INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA

***CAMPUS* LAGARTO**

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

ALAN FELIPE DO NASCIMENTO SANTANA

UM BENCHMARKING COM FRAMEWORKS FRONT-END JAVASCRIPT: UMA AVALIAÇÃO TÉCNICA SOBRE A CONSTRUÇÃO DE PROGRESSIVE WEB APPS

LAGARTO - SERGIPE

2024

ALAN FELIPE DO NASCIMENTO SANTANA

UM BENCHMARKING COM FRAMEWORKS FRONT-END JAVASCRIPT: UMA AVALIAÇÃO TÉCNICA SOBRE A CONSTRUÇÃO DE PROGRESSIVE WEB APPS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do *Campus* Lagarto do Instituto Federal de Educação, Ciência e Tecnologia, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Msc. George Leite Junior

LAGARTO - SERGIPE

2024

Santana, Alan Felipe do Nascimento.
S223b Um benchmarking com frameworks front-end javascript: uma avaliação técnica sobre a construção de progressive web apps / Alan Felipe do Nascimento Santana. – Lagarto, 2024.
45 f. ; il.

Monografia (Graduação) – Bacharelado em Sistemas de Informação. Instituto Federal de Educação Ciência e Tecnologia de Sergipe – IFS, 2024.
Orientador: Prof. MSc. George Leite Junior.

1. Informática. 2. Software-desenvolvimento. 3. Framework-programa de computador. 4. Ciência da computação. I. Instituto Federal de Educação Ciência e Tecnologia de Sergipe – IFS. II. Título.

CDU 004.4

ALAN FELIPE DO NASCIMENTO SANTANA

UM BENCHMARKING COM FRAMEWORKS FRONT-END JAVASCRIPT: UMA AVALIAÇÃO TÉCNICA SOBRE A CONSTRUÇÃO DE PROGRESSIVE WEB APPS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do *Campus* Lagarto do Instituto Federal de Educação, Ciência e Tecnologia, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Msc. George Leite Junior (Orientador)
Instituto Federal de Sergipe - IFS

Msc. Telmo Oliveira de Jesus
Instituto Federal de Sergipe - IFS

Msc. Ana Carla do Nascimento Santos
Universidade Tirandetes - UNIT

An. Sist. Matheus Augusto de Santana
PDCase

AGRADECIMENTOS

Primeiramente a Deus que permitiu que tudo isso acontecesse.

Aos meus pais, principalmente minha mãe, Maria Filomena, pelo apoio incondicional e incentivo ao longo do curso.

Aos meus irmãos, Adson, Alisson e Marcele, que sempre estiveram presentes, e me motivaram a continuar!

A minha namorada Luiza, que não me deixou desistir, e sempre acreditou em mim.

Aos meus amigos de curso, pelos desafios que passamos juntos, e trabalhos realizados.

Aos amigos Madson e Jefferson, que me acompanharam desde o inicio dessa jornada, me apoiando quando precisei.

Ao meu orientador George Leite Junior, que me ajudou a construir esse trabalho.

Agradeço também a todos os professores do Instituto Federal de Sergipe (IFS), Campus Lagarto, pelos ensinamentos em toda jornada.

“A simplicidade é o último grau de sofisticação.”

(Leonardo da Vinci)

RESUMO

Com a crescente utilização de dispositivos móveis com acesso a internet, e complexidade na criação de softwares que sejam desenvolvidos em menor tempo possível com qualidade e melhor experiência para o usuário, atendendo tanto navegadores web quanto *smartphones*, surgiram então vários *frameworks* e tecnologias que auxiliam no desenvolvimento dessas aplicações, e essa grande quantidade de *frameworks* pode gerar dúvidas sobre qual utilizar. Pensando nisso, esse trabalho tem como objetivo apontar os principais *frameworks* javascript no mercado utilizando também a tecnologia do *Progressive Web App* (PWA) para criação de um único sistema que pode ser acessado tanto em dispositivos móveis quanto em navegadores web, destacando suas características e realizando um comparativo entre esses *frameworks*. Para isso foi destacado as principais características e limitações do PWA, pretendendo então a partir disso ser capaz de decidir sobre utilizar ou não o PWA para atender os requisitos de um determinado projeto, foi feito também uma comparação entre os *frameworks* a nível de arquitetura, e foi definido caso de testes para obter resultados de benchmark do projeto desenvolvido com os *frameworks*, coletando os dados de análise do tamanho do *build*, tempo de renderização e por meio da ferramenta do Google Lighthouse, ter as métricas relacionadas a performance e uso do PWA. Por fim, esse estudo contribui para compreensão da escolha de um *framework* no momento de criação de um novo sistema.

Palavras-chave: Frameworks. Navegadores web. Dispositivos móveis. PWA. Benchmark.

ABSTRACT

With the increasing use of mobile devices with internet access, and the complexity of creating software that is developed in the shortest possible time with quality and better user experience, serving both web browsers and smartphones, several frameworks and technologies have emerged to assist in the development of these applications. However, this wide variety of frameworks can lead to confusion about which one to use. With this in mind, this work aims to highlight the main JavaScript frameworks available on the market, also utilizing Progressive Web App (PWA) technology to create a single system that can be accessed on both mobile devices and web browsers, emphasizing their features and conducting a comparison between these frameworks. To achieve this, the main characteristics and limitations of PWA were highlighted, aiming to help in deciding whether or not to use PWA to meet the requirements of a particular project. A comparison was also made between the frameworks at the architecture level, and test cases were defined to obtain benchmark results of the project developed with the frameworks, collecting analysis data on build size, rendering time, and using Google's Lighthouse tool to gather metrics related to performance and PWA usage. Finally, this study contributes to understanding the choice of a framework when creating a new system.

Keywords: Frameworks. Web browsers. Mobile devices. PWA. Benchmark.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
HTML	*Hypertext Markup Language*
IBGE	Instituto Brasileiro de Geografia e Estatística
JSON	*Javascript Object Notation*
PNAD	Pesquisa Nacional por Amostra de Domicílios
PWA	*Progressive Web App*
RxJS	*Reactive Extensions for JavaScript*
SPA	*Single-page Applications*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Com a crescente utilização de dispositivos móveis para acesso a internet, como indica a Pesquisa Nacional por Amostra de Domicílios (PNAD), divulgada pelo Instituto Brasileiro de Geografia e Estatística (IBGE) (IBGE, 2023), onde 92,5% dos domicílios possuem acesso a rede e destes 83,3% revelam acessar a internet por dispositivo móvel. Existe, portanto, uma complexibilidade de criação de softwares que estejam disponíveis com qualidade, desempenho, melhor experiência de usuário e capacidade de funcionar off-line, tanto para os celulares, quanto para os navegadores web. A partir deste cenário, foram criados vários *frameworks* que auxiliam o desenvolvimento de aplicações, atendendo aspectos técnicos de otimização, integridade e performance (DEVMEDIA, 2006).

Os *frameworks* têm um papel importante no fornecimento de estrutura e ferramentas que facilitam a criação de aplicações eficientes. Um *framework* é uma estrutura ou um conjunto de bibliotecas, ferramentas, componentes e instruções que fornecem um modelo de desenvolvimento para os desenvolvedores construírem aplicações de forma mais rápida, consistente e escalável. Essa abordagem ganhou popularidade devido às vantagens que oferece em termos de produtividade, manutenção e padronização. (FREEMAN; ROBSON, 2004).

E com a variedade de *frameworks* disponíveis, na maioria das vezes, a escolha de um pode gerar muitas dúvidas, pois como diz GIZAS (2012), além da alta qualidade de código e alta performance deve ser levado em consideração o suporte da comunidade, podendo pensar também na remoção do esforço duplicado para construção da aplicação em diferentes dispositivos.

De acordo com pesquisas feitas entre os anos de 2022 e 2023 , os *frameworks* tais como: ReactJs, Angular, VueJs e Svelte, se destacam como mais populares a serem utilizados no mercado e na comunidade de desenvolvedores (JETBRAINS, 2023).

Deste modo, esse estudo busca realizar uma análise comparativa entre os *frameworks* informados baseando-se na performance e outros critérios definidos, com a utilização de um *Progressive Web App* (PWA), levando em consideração que com essa tecnologia, é possível a criação de um único software que tem a capacidade de atender tanto dispositivos móveis, quanto navegadores web (DEVELOPER, 2022), proporcionando assim uma melhor perspectiva de qual *framework* utilizar em determinada situação.

Tendo em vista a utilização de um PWA no processo de criação de um software, será abordado quais as caracteristas e limitações na utilização do mesmo. Visando trazer um conhecimento do porquê utilizar essa tecnologia com os *frameworks* definidos.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Realizar de um *benchmark* entre os *frameworks* front-end javascript que suportam a criação de PWA destacando e avaliando seus componentes e funcionalidades.

1.1.2 Objetivos Específicos

a) Entender os princípios e características dos PWAs, além dos desafios da sua utilização no desenvolvimento web.
b) Analisar e comparar os *frameworks* à nível de arquitetura.
c) Identificar qual *framework* se destaca em menor tamanho de build e melhor tempo de renderização.
d) Destacar qual *framework* é mais eficaz em performance, a partir da ferramenta do Google Lighthouse.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 FRAMEWORKS

Existem algumas definições de *framework* que podem variar de acordo com alguns especialistas de software, como para FREEMAN e FREEMAN (2004), é uma estrutura conceitual e tecnológica que fornece funcionalidades genéricas e reutilizáveis para facilitar o desenvolvimento de software.

Já para GAMMA (1994), *frameworks* são definidos como "estruturas de suporte definidas em que outro software pode ser organizado e reutilizado".

E como diz JOHNSON (1988), sua importância reside na capacidade de fornecer estrutura, padronização e reutilização de código, acelerando o desenvolvimento e melhorando a qualidade do software .

Ou seja, como mostrado, um *framework* oferece uma estrutura sólida sobre a qual os aplicativos podem ser construídos, com isso faz com que os desenvolvedores se concentrem nas partes do projeto, em vez de se preocuparem com a implementação de funcionalidades comuns.

2.2 FRONT-END

Em relação ao desenvolvimento de software, front-end se refere a parte visual da aplicação, estando ligada diretamente com a usabilidade, acessibilidade, interface e experiência do usuário, garantindo assim a satisfação do cliente final (SMITH, 2018).

Segundo NIELSEN (1993), a usabilidade é uma medida da qualidade da experiência do usuário ao interagir com um produto ou sistema. Então, um front-end bem feito é essencial para criar uma experiência positiva para os usuários.

Para alcançar uma interface de usuário atraente e funcional, os desenvolvedores utilizam diversas tecnologias e ferramentas, incluindo HTML, CSS e JavaScript.

O *Hypertext Markup Language* (HTML) é fundamental para estruturar o conteúdo das páginas web, enquanto o *Cascading Style Sheets* (CSS) é responsável pelo estilo e aparência da interface, incluindo layout, cores e tipografia (MEYER, 2018).

Além disso, o JavaScript é uma linguagem de programação essencial para criar interatividade e dinamismo nas páginas web, possibilitando a manipulação do *Document Object Model* (DOM) e interações assíncronas com o servidor (FLANAGAN, 2011).

Para garantir uma experiência acessível a todos os usuários, é importante seguir as

diretrizes de acessibilidade web estabelecidas pelo *World Wide Web Consortium* (W3C). Essas diretrizes incluem práticas recomendadas para tornar o conteúdo web acessível a pessoas com diferentes habilidades e necessidades (CONSORTIUM), 2021).

Ou seja, o front-end não é apenas uma parte estética de uma aplicação, mas também uma garantia de experiência para o usuário final.

2.3 REACT JS

O React.js, uma biblioteca JavaScript de código aberto mantida pelo Facebook, é hoje uma das tecnologias mais populares para o desenvolvimento de aplicações para web.

O React.js é baseado no conceito de componentes, onde a interface do usuário é dividida em diversos componentes e cada componente é responsável por sua lógica. Como descreve KATZ e THOMAS (2017), "React permite a criação de componentes reutilizáveis, que podem ser compostos para formar interfaces de usuário complexas".

O React.js introduziu o conceito de Virtual DOM, uma representação em memória do DOM real, permitindo atualizações eficientes e minimizando manipulações diretas no DOM (FLEURET, 2019). Ao adotar o Virtual DOM, o React.js melhora significativamente a performance das aplicações web, já que quando comparada com a DOM real, ela identifica as alterações que serão necessárias para poder atualizar a visualização.

O React.js traz um conceito de reatividade que é uma característica fundamental. O estado (*state*) é gerenciado internamente pelos componentes e pode ser alterado de forma controlada, desencadeando automaticamente atualizações na interface de usuário (KACZANOWSKI, 2020). Esta capacidade de reatividade faz com que os componentes se atualizem automaticamente quando o estado dos dados é alterado, logo os desenvolvedores não precisam se preocupar com a manipulação manual da DOM, pois o React lida com essa tarefa de forma eficiente.

2.4 ANGULAR

Angular é um *framework* de código aberto desenvolvido e mantido pelo Google. Destinado a criação de aplicativos web dinâmicos e *Single-page Applications* (SPA). Ele se destaca por permitir uma abordagem modular e escalável, utilizando componentes reutilizáveis e separação clara de responsabilidades dentro da aplicação (FAIN, 2019).

Uma das características mais notáveis do Angular é sua arquitetura baseada em

componentes. Cada parte da interface do usuário é dividida em componentes reutilizáveis, que encapsulam lógica, estilos e templates HTML. Essa abordagem promove a modularidade, permitindo uma maior organização e manutenção do código (FREEMAN, 2018).

Outra característica marcante do Angular é sua capacidade de gerenciar fluxos de dados assíncronos através do *Reactive Extensions for JavaScript* (RxJS), uma biblioteca que facilita a implementação de programação reativa, otimizando a manipulação de eventos e chamadas de *Application Programming Interface* (API) de maneira eficiente e organizada (AHSAN, 2019).

Suas diretivas permanecem como uma característica chave no Angular moderno, relacionado ao antigo AngularJs, permitindo que os desenvolvedores criem novos comportamentos e funcionalidades diretamente no HTML. No entanto, agora as diretivas são mais focadas em adicionar comportamento a componentes existentes, com uma separação mais clara entre diretivas de estrutura (como ngIf e ngFor) e componentes de interface (FREEMAN, 2018).

Os módulos no Angular são utilizados para organizar a aplicação em partes funcionais, agrupando componentes, serviços e outros recursos relacionados. O conceito de módulos permite que os desenvolvedores carreguem apenas partes da aplicação quando necessário, melhorando a performance, especialmente em aplicativos de grande escala, por meio da técnica de *lazy loading* (PAPA, 2019).

2.5 VUE JS

Vue.js é um *framework* de JavaScript progressivo para a construção de interfaces de usuário, ele também é amplamente utilizado para o desenvolvimento de SPA fornecendo uma estrutura eficaz para criação de interface do usuário. Segundo JOHNSON (2019), Vue.js tem ganhado popularidade devido à sua flexibilidade e facilidade de integração com outros projetos e bibliotecas.

Uma das principais características do Vue.js é a componentização. Como apontado por SMITH (2020), o Vue.js permite que os desenvolvedores dividam suas interfaces de usuário em componentes reutilizáveis, facilitando assim a manutenção e o desenvolvimento ágil.

Além disso, o Vue.js utiliza um sistema de reatividade eficiente que permite atualizações automáticas da interface de usuário quando as informações essenciais mudam (JONES, 2018).

Vue.js assim como React.Js adota uma abordagem baseada em Virtual DOM para

melhorar o desempenho das aplicações. O Virtual DOM permite que o Vue.js faça atualizações eficientes na interface de usuário, minimizando as manipulações diretas no DOM, o que resulta em uma melhor experiência do usuário (BROWN, 2017).

2.6 SVELTE JS

Svelte.js, é um *framework* JavaScript que tem ganhado popularidade recentemente entre os desenvolvedores devido à sua abordagem inovadora para a construção de aplicações web. Ao contrário dos outros *frameworks* analisados aqui, como React e Vue.js, que dependem de uma camada de abstração em tempo de execução, o Svelte.js coloca grande parte desse trabalho para o momento de compilação, resultando em um código final mais eficiente e performático.

Conforme ressaltado por SMITH (2021), enquanto *frameworks* como React, Angular e Vue.js adotam uma abordagem baseada em componentes, o Svelte.js utiliza um processo de compilação que converte o código fonte em JavaScript vanilla otimizado. Isso elimina a necessidade de uma biblioteca de tempo de execução volumosa, resultando em aplicações web mais leves e rápidas.

Uma das características mais distintivas do Svelte.js é seu modelo reativo e declarativo. Conforme descrito por JOHNSON (2022), o Svelte.js permite que os desenvolvedores criem componentes reativos usando sintaxe simples de marcação e lógica JavaScript. Quando os dados subjacentes mudam, o Svelte.js atualiza automaticamente o DOM, proporcionando uma experiência de desenvolvimento intuitiva e eficaz.

2.7 PROGRESSIVE WEB APP (PWA)

Progressive Web Apps (PWAs) são uma forma inovadora de desenvolver aplicativos web que proporcionam uma experiência de usuário semelhante à de aplicativos nativos, combinando os melhores aspectos da web e das aplicações móveis. O termo " *Progressive Web App*" foi criado por Alex Russell e Frances Berriman em 2015 e desde então aproximadamente 54 mil websites adotaram essa tecnologia (VISIONVIX, 2023).

As PWAs são projetadas para funcionar em qualquer navegador e em qualquer plataforma, proporcionando uma experiência uniforme aos usuários, independentemente do dispositivo que estão utilizando (RUSSELL; BERRIMAN, 2015).

Elas oferecem a capacidade de serem acessadas a partir do navegador, eliminando a necessidade de downloads e instalações, o que as torna uma opção atraente para os usuários

(DEVELOPERS, 2021).

As PWAs possuem diversas características distintivas, incluindo a capacidade de trabalhar *offline*, e também a capacidade de serem instaladas na tela inicial dos dispositivos móveis (DEV, 2021).

Além disso, elas são seguras, pois são servidas através de HTTPS, garantindo assim a integridade e a segurança dos dados dos usuários (DEVELOPER, 2022).

3 TRABALHOS RELACIONADOS

3.1 COMPARAÇÃO ENTRE OS PRINCIPAIS FRAMEWORKS JAVASCRIPT DE FRONT-END PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB

Em JUNIOR (2022) foi feita uma análise comparativa entre os três *frameworks* mais utilizados no mercado React, Angular e Vue. Foi realizado uma pesquisa em campo com os desenvolvedores *front-end* para entender a experiência deles com os *frameworks* mais utilizados na comunidade, foi feito também uma comparação entre as ferramentas que são utilizadas junto ao *framework* para o desenvolvimento da aplicação, além de uma comparação para obter o desempenho de tempo de compilação, tempo de renderização, tempo de *build* e tempo de execução dos métodos de CRUD (*Create, Read, Update e Delete*).

Os resultados obtidos por JUNIOR (2022) trouxe uma perspectiva que no mercado o *framework* mais utilizado é o React, seguido por Angular e o menos utilizado foi o Vue, podendo entender o que leva a escolha de um *framework* no momento da criação de uma aplicação, além de indicar que a partir dos testes realizados para indicação de desempenho, o React acabou tendo os piores resultados, enquanto o Vue e Angular foram mais eficazes que quase todos os testes realizados.

Assim como no estudo feito por JUNIOR (2022), o presente trabalho busca comparar os *frameworks* mais populares no mercado, porém traz outra forma de comparação, como análise de arquitetura, tamanho de *build* de cada *framework* e a utilização deles junto a tecnologia de PWA, além utilizar as versões mais recentes dos *frameworks* e incluir o *framework* Svelte na comparação, completando o que foi proposto para trabalhos futuros na pesquisa de JUNIOR (2022).

3.2 UM COMPARATIVO ENTRE FRAMEWORKS JAVASCRIPT PARA DESENVOLVIMENTO DE APLICAÇÕES FRONT-END

Em ALMEIDA (2018) foi feito uma pesquisa para entender quais os *frameworks* mais utilizados no mercado, estabelecendo alguns critérios como popularidade e comunidade. Foi desenvolvido um projeto utilizando os *frameworks* escolhidos para poder realizar um teste de *benchmark* e obter os resultados de desempenho a partir dos métodos de CRUD (*Create, Read, Update e Delete*).

Os resultados obtidos por ALMEIDA (2018), mostrou que os *frameworks* mais

utilizados no mercado a partir dos critérios definidos são React, Vue e Angular. Enquanto a resultado de desempenho, houve poucas diferenças entre esses *frameworks* em geral, o Vue e Angular foram mais eficazes em quase todos os casos de testes, e o React acabou tendo números menos satisfatórios.

O presente trabalho, assim como no estudo feito por ALMEIDA (2018), também busca realizar um teste de *benchmark* comparando o desempenho dos *frameworks*, a partir de um projeto desenvolvido. Entretanto, traz uma comparação a mais em relação a tamanho de *build*, a performance foi medida junto a biblioteca do Google Lighthouse e foi incluído o *framework* Svelte para comparação, além de trazer uma perspectiva de como funciona a tecnologia do PWA junto a esses *frameworks*.

4 METODOLOGIA

Este trabalho apresenta um estudo exploratório e descritivo, com abordagem qualitativa, realizado mediante aplicação de *benchmark*. Ou seja, é caracterizado pela busca de aprofundamento do assunto abordado, de maneira a adquirir melhor entendimento e, posteriormente, registrar e descrever as informações obtidas (GIL, 2014). O *benchmark*, prática na qual o estudo é fundamentado, é um processo sistemático utilizado para avaliar, comparar, levantar dados e construir processos (SPENDOLINI, 2005).

4.1 REALIZAR PESQUISA E COMPREENDER OS PRINCÍPIOS E CARACTERÍSTICAS DO PROGRESSIVE WEB APP (PWA)

Este passo consistiu em realizar uma pesquisa mais aprofundada sobre o conceito de PWA, levando em consideração entender seus princípios e suas características principais, e também quais os desafios da sua utilização.

4.2 REALIZAR E ANALISAR PESQUISA A NÍVEL DE ARQUITETURA ENTRE OS FRAMEWORKS.

Este passo consistiu em realizar pesquisas mais centradas na arquitetura de cada um dos *frameworks*, podendo assim fazer uma análise mais aprofundada e comparação entre cada uma delas.

4.3 DESENVOLVER PROJETO PARA TESTES

Foi criado uma SPA utilizando cada um dos *frameworks* mencionados, essa aplicação consiste na criação de um *ShopList* (Lista de compras), com componente de formulário simples contendo um campo de texto para inserir o nome produto, e outro para quantidade de itens, com um botão para salvar, e uma listagem de produtos previamente carregada com 3000 registros em um arquivo *Javascript Object Notation* (JSON) e também os novos produtos castrados, com sua respectiva quantidade, sendo possível assim editar a quantidade de itens, ajustar o valor, e excluir os produtos.

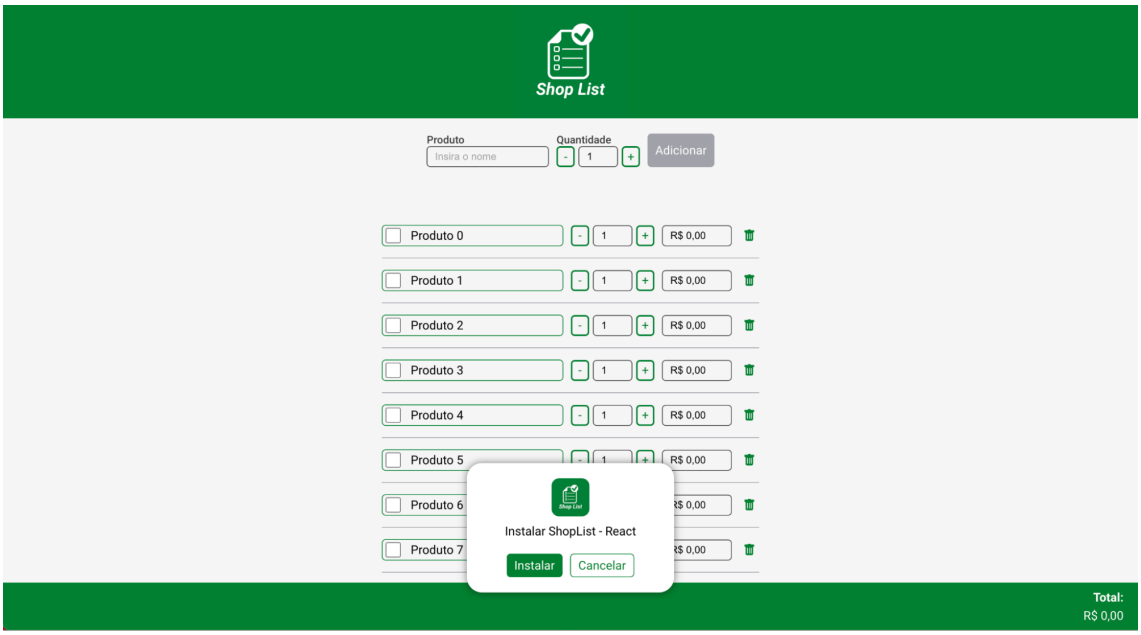

Figura 1 – Aplicação desenvolvida para testes

Fonte**:** Elaborado pelo autor

Podendo ser acessado nos links a seguir:

ReactJs:

- Codigo Fonte: https://github.com/alanflpns/shoplist-react

- Aplicação: https://shoplist-react.vercel.app/

Angular:

- Codigo Fonte: https://github.com/alanflpns/shoplist-angular

- Aplicação: https://shoplist-angular.vercel.app/

VueJs:

- Codigo Fonte: https://github.com/alanflpns/shoplist-vue

- Aplicação: https://shoplist-vue.vercel.app/

SvelteJs:

- Codigo Fonte: https://github.com/alanflpns/shoplist-svelte

- Aplicação: https://shoplist-svelte.vercel.app/

4.4 REALIZAR E IDENTIFICAR OS RESULTADOS DOS TESTES DE BENCHMARK

Este passo consistiu em realizar os testes de aplicação em que foi desenvolvida com os *frameworks*, sendo possível então analisar e identificar os resultados da comparação de *benchmark* de cada aplicação, levando em consideração o tamanho do *build* e tempo de renderização de cada um, onde também foi feito o levantamento de métricas por meio do Google Lighthouse relacionadas a performance e PWA, sendo possível também utilizar de maneira off line tanto em navegadores web, quanto em dispositivos móveis com o PWA.

4.5 AMBIENTE DE COMPARAÇÃO

O ambiente utilizado para realização testes da aplicação foi um computador com a seguinte configuração:

Modelo: MacBook Pro (2023)

Sistema Operacional: macOS Sonoma (14.2.1)

Processador: Apple M2 Pro

Memória(RAM): 16 Gb DDR5

Navegador: Google Chrome versão 120.0.6099.199

5 RESULTADOS

5.1 PRINCÍPIOS E CARACTERÍSTICAS DO PROGRESSIVE WEB APP (PWA)

Após realizar uma pesquisa mais aprofundada sobre o PWA, foi constatado então que são aplicações web que utilizam tecnologias modernas para desenvolver soluções semelhantes a de um aplicativo nativo para dispositivos móveis.

Utiliza então algumas linguagens base como Javascript, HTML e CSS, ou podendo também fazer o uso de algum *framework* para auxílio no desenvolvimento, em prática é um aplicação web em forma de aplicação *mobile*, porém com uma complexibilidade um pouco maior tendo em vista que existem alguns princípios e características específicas, como as listadas a seguir.

5.1.1 Service worker

É uma das principais características do PWA, um *script* que executa em segundo plano no navegador, independente da página web, servindo com um *proxy* entre a aplicação e a rede. Com ele é possível então implementar os recursos fundamentais para melhorar a experiência do usuário como, a capacidade de funcionar offline, organização de *cache* dos arquivos estáticos da aplicação, sincronização de dados em segundo plano e até mesmo envio de notificações push.

5.1.1.1 Funcionamento offline

Com o cacheamento o *service worker* permite que os desenvolvedores controle o *cache* da aplicação, possibilitando que os arquivos CSS, javascript, imagens e até mesmo dados sejam armazenados localmente no dispositivo do usuário, podendo assim fazer com que a aplicação funcione de forma offline, nesse caso então com os dados guardados em cache ele é recuperado quando não há então conexão com a rede.

5.1.1.2 Sincronização em segundo plano

O *service worker* permite que a aplicação agende tarefas para serem executadas em segundo plano, mesmo quando a aplicação não está aberta no navegador, isso é útil para realizar operações assíncronas, como enviar dados para um servidor, ou por exemplo sincronizar

mensagens quando uma conexão de internet estiver disponível.

Quando um usuário acessa a aplicação, o *service worker* é registrado no navegador, normalmente na primeira visita, então quando a aplicação tentar realizar uma operação assíncrona, mas não há conectividade na rede no momento, ela pode solicitar ao *service worker* que registre um evento para essa operação específica, e como ele permanece em execução em segundo plano, aguardando até que a conectividade seja restaurada, quando estiver disponível novamente, o *service worker* recebe um evento de sincronização indicando que agora pode executar as operações pendentes.

5.1.1.3 Notificações push

O *service worker* pode ser utilizado também para receber notificações push sendo útil para alertar os usuários de novos conteúdos. Quando um usuário acessa a aplicação, o *service worker* é registado no navegador, como mostrado no tópico anterior, com isso então quando a aplicação enviar uma notificação para o usuário, ela solicita ao *service worker* que registre o evento de notificação push, essa solicitação pode ocorrer mesmo que a aplicação não esteja aberta no momento, porque o *service worker* está sempre em execução em segundo plano, esperando eventos, então quando recebe a notificação o *service worker* pode exibir para o usuário o conteúdo ou mensagem da aplicação.

5.1.2 Atualizações automáticas

As aplicações que utilizam PWA são automaticamente atualizadas quando sobe uma nova versão, isso garante que os usuários sempre tenham a versão mais recente do aplicativo, sem que seja necessário baixar e instalar as versões manualmente. Isso é importante para corrigir bugs, melhorar desempenho, e adicionar novas funcionalidades, sem depender de esforços significativos por parte dos usuários.

O *service worker* pode ser programado para verificar periodicamente se há alguma atualização disponível, e isso pode ser feito adotando várias estratégias como em intervalos de tempo ou gatilhos específicos de eventos. Se uma atualização estiver disponível o *service worker* baixa os novos recursos e armazena em cache as versões atualizadas, e essa operação ocorre em segundo plano, sem interromper a experiência do usuário.

5.1.3 Segurança

Para utilização do PWA, é obrigatório o uso do protocolo HTTPS, isso significa que toda comunicação entre o navegador e o servidor da aplicação é criptografada, proporcionando assim uma camada de segurança, mesmo que a aplicação não utilize um backend e tenha seus dados armazenados de maneira estática pelo frontend.

O uso do HTTPS ajuda a prevenir ataques *Man-in-the-Middle*, que é quando um atacante tenta interceptar ou modificar a comunicação entre o usuário e servidor, provendo arquivos falsos para, por exemplo, capturar os dados do usuário.

Então não é possível carregar o funcionamento do service worker e utilizar as funcionalidades do PWA, em modo de desenvolvimento (*localhost*), tendo assim que subir a aplicação em um servidor que utilize HTTPS.

5.1.4 Desafios

Mesmo o conceito do PWA não sendo algo novo, existem ainda algumas limitações para sua utilização, já que ainda não tem todo o suporte para seu funcionamento completo.

Um dos principais desafios enfrentados é justamente a compatibilidade para diferentes navegadores e plataformas, apesar da Google ter um avanço maior com o PWA, a Apple não está tão preparada assim para algumas funcionalidades. Por exemplo, o Safari, navegador padrão do iOS, impõe um limite de armazenamento menor para o *service worker* em comparação aos outros navegadores, o que pode restringir a quantidade de recursos que podem ser armazenados localmente, impactando no funcionamento offline.

Outra limitação para o iOS se encontra no envio de notificações push, que embora o safari tenha suporte, a implementação e a experiência podem ser mais restritas, podendo não ter o mesmo nível de integração ou flexibilidade das outras plataformas.

E talvez um dos recursos mais utilizados e com um maior desafio para o iOS na utilização do PWA, é em relação à instalação da aplicação na tela inicial do dispositivo. Como o Google Chrome do iOS é baseado no mecanismo de renderização do safari, ele tem algumas restrições para o uso do PWA, e com isso não é possível e não é disponibilizado a instalação da aplicação pelo navegador, já no Safari, que tem um comportamento mais integrado com o sistema operacional, é possível adicionar a aplicação na tela inicial do dispositivo. Diferente do Android, que é disponibilizado com mais facilidade a instalação, podendo deixar a aplicação web como um aplicativo no dispositivo.

5.2 ARQUITETURA

Como resultado do presente estudo, podemos dizer que o React utiliza o V (*View*) do padrão MVC (*Model-View-Controller*) tendo em vista que é baseada no conceito de componentes e interface do usuário, e faz o uso da Virtual DOM como citado no tópico 2.3. Porém, dependendo de sua implementação, quando utilizada com gerenciamento de estados e interações, sua arquitetura pode se assemelhar ao padrão MVU (*Model-View-Update*), isso ocorre porque quando o estado da aplicação (*Model*) é alterado a partir de interações do usuário (*Update*), novas versões do estado são geradas, atualizando assim a interface da aplicação (*View*).

Já o Angular, adota também uma arquitetura baseada em componentes, e embora a documentação oficial não defina explicitamente uma arquitetura fixa, a abordagem utilizada é muito inspirada no conceito de MVVM (*Model-View-ViewModel*), em que os componentes funcionam como a "*View*"e "*ViewModel*", e os serviços representam a camada de "*Model*". Além disso, o Angular faz uso extensivo de *Observables* (através da biblioteca RxJS) para gerenciar fluxos de dados assíncronos e eventos, o que facilita a construção de aplicações escaláveis e reativas.

O Vue afirma em sua documentação que é utilizado o padrão MVVM (*Model-View-ViewModel*) devido a forma com gerencia a integração entre a lógica de UI(*ViewModel*), a camada de apresentação (*View*) e a lógica de negócio ou modelo de dados (*Model*).

E o Svelte segue um padrão que se assemelha ao MVU (*Model-View-Update*) onde o modelo (*Model*) é representado pelo estado da aplicação, tendo sua camada de apresentação, ou seja, a representação visual do estado da aplicação (*View*) e a lógica de atualização (*Update*) é acionada por eventos da interface do usuário, chamadas da API ou outras ações, e quando o estado (*Model*) é atualizado, reflete as mudanças da *View*.

5.3 EXECUÇÃO DOS TESTES

5.3.1 Tamanho do build

Para que uma aplicação seja carregada, o navegador deve baixar os arquivos necessários, e com o uso de um *framework* esses arquivos são gerados no momento da construção do pacote, que será enviado para o servidor.

Para analisar o tamanho do *build*, foi enviado a aplicação para plataforma de hospe-

dagem da Vercel[1], onde com ela conseguimos construir e realizar *deploy*[2], utilizando assim o servidor gratuito e disponibilizando para o ambiente de produção.

Com isso temos o seguinte resultado:

Figura 2 – Resultado do tamanho do build - React Js

Fonte**:** Elaborado pelo autor

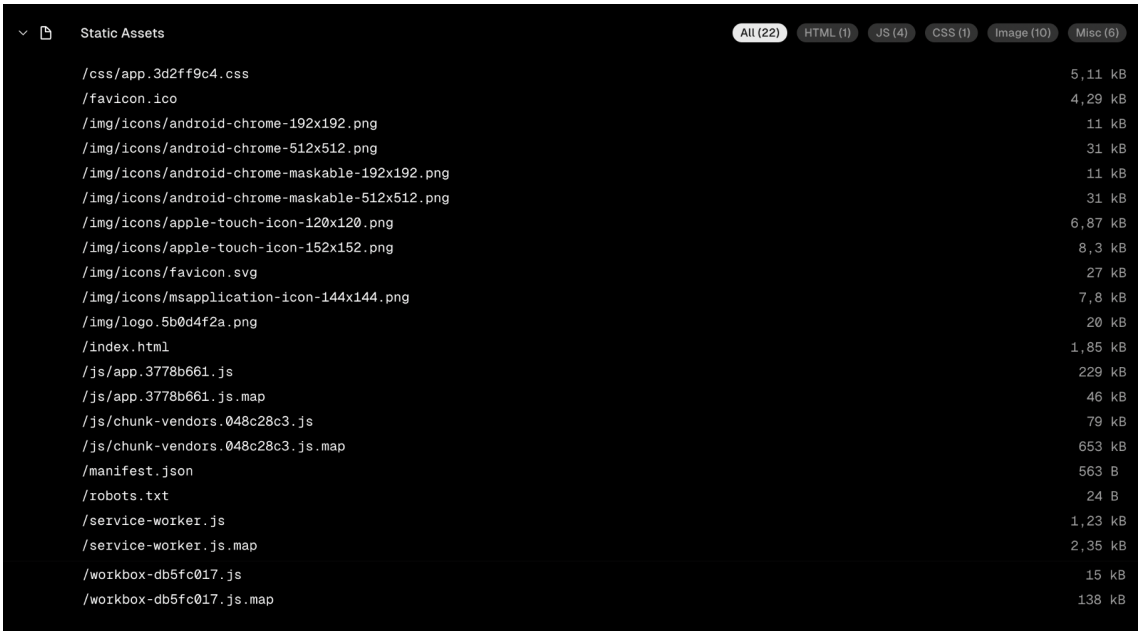

Figura 3 – Resultado do tamanho do build - Vue Js

Fonte**:** Elaborado pelo autor

[1] https://vercel.com/

[2] Processo de colocar no ar uma aplicação ja concluida.

Figura 4 – Resultado do tamanho do build - Angular

Fonte**:** Elaborado pelo autor

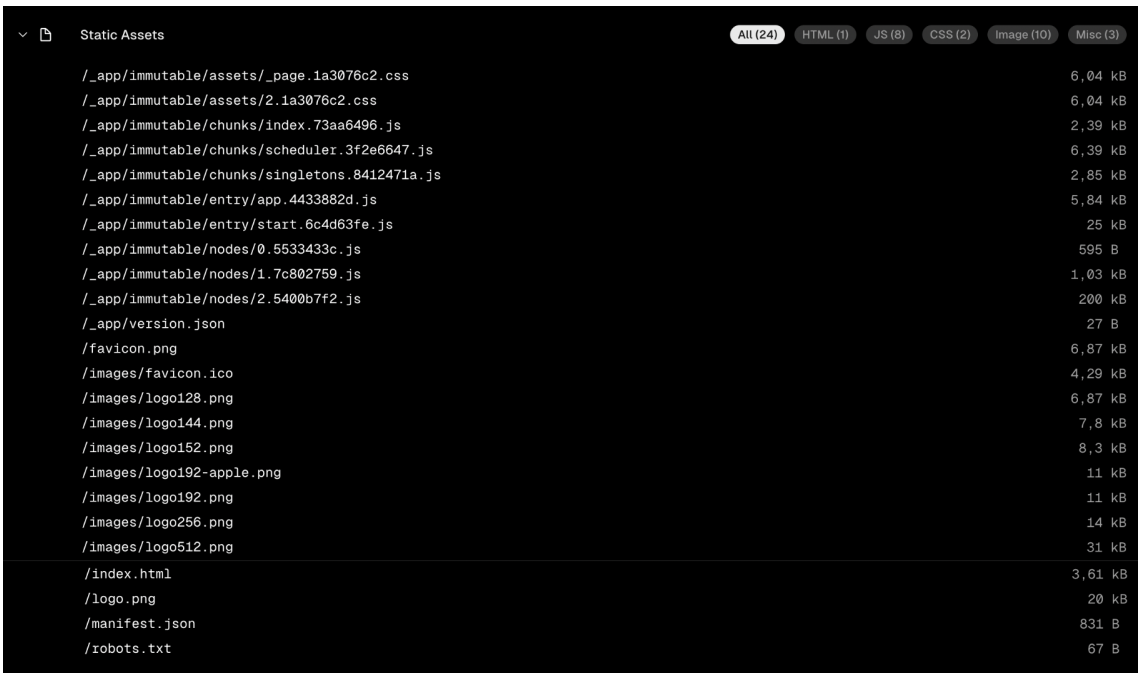

Figura 5 – Resultado do tamanho do build - Svelte Js

Fonte**:** Elaborado pelo autor

Observando os resultados acima, podemos perceber que como mostrado na figura 3, o Angular é o que menos gera arquivos estáticos no seu *build*, com 19, enquanto o Svelte na figura 4, gera a maior quantidade entre os frameworks, com 24 arquivos, o React e o Vue gera tem ambos a mesma quantidade, com 22 arquivos.

Já em relação ao tamanho do *build*, temos o Svelte com o menor tamanho entre os *frameworks*, com 345.85 kB no total, enquanto o Vue é o que tem um maior tamanho, com 1329.60 kB no total, logo em seguida o React com 1246.81 kB e o Angular com 624.97 kB no total.

5.3.2 Tempo de renderização

O tempo de renderização foi analisado através do recurso *Network* do Google Chrome, onde há um item chamado *Load*, que, segundo a documentação, é acionado quando a página é totalmente carregada, o que inclui os elementos DOM e Javascript.

Como podem haver variações nesse carregamento, o teste foi executado 20 vezes para cada *framework*, tendo assim como resposta, o menor tempo e o maior tempo em que foi executado cada, com isso conseguimos calcular e ter um tempo médio de cada *framework*.

A seguir temos os resultados:

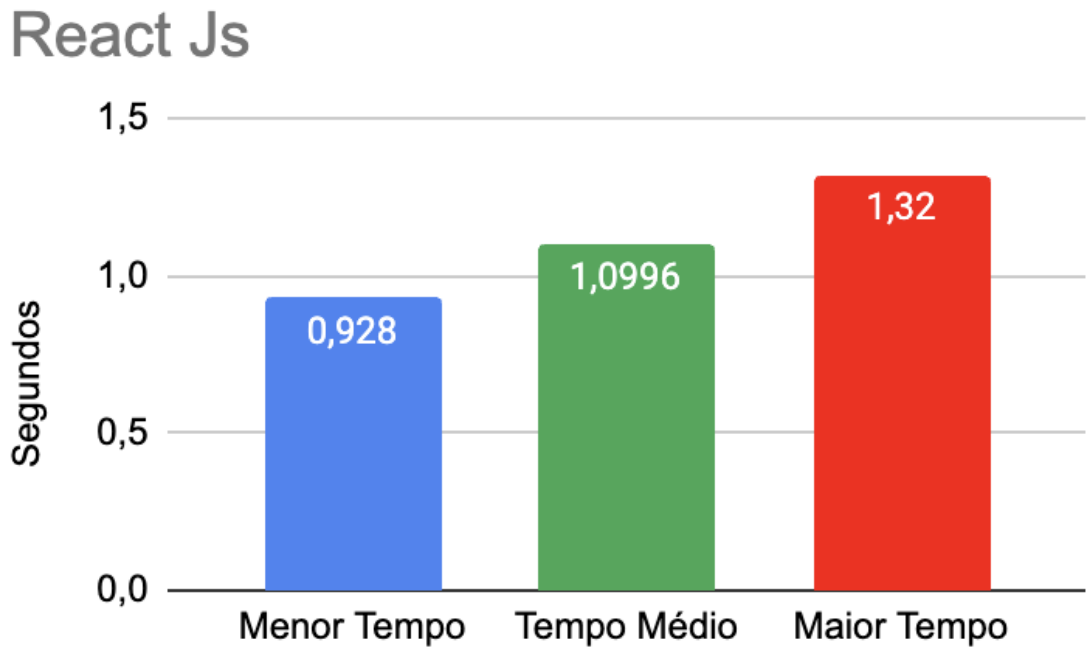

Figura 6 – Resultado do tempo de renderização - React Js

Fonte**:** Elaborado pelo autor

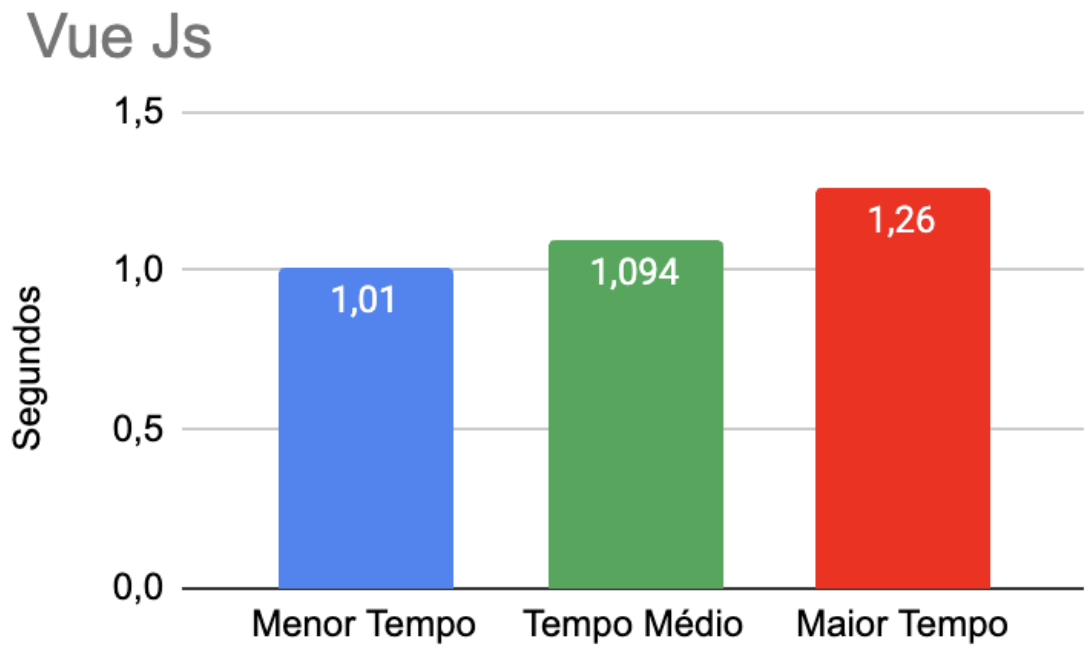

Figura 7 – Resultado do tempo de renderização - Vue Js

Fonte**:** Elaborado pelo autor

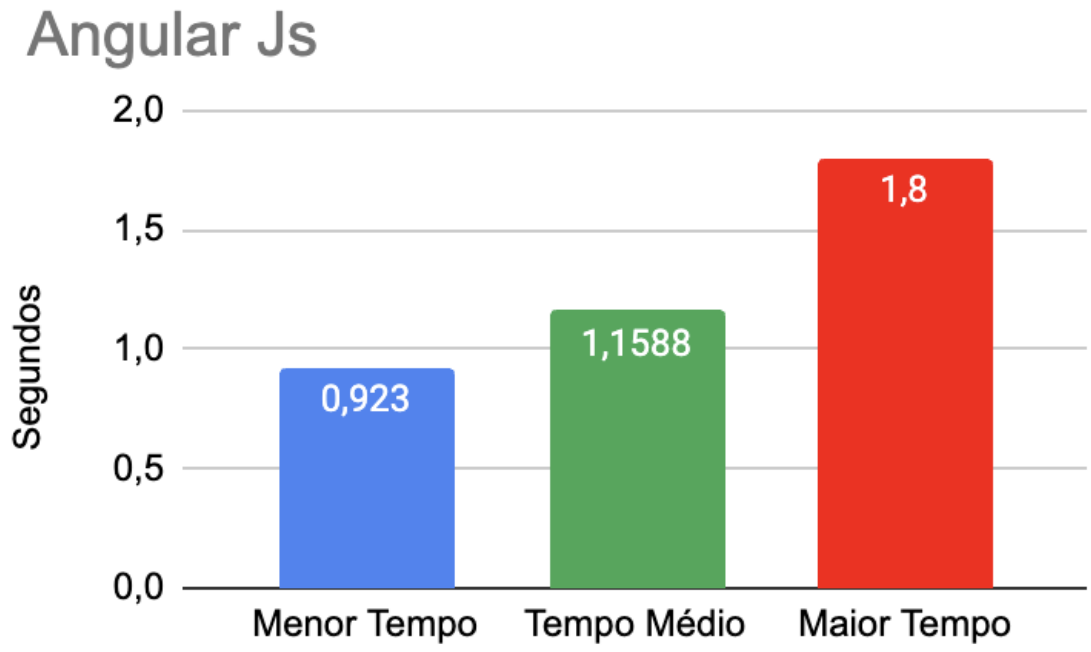

Figura 8 – Resultado do tempo de renderização - Angular

Fonte**:** Elaborado pelo autor

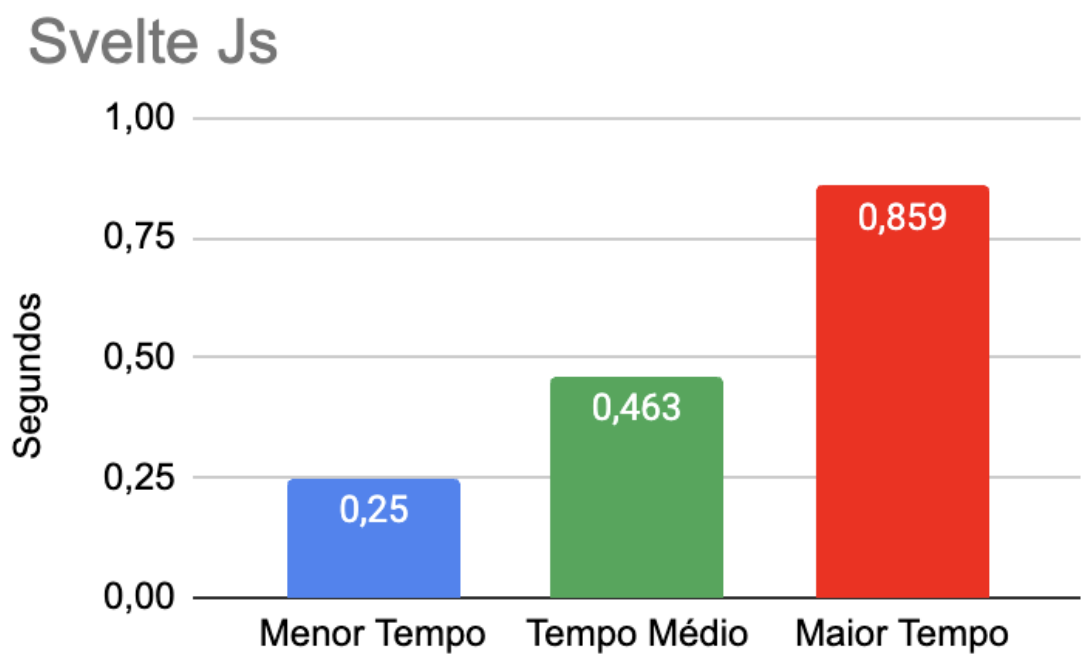

Figura 9 – Resultado do tempo de renderização - Svelte Js

Fonte**:** Elaborado pelo autor

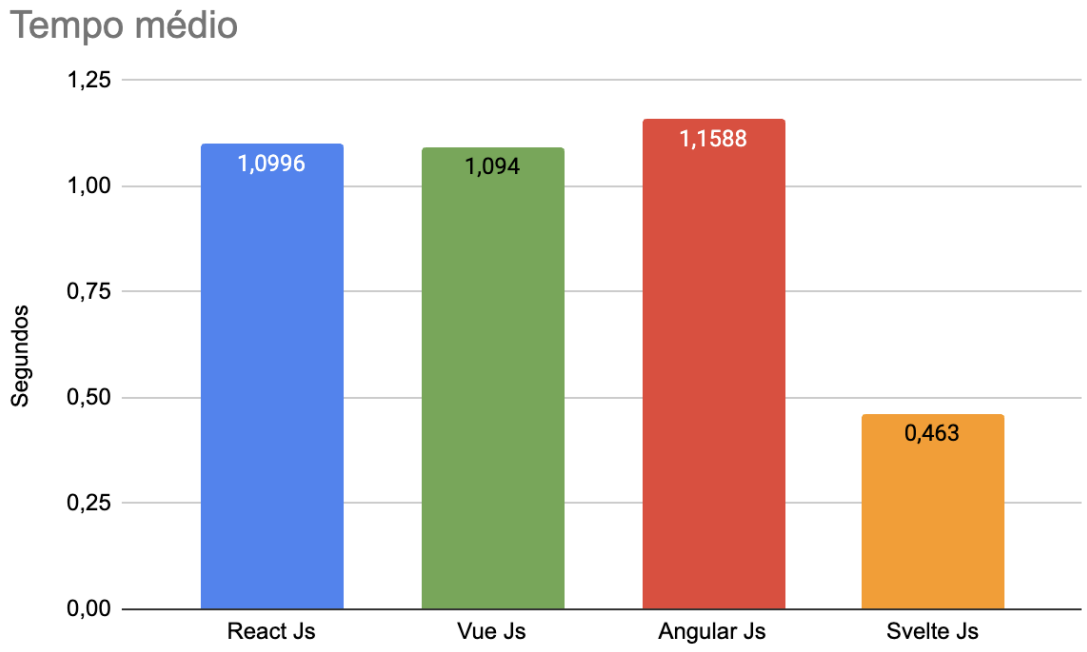

Figura 10 – Resultado do tempo de renderização - Frameworks

Fonte**:** Elaborado pelo autor

Com os testes executados, podemos notar que o Svelte é o *framework* em que teve os melhores casos, com o menor tempo de apenas 0.25 ms, com o maior tempo de 0.859 ms, e com um tempo médio de 0.463 ms, sendo assim o mais performático entre os outros, sendo que seu maior tempo, foi abaixo até do tempo médio dos outros *frameworks*.

O React e o Vue vem em seguida com um tempo bastante semelhantes, com uma média quase que igual, 1.0996 ms e 1.094 ms respectivamente, porém a variação de tempo do React chega a ser maior, com um menor tempo de 0.928 ms e maior tempo de 1.32 ms, sendo assim o Vue mais estável na variação, com um menor tempo de 1.01 ms e maior tempo de 1.26 ms.

E por fim temos o Angular com a maior média no tempo de renderização entre os *frameworks*, com 1.1588 ms, sendo esse o pior caso dos testes.

5.3.3 Performance e uso do PWA

Para os testes do cenário de performance da aplicação, e se há o uso do PWA, foi utilizado a ferramenta do Google Lighthouse para colher os resultados.

Para verificar a performance, é colhidos as seguintes 5 métricas da sessão de desempenho do Google Lighthouse:

a) ***First contentful paint*** **(Primeira exibição de conteúdo)**:

A primeira exibição de conteúdo (FCP, na sigla em inglês), mede quanto tempo o navegador leva para renderizar a primeira parte do conteúdo DOM, depois que um usuário navega até sua página.

Com isso o Google Lighthouse determina a pontuação do FCP da seguinte maneira:

0 a 1.8 segundos: verde (rápido)

1.8 a 3 segundos: Laranja (moderado)

Mais de 3 segundos: Vermelho (lento)

b) ***Largest contentful paint*** **(Maior exibição de conteúdo)**:

A maior exibição de conteúdo (LCP, na sigla em inglês), mede quando o maior elemento de conteúdo na janela de visualização é renderizado na tela, isso é aproximado quando o conteúdo principal da página está visível para os usuários.

Temos a seguintes pontuações:

0 a 2.5 segundos: verde (rápido)

2.5 a 4 segundos: Laranja (moderado)

Mais de 4 segundos: Vermelho (lento)

c) ***Total blocking time*** **(Tempo total de bloqueio)**:

O tempo total de bloqueio (TBT, na sigla em inglês), mede o tempo total em que uma página é impedida de responder à entrada do usuário, como cliques do mouse, toques na tela ou pressionamentos no teclado, a soma é calculada adicionando a parte de bloqueio de todas as tarefas longas entre a primeira exibição do conteúdo e o tempo para interação da página, qualquer tarefa executada por mais de 50 ms é uma tarefa longa, o período após 50 ms é a parte do bloqueio, por exemplo, se o lighthouse detectar uma tarefa de 70 ms, a parte do bloqueio será de 20 ms.

Então temos as seguintes pontuações:

0 a 200 milissegundos: verde (rápido)

200 a 600 milissegundos: Laranja (moderado)

Mais de 600 milissegundos: Vermelho (lento)

d) ***Cumulative layout shift*** **(Mudança de layout cumulativa)**:

A mudança de layout cumulativa (CLS, na sigla em inglês), mede a estabilidade visual da página, porque ajuda a quantificar a frequência com que os usuários experimentam mudanças inesperadas de layout, essas mudanças podem atrapalhar a experiência do usuário de várias maneiras, fazendo com que ele se perca durante a leitura (se o texto se mover repentinamente) ou fazendo com que cliquem no link ou botão errado, em alguns casos, isso pode causar muitos danos.

Com isso temos as seguintes pontuações:

0 a 0.1 segundos: verde (rápido)

0.1 a 0.25 segundos: Laranja (moderado)

Mais de 0.25 segundos: Vermelho (lento)

e) ***Speed index*** **(Índice de velocidade)**:

O índice de velocidade mede a rapidez com que o conteúdo é exibido visualmente durante o carregamento da página, o lighthouse primeiro captura um vídeo do

carregamento da página no navegador e calcula a progressão visual entre os frames.

O lighthouse usa o módulo speedline Node.js para gerar a pontuação do índice de velocidade, então temos a seguintes pontuações:

0 a 3.4 segundos: verde (rápido)

3.4 a 5.8 segundos: Laranja (moderado)

Mais de 5.8 segundos: Vermelho (lento)

A partir disso temos os seguintes resultados dos frameworks:

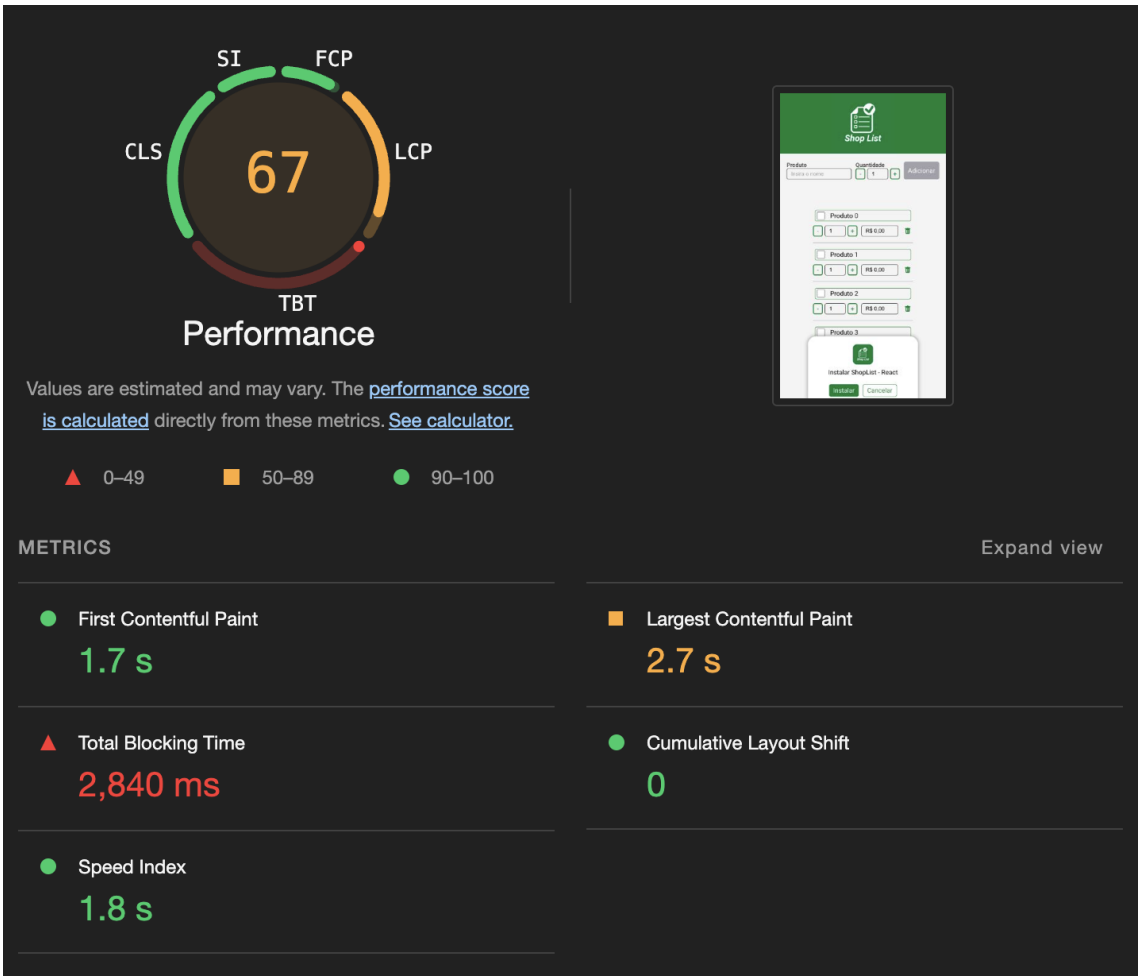

Figura 11 – Resultado da performance - React Js

Fonte**:** Elaborado pelo autor

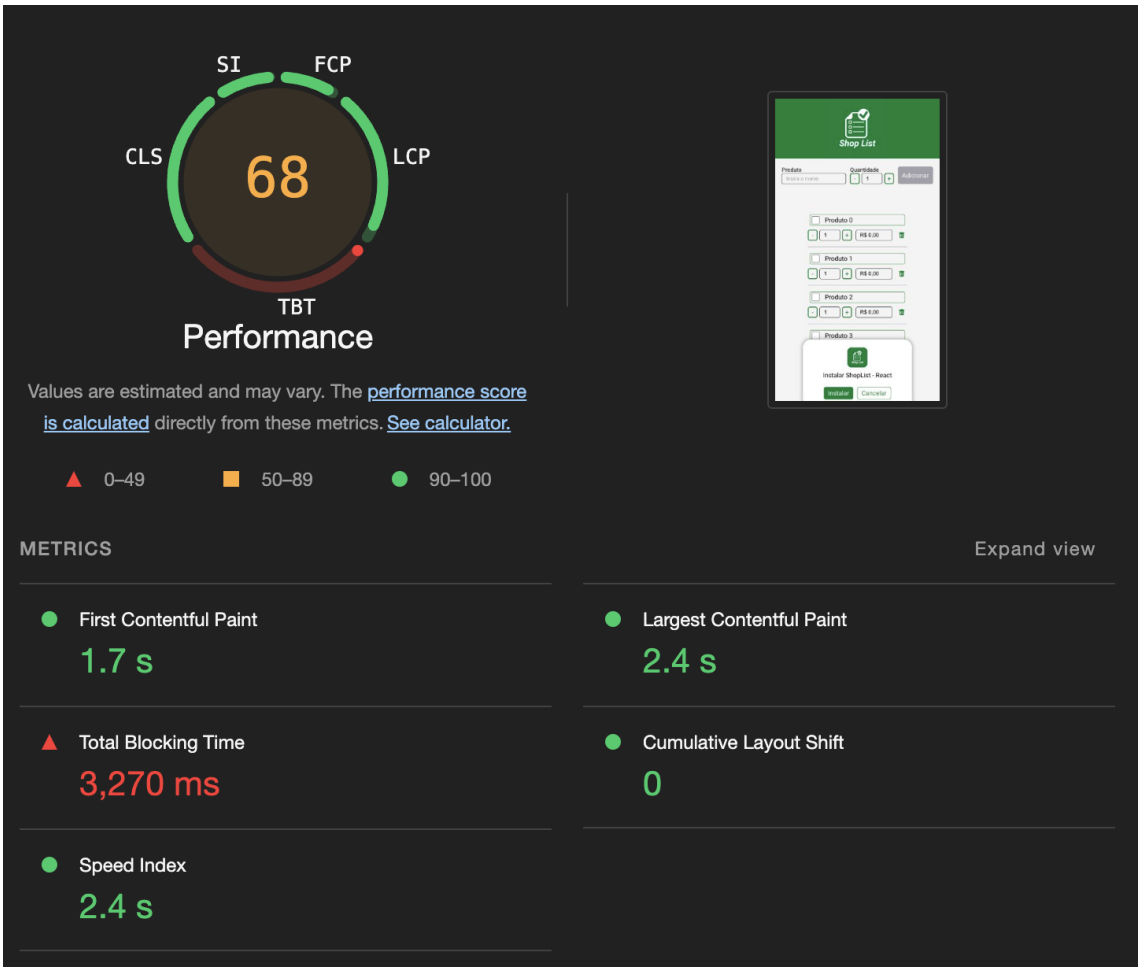

Figura 12 – Resultado da performance - Vue Js

Fonte**:** Elaborado pelo autor

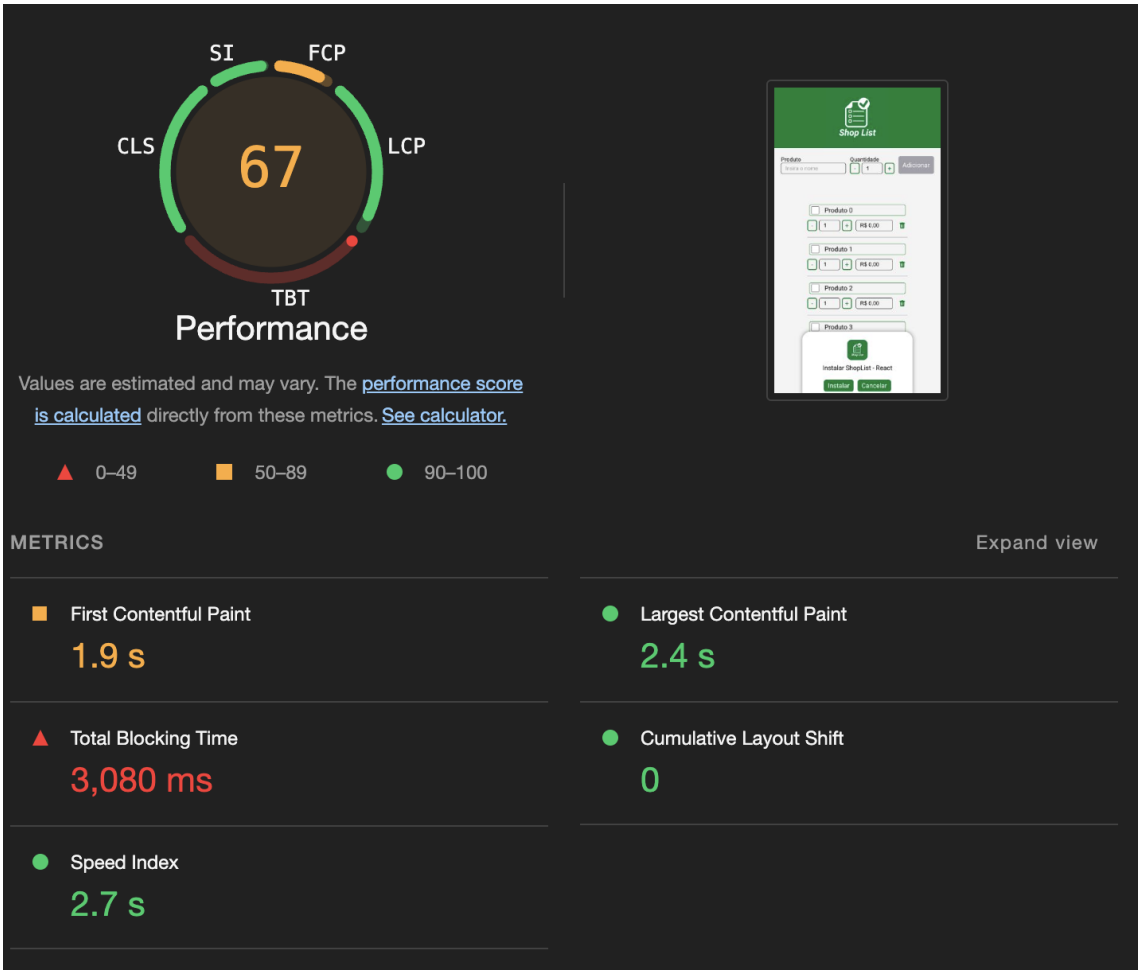

Figura 13 – Resultado da performance - Angular

Fonte**:** Elaborado pelo autor

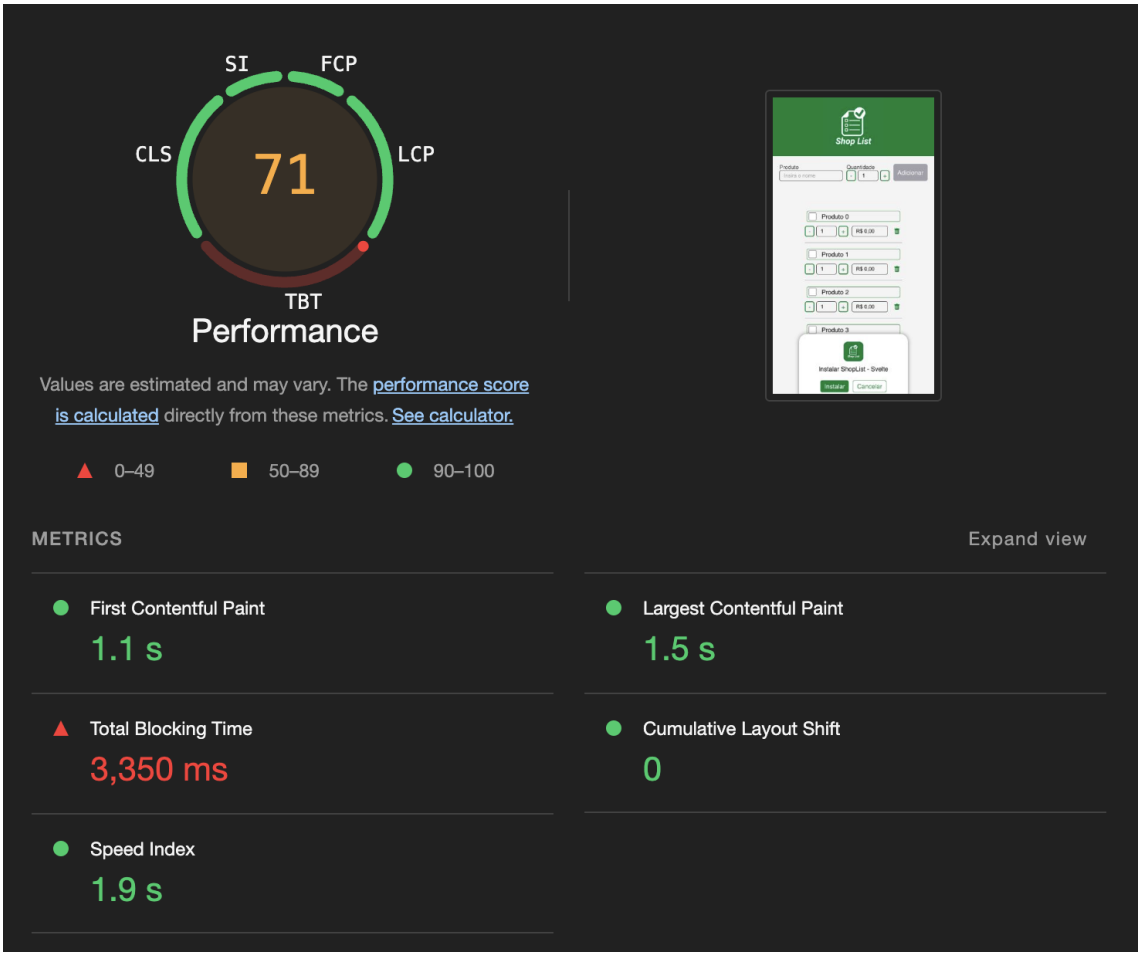

Figura 14 – Resultado da performance - Svelte Js

Fonte**:** Elaborado pelo autor

A partir dos resultados colhidos acima, podemos observar que, em relação a *First contentful paint* (Primeira exibição de conteúdo), o Svelte se destaca com apenas 1.1 segundos de execução, sendo assim o mais rápido a renderizar a primeira parte do conteúdo da DOM, enquanto o Angular foi o mais lento nesse cenário, com 1.9 segundos tendo a pontuação com velocidade moderada, seguindo a pontuação do lighthouse, o React e o Vue tiveram o mesmo tempo, com 1.7 segundos.

Falando do *Largest contentful paint* (Maior exibição do conteúdo), o Svelte também foi o mais rápido com 1.5 segundos para renderizar o maior conteúdo da janela de visualização, o React foi o mais lento nesse cenário com 2.7 segundos para executar, o Vue e o Angular tiveram o mesmo tempo de execução, com 2.4 segundos.

No *Total blocking time* (Tempo total de bloqueio), todos os *frameworks* atingiram a pior pontuação, tendo 600 ms em tempo de bloqueio, o React conseguiu ser o melhor dentre os outros com 2.840 ms de bloqueio, enquanto o Svelte nesse cenário foi o que teve o pior tempo, com 3.350 ms, o Vue veio logo em seguida com o tempo bem próximo de 3.270 ms, o Angular veio em seguida com 3.080 ms.

Em *Cumulative layout shift* (Mudança de layout cumulativa), todos os frameworks tiveram o tempo 0 segundos de execução, sendo assim a melhor nota, e tendo assim todos uma boa estabilidade visual na página, sem mudanças inesperadas de layout.

Já em relação ao *Speed index* (Índice de velocidade), que calcula o tempo em que o conteúdo é exibido visualmente durante o carregamento da página, o React teve o melhor tempo com 1.8 segundos para executar, logo em seguida o Svelte com um tempo muito próximo de 1.9 segundos, depois temos o Vue com um tempo um pouco maior de 2.4 segundos para executar, e por fim o Angular com o pior tempo entre os *frameworks*, de 2.7 segundos.

A partir dessas métricas, o google lighthouse calcula uma média ponderada das pontuações, gerando assim uma pontuação de desempenho geral, assim temos o Svelte com a melhor média de performance dentre os *frameworks* com 71 pontos, logo em seguida temos o Vue com 68 pontos, e por fim o React e o Angular com a mesma média de 67 pontos.

Para verificar o uso correto do PWA na aplicação, a partir do teste executado pelo Google Lighthouse, é feito uma verificação que validam os aspectos de um *Progressive Web App*:

a) *Service worker* sendo implementado na aplicação, e com ele o arquivo de manifesto, onde com isso disponibiliza ao usuário de instalar e adicionar a aplicação na tela inicial do celular ou computador.

b) Uma tela inicial temática que garante uma experiência de qualidade quando os usuários iniciam seu aplicativo pela tela inicial do aparelho.

c) A barra de endereço do navegador pode ter um tema que corresponda ao seu site.

d) Se a largura do conteúdo da aplicação, corresponde a largura da janela de visualização, estando assim adaptável para telas de dispositivos móveis.

e) Se contém um <meta name="viewport» que não apenas otimiza seu aplicativo para tamanhos de tela de dispositivos móveis, mas também evita um atraso de 300 milissegundos na entrada do usuário.

f) Um ícone mascarável que garante que a imagem preencha toda a forma sem ficar em formato *letterbox* (caixa preta ao redor) ao instalar o aplicativo em um dispositivo.

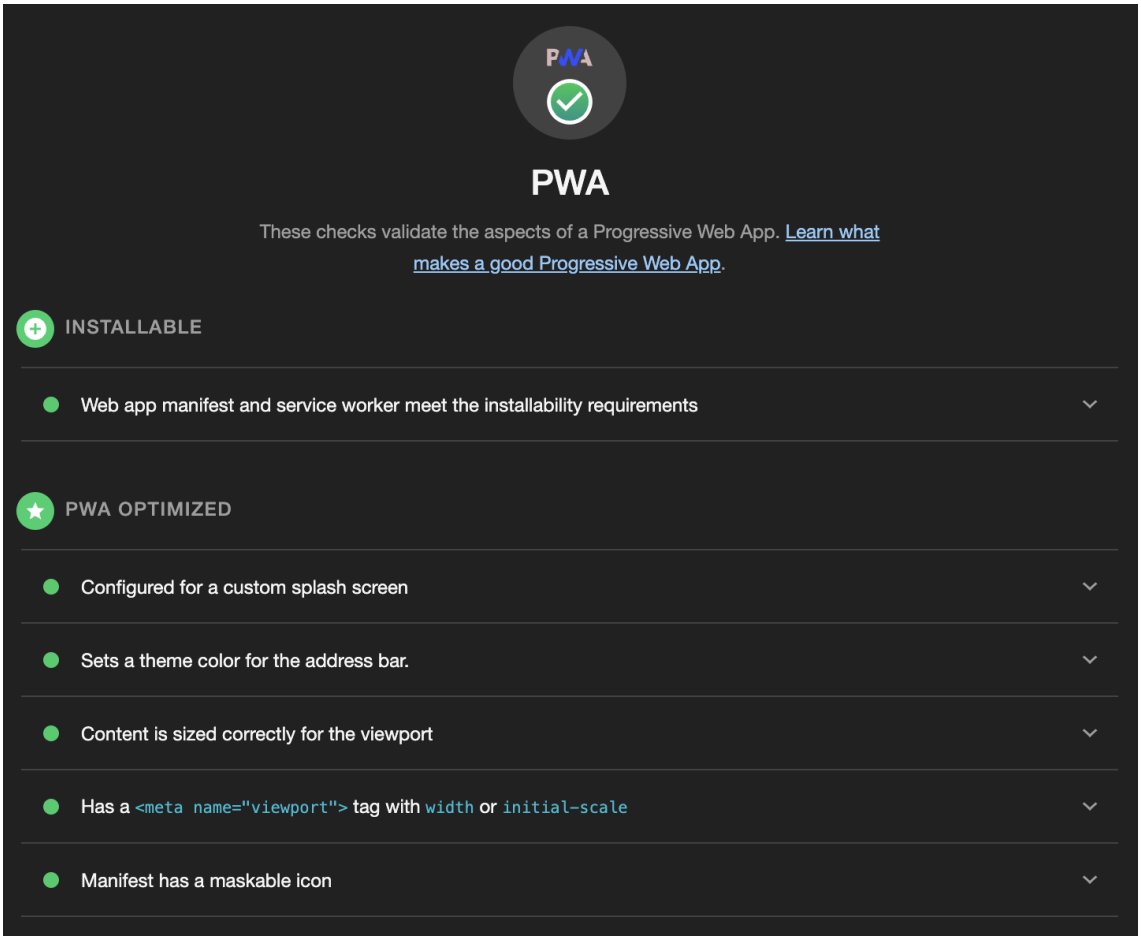

Figura 15 – Aspectos do PWA

Fonte**:** Elaborado pelo autor

Como mostrado na figura 11, todos os frameworks estão com o service worker sendo implementado na aplicação, e como isso estão aptos a utilizar as funcionalidades que o PWA

disponibiliza.

6 CONCLUSÕES E TRABALHOS FUTUROS

Durante o desenvolvimento deste trabalho, foi possível pesquisar e entender o conceito da tecnologia do PWA para a facilidade do desenvolvimento de aplicações, e comparar os *frameworks* mais populares do mercado que ajudam nesse desenvolvimento.

A partir disso podemos ver o uso da implementação do PWA, a sua aceitação nos dispositivos móveis, podendo então destacar suas principais características e limitações da utilização. Também podemos perceber que o uso *frameworks* facilita e agiliza o tempo no desenvolvimento de uma aplicação, o que é uma necessidade para criação de novos sistemas mais complexos, que seja utilizado tanto em navegadores web, quanto em dispositivos móveis.

Portanto, conclui-se que o PWA é uma opção viável e promissora dependendo da necessidade e objetivos específicos do projeto, no entanto é importante considerar suas limitações e se o projeto vai se beneficiar das suas características para atender os requisitos do usuário, cabe então na a decisão de utilizar ou não, avaliar cuidadosamente com base nas demandas específicas de cada projeto.

Com relação ao comparativo entre os *frameworks*, foi possível observar que por se tratarem da mesma linguagem base, a uma grande similaridade no desenvolvimento, tendo cada um sua particularidade o que destaca cada *framework* em diferentes pontos.

Todos eles são compatíveis com o uso do PWA e podem ser usados para criação de sistemas complexos, e há poucas diferenças entre eles no geral. O Svelte se mostrou muito eficaz em todos os casos de teste, se destacado entre os *frameworks* por sua velocidade de carregamento e performance, porém ele não é muito utilizado ainda no mercado por se tratar de um *framework* mais recente, mas já mostra uma eficácia que pode ser considerada pro futuro, já o React traz uma vantagem em relação ao mercado atual e a comunidade de desenvolvedores, porém não é o mais rápido e performático, tendo um tempo muito semelhante ao Vue, que também é bastante utilizado e mais similar ao React em relação ao uso no mercado, enquanto o Angular que é muito utilizado ainda em aplicações mais antigas, foi o que menos se destacou em velocidade e performance tendo números mais elevados.

Então, podemos concluir que cada *framework* pode trazer uma determinada vantagem ao projeto, e assim como o PWA, cabe decidir qual o índice mais importante a ser avaliado para criação da aplicação, tendo como base os requisitos do projeto que irá ser desenvolvido.

6.1 TRABALHOS FUTUROS

Como trabalhos futuros, uma continuação desse estudo, seria a inclusão de outros *frameworks* disponíveis, como por exemplo o SolidJs[1] ou Qwik[2], ou até mesmo outros que não sejam exclusivamente JavaScript, como o Flutter[3].

Além disso, pode ser atualizado os projetos atuais, paras versões mais novas dos *frameworks* que foram lançadas, incluindo a utilização do NextJs[4] (baseado no React) e NuxtJs[5] (baseado no Vue), que oferecem soluções para renderização do lado do servidor (SSR) e outras otimizações, o que poderia melhorar o desempenho dos projetos e a experiência do desenvolvedor.

Outro ponto importante seria analisar as novas atualizações do iOS, se houve uma melhoria nas limitações das funcionalidades e verificar o impacto delas no uso do PWA nos ecossistemas da Apple, e também realizar um estudo de caso em empresas que implementaram o PWA com sucesso, identificando os benefícios e desafios enfrentados.

[1] https://www.solidjs.com/
[2] https://qwik.dev/
[3] https://flutter.dev/
[4] https://nextjs.org/
[5] https://nuxt.com/

REFERÊNCIAS

AHSAN, U. Rxjs in angular: Reactive programming with angular and rxjs. Packt Publishing, 2019.

ALMEIDA, F. E. V. D. Um comparativo entre frameworks javascript para desenvolvimento de aplicações front-end. 2018.

BROWN, D. Vue.js 2 cookbook. Packt Publishing, 2017.

CONSORTIUM), W. W. W. W. **Web Content Accessibility Guidelines (WCAG)**. 2021. Acessado em: nov. 2022. Disponível em: <https://www.w3.org/WAI/WCAG21/quickref/>.

DEV, W. **Progressive Web Apps**. 2021. Acessado em: dez. 2023. Disponível em: <https://web.dev/learn/pwa/progressive-web-apps?hl=pt-br>.

DEVELOPER, M. **Progressive Web Apps**. 2022. Acessado em: out. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/Progressive_web_apps.>

DEVELOPERS, G. **Progressive Web Apps**. 2021. Acessado em: mai. 2023. Disponível em: <https://developers.google.com/web/progressive-web-apps>.

DEVMEDIA. **Frameworks e padrões de projeto**. 2006. Acessado em: out. 2022. Disponível em: <https://www.devmedia.com.br/frameworks-e-padroes-de-projeto/1111>.

FAIN, Y. Angular development with typescript. Manning Publications, 2019.

FLANAGAN, D. Javascript: The definitive guide. O'Reilly Media, 2011.

FLEURET, T. React and react native. Apress, 2019.

FREEMAN, A. Pro angular 6. Apress, 2018.

FREEMAN, E.; FREEMAN, E. Head first design patterns. O'Reilly Media, 2004.

FREEMAN, E.; ROBSON, E. Head first design patterns. O'Reilly Media, 2004.

GAMMA, E. Design patterns: Elements of reusable object-oriented software. Addison-Wesley, 1994.

GIL, A. C. Métodos e técnicas de pesquisa social. 6. ed. reimpr. São Paulo: Atlas, 2014.

GIZAS, A. B. Comparative evaluation of javascript frameworks. Lyon, France: HPCLab, 2012.

IBGE. **Internet foi acessada em 72,5 milhões de domicílios do país em 2023**. 2023. Acessado em: set. 2024. Disponível em: <https://anda.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/41024-internet-foi-acessada-em-72-5-milhoes-de-domicilios-do-pais-em-2023>.

JETBRAINS. **JavaScript Developer Ecosystem Report 2023**. 2023. Acessado em: nov. 2022. Disponível em: <https://www.jetbrains.com/lp/devecosystem-2023/javascript/>.

JOHNSON, A. Mastering vue.js 2. Packt Publishing, 2019.

JOHNSON, A. Reactive and declarative programming with svelte.js. **Web Technologies Review**, v. 18, n. 3, p. 112–124, 2022.

JOHNSON, R. E. Documenting frameworks with patterns. **ACM SIGPLAN Notices**, v. 23, n. 11, p. 63–79, 1988.

JONES, C. Vue.js in action. Manning Publications, 2018.

JUNIOR, A. C. M. D. S. Comparação entre os principais frameworks javascript de front-end para o desenvolvimento de aplicações web. 2022.

KACZANOWSKI, T. React explained: Your step-by-step guide to react. Leanpub, 2020.

KATZ, L.; THOMAS, M. React design patterns and best practices. Packt Publishing, 2017.

MEYER, E. A. Css pocket reference: Visual presentation for the web. O'Reilly Media, 2018.

NIELSEN, J. Usability engineering. San Francisco: Morgan Kaufmann, 1993.

PAPA, J. Angular: From angularjs to angular. Pluralsight, 2019.

RUSSELL, A.; BERRIMAN, F. **Progressive Web Apps: Escaping Tabs Without Losing Our Soul**. 2015. Acessado em: mai. 2023. Disponível em: <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>.

SMITH, B. Vue.js: Up and running. O'Reilly Media, 2020.

SMITH, J. Front-end web development: The big nerd ranch guide. Big Nerd Ranch, 2018.

SMITH, J. The future of front-end frameworks. **Journal of Web Development**, v. 35, n. 2, p. 45–57, 2021.

SPENDOLINI, M. Benchmarking. Bogotá: Editorial Norma, 2005.

VISIONVIX. **Progressive Web App (PWA) Statistics 2024**. 2023. Acessado em: agt. 2024. Disponível em: <https://visionvix.com/progressive-web-app-statistics-2023>.