipulação de estado da plataforma, bem como chamadas assíncronas e componentes genéricos para utilização nas demais aplicações. O módulo “library-config” possui a lógica necessária para a integração e carregamento dos demais Micro-frontends. O carregamento e gerenciamento do sistema é coordenado por um *Application Shell* chamado Single-Spa. Todos os cinco Micro-frontends utilizam a biblioteca ReactJS, além dos padrões de design Hooks[14] e Context API[15].

Para a arquitetura de microsserviços presente no *back-end*, foram projetadas duas aplicações para proporcionar controladores REST para as aplicações *front-end.* Os projetos foram criados utilizando Java 18[16], Spring Boot[17] (framework versão 3.1.0) e Maven[18] (versão 2). Todos os tokens gerados devem ser armazenados em uma tabela de banco de dados, para posterior consulta durante a autenticação. Por fim, para realizar a persistência dos dados utilizou-se um único Sistema de Gerenciamento de Banco de Dados (SGBD) relacional, a fim de auxiliar a implementação.

[14] https://legacy.reactjs.org/docs/hooks-intro.html
[15] https://react.dev/reference/react/useContext
[16] https://www.oracle.com/java/technologies/javase/jdk18-archive-downloads.html
[17] https://spring.io/projects/spring-boot
[18] https://maven.apache.org/index.html

Figura 7 - Arquitetura proposta



Fonte: Da autora.

3.4 IMPLEMENTAÇÃO

A partir da proposta de arquitetura, criou-se sete projetos e seus repositórios no GitHub[19]. A aplicação “library-authentication” utiliza dependências para gerenciamento de conexões com o SGBD e manipulação de *Json Web Token* (io.jsonwebtoken). O serviço de autenticação é responsável por registrar novos usuários e também gerencia os novos acessos à plataforma por meio do *login*. Ainda, o projeto “library-books-ms” possui dois controladores que gerenciam as operações de CRUD (*create, read, update e delete*) de livros e locações.

O projeto “library-config” é o componente principal da arquitetura, importando e declarando os demais Micro-frontends, por meio do arquivo index.ejs (Figura 8), além de gerenciar seus carregamentos com base em rotas definidas no arquivo *tcc-root-config.js* (Figura 9). O *framework* Single-SPA apresenta, em sua estrutura padrão, um arquivo *Javascript* na pasta *source* do projeto que fornece funções para gerenciar o ciclo de vida do Micro-frontend (Figura 10).

[19] https://github.com/

Figura 8 - Arquivo index.ejs do projeto library-config

```
<script type="systemjs-importmap">
  {
    "imports": {
      "@tcc/root-config": "//localhost:9000/tcc-root-config.js",
      "@tcc/library-books": "//localhost:8500/tcc-library-books.js",
      "@tcc/library-navbar": "//localhost:8400/tcc-library-navbar.js",
      "@tcc/library-users": "//localhost:8300/tcc-library-users.js",
      "@tcc/library-utility-module": "//localhost:8100/tcc-library-utility-module.js"
    }
  }
</script>
```

Fonte: Da autora.

Figura 9 - Arquivo tcc-root-config.js do projeto library-config

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

registerApplication({
  name: "@tcc/library-books",
  app: () =>
    System.import("@tcc/library-books"),
  activeWhen: (location) => location.pathname === "/books",
});

registerApplication({
  name: "@tcc/library-users",
  app: () =>
    System.import("@tcc/library-users"),
  activeWhen: (location) => location.pathname === "/auth",
});

registerApplication({
  name: "@tcc/library-navbar",
  app: () => System.import("@tcc/library-navbar"),
  //activeWhen: (location) => location.pathname === "/library-navbar"
  activeWhen: ["/"]
});


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

Fonte: Da autora.

Figura 10 - Função para gerenciamento do ciclo de vida da aplicação

```
import React from "react";
import ReactDOM from "react-dom";
import singleSpaReact from "single-spa-react";
import App from "./App";

const lifecycles = singleSpaReact({
  React,
  ReactDOM,
  rootComponent: App,
  errorBoundary(err, info, props) {
    // Customize the root error boundary for your microfrontend here.
    return null;
  },
});

export const { bootstrap, mount, unmount } = lifecycles;
```

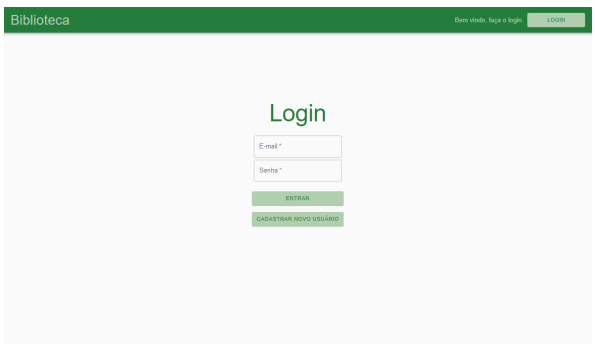
Fonte: Da autora.

O projeto “library-utility-module” é um pacote de utilitários, contendo chamadas para os microsserviços, funções para gerenciamento do *web storage*, criação de contextos para os demais Micro-frontends e contém componentes ReactJS genéricos. Já o projeto “library-navbar” contém funções para roteamento das telas.

Com relação aos outros dois projetos, a aplicação “library-users” é responsável por realizar a chamada ao *controller* REST do microsserviço “library-authentication”, gerando novos usuários ou realizando a autenticação. O token é salvo no *local storage* da aplicação e utilizado nos demais projetos. A aplicação “library-books” contém as interfaces de usuário para listagem e cadastro de livros e também funções para realizar a locação.

Referente aos padrões aplicados, o *Hooks* permite a utilização de funções e manipulação de estados da aplicação em componentes funcionais. O Context API, por sua vez, é um *hook* utilizado para transportar informações de diferentes componentes dentro de um contexto sem a necessidade de utilizar *properties*, característica da biblioteca ReactJS; neste sentido, é criado um contexto para cada informação armazenada e declarado no componente principal de um determinado módulo, de forma que todos os componentes contidos por ele estejam aptos a manipular a informação dentro do contexto gerado. As Figuras 11 e 12 demonstram a identidade visual da aplicação.

Figura 11 - Login da aplicação.



Fonte: Da autora.

Figura 12 - Listagem de livros da aplicação.

Fonte: Da autora.

4 AVALIAÇÃO E EXPERIÊNCIAS

A avaliação para analisar e comparar as aplicações implementadas, Micro-frontends e SPA, foi realizada com base nas métricas de tempo médio de *build*, tamanho do pacote gerado ao realizar o *build*, tamanho do projeto (sem a pasta *node_modules*), tempo médio de *start* e tempo médio de carregamento da página *home*. Todos os testes foram realizados em máquina local, com as seguintes especificações: processador Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz; RAM 16GB; Windows 11 Home x64 e Hard Disk SSD 240GB. Ainda, foram performados testes utilizando a extensão de navegador Google Lighthouse[20], visto que ferramentas como esta são comuns para testes de desempenho em pesquisas envolvendo software (HERICKO et al, 2021). Os resultados obtidos foram organizados em uma tabela comparativa.

Também, vale ressaltar que as configurações utilizadas para *build* e *webpack* são as disponibilizadas na geração do projeto por meio dos *frameworks* utilizados. Outrossim, as avaliações foram realizadas analisando apenas as aplicações pertencentes ao *front-end*. O tempo médio foi obtido após três execuções de *build*, com exceção da primeira geração do pacote. Em vista disto, a Tabela 1 demonstra os resultados obtidos.

Tabela 1 - Testes dos Micro-Frontends e SPA para o tempos de build, tamanho do pacote e tempo médio de carregamento do primeira páginas.

Aplicação	Primeiro build	Média de builds seguintes	Tamanho pacote de build	Tamanho projeto (sem node modules)	Média de tempo para start da aplicação	Tempo para carregamento da página home
library-config	10863 ms	920 ms	16,0 KB	508 KB	2168 ms	55 ms
library-utility-module	15169 ms	5469 ms	920 KB	1,92 MB	4714 ms	-
library-navbar	18189 ms	6152 ms	0,98 MB	1,99 MB	5327 ms	-
library-users	16293 ms	5913 ms	1,30 MB	2,29 MB	4366 ms	-
library-books	26354 ms	11299 ms	3,31 MB	3,88 MB	5731 ms	-
library-spa	34820 ms	19010 ms	3,95 MB	5,26 MB	6923 ms	53 ms

Fonte: Da autora.

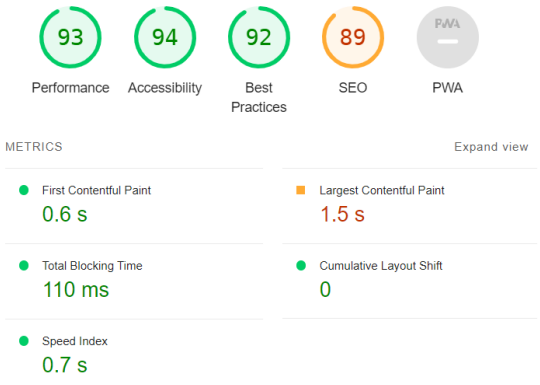
[20] https://developer.chrome.com/docs/lighthouse/overview/

O tempo de primeiro *build* é superior às demais execuções por conta da criação e indexação dos arquivos, pois caso a pasta não seja excluída, os arquivos serão atualizados nas demais ocasiões. Também, pode-se observar que o tempo de geração dos arquivos de *build* é maior em todas as execuções para a aplicação SPA. Ainda, é válido afirmar que a mesma regra é aplicada para o tamanho de pacote de *build*, tamanho do projeto, tempo médio para *start* da aplicação e carregamento da página home. Este último apresenta uma diferença irrelevante, uma vez que o usuário não percebe qualquer atraso no carregamento.

Utilizando a ferramenta Google Lighthouse, foram encontrados dados semelhantes para os Micro-frontends (Figura 13) e a aplicação SPA (Figura 14). A ferramenta analisou o tempo mínimo e máximo para carregamento do primeiro componente em tela, tempo de bloqueio até o usuário poder interagir com a página, movimento dos componentes de tela e indexação da página, acessibilidade, boas práticas e ferramentas de otimização (SEO).

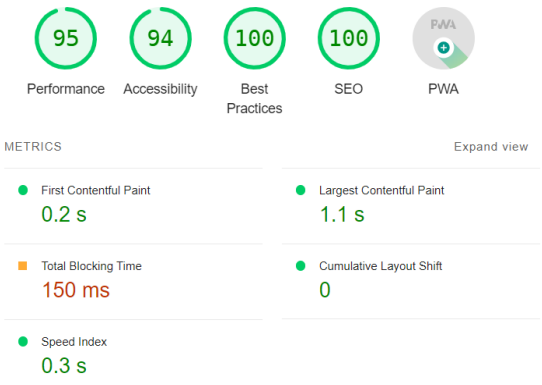
De acordo com os dados obtidos, a aplicação SPA performou de maneira superior aos Micro-Frontends. Tal resultado inferior dos Micro-Frontends se deve por conta dos seguintes fatores: falta de contraste nas cores utilizadas, console contendo erros, HTML sem tag “meta” com a descrição da aplicação e Web App sem informações sobre funcionamento como PWA.

Figura 13 -Teste de performance para a aplicação Micro-Frontends.



Fonte: Da autora.

Figura 14 - Teste de performance para a aplicação SPA.



Fonte: Da autora.

Ainda, foi observado que a versão do *webpack* para execução do *build* dos Micro-frontends é desigual entre as aplicações. Os projetos “library-config” e “library-books” utilizam a versão 5.79.0; já os projetos “library-navbar” e “library-users” utilizam a versão 5.82.0; ainda, o projeto “library-utility-module” utiliza a versão 5.80.0. Tal diferença de versões observada no arquivo package.json foi gerada no momento da criação dos projetos utilizando o *framework* Single-SPA, mesmo que para aplicações semelhantes, como o “library-books” e “library-users”. Outrossim, evidenciou-se que a aplicação “library-books” ultrapassa o limite definido pelo próprio *framework* para tamanho de arquivos e *assets*.

Por fim, com base na pesquisa e implementação realizada, considerando o contexto do desenvolvimento do presente trabalho, avaliou-se algumas características da adoção da abordagem de Micro-frontends, sendo organizadas em vantagens e desafios (Tabela 2).

Tabela 2 – Vantagens e desafios em relação à adoção da arquitetura de Micro-frontends.

Vantagens	Desafios
● Versatilidade na escolha da tecnologia; ● Cada projeto é independe, seja em relação ao repositório, tecnologias, padrões de projeto ou regras para deploy; ● Cada funcionalidade pode ser entregue à produção isoladamente; ● Podem e devem ser criadas regras para gerenciamento de falhas.	● Muita diferença entre projetos pode ocasionar em diminuição da performance dos desenvolvedores durante trocas de projeto; ● Falhas, se não forem isoladas, podem comprometer o funcionamento da aplicação inteira; ● Não houve ganho expressivo de performance em relação às aplicações SPA convencionais.

Fonte: Da autora.

5 CONSIDERAÇÕES FINAIS

Diante da análise realizada e conforme resultados obtidos na avaliação, observa-se que a arquitetura de Micro-frontends, utilizando o *framework* Single-SPA e a biblioteca ReactJS, não apresenta um ganho expressivo de performance em relação às aplicações SPA convencionais construídas com os mesmos parâmetros. Portanto, a sua utilização não é determinada pela performance em si, mas sim pela necessidade do projeto e do time. Como mencionado anteriormente, a finalidade é

auxiliar na organização de times e deploy de aplicações com muitas funcionalidades e regras de negócio abrangentes. Portanto, é útil quando existe a necessidade de coordenação de times com mais de dez pessoas.

Ainda, a implementação utilizando o *framework* Single-SPA mostrou-se simples, por existirem manuais para a geração dos projetos, onde é necessário apenas seguir as regras documentadas. A dificuldade mais notável da implementação é a construção das rotas, onde são utilizados redirecionamentos para o carregamento das micro-aplicações por meio do projeto “library-config”, visto que não foi possível encontrar uma forma para exportar componentes isolados do Micro-frontend quando este utiliza o arquivo das especificações para o ciclo de vida da aplicação, declarando apenas o *root component*. Em relação aos microsserviços, a elaboração demonstra-se compreensível e acessível de realizar, porém com uma pequena dificuldade referente às implementações sugeridas de versões mais antigas do *framework* Spring-security e compatibilidade com versões mais novas do *framework* Spring-boot, utilizada para a criação dos projetos, durante a implementação do filtro para acesso de rotas protegidas. Também, referente ao método de autenticação utilizado, por meio de *JWT,* é possível identificar uma lacuna na segurança da aplicação por conta do armazenamento no *local storage*.

Para trabalhos futuros, sugere-se a exploração de possíveis alterações identificadas na estrutura dos componentes, bem como a evolução do *layout* e experiência do usuário. Também, ajustar ou reestruturar o projeto library-config para que esteja de total acordo com os padrões do *framework* utilizado, além de aprimorar a segurança da aplicação e armazenamento de tokens. Ainda, há possibilidade da utilização de mais frameworks como Angular e VueJS para o desenvolvimento de novos módulos. Por fim, eventualmente pode ser realizado o deploy das aplicações para cloud, a fim de aplicar os mesmos métodos de testes locais.

REFERÊNCIAS

DUARTE, G. TAMBERLINI, V. NEVES, N. *Proposta de arquitetura baseada em micro serviço para autenticação utilizando openID.* Mackenzie, 2021.

FOWLER, S. *Microsserviços Prontos Para a Produção: Construindo Sistemas Padronizados em uma Organização de Engenharia de Software.* 1 ed. São Paulo: Novatec Editora, 2017.

GEERS, M. *Micro Frontends in action.* 1st ed. Shelter Island: Manning Publications, 2020.

HERICKO, A. et al. *Towards representative web performance measurements with Google Lighthouse*. 7th Student Computer Science Research Conference (StuCoSReC), 2021.

HOSANAGAR, K. et al. *Service adoption and pricing of content delivery network (CDN) services.* Management Science, v. 54, n. 9, p. 1579-1593, 2008.

JADHAD, M. SAWANT, B. DESHMUKH, A. *Single page application using angularjs.* International Journal of Computer Science and Information Technologies, v. 6, n. 3, p. 2876-2879, 2015.

KARIM, N. SHUKUR, Z. AL-BANNA, M. *UIPA: user authentication method based on user interface preferences for account recovery process.* Journal of Information Security and Applications, v. 52, p. 102466, 2020.

MEZZALIRA, L. *Building Micro-Frontends: Scaling Teams and Projects, Empowering Developers.* 1st ed. Sebastopol: O'Reilly Media, 2022.

MONTESI, F. WEBER, J. *Circuit breakers, discovery, and API gateways in microservices.* arXiv preprint arXiv:1609.05830, 2016.

NEWMAN, S. *Migrando sistemas monolíticos para microsserviços: Padrões evolutivos para transformar seu sistema monolítico.* 1 ed. São Paulo: Novatec Editora, 2020.

NEWMAN, S. *Building microservices.* 1st ed. Sebastopol: O'Reilly Media, Inc., 2015.

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.

PELTONEN, S. MEZZALIRA, L. TAIBI, D. *Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review.* Information and Software Technology, v. 136, p. 106571, 2021.

WANG, D. et al. *A novel application of educational management information system based on micro frontends.* Procedia Computer Science, v. 176, p. 1567-1576, 2020.