Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.7413

Artigo de Pesquisa/Research Paper

Impacto do Gerenciamento de Estado com a Ferramenta Context API em Single Page Applications

Impact of State Management Using the Context API in Single Page Applications

Vitor Gabriel Comin [Universidade da Região de Joinville (Univille) | *vitorgabrielcomin@gmail.com*]
Leanderson André [Universidade da Região de Joinville (Univille) | *leandersonandre@univille.br*]

Universidade da Região de Joinville (Univille), Joinville, SC, Brasil.

Resumo. Este artigo apresenta uma pesquisa que teve como objetivo identificar o impacto do gerenciamento de estado com a ferramenta Context API em Single Page Applications (SPAs). O estudo foi conduzido utilizando o framework React e a linguagem de programação TypeScript, comparando tempos de carregamento de scripts em abordagens com e sem o uso da ferramenta Context API. Os experimentos foram realizados em dois cenários de componentização: plana, com componentes em um único nível hierárquico, e profunda, com cinco níveis de aninhamento entre os componentes. Nos testes com componentização plana, as diferenças de tempos de carregamento entre as abordagens foram mínimas, com variações estatisticamente significativas apenas em cargas intermediárias. Já nos cenários com componentização profunda, os resultados indicaram tempos de carregamento consistentemente menores com o uso da Context API. A análise estatística, realizada por meio do teste de Wilcoxon, reforçou esses resultados. Assim, conclui-se que, em determinadas condições, especialmente em aplicações com maior profundidade estrutural, o uso da Context API pode estar associado a um comportamento mais eficiente em termos de desempenho, representado pelos tempos de carregamento da interface.

Abstract. This article presents a study aimed at identifying the impact of state management using the Context API tool on Single Page Applications (SPAs). The research was conducted using the React framework and the TypeScript programming language, comparing script loading times between approaches with and without the use of the Context API. Experiments were performed in two componentization scenarios: flat, with components organized in a single hierarchical level, and deep, with five levels of nested components. In the flat componentization tests, differences in loading times between the approaches were minimal, with statistically significant variations observed only at intermediate loads. In contrast, results from the deep componentization scenarios indicated consistently lower loading times with the use of the Context API. Statistical analysis, conducted through the Wilcoxon test, reinforced these results. Thus, it is concluded that under certain conditions, especially in applications with greater structural depth, the use of the Context API may be associated with more efficient performance behavior, as represented by interface loading times.

Palavras-chave: Gerenciamento de Estado, Desenvolvimento Web, Single Page Applications (SPAs), Context API

Keywords: State Management, Web Development, Single Page Applications (SPAs), Context API

Recebido/Received: 16 January 2026 • **Aceito/Accepted:** 27 March 2026 • **Publicado/Published:** 11 April 2026

1 Introdução

O uso da tecnologia vem crescendo em ritmo acelerado, estando cada vez mais presente na vida das pessoas. Segundo o relatório *Digital 2023: Global Overview Report* [We Are Social and Hootsuite, 2023], o número de usuários de internet aumentou significativamente nos últimos anos, tendo 5,16 bilhões de usuários de internet no mundo, o que significava 64,4% da população do mundo na época do estudo.

Diante da crescente importância da internet, o desenvolvimento de páginas web busca atender à demanda por interfaces rápidas e eficientes, pois problemas de desempenho podem levar à perda de usuários e prejudicar a reputação de um site. O desempenho é essencial para o sucesso de empreendimentos on-line, já que sites mais rápidos atraem e retêm mais visitantes (Google, 2023). Além disso, com a popularização dos dispositivos móveis, a expectativa por respostas rápidas aumentou, e a tolerância a atrasos diminuiu. Segundo o relatório *The Need for Mobile Speed*, 53% dos usuários abandonam um site móvel se ele levar mais de três segundos para carregar, evidenciando o impacto direto da performance na experiência do usuário e nas conversões [Google, 2016].

Neste contexto de crescente demanda por desempenho, as *Single Page Applications* (*SPAs*) se tornaram populares no desenvolvimento web por ser uma das abordagens mais eficazes quando o objetivo é trazer desempenho para a interface. Essa abordagem permite que a interface seja atualizada de forma fluida, sem necessidade de recarregar a página inteira. As SPAs também trazem um processamento assíncrono de dados, o que faz com que as informações sejam carregadas e exibidas apenas quando necessário, poupando esforço da aplicação. De acordo com Jadhav, Sawant e Deshmukh [Jadhav *et al.*, 2015], as SPAs economizam largura de banda e tempo no ciclo de requisição e resposta, resultando em uma experiência de usuário mais rápida e eficiente.

Para aproveitar as vantagens das SPAs, é essencial um gerenciamento eficaz do estado da aplicação, que representa o status atual da interface e os dados exibidos. Um gerenciamento eficiente do estado pode reduzir o processamento e o uso de memória, proporcionando uma experiência mais fluida e responsiva. Além disso, ele suporta recursos como cache de

dados, permitindo o compartilhamento de informações entre componentes e reduzindo o tempo de cálculos ou requisições, resultando em interfaces mais estáveis e ágeis [Techtarget, 2024].

Entretanto, estudos anteriores, como os de Le [Le, 2021] e Flovén [Flovén, 2020], que investigaram o impacto de diferentes abordagens de gerenciamento de estado no desempenho de aplicações web, reconhecem a necessidade de novas pesquisas para se obter conclusões mais abrangentes e confiáveis. Em especial, esses trabalhos apontam que ainda há poucos estudos sobre o tema, sobretudo em aplicações de maior escala e com diferentes estruturas de componentização, o que evidencia uma lacuna relevante a ser explorada com maior profundidade.

Considerando esse contexto e as lacunas apontadas por estudos anteriores, este estudo busca responder à seguinte questão de pesquisa: como o gerenciamento de estado com a ferramenta Context API impacta SPAs? O fenômeno central abordado é a relação entre o uso da Context API e o desempenho percebido na interface dessas aplicações, avaliado com base nos tempos de carregamento. Para tanto, esta pesquisa tem como objetivo identificar o impacto do gerenciamento de estado com a ferramenta Context API em SPAs. Sendo uma ferramenta simples e eficiente para gerenciamento de estado no *framework* React, e uma das principais soluções utilizadas atualmente [Awari, 2023], o estudo visa contribuir com evidências que auxiliem desenvolvedores na escolha de soluções mais eficazes para o gerenciamento de estado em aplicações web.

2 Revisão da Literatura

O desenvolvimento de aplicações web tem evoluído rapidamente para atender às crescentes expectativas dos usuários por experiências mais ágeis, interativas e eficientes. Nesse cenário, surgem arquiteturas e práticas que buscam equilibrar desempenho e usabilidade, como as SPAs e os mecanismos de gerenciamento de estado. A seguir, serão apresentados os principais conceitos relacionados a essas abordagens, com ênfase nos estudos que investigam como as ferramentas de gerenciamento de estado influenciam o desempenho de aplicações modernas.

2.1 Desempenho e Arquitetura: SPAs e o Papel do Gerenciamento de Estado

O tempo de carregamento de uma página web continua sendo determinante para a retenção e satisfação do usuário. Embora usuários de dispositivos móveis tolerem tempos ligeiramente mais longos, a experiência deteriora significativamente após 7 a 10 segundos de espera, resultando em maior taxa de desistência [Arapakis *et al*., 2021]. A otimização do tempo de carregamento é essencial não apenas para a retenção de usuários, mas também para a maximização de retorno financeiro, sendo um fator determinante para a competitividade digital. A Google [Google, 2023] enfatiza que sites rápidos aumentam as taxas de conversão e satisfação, destacando a importância de métricas como o Largest Contentful Paint (LCP) para melhorar os resultados financeiros.

As SPAs surgem como uma solução eficiente para oferecer uma navegação mais ágil e interativa, pois atualizam apenas os componentes necessários em vez de recarregar a página inteira. Segundo Jadhav, Sawant e Deshmukh [Jadhav *et al*., 2015], as SPAs carregam todos os recursos essenciais na primeira solicitação, atualizando apenas partes específicas conforme o usuário interage, o que resulta em uma experiência mais fluida e otimizada no uso de banda e tempo de resposta. Isso contribui para uma navegação contínua e melhora a performance em dispositivos móveis e desktop.

Nesse contexto, o gerenciamento de estado se torna essencial para garantir que a interface do usuário (UI) reflita o estado atual da aplicação de forma precisa. De acordo com Alkhateeb [Alkhateeb, 2024], o gerenciamento de estado organiza e mantém as informações sobre o que ocorre na aplicação, permitindo que a UI seja atualizada conforme as interações do usuário. O React, uma das bibliotecas mais populares para o desenvolvimento de SPAs, oferece uma série de recursos para facilitar o gerenciamento de estado. Uma dessas ferramentas é a Context API, que permite o compartilhamento de dados entre componentes sem a necessidade de bibliotecas externas. O uso do React, e da Context API em particular, tem sido amplamente adotado em projetos de grande escala devido à sua eficiência e flexibilidade [Devographics, 2023].

Além das abordagens tradicionais de gerenciamento de estado, a literatura recente também propõe classificações das diferentes arquiteturas utilizadas em aplicações React. De acordo com Droń e Szandała [Droń and Szandała, 2026], as soluções de gerenciamento de estado podem ser organizadas em categorias como abordagens centralizadas, que utilizam um estado global único, como Redux e Zustand; abordagens reativas, baseadas em observáveis e propagação automática de mudanças, como o MobX; abordagens baseadas em contexto, como a Context API do React, que permite o compartilhamento de estado entre componentes sem a necessidade de bibliotecas externas; e abordagens atômicas, como Recoil e Jotai, que modelam o estado em unidades menores e independentes.

Nesse contexto, a Context API pode ser compreendida como uma solução nativa e de menor complexidade dentro do ecossistema React, sendo mais adequada para aplicações de pequeno a médio porte. Embora apresente vantagens como simplicidade e ausência de dependências externas, estudos indicam que sua escalabilidade pode ser limitada em aplicações com maior complexidade estrutural, especialmente quando comparada a soluções mais robustas [Droń and Szandała, 2026].

2.2 Karl Fredrik Flovén (2020): *State Management Models Impact on Run-time Performance in Single Page Applications*

O estudo de Karl Fredrik Flovén [Flovén, 2020], desenvolvido como dissertação de mestrado, investiga o impacto de diferentes modelos de gerenciamento de estado no desempenho de SPAs. A pesquisa compara dois modelos principais: o gerenciamento global, que centraliza o estado em um único local acessado por diversos componentes, e o gerenciamento relativo, onde o estado é distribuído e propagado diretamente entre componentes específicos. O objetivo do autor é oferecer insights práticos para desenvolvedores, auxiliando na escolha da abordagem mais adequada com base no impacto de cada

modelo no desempenho da aplicação.

Para realizar a análise, Flovén utilizou ClojureScript e a biblioteca Reagent, implementando o gerenciamento de estado com a ferramenta Re-frame. Os experimentos foram conduzidos em dois cenários: uma estrutura ampla, onde todos os componentes são filhos de um único pai, e uma estrutura profunda, com múltiplos níveis aninhados. Os resultados indicaram que o modelo relativo teve melhor desempenho em estruturas amplas, enquanto o modelo global se mostrou mais eficiente em estruturas profundas. O autor destaca a necessidade de estudos adicionais para validar esses achados em diferentes contextos e tecnologias. Essa pesquisa é relevante para o presente estudo, pois fornece uma base comparativa sobre o impacto do gerenciamento de estado, embora aqui a análise seja conduzida com a Context API do React, amplamente utilizada no desenvolvimento de SPAs.

2.3 Thanh Le (2021): *Comparison of State Management Solutions between Context API and Redux Hook in ReactJS*

O estudo de Thanh Le [Le, 2021], desenvolvido como tese de bacharelado, compara os métodos de gerenciamento de estado Context API e Redux Hook no React. A pesquisa tem como objetivo fornecer uma análise prática para auxiliar desenvolvedores na escolha da solução mais adequada para otimizar o desempenho de uma aplicação. Para isso, o autor adota critérios como rastreamento de mudanças, pacotes instalados, complexidade, consumo de recursos, tempo de processamento e escalabilidade. Apesar dos achados, Thanh Le reconhece limitações na pesquisa e sugere a necessidade de estudos adicionais para validar os resultados em diferentes cenários.

Na metodologia, foi criado um projeto de portfólio em React, no qual duas versões foram desenvolvidas: uma utilizando Context API e outra com Redux Hook. A análise mostrou que o Redux consumiu mais memória, mas apresentou um tempo de processamento menor (243 ms) em comparação à Context API (326 ms). O autor concluiu que a Context API, por ser mais simples e fácil de configurar, é recomendada para projetos menores, enquanto o Redux, apesar da maior complexidade, demonstrou vantagens no rastreamento de mudanças e na eficiência em projetos de maior escala.

2.4 Considerações sobre os Estudos Analisados

Os estudos de Flovén [Flovén, 2020] e Thanh Le [Le, 2021] abordam o gerenciamento de estado em aplicações web, destacando a necessidade de pesquisas adicionais para uma análise mais aprofundada. Flovén aponta que as métricas utilizadas para medir desempenho podem apresentar resultados imprecisos [Enghardt *et al*., 2019] e que o desempenho medido nem sempre reflete a percepção do usuário [Borzemski and Kędras, 2019]. Thanh Le [Le, 2021], por sua vez, reconhece limitações em seu estudo, como a simplicidade da aplicação e a coleta manual dos dados, sugerindo a necessidade de comparações mais amplas com outros *frameworks*. O presente estudo tem como objetivo explorar o impacto do gerenciamento de estado utilizando a Context API em diferentes cenários de carga de dados, buscando contribuir com uma análise mais robusta sobre seu desempenho em situações variadas.

3 Procedimentos metodológicos

Este estudo quantitativo investigou o impacto do uso de uma ferramenta de gerenciamento de estado no desempenho de uma aplicação SPA, avaliando o tempo de resposta da aplicação ao processar diferentes volumes de dados para comparar o desempenho médio das abordagens com e sem gerenciamento de estado.

3.1 Caracterização do delineamento de pesquisa

Trata-se de uma pesquisa quantitativa, experimental e comparativa que investiga o impacto do uso da Context API no desempenho de SPAs desenvolvidas em React.js com TypeScript. Segundo Creswell [Creswell, 2009], a pesquisa quantitativa é adequada para a coleta e análise de dados numéricos, permitindo a verificação de hipóteses por meio de experimentos controlados.

O estudo comparou os impactos em desempenho da presença e da ausência da Context API em dois projetos com diferentes níveis de componentização, plana e profunda, buscando analisar variações no tempo de resposta conforme a complexidade estrutural da aplicação.

3.2 Caracterização da amostra

Os experimentos foram realizados em um ambiente com as seguintes configurações:

Tabela 1. Configuração do ambiente experimental

Componente	Especificação
Processador	Intel i5 de 10ª geração
Memória RAM	8 GB
Armazenamento	240 GB SSD
Sistema Operacional	Windows 11 Home
Versão do React	18.3.1
Linguagem de Programação	TypeScript (versão 4.9.5)
Navegador	Google Chrome 136.0.7103.114 (64 bits)

Além das especificações apresentadas na Tabela 1, Os testes foram conduzidos em ambiente controlado, com o mínimo possível de interferências externas, como execução de aplicações em segundo plano. O navegador foi utilizado em condições padronizadas, com cache limpo entre as execuções, a fim de reduzir efeitos de armazenamento temporário nos resultados. Também foram mantidas condições consistentes de rede e processamento ao longo dos experimentos, buscando garantir a reprodutibilidade e a confiabilidade dos dados coletados.

3.3 Técnicas e ferramentas de coleta e análise de dados

A coleta e análise de dados ocorreram por meio de experimentos computacionais que executaram diferentes versões em cada aplicação: uma utilizando Context API e outra sem gerenciamento de estado. Para mensurar o desempenho, listas de endereços fictícios foram enviadas entre componentes do mesmo nível, variando o tamanho dos arrays entre 64, 128, 256, 512, 1024, 2048 e 4096 itens. Os dados foram gerados com a biblioteca Faker.js, que cria informações realistas para garantir variabilidade e replicabilidade dos testes. Cada experimento foi repetido 50 vezes, registrando os tempos de execução em milissegundos para posterior análise estatística. Para simular uma carga de trabalho realista, foi inserida uma

função com esforço computacional intensivo nos componentes, aproximando as condições encontradas em aplicações reais.

O tempo real registrado considerou apenas a soma dos tempos referentes a processamento de scripts, renderização e exibição, excluindo carregamento, sistema e mensagens, conforme fórmula adaptada com base em Flovén [Flovén, 2020]:

$$T_{\text{real}} = T_{\text{script}} + T_{\text{render}} + T_{\text{exib}} \quad (1)$$

em que T_{real} representa o tempo total observado, T_{script} corresponde ao tempo de processamento de scripts, T_{render} ao tempo de renderização da interface e T_{exib} ao tempo de exibição na tela.

Essa decisão metodológica, distinta da adotada por Flovén [Flovén, 2020], teve como objetivo isolar o impacto do gerenciamento de estado no desempenho da aplicação, uma vez que os tempos excluídos não variam entre as abordagens analisadas ou, como no caso do tempo de sistema, são independentes da lógica da aplicação. A exclusão específica do tempo de sistema também contribuiu para a redução do desvio padrão das amostras, ao eliminar variações associadas a fatores externos ao comportamento da aplicação em si.

As duas abordagens (com e sem gerenciamento de estado) foram testadas em dois projetos distintos: um com componentização plana e outro com componentização profunda, conforme definição de Flovén [Flovén, 2020]. Na componentização plana, os componentes são organizados em um único nível hierárquico. Já a componentização profunda apresenta múltiplos níveis aninhados de componentes, como ilustrado na Figura 1. No presente trabalho, o projeto de componentização profunda foi desenvolvido com nível 5 de profundidade.

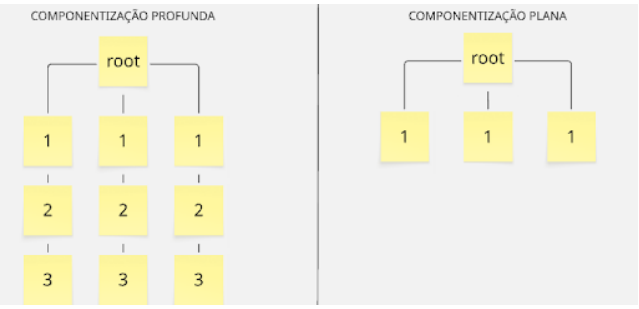

Figura 1. Representação da componentização profunda e da componentização plana.

As amostras de tempo obtidas em cada cenário — combinando tipo de componentização, uso do gerenciamento de estado e tamanho das listas — foram coletadas a partir das capturas da aba "Desempenho"das ferramentas de desenvolvedor do Google Chrome, organizadas em planilhas utilizando o Google Planilhas para cálculo das médias, desvios padrão e elaboração de gráficos. Posteriormente, essas amostras foram submetidas ao teste de Wilcoxon, um método estatístico não paramétrico, com o objetivo de verificar a significância das diferenças entre as abordagens e garantir a confiabilidade dos resultados [Siegel and Castellan, 2006].

4 Análise dos dados e discussão dos resultados

Os testes foram realizados nas duas abordagens: sem ferramenta de gerenciamento de estado (gerenciamento relativo) e com o uso da Context API (gerenciamento global). A Tabela 2 apresenta as médias e desvios padrão calculados a partir de 50 amostras para cada quantidade de itens em ambas as abordagens na componentização plana, enquanto a Tabela 3 traz os mesmos dados para a componentização profunda.

Tabela 2. Tempos de carregamento (ms) e desvios padrão para gerenciamento de estado relativo e global na componentização plana

Configuração	64	128	256	512	1024	2048	4096
Gerenciamento de Estado Relativo							
Média (ms)	833,42	1011	1316,52	1914,52	3065,1	5301,34	9709,76
Desvio padrão	30,44	49,83	62,25	83,17	76,15	209,75	255,69
Gerenciamento de Estado Global							
Média (ms)	833,46	1000,72	1320,94	1921,16	3010	5177,94	9668,9
Desvio padrão	16,26	27,43	53,24	58,02	106,48	123,22	357,30

Tabela 3. Tempos de carregamento (ms) e desvios padrão para gerenciamento de estado relativo e global na componentização profunda

Configuração	64	128	256	512	1024	2048	4096
Gerenciamento de Estado Relativo							
Média (ms)	2774,72	2952,86	3260,44	3857,24	4996,22	7271,92	11862,98
Desvio padrão	40,62	72,59	42,52	67,74	97,28	153,44	419,98
Gerenciamento de Estado Global							
Média (ms)	842,12	1030,9	1327,92	1872,02	3079,36	5287,36	9779,06
Desvio padrão	30,16	56,63	83,21	46,82	81,85	130,22	366,95

As Figuras 2 e 3 exibem gráficos Box Plot que representam a distribuição dos resultados para as abordagens nas componentizações plana e profunda, respectivamente, facilitando a comparação visual entre elas. Esses gráficos ajudam a entender como os valores estão distribuídos e onde estão os resultados mais comuns e os mais distantes.

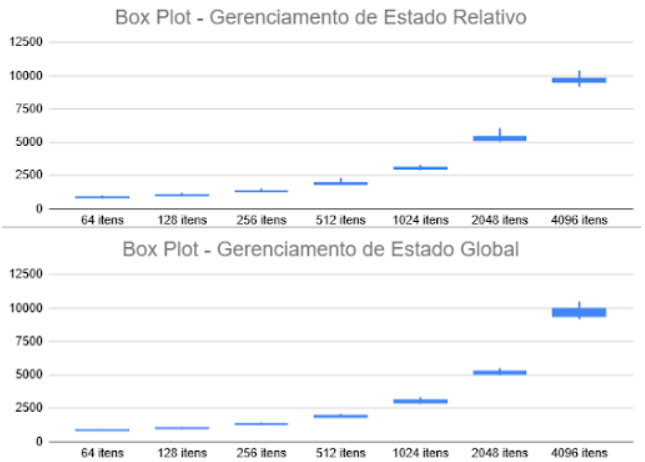

Figura 2. Gráficos Box Plot dos resultados na Componentização Plana

4.1 Componentização Plana

Na configuração de componentização plana, os resultados, conforme apresentados na Figura 4, indicaram que os tempos médios de carregamento entre as abordagens com e sem o Context API foram bastante próximos em todas as faixas de tamanhos dos arrays testados. A análise gráfica evidenciou

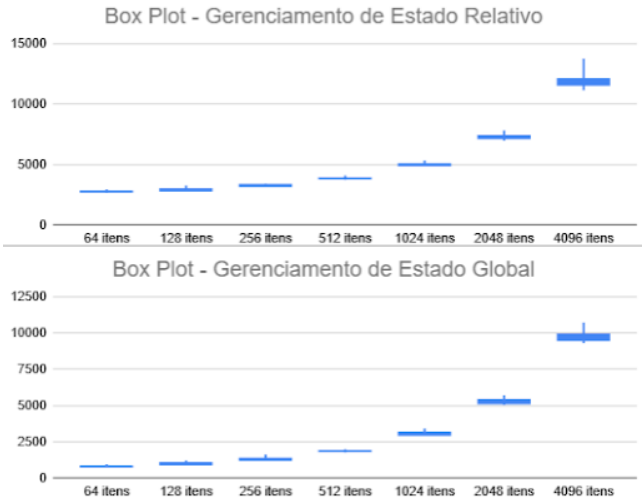

Figura 3. Gráficos Box Plot dos resultados na Componentização Profunda

curvas praticamente sobrepostas, o que sugere uma diferença pouco perceptível no desempenho das duas abordagens sob cargas pequenas e médias. Entretanto, ao se aplicar o teste estatístico de Wilcoxon, foi possível identificar diferenças estatisticamente significativas nas cargas de 1024 e 2048 itens, com valores-p de 0,01 e 0,00, respectivamente. Esses achados indicam que, sob determinadas condições de maior carga, o uso do gerenciamento global de estado pode introduzir um impacto negativo no desempenho, mesmo que a diferença seja sutil e não necessariamente perceptível na análise visual dos gráficos.

Apesar dessas evidências, é importante destacar que, para a carga máxima avaliada, com 4096 itens, não houve diferença estatisticamente significativa entre as abordagens, conforme indicado pelo valor-p de 0,54. Esse resultado sugere que não é possível afirmar que o aumento progressivo da carga de dados, por si só, é suficiente para estabelecer uma tendência clara de impacto no desempenho decorrente do uso do gerenciamento global de estado. Portanto, embora diferenças significativas tenham sido detectadas em determinados volumes de dados, não se pode generalizar que cargas maiores resultarão, necessariamente, em um impacto estatisticamente relevante no desempenho quando se utiliza a Context API em cenários de componentização plana.

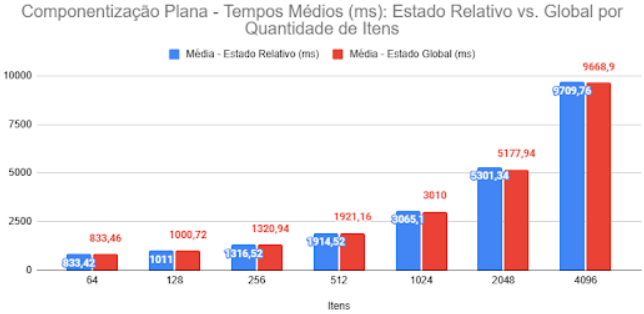

Figura 4. Componentização Plana — Tempos Médios (ms): Comparação entre Gerenciamento de Estado Relativo e Global por Quantidade de Itens

4.2 Componentização Profunda

No cenário de componentização profunda, as diferenças de desempenho entre as abordagens foram mais evidentes, conforme apresentado na Figura 5. A análise gráfica demonstrou que, conforme a quantidade de itens aumentava, o tempo médio de carregamento da abordagem sem gerenciamento de estado (relativa) apresentava um crescimento exponencial, enquanto a utilização do Context API resultava em tempos consideravelmente menores em todas as faixas de tamanho avaliadas.

Essa superioridade do Context API foi confirmada pelo teste estatístico de Wilcoxon, que apresentou valor-p igual a 0,00 para todas as combinações de tamanhos de arrays, indicando diferenças estatisticamente significativas em todos os casos. Dessa forma, pode-se concluir que, na componentização profunda, o uso de uma ferramenta de gerenciamento de estado global, como a Context API, mostrou-se mais eficiente, especialmente à medida que a complexidade da interface e a carga de dados aumentam.

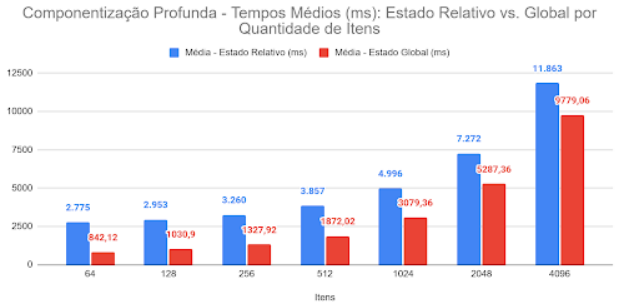

Figura 5. Componentização Profunda — Tempos Médios (ms): Comparação entre Gerenciamento de Estado Relativo e Global por Quantidade de Itens

5 Conclusão

Diante da crescente importância do desempenho no desenvolvimento web e da popularidade das SPAs como solução para interfaces mais ágeis, e considerando a relevância de soluções eficientes para o carregamento de dados e a atualização da interface — especialmente em estruturas mais complexas —, este estudo teve como objetivo identificar o impacto do gerenciamento de estado com a ferramenta Context API em SPAs.

Os resultados indicam que, em aplicações com componentização plana, os tempos médios de carregamento entre as abordagens com e sem o uso da Context API foram bastante próximos, exibindo curvas semelhantes para a maioria das faixas de tamanho dos arrays testados. Entretanto, o teste de Wilcoxon revelou diferenças estatisticamente significativas para cargas intermediárias de 1024 e 2048 itens, enquanto para a carga máxima de 4096 itens não foram observadas diferenças significativas entre as abordagens.

Já no cenário de componentização profunda, os dados apontaram que a abordagem com gerenciamento de estado relativo apresentou tempos médios de carregamento maiores em todos os cenários, quando comparada ao uso do gerenciamento global por meio da Context API. Esta última apresentou tempos inferiores em todas as faixas de tamanhos analisadas, resultado que foi confirmado pelo teste estatístico, com valor-p igual a 0,00 para todos os tamanhos de arrays, indicando diferenças estatisticamente significativas.

Diante dos resultados obtidos, conclui-se que o impacto do gerenciamento de estado com a ferramenta Context API

em SPAs, objetivo deste trabalho, varia conforme a complexidade estrutural da aplicação. Em cenários com componentização plana, os efeitos no tempo de carregamento da interface foram pouco expressivos, com diferenças estatisticamente significativas apenas em cargas intermediárias, nas quais o uso da Context API esteve associado a tempos de carregamento inferiores. Já na componentização profunda, a ferramenta apresentou tempos de carregamento inferiores em todos os testes, com diferenças estatisticamente significativas em todas as faixas avaliadas, sugerindo uma possível vantagem em estruturas mais complexas. Dessa forma, em resposta à questão de pesquisa — como o gerenciamento de estado com a ferramenta Context API impacta SPAs? — os achados concluem que esse impacto pode ser positivo em contextos específicos, especialmente em aplicações com maior profundidade estrutural, ainda que não se possa generalizar os resultados para todos os cenários.

Como perspectivas para trabalhos futuros, propõe-se expandir a análise para outras arquiteturas de componentização e ambientes de execução, incluindo dispositivos móveis, navegadores distintos e variações de hardware. Também seria relevante comparar a Context API com outras soluções de gerenciamento de estado amplamente utilizadas, como Redux, MobX, Recoil e Zustand, considerando não apenas o desempenho, mas também aspectos como consumo de memória, facilidade de implementação, complexidade arquitetural e escalabilidade. Essa comparação permitiria posicionar de forma mais abrangente a Context API em relação às diferentes categorias de gerenciamento de estado presentes na literatura, contribuindo para uma análise mais completa do seu papel em aplicações reais.

Declarações complementares

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a meu pai, José Ivo Comin, e minha mãe, Roseli dos Santos Comin, pelo investimento contínuo em meus estudos e no meu futuro, e pelo apoio incondicional em todas as etapas da minha formação. Seu incentivo e dedicação têm sido fundamentais para que eu alcance cada vez mais conquistas acadêmicas e pessoais.

Contribuições dos autores

Vitor Gabriel Comin foi responsável pela concepção do estudo, desenvolvimento da metodologia, implementação do software, coleta e análise de dados, e redação do manuscrito. Leanderson André contribuiu com supervisão, validação dos resultados, revisão crítica e administração do projeto. Todos os autores leram e aprovaram o manuscrito final.

Conflitos de interesse

Os autores declaram que não têm nenhum conflito de interesses.

Disponibilidade de dados e materiais

Os conjuntos de dados e softwares gerados e/ou analisados durante o estudo estão disponíveis publicamente nos seguintes repositórios GitHub:

- Galeria de Fotos – Componentização Profunda
- Galeria de Fotos – Gerenciamento de Estado Relativo
- Código do Teste de Wilcoxon

Referências

Alkhateeb, B. (2024). Understanding state management: A comprehensive guide. Disponível em: https://medium.com/@bakrialkhateeb.dev/overview-of-state-management-in-software%2Ddevelopment-54c489011b90. Acesso em: 8 nov. 2024.

Arapakis, I., Park, S., and Pielot, M. (2021). Impact of response latency on user behaviour in mobile web search. https://ar5iv.labs.arxiv.org/html/2101.09086. Acesso em: 8 nov. 2024.

Awari (2023). Gerenciamento de estado no front-end: tudo o que você precisa saber. Disponível em: https://fluency.io/br/blog/gerenciamento%2Dde-estado-no-front-end-tudo-o-que-voce%2Dprecisa-saber-2/. Acesso em: 16 out. 2024.

Borzemski, L. and Kędras, M. (2019). Information systems architecture and technology. In *Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2019*, pages 285–301. Springer. DOI: 10.1007/978-3-030-30440-9_27.

Creswell, J. W. (2009). *Research design: qualitative, quantitative, and mixed methods approaches*. SAGE Publications, Thousand Oaks, CA, 3 edition.

Devographics (2023). State of javascript 2023: Front-end frameworks. Disponível em: https://2023.stateofjs.com/pt-BR/libraries/front-end-frameworks/. Acesso em: 10 nov. 2024.

Droń, M. and Szandała, T. (2026). Web application state management: A review of leading react frameworks. *IEEE Software*. DOI: 10.1109/MS.2025.3621269.

Enghardt, T., Zinner, T., and Feldmann, A. (2019). Passive and active measurement. In *PAM 2019*, pages 286–303. Springer. DOI: 10.1007/978-3-030-15986-3_19.

Flovén, K. F. (2020). State management models impact on run-time performance in single page applications. Disponível em: https://scholar.googleusercontent.com/scholar?q=cache:KSaDK1fWrPwJ:scholar.google.com/. Acesso em: 18 set. 2024.

Google (2016). The need for mobile speed. Disponível em: https://blog.google/products/admanager/the-need-for-mobile-speed/. Acesso em: 16 out. 2024.

Google (2023). Why does speed matter? Disponível em: https://web.dev/learn/performance/why-speed-matters?hl=pt-br. Acesso em: 24 set. 2024.

Jadhav, M. A., Sawant, B. R., and Deshmukh, A. (2015). Single page application using angularjs. *International Journal of Computer Science and Information Technologies*. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4c272a54fad90a38e2a218c54653303f1203438d. Acesso em: 16 out. 2024.

Le, T. (2021). Comparison of state management solutions between context api and redux hook in reactjs. Disponível em: https://urn.fi/URN:NBN:fi:

amk-202104134744. Acesso em: 18 set. 2024.

Siegel, S. and Castellan, N. J. (2006). *Estatística não-paramétrica para ciências do comportamento*. Artmed, Porto Alegre, 2 edition.

Techtarget (2024). What is state management? Disponível em: https://www.techtarget.com/searchapparchitecture/definition/state-management. Acesso em: 24 set. 2024.

We Are Social and Hootsuite (2023). Digital 2023: Global overview report. Disponível em: https://wearesocial.com/wp-content/uploads/2023/03/Digital-2023-Global-Overview-Report.pdf. Acesso em: 7 set. 2024.