Abdul Kevin Alexis

Análise comparativa entre *frameworks Single Page Application* (SPA)

Belo Horizonte

2023

Abdul Kevin Alexis

Análise comparativa entre *frameworks Single Page Application* (SPA)

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

Departamento de Computação

Curso de Engenharia da Computação

Orientador: Prof. Dr. Edson Marchetti da Silva

Belo Horizonte

2023

Centro Federal de Educação Tecnológica de Minas Gerais

Curso de Engenharia de Computação

Avaliação do Trabalho de Conclusão de Curso

Aluno: Abdul Kevin Alexis

Título do trabalho: Análise comparativa entre frameworks Single Page Application (SPA)

Data da defesa: 06/12/2023

Horário: 08:00

Local da defesa:

https://conferenciaweb.rnp.br/conference/rooms/edson-marchetti-da-silva-2/invite?showsuggestion=false

O presente Trabalho de Conclusão de Curso foi avaliado pela seguinte banca:

Professor Edson Marchetti da Silva – Orientador
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professora Kecia Aline Marques Ferreira – Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Professor Eduardo Cunha Campos – Membro da banca de avaliação
Departamento de Computação
Centro Federal de Educação Tecnológica de Minas Gerais

Dedico este trabalho a todos da área de Engenharia de Computação ou de Software.

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador e às pessoas que contribuíram na minha formação.

Em seguida, agradeço o Povo Brasileiro, este país que me acolheu e apoiou durante toda a minha estadia aqui. Estou muito grato pela oportunidade e crescimento durante meu estudo.

Também gostaria de agradecer àqueles que me permitiram viajar para o Brasil. Sem a sua preciosa ajuda, não teria conseguido realizar meu sonho.

Agradeço especialmente a:

- Esther Fleurijuste, minha querida, por seu apoio incondicional;
- Samuel Michel, por sua ajuda na obtenção da bolsa de estudos;
- Marie Alice Sirène, minha madrinha, por sempre acreditar em mim;
- Nicolas Dorvilus, que descansou em paz, por seu apoio e suporte incondicional;
- Beat Lockslina Edmond, por sua generosidade e solidariedade;
- Guerlovecia Pierre, por sua ajuda na correção;
- Padre Benh Donais (CJMS), por seu incentivo e orientação;
- Socolavim (Fils-Aimé) e DGS Communication (Gary Dalencourt), pelo apoio financeiro;
- Carline Alexis, Dor Erlens, Rose Darline Camille, Maxime Termitus, @BUC (Laf, Mahatma, AMI, 2tay, Sly...), sou muito grato a todos vocês por fazerem parte dessa jornada. Seu apoio foi fundamental para que eu pudesse alcançar meus objetivos.

Obrigado!

"Crença + Perseverança = Realização ou Sucesso"
(Autoria Propriá)

Resumo

O presente trabalho apresenta os resultados de um estudo experimental realizado para comparar três *frameworks* de desenvolvimento *front-end*: Angular, React e Vue. Os *frameworks* Single Page Application (SPA) são uma das principais tendências de desenvolvimento web atual. Eles permitem criar aplicações web que carregam apenas uma página, o que proporciona uma experiência de usuário mais fluida e responsiva. Com a proliferação de *frameworks* SPA disponíveis, escolher o ideal para um determinado projeto pode ser uma tarefa desafiadora. O objetivo deste trabalho é realizar uma análise comparativa entre os *frameworks* Angular, React e Vue, considerando o tamanho do pacote, tempo de carregamento, tempo de renderização, uso de memória e tempo de execução. Como método, foi criado um aplicativo baseado na arquitetura microsserviço usando cada uma das alternativas, em seguida, esses aplicativos foram comparados para avaliar qual apresenta melhor desempenho. A análise comparativa revelou que o React apresenta o melhor desempenho geral, seguido pelo Vue e pelo Angular. Em termos de tamanho do pacote gerado para pôr em produção, o Vue tem o menor *bundle*, o React tem o segundo menor e o Angular tem o maior entre os três. Em termos de tempo de carregamento, o React e o Vue são os mais rápidos, seguidos pelo Angular. Em termos de tempo de renderização, o Angular e o React, que têm um desempenho similar, renderizam mais rápidos, seguido pelo Vue, que é mais lento na renderização. Em termos de uso de memória, o Vue faz uso de menos memória, O React vem segunda posição seguido pelo Angular que mais usa memoria. Em termos de tempo de execução, o React tem o melhor desempenho em todas as operações de CRUD, seguido pelo Vue e pelo Angular. Com base nos resultados obtidos, foi recomendado o React ou o Vue como opção para aplicativos que precisam ser carregados rapidamente. Para aplicativos que precisam usar menos memória, o Vue é a melhor opção. Para aplicações que precisam ter um bom desempenho em operações de CRUD, o React é a melhor opção. Os resultados deste estudo contrastam com os de Kaluža, Troskot e Vukelić (2018), que concluíram que não há um framework JavaScript ideal para MPA e SPA. Ziani (2021) também realizou uma análise comparativa de frameworks JavaScript, mas seu estudo foi qualitativo, enquanto o presente estudo é quantitativo. Portanto, os resultados deste estudo corroboram a afirmação de Diniz-Junior et al. (2022), de que o Angular apresenta melhor desempenho em termos de renderização. Os resultados deste estudo são relevantes para engenheiros de software iniciantes, pois fornecem informações valiosas para a escolha da melhor tecnologia de SPA para software baseado na arquitetura microsserviço.

Palavras-chave: *frameworks*. SPA. Desenvolvimento Web.Angular. React. Vue

Abstract

This work presents the results of an experimental study carried out to compare three front-end development frameworks: Angular, React and Vue. Single Page Application (SPA) frameworks are one of the main trends in current web development. They allow you to create web applications that load just one page, which provides a more fluid and responsive user experience. With the proliferation of SPA frameworks available, choosing the ideal one for a given project can be a challenging task. The objective of this work is to carry out a comparative analysis between the Angular, React and Vue frameworks, considering package size, loading time, rendering time, memory usage and execution time. As a method, an application based on microservice architecture was created using each of the alternatives, then these applications were compared to evaluate which one presents better performance. The comparative analysis revealed that React has the best overall performance, followed by Vue and Angular. In terms of bundle size generated to put into production, Vue has the smallest tbundle, React has the second smallest and Angular has the largest of the three. In terms of loading time, React and Vue are the fastest, followed by Angular. In terms of rendering time, Angular and React, which have similar performance, render faster, followed by Vue, which is slower in rendering. In terms of memory usage, Vue uses less memory, React comes second followed by Angular which uses the most memory. In terms of execution time, React has the best performance in all CRUD operations, followed by Vue and Angular. Based on the results obtained, React or Vue was recommended as an option for applications that need to load quickly. For applications that need to use less memory, Vue is the best option. For applications that need to perform well in CRUD operations, React is the best option. The results of this study contrast with those of Kaluža, Troskot and Vukelić (2018), who concluded that there is no ideal JavaScript framework for MPA and SPA . Ziani (2021) also carried out a comparative analysis of JavaScript frameworks, but his study was qualitative, while the present study is quantitative. Therefore, the results of this study corroborate the statement by Diniz-Junior et al. (2022), that Angular presents better performance in terms of rendering. The results of this study are relevant for beginning software engineers, as they provide valuable information for choosing the best SPA technology for software based on microservice architecture.

Keywords: frameworks. SPA. Web Development.Angular. React. Vue

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AHP	Analytic Hierarchy Process
API	Application Programming Interface
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DOM	Document Object Model
HTML	Hyper Text Markup Language
IDE	Integrated Development Environment
MPA	Multi-page application
REST	REpresentational State Transfer
SEO	Search Engine Optimization
SGBD	Sistema de Gerenciamento de Banco de Dados
SPA	Single Page Aplication

Sumário

1 Introdução

Atualmente, o mundo assiste a um rápido processo de evolução na necessidade de facilidades de informação em termos de quantidade, qualidade e acesso. As pessoas estão mais conectadas em todo o mundo e podem trabalhar com comunidades fora de suas redes habituais (NEWMAN et al., 2016).

Perante esses fatos, os desenvolvedores precisam escolher, dentre as diferentes arquiteturas e tecnologias existentes, qual melhor se adequa à sua demanda em cada caso. De acordo com Kornienko, Mishina e Melnikov (2021), as soluções da Web são preferidas às plataformas móveis ou de *desktop* ao criar novos serviços. Essa escolha se justifica pelo fato que quaisquer recursos on-line não impõem requisitos rígidos de hardware ao usuário (memória de armazenamento, memória de acesso aleatório, consumo de energia da unidade central do processador), não vinculam a interação a um tipo específico de plataforma (computador pessoal, *smartphone*, dispositivo especializado). Ou seja, as aplicações web podem ser acessadas fazendo uso apenas de um navegador web.

De acordo com seu *design*, uma aplicação web pode ser desenvolvida como *Multi Page Application* (MPA) ou *Single Page Application* (SPA) (KALUŽA; TROSKOT; VUKELIĆ, 2018). A MPA é o modelo convencional de construção de software *front-end*, no qual o navegador renderiza uma página inteira para cada rota ou aba na aplicação para apresentar os dados. Geralmente, os dados são renderizados no lado servidor e apenas um documento HTML é mandado para o lado cliente a cada requisição. No entanto, a SPA é uma combinação de soluções, dividindo uma aplicação web em duas partes (*front-end* e *back-end*) e organizando a interação entre elas. A parte de *back-end* lida com a lógica, processa solicitações, trabalha com o banco de dados, enquanto a parte de *front-end* forma a visão externa com base nos dados recebidos do *back-end* e exibe o resultado no navegador. A troca de informações entre as partes cliente e servidor do aplicativo, por norma, ocorre por meio de uma interface de programação de aplicativos especialmente desenvolvida. As aplicações web modernas, que não são desenvolvidas de maneira tradicional, são baseadas no conceito SPA.

De acordo com W3Techs (2022), a JavaScript é usada como linguagem de programação do lado do cliente por 97,9% de todos os sites. Por existirem muitos *frameworks* JavaScript disponíveis, escolher um para realizar um projeto pode ser uma tarefa muito desafiadora. Nesse contexto, segundo a StackOverflow (2021) as alternativas de desenvolvimento mais utilizadas são React.js, JQuery, Express, Angular, Vue.js. Sendo assim, quais critérios adotar para escolher um *framework* dentre os mais utilizados?

Segundo as conclusões apresentadas no trabalho de Ziani (2021), não há uma

resposta categórica confiável e unanimemente aceita sobre qual o melhor *framework* JavaScript para usar no desenvolvimento de uma aplicação web de *front-end*. Similarmente, Ferreira e Zuchi (2018) concluíram que cada *framework* pode trazer determinada vantagem ao projeto, e cabe ao desenvolvedor definir quais características são mais importantes para sua aplicação.

Existem várias alternativas de SPA diferentes disponíveis, cada uma com seus próprios pontos fortes e fracos, ou seja, cada *framework* tem seu próprio conjunto exclusivo de recursos e capacidades. Angular, React e Vue são três entre os *frameworks* JavaScript mais populares para desenvolvimento de aplicativo de página única. Eles são todos baseados em componentes, o que facilita a modularização e a manutenção do código. Além disso, todos eles oferecem uma ampla gama de recursos e bibliotecas, o que os torna adequados para uma variedade de aplicações.

A escolha desses três frameworks para uma análise comparativa é justificada pelos seguintes motivos:

- relevância: Esses *frameworks* são os mais utilizados no mercado, o que garante que os resultados da análise sejam relevantes para uma ampla gama de profissionais;
- diversidade: Os *frameworks* apresentam diferentes abordagens de desenvolvimento, o que permite uma análise mais abrangente;
- potencial de contribuição: A análise comparativa pode fornecer *insights* valiosos para engenheiros de software que estão avaliando qual *framework* utilizar para seus projetos.

Portanto, realizar a melhor escolha para um determinado projeto dependerá dos requisitos específicos.

1.1 Objetivo

O propósito deste trabalho é realizar uma análise comparativa entre a plataforma SPA Angular, a biblioteca React e o *framework* Vue.[1] A análise comparativa das ferramentas para o desenvolvimento SPA pode ser realizada usando uma variedade de métodos. Um método comum é criar uma tabela que compare as diferentes alternativas testadas a partir de vários critérios, tais como: facilidade de uso, desempenho e recursos. Outro método é criar um aplicativo de demonstração para cada uma das alternativas testadas, em seguida, comparar os aplicativos para ver qual apresenta melhor desempenho. Para

[1] No contexto deste trabalho, cabe ressaltar a complexidade inerente à busca por um termo consensual que englobe as três tecnologias de SPA: a plataforma Angular, a biblioteca React e o framework Vue. Com o intuito de simplificar a comunicação e evitar ambiguidades, adotaremos o termo "*framework*"de maneira genérica para referenciar essas tecnologias ao longo do documento.

isso foi implementado um aplicativo na arquitetura de microsserviços usando cada uma dessas alternativas na elaboração do *front-end*. A partir desses aplicativos desenvolvidos, foram realizados testes para medir o tamanho do código gerado, o tempo de carregamento, além de mensurar para diferentes cenários de carga, o tempo de renderização, o uso de memória e o tempo de execução em cada caso.

1.2 Justificativa

A realização deste trabalho se justifica, porque uma análise comparativa de *frameworks* SPA pode ser uma ferramenta valiosa para desenvolvedores que estão considerando usar uma ferramenta SPA para um novo projeto. Ao comparar as diferentes alternativas, os desenvolvedores podem identificar qual delas atende melhor as necessidades de seu projeto.

A seguir são apresentados alguns dos benefícios nos quais essa análise comparativa buscou fazer:

- ajudar os desenvolvedores a identificar a alternativa de desenvolvimento *front-end* SPA que melhor atende às necessidades de seu projeto;
- auxiliar os desenvolvedores a economizar tempo e dinheiro, evitando a necessidade de experimentar diferentes alternativas;
- ajudar os desenvolvedores a escolher um *framework* suportado por uma grande comunidade de usuários;
- auxiliar os recém-formados ou os iniciantes a escolher a melhor, dentre as várias alternativas existentes, para estudar e se desenvolver.

No geral, uma análise comparativa de *frameworks* SPA é uma ferramenta valiosa para desenvolvedores que estão considerando usar uma *framework* SPA para um novo projeto. Ao realizar essa análise, os desenvolvedores podem ter certeza de que estão fazendo a melhor escolha para suas necessidades e que estão bem informados sobre as diferentes ferramentas e como elas funcionam.

1.3 Estrutura do trabalho

O restante deste trabalho está organizado da seguinte maneira. O capítulo 2 apresenta o referencial teórico. O capítulo 3 trata dos trabalhos relacionados. O capítulo 4 discute a metodologia proposta. O capítulo 5 apresenta os resultados, enquanto o capítulo 6 apresenta as conclusões.

2 Referencial Teórico

Este capítulo apresenta o referencial teórico que fundamenta esta pesquisa, fornecendo as bases conceituais e teóricas necessárias para a compreensão do tema em estudo. Serão abordados os principais conceitos, teorias e estudos que embasam a análise.

2.1 O início da Web

É muito comum ouvir os nomes Google, Facebook, Twitter, Instagram..., etc. antes de ouvir a expressão "rede social". Afinal, esses gigantes da tecnologia impulsionaram a busca de informações e as interações entre as pessoas. Devido à necessidade de comunicação contínua visando permanecer conectado a tudo o que acontece, é difícil viver sem essas tecnologias no mundo de hoje. Elas são ferramentas indispensáveis para realizar as interações pessoais e de trabalho, fazendo parte da nossa vida diária. E tudo isso começou com uma tecnologia chamada Web na década de 1980.

De acordo com Berners-Lee et al. (1994), a web representa uma série de coisas, que devem ser distinguidas. Essas coisas incluem:

- a ideia de um mundo de informação sem limites em que todos os itens tenham uma referência pela qual possam ser recuperados;
- o sistema de endereços (URI) que o projeto implementou para tornar este mundo possível, apesar de muitos protocolos diferentes;
- um protocolo de rede (HTTP) usado por servidores web nativos oferecendo desempenho e recursos não disponíveis de outra forma;
- uma linguagem de marcação (HTML) que todo cliente W3 deve entender, e é usada para a transmissão de coisas básicas, como texto, menus e informações simples de ajuda on-line pela rede;
- o conjunto de dados disponível na Internet usando todos ou alguns dos itens listados acima.

Berners-Lee é considerado o pai da Web. As principais bases inventadas por ele deram surgimento à web tal como conhecemos hoje em dia. No entanto, não foram suficiente para tornar a web suficientemente interativa do lado do *browser*. De acordo com o site Mozilla (2023), sempre que uma página da web apresenta recursos além da exibição de informações estáticas, como a exibição de conteúdo atualizado periodicamente, mapas

interativos ou gráficos animados em duas ou três dimensões, é altamente provável que o JavaScript esteja envolvido nesse processo. Portanto, a Seção 2.2 traz uma breve introdução sobre esse tema.

2.2 JavaScript

O JavaScript, às vezes abreviada para JS, é uma linguagem leve, interpretada e baseada em objetos com função *first-class*,[1] mais conhecida como a linguagem de *script* para páginas Web. Ela é usada também em vários outros ambientes sem *browser*, tais como: node.js, Apache CouchDB e Adobe Acrobat. O JavaScript é uma linguagem baseada em protótipos, multi-paradigma e dinâmica, suportando estilos de orientação a objetos, imperativos e declarativos como, por exemplo, a programação funcional (JAVASCRIPT, 2021).

JavaScript é uma linguagem de programação interpretada, dinâmica e de alto nível, amplamente utilizada na web para criar interatividade e efeitos dinâmicos em páginas da web. Foi desenvolvida por Brendan Eich em 1995 enquanto trabalhava na Netscape Communications Corporation. Ela é baseada em C e Java, e possui uma sintaxe similar, mas diferente de ambas em termos de objetivos e propósito. Também é amplamente suportada por todos os principais navegadores da web e pode ser utilizada tanto no lado do cliente quanto no lado do servidor. Com a popularização de bibliotecas e *frameworks* como jQuery, AngularJS, React e Vue, a JavaScript tornou-se ainda mais popular, o que é uma característica valorizada pelos desenvolvedores de software. Uma das principais vantagens do JavaScript é a capacidade de criar aplicações tanto *Multi Page Application* (MPA) quanto *Single Page Application* (SPA). Esses dois conceitos serão discutidos a seguir.

Segundo Kaluža, Troskot e Vukelić (2018), SPAs são um tipo de aplicativo da web que carrega todo o seu conteúdo em uma única página HTML. Isso significa que os usuários não precisam esperar que a página seja recarregada ao interagir com ela, o que pode fornecer uma experiência de usuário mais integrada e responsiva. As SPAs estão se tornando cada vez mais populares, pois podem ser usadas para criar aplicativos da Web mais interativos e envolventes do que as páginas da Web tradicionais. Por outro lado, MPAs são as páginas da Web tradicionais, cada conteúdo é carregado em uma página diferente, que é recarregada a cada interação do usuário.

Uma SPA destaca-se, em relação a uma MPA, pelo melhor desempenho, menor

[1] A linguagem de programação é considerada tendo características de função *first-class* quando suas funções são tratadas de forma equivalente a outras variáveis dentro do sistema. Em outras palavras, nesse tipo de linguagem, é possível utilizar funções como argumentos de outras funções, retorná-las como resultado de uma função e atribuí-las a variáveis, de forma semelhante ao tratamento dado a outras entidades de dados.

consumo de recursos, maior interatividade e melhor experiência do usuário. Porém, possui maior complexidade de desenvolvimento, dificuldade de *Search Engine Optimization* (SEO) e segurança. Por outro lado, uma MPA tem simplicidade de desenvolvimento, facilidade de SEO e segurança. Porém, tem pior desempenho, maior consumo de recursos, menor interatividade e pior experiência do usuário(SHARMA et al., 2019).

A escolha entre SPA e MPA depende dos requisitos e objetivos da aplicação web. Se a aplicação requer alto desempenho, interatividade e usabilidade, uma SPA pode ser mais adequada. Se a aplicação requer simplicidade, SEO e segurança, uma MPA pode ser melhor escolha.

2.3 Angular

Angular é uma plataforma de código aberto para o desenvolvimento de aplicativos web mantida pelo Google. Ele é baseado em TypeScript e foi lançado pela primeira vez em 2010. O Angular é conhecido por sua arquitetura de componentes, que permite aos desenvolvedores criar aplicativos web de maneira modular e escalável. Ele fornece uma série de ferramentas para criar aplicativos web, incluindo o gerenciamento de estado, validação de formulários, roteamento, serviços e muito mais. Angular é utilizado por empresas de todos os tamanhos, incluindo pequenas *startups* e grandes corporações, para criar aplicativos web de alto desempenho. Ela é uma das ferramentas JavaScript mais populares e amplamente utilizadas em todo o mundo (ANGULAR, 2023).

2.4 React

React é uma biblioteca JavaScript de código aberto desenvolvida pelo Facebook e mantida por uma comunidade de desenvolvedores. Ela foi lançada em 2013 e é utilizada para criar interfaces de usuário em aplicativos web e móveis. React se concentra na criação de componentes reutilizáveis, permitindo aos desenvolvedores construir aplicativos web de maneira eficiente e escalável. Ela utiliza o virtual DOM (VDOM) que é um conceito de programação onde uma representação ideal, ou “virtual”, da interface do usuário é mantida em memória e sincronizada com o DOM “real” por uma biblioteca como o ReactDOM. Esse processo é chamado de reconciliação. O que permite ao React renderizar e atualizar rapidamente a interface do usuário sem a necessidade de atualizar toda a página. Além disso, o React fornece recursos para gerenciamento de estado, roteamento, validação de formulários e outras funcionalidades importantes para a criação de aplicativos web. Devido a sua eficiência, escalabilidade e facilidade de uso, o React é amplamente utilizado por empresas de todos os tamanhos, incluindo *startups* e grandes corporações, para desenvolver aplicativos web de alta qualidade (REACT, 2023).

2.5 Vue

O Vue.js é um *framework* JavaScript de código aberto para construção de interfaces de usuário. Ele foi criado por Evan You em 2014 e é mantido por uma comunidade ativa de desenvolvedores. O Vue se destaca por sua abordagem de componentização, que permite aos desenvolvedores criar aplicativos web de maneira modular e escalável. Além disso, o Vue tem uma estrutura de dados reativa, o que permite aos desenvolvedores criar interfaces dinâmicas que atualizam automaticamente conforme os dados mudam. O Vue também inclui recursos para roteamento, validação de formulários, animações, entre outros, tornando-o uma escolha popular para a criação de aplicativos web. O Vue é conhecido por sua curva de aprendizado suave e documentação clara, tornando-o uma opção popular para desenvolvedores iniciantes e experientes. Ele é amplamente utilizado por empresas de todos os tamanhos para criar aplicativos web de alta qualidade (VUE, 2023).

2.6 Arquitetura de Microsserviços

A arquitetura de microsserviços é um estilo arquitetônico de software que se concentra na construção de aplicativos como um conjunto de serviços independentes e colaborativos. Cada serviço é construído em torno de uma única funcionalidade de negócios e é implantado e gerenciado independentemente dos outros serviços. Esses serviços geralmente se comunicam uns com os outros por meio de uma Application Programming Interface (API) bem definida e podem ser desenvolvidos em diferentes linguagens de programação e usando diferentes tecnologias (MICROSOFT, 2023).

Segundo Lewis e Fowler (2014), a arquitetura de microsserviços é baseada em princípios de *design*, como a separação de responsabilidades, a alta coesão e o acoplamento fraco. Esses princípios garantem que cada serviço seja altamente especializado em uma única tarefa, o que facilita a manutenção e evolução dos serviços. A arquitetura de microsserviços também facilita a escalabilidade horizontal dos serviços, permitindo que cada serviço seja escalado de forma independente, conforme necessário.

Nessa mesma linha, de acordo com Newman (2021), a arquitetura de microsserviços é uma abordagem para construir sistemas de software que enfatiza a modularidade, escalabilidade, manutenção e independência dos serviços. Cada microsserviço é responsável por uma única função e é construído em torno de um contexto de negócios específico. Esses serviços são altamente independentes e podem ser implantados e gerenciados separadamente, o que torna o desenvolvimento e a manutenção de aplicativos mais ágeis e eficientes.

Em resumo, o conceito de arquitetura de microsserviço foi utilizado no trabalho para fornecer um contexto para a análise comparativa dos *frameworks* Angular, React e Vue. A arquitetura de microsserviço é uma abordagem de design de software que divide um

sistema em unidades menores, chamadas microsserviços. Cada microsserviço é responsável por uma função específica do sistema. Isso facilita a manutenção, evolução e escalabilidade dos serviços.

2.7 Arquitetura REST

A arquitetura REpresentational State Transfer (REST), segundo Fielding (2000), é um estilo de arquitetura de software que se baseia em princípios de comunicação na Web. Ela é amplamente utilizada para construir APIs para aplicativos web e móveis. A arquitetura REST define uma série de regras e recomendações para a estruturação de APIs, incluindo a utilização de recursos HTTP como GET, POST, PUT e DELETE para representar ações sobre recursos da aplicação. Além disso, a arquitetura REST usa URLs para identificar recursos e utiliza respostas HTTP padronizadas para indicar o resultado de uma solicitação. A arquitetura REST é escalável e permite a criação de aplicativos distribuídos, já que diferentes aplicativos podem acessar e compartilhar recursos por meio de uma API REST. Isso torna a arquitetura REST uma escolha popular para a criação de aplicativos que precisam ser integrados com outros sistemas ou compartilhar dados entre diferentes plataformas. Portanto, a arquitetura REST foi utilizada no trabalho para definir como os microsserviços se comunicam entre si. A arquitetura REST é um padrão de arquitetura de software que define um conjunto de restrições para a comunicação entre sistemas. Essas restrições garantem que os microsserviços sejam interoperáveis e fáceis de integrar.

3 Trabalhos Relacionados

Este capitulo apresenta os estudos relacionados à análise comparativa entre *frameworks* SPA.

Kaluža, Troskot e Vukelić (2018) realizaram uma análise comparativa entre *frameworks front-end* para desenvolvimento web. O objetivo do artigo foi demonstrar a adequação dos *frameworks* disponíveis para a criação de aplicações web SPA *vs* MPA, definir as principais funcionalidades que permitem o desenvolvimento mais customizado de ambos os tipos de aplicação, além de identificar se existe um *framework front-end* otimizado para criar aplicativos MPA e SPA. Os autores escolheram os três *frameworks front-end* mais populares com objetivo de analisar: se apenas *js import* é suficiente para começar; se a estrutura inclui muita sobrecarga; se o fluxo de trabalho é necessário; se há renderização lateral; se há necessidade de escrever e manter uma grande quantidade de código complexo; se o desenvolvedor deve confiar em pacotes de terceiros; qual é a possibilidade de compactar o aplicativo, minimizando o arquivo *bundle*[1]; além de se é possível o gerenciamento do estado dos dados. Para atender ao objetivo, eles escolheram no Github os *frameworks* mais populares que estavam ainda em crescimento naquele momento, descartando os demais que estavam estagnados. Segundo esses autores, o React tinha 86.000 estrelas e Vue.js 81.000. Por outro lado, o Angular estava sendo impulsionado pelos gigantes Google e Microsoft. Oitos perguntas foram formuladas e respondidas baseadas no objetivo. As respostas foram utilizadas como métricas, expressos em texto, os valores possíveis das respostas foram: Não, Parcialmente e Sim. Além disso, foi atribuída a cada resposta um peso em uma escala de 0 a 2. Esses valores servem para reforçar o quão forte se dá cada uma das alternativas textuais utilizadas. Eles chegaram a conclusão que Angular é o melhor *framework* para SPA, mas por outro lado, se mostrou o mais inadequado para desenvolvimento MPA. O Vue é o melhor para MPA. Já o React ficou em uma situação intermediária em ambos os casos.

No trabalho de Ziani (2021) foi realizado uma análise comparativa e qualitativa de *frameworks* JavaScript. O trabalho buscou responder a três perguntas:

- Existe um método eficaz para comparar e selecionar um *frameworks* JavaScript como parte do desenvolvimento de um aplicativo web?
- Quais são os critérios usados para comparar *framework* JavaScript?

[1] O *bundle* é a compilação de produção otimizada para a Web criada a partir das ferramentas de compilação de cada estrutura. Seus conjuntos de ferramentas resultam em um arquivo HTML e um arquivo JavaScript minificado compactado na Web, pronto para implantação de produção. Para o JavaScript nativo, utiliza-se um arquivo HTML estático que encapsula o *script* equivalente, contendo as funções escritas no *framework*.

– Se todos os *frameworks* JavaScript são levados em consideração nas análises?

De acordo com Ziani, as respostas para as duas primeiras perguntas foram obtidas por meio da busca na literatura. Ou seja, para primeira pergunta parece que não existe um método único e padronizado de comparação de *frameworks* reconhecido pela comunidade científica. Em relação à segunda pergunta, Ziani identificou quatro grupos de critérios:

- critérios mensuráveis;
- características desejadas/esperadas;
- o resultado da execução de software externo;
- critérios que são estáticos ou mensuráveis, mas não confiáveis porque mudam.

E por último, Ziani não encontrou uma solução atemporal, pois novos *frameworks* são criados a cada ano, os *frameworks* existentes são aprimorados e às vezes refatorados sem oferecer compatibilidade com versões anteriores (ex: Angular V1 *vs* Angular V2) e finalmente pode acontecer que os *frameworks* não tenham mais suporte. Portanto, Ziani propôs usar a método *Analytic Hierarchy Process* (AHP) usando árvore de decisão para comparar diferentes *frameworks* JavaScript. Esse método considera como critério de escolha o esforço de aprendizagem; o esforço de desenvolvimento; a estabilidade, a atratividade e o desempenho como critérios. Como conclusão, Ziani apontou o Svelte como o melhor *framework* dentre aqueles que foram testados: Angular, Preact, React, Svelte e Vue.

Diniz-Junior et al. (2022) avaliaram o desempenho de tecnologias de renderização da Web baseadas em JavaScript: Angular, React e Vue. O trabalho teve como objetivo avaliar o desempenho de tecnologias de renderização web baseadas em JavaScript, considerando o tempo de interação, o tempo de manipulação do DOM e o tamanho do *bundle*. Os autores buscaram as principais métricas de avaliação na literatura e analisaram DOM virtual e incremental criando uma aplicação padronizada. Testaram manipulações DOM usando três estruturas que representam essas duas técnicas de renderização: Angular para incremental e React e Vue para virtual. Como resultados, os autores buscaram contribuir com:

– uma metodologia para comparar *framework* de renderização Web incremental e virtual baseada em JavaScript;

– um estudo de caso comparativo com os *frameworks* Angular, Vue e React;

– um aplicativo da Web genérico capaz de criar, editar e excluir determinada quantidade de elementos na janela do navegador, seguindo o ciclo de vida (função do *framework* que controla os estados de uma aplicação) proposto por cada *framework* e biblioteca.

Diniz-Junior et al. (2022) concluíram que o React teve dificuldade em criar elementos de DOM. O Angular e o Vue levaram 270-290 ms em média respectivamente, enquanto React levou mais de 5000 ms. Os dois primeiros são baseados em DOM virtual e produziram resultados diferentes. Angular, por sua vez, com sua implementação DOM incremental, produziu resultados semelhantes aos vistos na criação de elementos para estruturas Vue, com uma diferença média de cerca de 20 ms considerando 50.000 elementos e o melhor desempenho na edição de cenários DOM. Como resultado, a principal vantagem sobre o VDOM ou VNodes é que eles podem ser atualizados sem a necessidade de um intermediário para combinar os elementos na memória. Por fim, os autores relatam que a solução proposta pelo DOM virtual implementado dentro do *framework* Vue teve uma pequena vantagem geral, tendo um bom desempenho em todas as funções e mantendo picos abaixo da marca de 1s no maior tamanho de amostra.

O trabalho de Ziani (2021) é diferente deste trabalho pelo método usado para atribuir peso aos critérios, o que é subjetivo. Ou seja, os pesos atribuídos a cada variáveis proposta por Ziani (2021) segue uma lógica definida pelo autor que poderia ser outra, se fosse proposto por outra pessoa. O trabalho de Kaluža, Troskot e Vukelić (2018) difere deste trabalho por levar em consideração o conceito de MPA. Já o trabalho de Diniz-Junior et al. (2022) focalizou muito na questão de renderização. No entanto, todos esses trabalhos se assemelham por levar em consideração os três *frameworks* Angular, React e Vue. Portanto, este trabalho pode contribuir para o conhecimento científico sobre *frameworks* JavaScript para desenvolvimento de SPAs baseadas em microsserviços. Por exemplo, a análise pode identificar padrões e tendências que podem ser úteis para futuros estudos. Além disso, a arquitetura de microsserviço é uma abordagem de design de software cada vez mais popular, e os *frameworks* Angular, React e Vue são os mais utilizados para desenvolvimento de aplicativo de página única. Uma análise comparativa desses *frameworks* no contexto da arquitetura de microsserviço é relevante para uma ampla gama de profissionais. A seguir, no capítulo 4, é apresentada a metodologia que foi adotada neste trabalho visando comparar os *frameworks* SPA.

4 Metodologia

Este capítulo apresenta a metodologia proposta para alcançar os objetivos traçados neste trabalho. A metodologia para comparar os *frameworks* SPA desenvolvendo um aplicativo com arquitetura de microsserviços para testar o desempenho consistiu em uma série de etapas descritas a seguir:

4.1 Revisão bibliográfica

Foi realizada uma revisão de literatura para obter informações sobre os *frameworks*, suas principais características, vantagens e desvantagens, e como compará-los em relação aos critérios como tamanho do pacote, tempo de carregamento, tempo de renderização, uso de memória e tempo de execução. Três estudos recentes analisaram *frameworks* SPA para desenvolvimento web. Kaluža, Troskot e Vukelić (2018) compararam *frameworks* para SPA e MPA, identificando as principais funcionalidades e adequação para cada tipo de aplicação. Ziani (2021) buscou um método eficaz para comparar e selecionar *frameworks* JavaScript, identificando critérios e *frameworks* mais usados. Diniz-Junior et al. (2022) avaliaram o desempenho de *frameworks* JavaScript, considerando tempo de interação, manipulação do DOM e tamanho do *bundle*.Os estudos indicam que não existe um framework *front-end* ideal para todos os casos. A escolha do framework deve ser feita considerando o tipo de aplicação, as necessidades do usuário e os critérios de avaliação relevantes. A revisão bibliográfica também forneceu uma compreensão geral do conceito de arquitetura de microsserviços e como ela é implementada nos aplicativos.

4.2 Definição da arquitetura e desenvolvimento do aplicativo

Com base na revisão bibliográfica, foram definidos os critérios, as métricas (tamanho do pacote (*bundle*), tempo de carregamento, tempo de renderização, uso de memória e tempo de execução) de avaliação e os *frameworks* a serem avaliados comparativamente. Desse modo, foram definidos os requisitos a serem implementados pelo aplicativo a ser construído utilizando as três alternativas propostas. A arquitetura de microsserviços foi definida para garantir que o aplicativo seja escalável e possa lidar com excesso de tráfego. Isso é necessário para a definição do escopo do trabalho. O objetivo foi desenvolver o mesmo aplicativo com as três *frameworks* e compará-los usando as métricas definidas;

4.2.1 Infraestrutura

Para atender aos requisitos metodológicos propostos, foi imprescindível empregar um dispositivo computacional com capacidade de armazenamento e processamento adequados para o desenvolvimento e, se necessário, a simulação do aplicativo. Foram empregadas as seguintes tecnologias para realizar esses testes:

- notebook Vaio FE15 com 12GB de memória, processador Intel® Core™ i5-8250U 3,40 GHz, executando o sistema operacional Windows 10;
- o Microsoft Edge 119.0.2151.58 para execução do *front-end*;
- Docker[1] para a conteinerização [2] dos serviços;
- o IDE VS Code para a implementação dos códigos;
- o SGBD MYSQL 5.7 para amazenamento dos dados;
- o PM2[3] 5.3.0 para gerenciamento dos processos;
- Node.js 16.16.0;
- Angular 16.2.0, última versão de suporte de longo prazo no momento de realizar o experimento;
- React 18.2.0, última versão de suporte de longo prazo no momento de realizar o experimento;
- Vue 3.3.4, última versão de suporte de longo prazo no momento de realizar o experimento.

4.2.2 Descrição do aplicativo

O aplicativo utilizado para teste pode ser considerado uma parte de um sistema de *e-commerce*. O aplicativo elaborado se concentra na gestão dos produtos disponíveis para venda na loja on-line, incluindo a sua criação, leitura, atualização e remoção (CRUD), bem como o armazenamento das imagens associadas a esses produtos.

Dessa forma, um *e-commerce* pode se beneficiar do sistema descrito para gerenciar os produtos de forma eficiente, tornando-os facilmente acessíveis aos clientes por meio de interface do usuário desenvolvida com Vue, React e Angular. O sistema também pode

[1] Docker é um conjunto de produtos de plataforma como serviço que usam virtualização em nível de sistema operacional para entregar software em pacotes chamados contêineres. Os contêineres são isolados uns dos outros e agrupam seus próprios softwares, bibliotecas e arquivos de configuração.

[2] Conteinerização é o empacotamento do código de software com todos os componentes necessários, como bibliotecas, frameworks e outras dependências, para que fiquem isolados em seu próprio contêiner.

[3] PM2 é um gerenciador de processos para o *runtime* JavaScript Node.js

ajudar a garantir a integridade e consistência dos dados dos produtos, evitando problema como informações incorretas ou imagens perdidas.

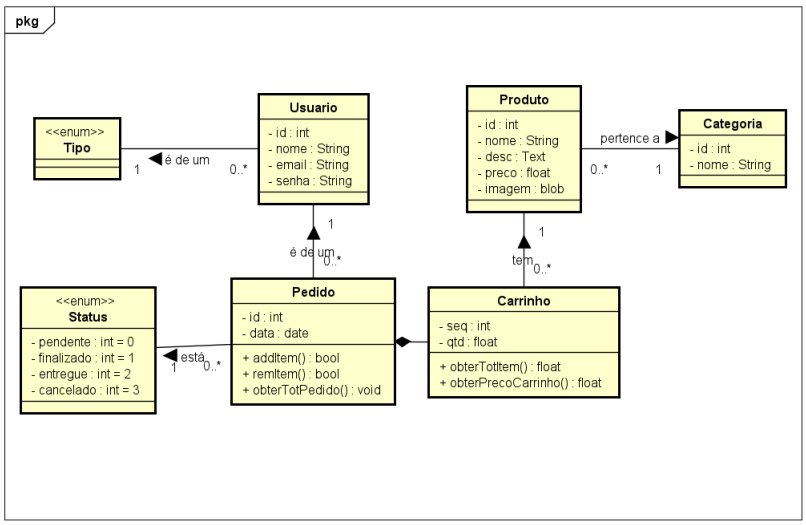
Portanto, o aplicativo é composto de duas camadas:

- *back-end* : um serviço API que se comunica com o SGBD MySQL;
- *front-end*: a interface do aplicativo implementado com os três *frameworks* (Angular, React, Vue.js).

A Figura 1 mostra um diagrama de classe que representa as classes do aplicativo. Esse diagrama de classe correspondem aos requisitos do sistema que são:

- permitir que os usuários autorizados criem, leiam, atualizem e excluam produtos;
- possibilitar associar imagens a cada produto;
- permitir aos usuários pesquisar produtos por nome e categoria;
- permitir aos usuários adicionem produtos ao carrinho e finalizar a compra.

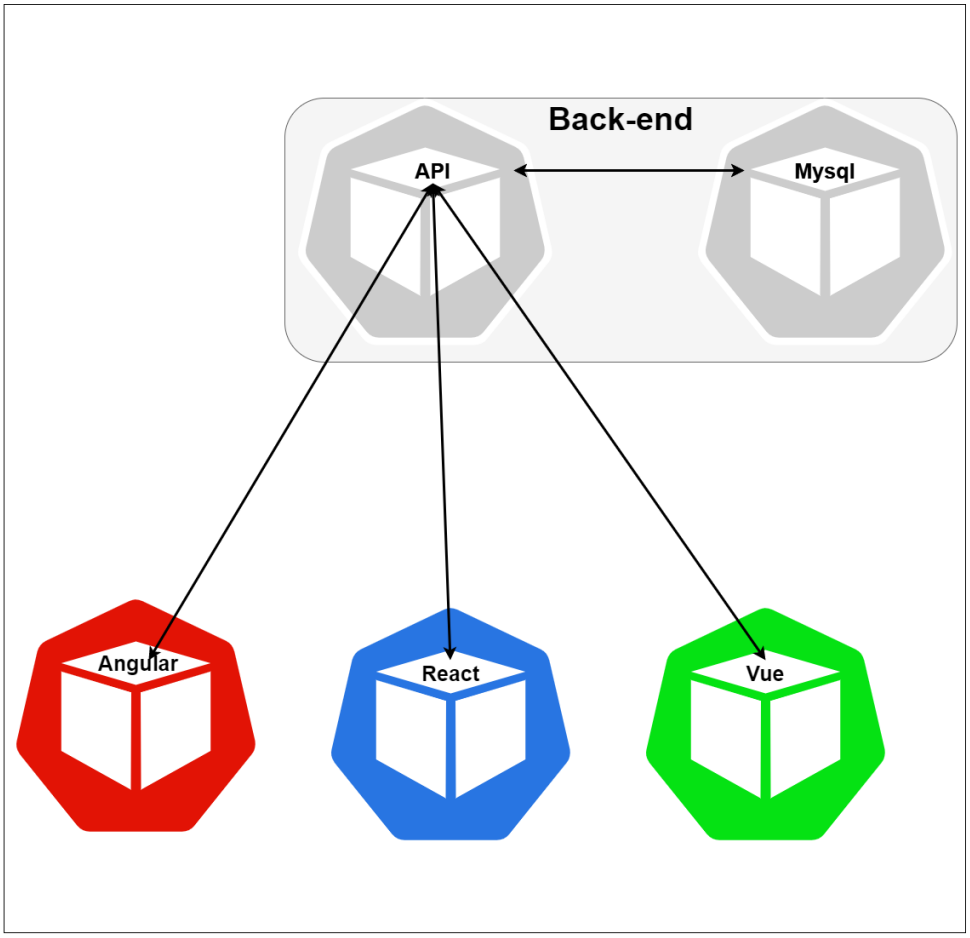
Figura 1 – Diagrama de classe



Fonte: do Autor(2023).

Para simular um ambiente de arquitetura de microsserviço real, foi utilizado o conceito de conteinerização para a implementação de cada serviço. A Figura 2 mostra um diagrama que representa a arquitetura do aplicativo. Cada cubo representa um contêiner em que foi implementado um serviço que é um componente do aplicativo. Neste caso, o *front-end* foi implementado usando as tecnologias SPA Angular, React e Vue para poder realizar a análise comparativa.

Figura 2 – Arquitetura do aplicativo



Fonte: do Autor(2023).

4.3 Testes de desempenho

Depois que o aplicativo foi desenvolvido, realizou-se o teste de desempenho para compará-los. O teste de desempenho visa medir o tamanho do pacote, o tempo de carregamento, o tempo de renderização, o uso de memória e o tempo de execução em diferentes cenários de carga. Porém, para ser possível o teste, sem considerar a complexidade de interação em um *e-commerce*, foram limitadas as operações na interface à:

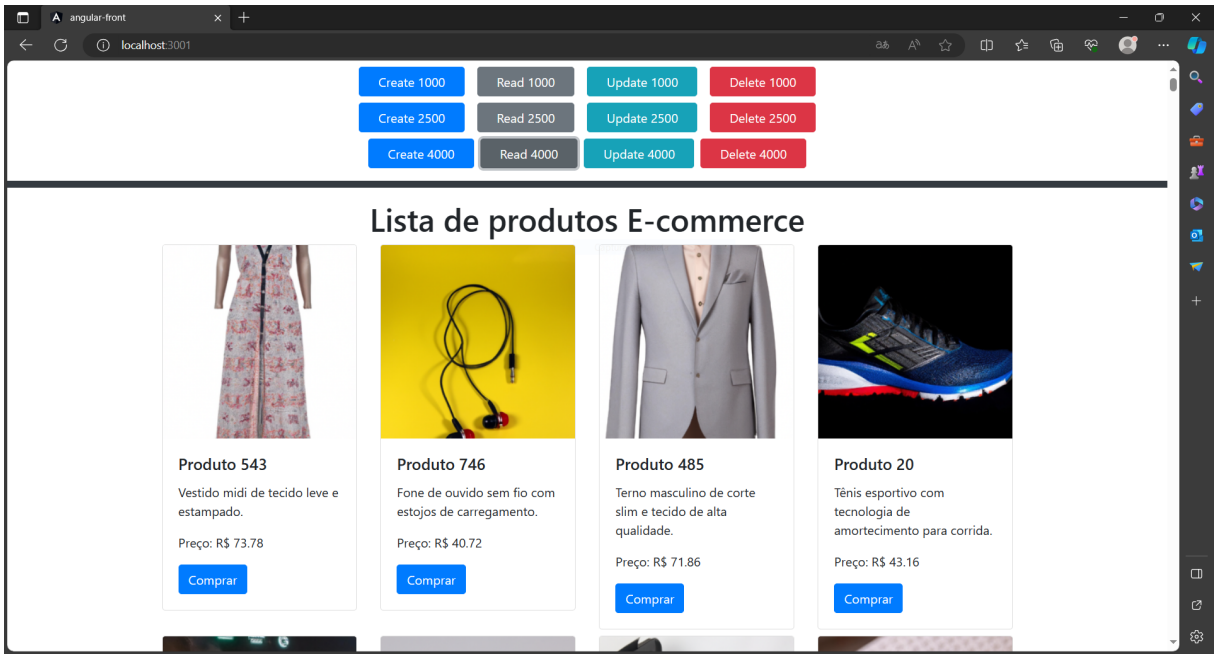
- criação de 1000, 2500 e 4000 produtos;
- leitura de 1000, 2500 e 4000 produtos;
- edição de 1000, 2500 e 4000 produtos;
- remoção de 1000, 2500 e 4000 produtos.

Inicialmente, a carga foi definida usando como exemplo uma carga de 1000, 5000 e 10000 produtos. Isso representa um cenário de carga pequena, um de carga media e

um de carga alta. Porém, a maquina usada não consegue fazer o processamento porque o processamento de imagem requer muitos recursos que a maquina não tinha a disposição. Portanto, foi reduzida a carga até chegar nesses valores 1000, 2500 e 4000, representando respectivamente o cenário de carga pequena, media e alta que a maquina conseguiu processar. O teste do tempo de execução das operações de CRUD foi realizado a partir das interações dos usuários usando um botão que executa cada uma das operações com a carga indicada. O desempenho de cada *framework* foi avaliado por meio da captura do tempo de execução dentro do código, usando a API desempenho do JavaScript. A coleta das outras métricas foi obtida a partir da ferramenta de desenvolvedor Devtools do Microsoft Edge.

A Figura 3 mostra a interface do aplicativo em que foi realizada os diferentes testes. Essa interface é a mesma gerada usando cada um dos *frameworks*. Para que os produtos reflitam a realidade, foi utilizado o ChatGPT [4] para gerar cento e cinquenta descrições de produto como mostrada na Figura 4 e em seguida foi usado a API da IA do gerador de imagem DALL.E2 [5] da OpenAI para gerar as imagens dos produtos baseado nas descrições fornecida pela ChatGPT. O *script* para gerar as imagens é apresentada na Figura 5. No total, foram geradas 1.500 imagens, dez para cada descrição de produto. Essas imagens foram usadas para realizar o teste.

Figura 3 – Interface de teste

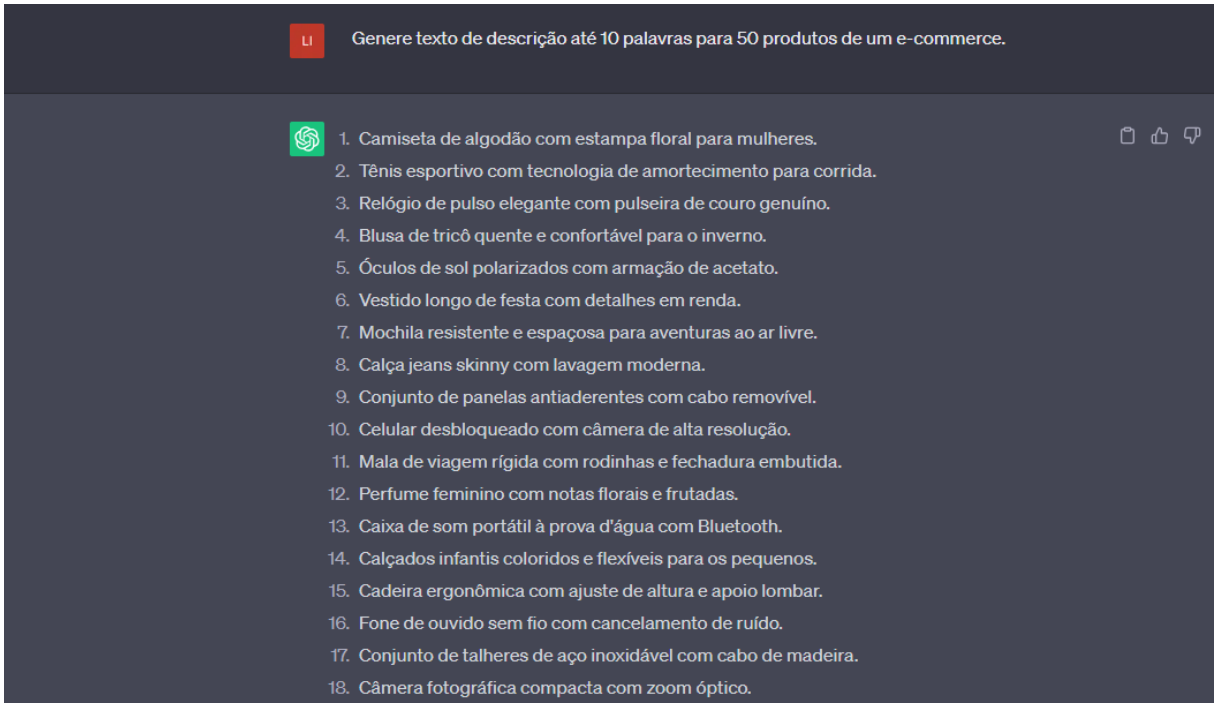


Fonte: do Autor(2023).

[4] ChatGPT é um *chatbot on-line* de inteligência artificial desenvolvido pela OpenAI, lançado em novembro de 2022. O nome "ChatGPT"combina "Chat", referindo-se à sua funcionalidade de *chatbot*, e "GPT", que significa *Generative Pre-trained Transformer*, um tipo de modelo de linguagem grande e uma estrutura proeminente para inteligência artificial generativa.

[5] O DALL.E2 é um sistema de IA que pode criar imagens e arte realistas a partir de uma descrição em linguagem natural.

Figura 4 – Prompt de geração de descrição de produto usando ChatGPT



Fonte: ChatGPT, do Autor(2023).

Figura 5 – Script que gerador de imagem de produto usando descrição fornecido pelo ChatGPT a partir de chamada via API da OpenAI

```
// Função para gerar 10 produtos
async function gerarProdutos() {
  const produtos = [];
let c1=0,c2=0;
  for (let i = 0; i < descriptions.length; i++) {
    if(i!=0 && i%5==0){
      console.log('Esperando 2 minutos');
      await new Promise((resolve) => {
        setTimeout(() => {
          resolve();
        }, 2*60000); // 60000 milissegundos = 1 minuto
      });
    }
    // console.log(`'description ${i+1}:>> '`, descriptions[i]);
    const response = await axios.post('https://api.openai.com/v1/images/generations',
    {
      'prompt':  descriptions[i],
      'n': 10,
      'size': '256x256'
    },
    {
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY
      }
    }
  );

```

Fonte: do Autor(2023).

4.4 Avaliação comparativa dos frameworks

Com base nos resultados dos testes de desempenho, foi realizada uma avaliação comparativa dos aplicativos implementados. A avaliação mostrou qual das alternativas é a melhor opção em termos de tamanho do pacote, tempo de carregamento, tempo de renderização, uso de memória e tempo de execução.

A seguir, no capítulo 5, é apresentada os resultados esperados neste trabalho expressos por meio de tabelas e gráficos. Em seguida, é realizada uma análise dos resultados obtidos.

5 Resultados

Esse capitulo apresenta os resultados do estudo por métricas avaliadas.

5.1 Tamanho do pacote (*bundle*)

A Tabela 1 mostra o *bundle* dos aplicativos dos três *frameworks*, medido em KB. O Vue gerou o menor pacote, o React gerou o segundo menor e o Angular gerou o maior. Ou seja, o *bundle* do Vue é de 404 KB, do React é de 488 KB e do Angular é de 500 KB. Portanto, ter o menor pacote gerado é uma vantagem, pois nem todo dispositivo tem capacidade de memória para guardar um aplicativo.

Tabela 1 – Dados obtidos nos experimentos

	Angular	React	Vue
Bundle (KB)	500	488	404
Carregamento (ms)	7	6	6
Renderização (ms)	4	4	6
Heap JS (MB)	2,4-6,2	2,8-3,8	2,5-3,0

Fonte: do Autor(2023).

A Tabela 1 mostra que o Vue é o *framework* com o *bundle* menor, o React com o segundo menor e o Angular com o maior *bundle*. O tamanho do *bundle* gerado pelo Angular é 2,46% maior que o do React e 23,76% maior que o do Vue. Isso significa que o aplicativo implementado por Vue requer menos largura de banda, o da React vem em segundo lugar e o da Angular é o que mais precisa de largura de banda.

Essas diferenças podem ser explicadas por uma série de fatores, incluindo as características dos *frameworks*, as bibliotecas que eles incluem e as opções de configuração que oferecem. O Angular é um *framework* completo, que inclui uma ampla gama de recursos, como injeção de dependência, roteamento e *template*. Isso pode contribuir para o seu tamanho maior. O React é um *framework* mais focado na renderização de componentes, enquanto o Vue é um *framework* mais modular, que permite aos desenvolvedores escolher quais bibliotecas e recursos incluir. Isso pode explicar o tamanho menor desses dois *frameworks*.

É importante ressaltar que o *bundle* não é o único fator a ser considerado na escolha de um *framework*. Outros fatores importantes incluem a facilidade de uso, a flexibilidade e a comunidade de suporte. No entanto, o *bundle* pode ser um fator importante para

aplicações que precisam ser carregadas rapidamente ou que têm restrições de largura de banda.

5.2 Tempo de carregamento

A Tabela 1 mostra o tempo de carregamento dos aplicativos dos três *frameworks*, medido em milissegundos. O aplicativo do React e o do Vue têm o menor tempo de carregamento, seguido pelo Angular. Ou seja, o do React e o do Vue têm um tempo de carregamento de 6 milissegundos, enquanto o do Angular tem um tempo de carregamento de 7 milissegundos. Essa diferença de 1 milissegundo representa um aumento de 16,67% em relação ao tempo de carregamento do React e do Vue.

Essa diferença é pequena, mas pode ser significativa em aplicações em que o tempo de carregamento é crítico. Por exemplo, em aplicativos móveis, onde a largura de banda é limitada, um aumento de 1 milissegundo pode representar uma diferença perceptível na experiência do usuário.

Existem várias razões pelas quais o Angular tem um tempo de carregamento ligeiramente maior do que o React e o Vue. Uma das razões é que o Angular é um *framework* mais complexo, com mais código e bibliotecas. Outra razão é que o Angular usa um compilador para converter o código TypeScript em JavaScript, o que adiciona um passo adicional ao processo de carregamento.

No geral, a diferença de tempo de carregamento entre o React, o Vue e o Angular é pequena. No entanto, essa diferença pode ser significativa em aplicações nas quais o tempo de carregamento é crítico.

5.3 Tempo de renderização

A Tabela 1 mostra o tempo de renderização dos aplicativos dos três *frameworks*, medido em milissegundos. O do Angular e o do React têm o menor tempo de renderização, seguido pelo Vue com o maior tempo. As diferenças de tempo de renderização entre os três *frameworks* são pequenas, mas significativas. O do Angular e o do React têm o mesmo tempo de renderização, de 4 milissegundos. O do Vue, por sua vez, tem um tempo de renderização 50% maior, de 6 milissegundos.

Essa diferença de 2 milissegundos pode parecer pequena, mas pode ser significativa em aplicações que requerem um alto desempenho de renderização. Por exemplo, em aplicações que exigem que o conteúdo da tela seja atualizado rapidamente, como jogos ou aplicativos de *streaming* de vídeo, uma diferença de 2 milissegundos pode ser perceptível pelo usuário.

Além disso, a diferença de 50% entre o do Vue e os outros dois *frameworks* pode ser significativa em aplicações que exigem renderizações frequentes. Por exemplo, em aplicações que requerem que o conteúdo da tela seja atualizado com base em dados externos, como o tempo ou o clima, uma diferença de 50% pode resultar em um consumo de recursos maior.

Em resumo, as diferenças de tempo de renderização entre os três *frameworks* são pequenas, mas significativas. O do Vue tem um tempo de renderização 50% maior que o do Angular e o do React. Essa diferença pode ser significativa em aplicações que requerem um alto desempenho de renderização ou renderizações frequentes.

Aqui estão alguns exemplos de como as diferenças de tempo de renderização podem afetar a experiência do usuário:

Em um jogo, por exemplo, uma diferença de 2 milissegundos pode resultar em um atraso perceptível na resposta do jogo aos comandos do usuário. Em um aplicativo de *streaming* de vídeo, uma diferença de 2 milissegundos pode resultar em um atraso perceptível na atualização da imagem na tela. Em um aplicativo que exibe a hora atual, uma diferença de 50% pode resultar em uma atualização da hora que é 25% mais lenta. Portanto, ao escolher um *framework* para sua aplicação, é importante considerar o tempo de renderização, especialmente se sua aplicação exigir um alto desempenho ou renderizações frequentes.

5.4 Uso de memória

A Tabela 1 mostra o uso de memória dos três aplicativos implementados usando os*frameworks*, medido em megabytes. A Figura 6 mostra que o aplicativo implementado usando Vue é o que apresenta o menor uso de memória, com um consumo que varia de 2,5 a 3,0 megabytes. Essa diferença pode ser atribuída a um modelo de componentes semelhante ao do React, mas com algumas otimizações que reduzem o consumo de memória.

O aplicativo do React é o que apresenta o segundo maior uso de memória, com um consumo que varia de 2,8 a 3,8 megabytes. Essa diferença pode ser atribuída a um modelo de componentes mais simples, que requer menos memória para armazenar informações sobre os componentes e seus estados.

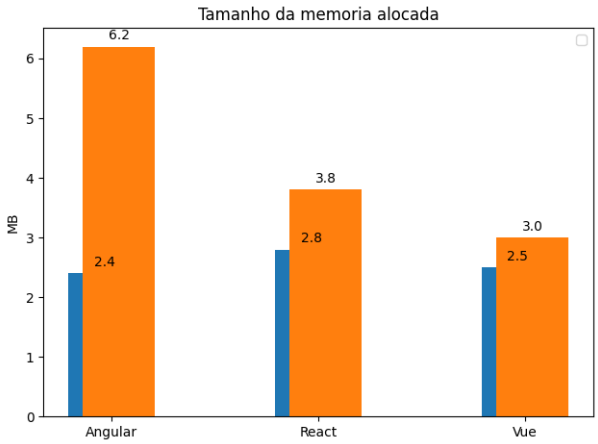
O aplicativo do Angular é o que apresenta o maior uso de memória, com um consumo que varia de 2,4 a 6,2 megabytes. Essa diferença pode ser atribuída a uma série de fatores, incluindo:

- o uso de um modelo de componentes mais complexo, que requer mais memória para armazenar informações sobre os componentes e seus estados;
- o uso de um sistema de injeção de dependências mais completo, que também requer

mais memória;

- o uso de uma biblioteca de componentes e serviços mais ampla, que também pode aumentar o consumo de memória.

Figura 6 – Figura do espaço (MB) alocado na memória inicialmente por cada *framework* em que azul representa o menor espaço alocado e laranja, o maior espaço alocado.



Fonte: do Autor(2023).

Em termos percentuais, o do Angular usa cerca de 63,15% a mais de memória do que o do React e de 106,67% a mais de memória do que o do Vue.

Essa diferença de uso de memória deve ser considerada ao escolher um *framework* para o desenvolvimento de uma aplicação web. A escolha do *framework* ideal dependerá de uma série de fatores, incluindo o tamanho e a complexidade da aplicação, os recursos necessários e o desempenho esperado.

Em aplicações pequenas e simples, a diferença de uso de memória pode não ser significativa. No entanto, em aplicações grandes e complexas, a diferença pode se tornar importante, especialmente se a aplicação for executada em dispositivos com recurso limitado.

5.5 Tempo de execução

A Tabela 2 mostra o tempo médio de execução de cada tarefa para cada tecnologia. Pode-se observar que, em geral, o Angular apresentou o pior desempenho, seguido pelo React e pelo Vue.

Tabela 2 – Tempo médio de execução x Processo

Execução		Tempo Médio (ms)		
		Angular	React	Vue
Create time	1000	17836	17603	16705
	2500	46553	44097	43906
	4000	77313	73034	73317
Read time	1000	43	41	44
	2500	67	57	68
	4000	102	94	97
Update time	1000	38234	36347	34932
	2500	101166	90668	89543
	4000	169862	153900	169808
Delete time	1000	16598	9283	15682
	2500	86612	23845	67686
	4000	178000	38799	151553

Fonte: do Autor(2023).

A Tabela 3 mostra o desvio padrão de execução de cada tarefa para cada tecnologia. O desvio padrão é uma medida da variabilidade dos resultados. Valores de desvio padrão baixos indicam que os resultados são mais consistentes, enquanto valores de desvio padrão altos indicam que os resultados são mais dispersos.

Tabela 3 – Desvio Padrão de execução x Processo

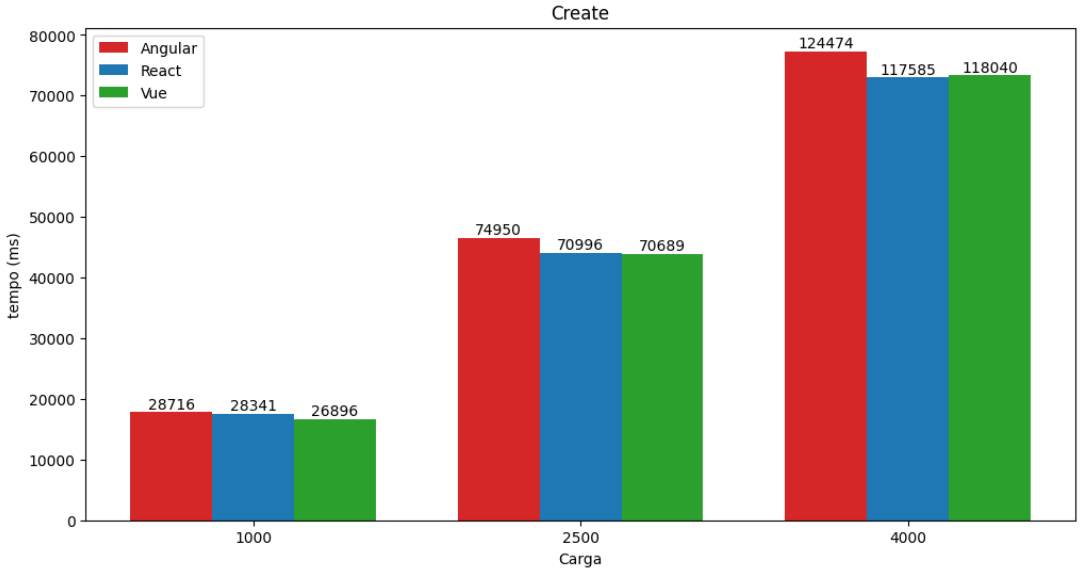
Execução		Desvio Padrão (ms)		
		Angular	React	Vue
Create time	1000	43	1504	21
	2500	474	383	601
	4000	1098	1596	456
Read time	1000	4	4	1
	2500	14	3	3
	4000	7	9	12
Update time	1000	661	1734	179
	2500	1711	898	1149
	4000	18031	5760	19270
Delete time	1000	242	131	265
	2500	702	615	2048
	4000	2141	710	14769

Fonte: do Autor(2023).

A Figura 7 mostra que o tempo médio gasto na operação de criação de produto para as diferentes cargas. O aplicativo do React e o do Vue têm o melhor desempenho, com a tendência que mostra que o React pode ser melhor que o do Vue se a carga continuar

aumentando. É possível verificar que o aplicativo do Angular tem o pior desempenho nesse quesito.

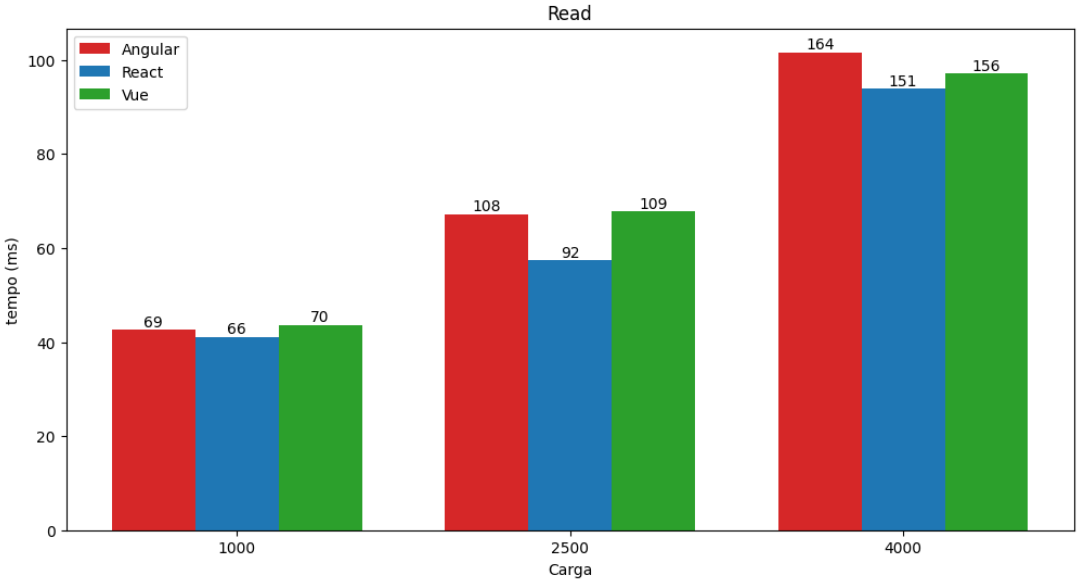
Figura 7 – Tempo de execução (ms) da operação de criação de produto



Fonte: do Autor(2023).

A Figura 8 mostra que o tempo médio gasto na operação de leitura de produto para as diferentes cargas. É possível verificar que o aplicativo do React tem o melhor desempenho nesse quesito. Pode-se afirmar que o do Vue tem o segundo melhor desempenho, apesar de que o do Angular tem um melhor desempenho considerando a carga de 2500 produtos, mas a diferença de 1 ms pode ser insignificante ao considerar uma margem de erro na medição. Portanto, o do Angular teve o pior desempenho nesse quesito.

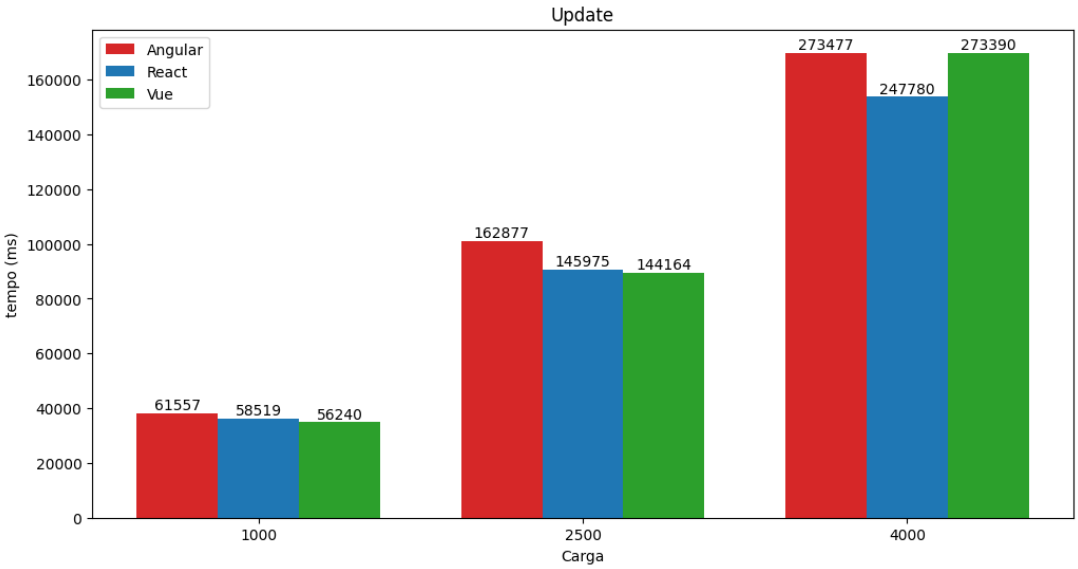
Figura 8 – Tempo de execução (ms) da operação de leitura de produto



Fonte: do Autor(2023).

A Figura 9 mostra que o tempo médio gasto na operação de atualização de produto para as diferentes cargas. É possível verificar que o aplicativo do React tem o melhor desempenho para a carga de 4000 produtos. Porém, o do Vue tem o melhor desempenho respectivamente nas cargas 1000 e 2500 produtos. Se considerar a pequena diferença entre o do Vue e o do React, pode-se afirmar que o do React tem o melhor desempenho e o do Vue, o segundo melhor desempenho. Portanto, o aplicativo do Angular tem o pior desempenho.

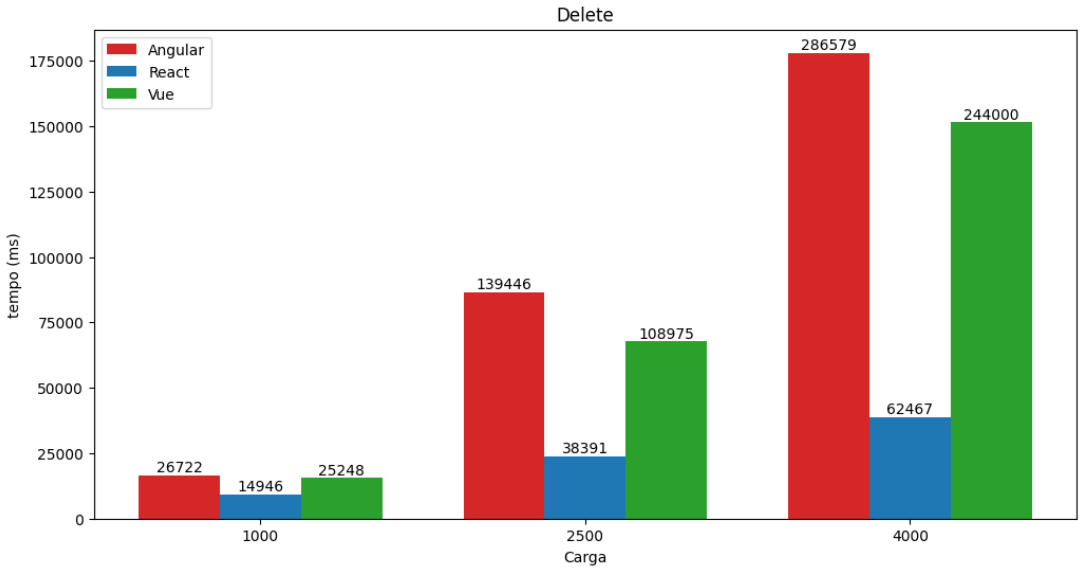
Figura 9 – Tempo de execução (ms) da operação de atualização de produto



Fonte: do Autor(2023).

A Figura 10 mostra que o tempo médio gasto para a operação de remoção de produto para as diferentes cargas. É possível verificar que o aplicativo do React tem o melhor desempenho nesse quesito. O do Vue tem um desempenho intermediário e o do Angular tem o pior desempenho. Portanto, para remoção de produto, o do React tem o melhor desempenho com uma margem considerável.

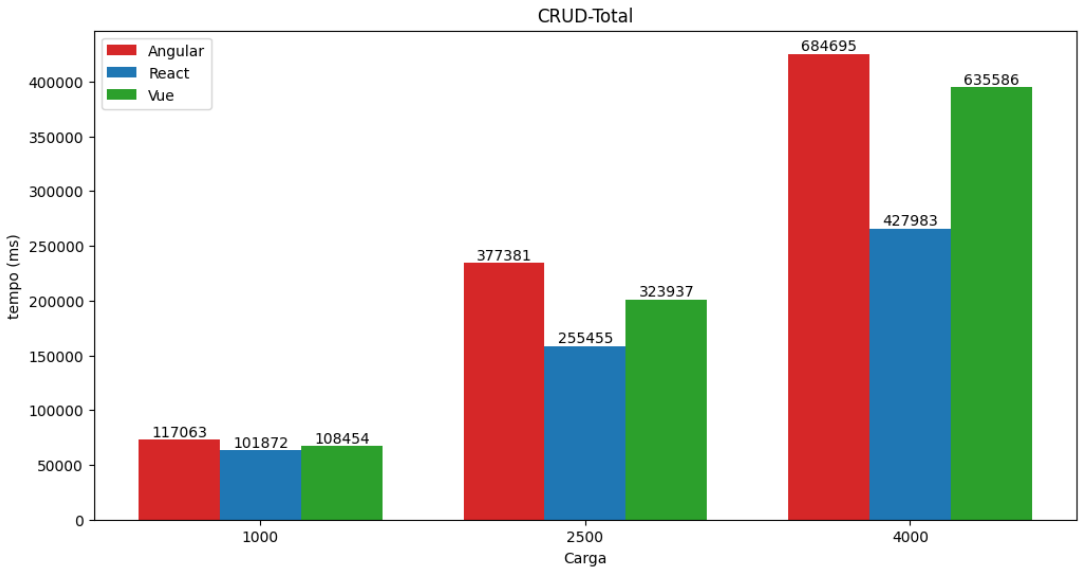
Figura 10 – Tempo de execução (ms) da operação de remoção de produto



Fonte: do Autor(2023).

A Figura 11 mostra que o tempo médio total gasto para a operação de CRUD de produto para as diferentes cargas. É possível verificar claramente que o aplicativo do React tem o melhor desempenho. O do Vue tem um desempenho intermediário e o do Angular tem o pior desempenho. Se considerar a operação CRUD como única, o do React se destaca com uma margem considerável, especialmente quando a carga é grande.

Figura 11 – Tempo total de execução (ms) da operação de CRUD de produto



Fonte: do Autor(2023).

Os achados desta pesquisa divergem dos obtidos por Kaluža, Troskot e Vukelić (2018), os quais concluíram que não existe um framework JavaScript ideal tanto para SPA quanto para MPA. Embora Ziani (2021) tenha conduzido uma análise comparativa de frameworks JavaScript, vale ressaltar que seu estudo foi qualitativo, ao passo que o presente trabalho adota uma abordagem quantitativa. Assim, os resultados aqui apresentados corroboram a afirmativa de Diniz-Junior et al. (2022), indicando que o Angular demonstra um desempenho superior em termos de renderização.

Vale ressaltar que os resultados obtidos nesta análise são baseados em um conjunto específico de dados e condições. É possível que os resultados sejam diferentes para outros conjuntos de dados ou condições. Portanto, este estudo apresenta algumas limitações que devem ser consideradas na interpretação dos resultados.

Em primeiro lugar, o estudo foi realizado em um ambiente controlado, com um conjunto específico de dados. Considerando que este trabalho adotou como abordagem a arquitetura de microsserviço, é possível que os resultados sejam diferentes com arquitetura diferente, ou em ambientes diferentes como Linux ou MacOS, ou com conjuntos de dados diferentes.

Em segundo lugar, o estudo se concentrou em um conjunto limitado de medidas de desempenho. É possível que outros fatores qualitativos, como a facilidade de uso ou a capacidade de manutenção, sejam mais importantes para alguns aplicativos do que os fatores considerados neste estudo.

Em terceiro lugar, o estudo não considerou as facilidades e recursos disponibilizados pelos *frameworks* que podem ajudar no desenvolvimento.

6 Conclusão

A partir dos resultados obtidos nesta análise comparativa entre os *frameworks* Angular, React e Vue, pode concluir que o aplicativo implementado usando React apresenta o melhor desempenho geral, seguido pelo do Vue e pelo do Angular em última posição.

Em termos de tamanho do pacote gerado para pôr em produção, o Vue gerou o menor *bundle*, o React gerou o segundo menor e o Angular gerou o maior entre os três. Isso significa que o aplicativo do Angular requer mais largura de banda para ser carregado, o que pode afetar o desempenho da aplicação em dispositivos com menor capacidade de processamento e também com acesso à internet mais lenta.

Em termos de tempo de carregamento, o aplicativo do React e o do Vue são os mais rápidos, seguido pelo Angular. Isso significa que as aplicações desenvolvidas com esses *frameworks* serão carregadas mais rapidamente, o que pode melhorar a experiência do usuário.

Em termos de tempo de renderização, o aplicativo Angular e o do React têm o mesmo desempenho. Eles renderizam mais rápidos do que o aplicativo do Vue. Isso significa que as aplicações desenvolvidas com esses *frameworks* terão um desempenho semelhante em termos de renderização de conteúdo.

Em termos de uso de memória, o aplicativo do Vue é o que menos usa memoria, seguido pelo do React e pelo do Angular em última posição. Isso significa que as aplicações desenvolvidas com o Angular podem consumir mais memória nas demais tecnologias testadas, o que pode afetar o desempenho da aplicação em dispositivos com menor capacidade de processamento.

Em termos de tempo de execução, o aplicativo do React tem o melhor desempenho em todas as operações de CRUD, seguido pelo do Vue e pelo Angular, em último lugar. Isso significa que as aplicações desenvolvidas com o React terão um desempenho melhor em operações de CRUD.

Com base nos resultados obtidos, as seguintes recomendações podem ser feitas:

- para aplicações que precisam ser carregadas rapidamente, o React ou o Vue são as melhores opções;
- para aplicações que precisam usar menos memória, o Vue é a melhor opção;
- para aplicações que precisam ter um bom desempenho em operações de CRUD, o React é a melhor opção.

Alguns trabalhos futuros que poderiam ser realizados para ampliar os resultados deste estudo incluem:

- realizar o estudo em um ambiente em nuvem, com diferentes tipos de aplicativos e dados;
- considerar outros fatores para a tomada de decisão, como a facilidade de uso ou a capacidade de manutenção;
- comparar os *frameworks* SPA em hardware *mobile* como Android ou IOS.

Referências

ANGULAR. Google, 2023. Disponível em: <https://angular.io/guide/what-is-angular>.

BERNERS-LEE, T. et al. The world-wide web. *Communications of the ACM*, ACM New York, NY, USA, v. 37, n. 8, p. 76–82, 1994.

DINIZ-JUNIOR, R. N. et al. Evaluating the performance of web rendering technologies based on javascript: Angular, react, and vue. p. 1–9, 2022.

FERREIRA, H. K.; ZUCHI, J. D. AnÁlise comparativa entre frameworks frontend baseados em javascript para aplicaÇÕes web. *Revista Interface Tecnológica*, v. 15, n. 2, p. 111–123, dez. 2018. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/502>.

FIELDING, R. T. *Representational State Transfer (REST)*. University of California, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>.

JAVASCRIPT. Mozilla Corporation's, 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>.

KALUŽA, M.; TROSKOT, K.; VUKELIĆ, B. Comparison of front-end frameworks for web applications development. *Zbornik Veleučilišta u Rijeci*, Veleučilište u Rijeci, v. 6, n. 1, p. 261–282, 2018.

KORNIENKO, D.; MISHINA, S.; MELNIKOV, M. The single page application architecture when developing secure web services. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2021. v. 2091, n. 1, p. 012065.

LEWIS, J.; FOWLER, M. *Microservices: a definition of this new architectural term*. Martin Fowler, 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>.

MICROSOFT. Microsoft, 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices>.

MOZILLA. Mozilla Corporation's, 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript>.

NEWMAN, R. et al. Web 2.0—the past and the future. *International Journal of Information Management*, Elsevier, v. 36, n. 4, p. 591–598, 2016.

NEWMAN, S. *Building microservices*. 2. ed. Sebastopol: O'Reilly Media, Inc., 2021.

REACT. Meta Platforms, Inc., 2023. Disponível em: <https://pt-br.reactjs.org/>.

SHARMA, D. et al. A brief review on search engine optimization. In: *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. [S.l.: s.n.], 2019. p. 687–692.

STACKOVERFLOW. *Stack overflow developer survey 2021*. 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks>.

VUE. MEvan You, 2023. Disponível em: <https://vuejs.org/>.

W3TECHS. *Usage statistics of JavaScript as client-side programming language on websites*. 2022. Disponível em: <https://w3techs.com/technologies/details/cp-javascript>.

ZIANI, A. *Comparaison quantitative et qualitative de frameworks de développement web*. Dissertação (Mestrado) — Université de Namur, 2021.

Apêndices

APÊNDICE A – Código

<https://github.com/lildiop2/tcc>

Fonte: do Autor(2023).