UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALMIR GONÇALVES CRISPINIANO

ESTUDO COMPARATIVO ENTRE FRAMEWORKS FRONTEND PARA A CRIAÇÃO DE UM PROGRESSIVE WEB APP (PWA)

CAMPINA GRANDE - PB

2021

ALMIR GONÇALVES CRISPINIANO

ESTUDO COMPARATIVO ENTRE FRAMEWORKS FRONTEND PARA A CRIAÇÃO DE UM PROGRESSIVE WEB APP (PWA)

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Everton L. G. Alves

CAMPINA GRANDE - PB

2021

C932e Crispiniano, Almir Gonçalves.
Estudo comparativo entre frameworks fronted para a criação de um progressive web app (PWA). / Almir Gonçalves Crisponiano. – 2021.

14 f.

Orientador: Prof. Dr. Everton Leandro Galdino Alves.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Aplicativos. 2. Desenvolvimento web. 3. Framework. 4. Javascript. 5. Progressive web app – PWA. I. Alves, Everton Leandro Galdino. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

ALMIR GONÇALVES CRISPINIANO

ESTUDO COMPARATIVO ENTRE FRAMEWORKS FRONTEND PARA A CRIAÇÃO DE UM PROGRESSIVE WEB APP (PWA)

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Everton L. G. Alves

Orientador – UASC/CEEI/UFCG

Professor Dr. Franklin de Souza Ramalho

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 20 de Outubro de 2021.

CAMPINA GRANDE - PB

RESUMO (ABSTRACT)

With the rise of Javascript as one of the main programming languages for the Web, several frameworks have emerged to help create complex and scalable projects. A current trend is the use of the Web through mobile devices. In this context, there is an increase in the demand for lightweight applications that replace native applications. This demand resulted in a methodology that combines features offered by browsers with the advantages of using a cell phone. The Progressive Web App (PWA) is a development methodology that seeks to bring to web applications, accessed by mobile browsers, the same experience (light and responsive) experienced in native applications. Currently, there is a huge variety of libraries and frameworks, which causes doubts when making the decision to build a PWA. The objective of this work is to carry out a comparative study between three technologies, for the implementation of a PWA. For this, the same system was developed using Angular, Vue and React to collect metrics and perform analysis on their advantages and disadvantages. As the technologies follow the same paradigm, the study presents the characteristics of each one, associating these with the implementation of a PWA that consumes a Rest API. Finally, guidelines were established to help web programmers choose among the various technologies available.

Keywords: Applications, Web Development, Framework, Javascript, PWA

Estudo Comparativo entre Frameworks Frontend para a Criação de um Progressive Web App (PWA)

Trabalho de Conclusão de Curso

Aluno: Almir Gonçalves Crispiniano
almir.crispiniano@ccc.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

Professor: Dr. Everton L. G. Alves
everton@computacao.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

RESUMO

Com a ascensão de Javascript como uma das principais linguagem de programação para Web, diversos frameworks surgiram para auxiliar a criação de projetos complexos e escaláveis. Uma tendência dos dias atuais é a utilização da Web através de dispositivos móveis. Neste contexto, ocorre um aumento na demanda por aplicações leves que substituam as aplicações nativas. Essa demanda resultou em uma metodologia que combina recursos oferecidos pelos navegadores com as vantagens do uso de um celular. O Progressive Web App (PWA) é uma metodologia de desenvolvimento que busca trazer para aplicações Web, acessadas por navegadores móveis, a mesma experiência (leve e responsiva) vivenciada nos aplicativos nativos. Atualmente, existe uma enorme variedade de bibliotecas e frameworks, o que causa dúvidas na tomada de decisão para a construção de um PWA. O objetivo deste trabalho é realizar um estudo comparativo entre três tecnologias, para a implementação de um PWA. Para isso, foi desenvolvido um mesmo sistema utilizando Angular, Vue e React para coletar métricas e realizar análises sobre suas vantagens e desvantagens. Como as tecnologias seguem o mesmo paradigma, o estudo apresenta as características, de cada uma, associando essas a implementação de um PWA que consume uma API Rest. Por fim, foram estabelecidos *guidelines* para ajudar programadores Web a escolher entre as várias tecnologias disponíveis.

Palavras-chave: Aplicativos, Desenvolvimento Web, Framework, Javascript, PWA

1 INTRODUÇÃO

Em meados dos anos 90, as páginas da internet eram estáticas, ou seja, todas as ordens dadas aos navegadores precisavam ser enviadas a um servidor externo para serem executadas e então retornarem ao seu ponto de partida. Todo esse processo acabava sendo custoso e pouco interativo, já que se passavam bons segundos até que um retorno pudesse ser visualizado. Passados alguns anos, em 1995, foi criada Javascript [9], uma linguagem de alto nível e multi paradigma que permite ao desenvolvedor implementar diversos itens de complexidade em páginas Web tais como animações, mapas, gráficos e informações que se atualizam em intervalos de tempo. Sua criação tem como propósito ser uma linguagem de programação *client-side*, ou seja, executada do lado do usuário, mais especificamente pelo navegador utilizado por ele.

Segundo dados do Índice Tiobe [2], a partir dos anos 2000 é possível ver a ascensão do Javascript como uma das dez principais linguagens de programação. Sua popularidade se dá pelo fato de possuir uma ampla comunidade junto a uma variedade de estruturas e bibliotecas que auxiliam desenvolvedores a criar aplicações complexas. A Figura 1 apresenta uma pesquisa do Stack OverFlow, [7] em 2020, na qual classificou Javascript como a linguagem de programação mais comumente utilizada entre os desenvolvedores pelo oitavo ano consecutivo. No entanto, mesmo com essas vantagens, o processo de construção de um software Web continuou sendo extremamente complexo e oneroso.

Uma das abordagens para aumentar a produtividade e qualidade dos artefatos de desenvolvimento, tem sido a utilização de frameworks de desenvolvimento. Frameworks [1] são coleções de funções e ferramentas que facilitam e criam um ambiente de desenvolvimento de software reusável, aumentando a produtividade, exigindo menor esforço e oferecendo códigos consistentes e robustos, com reconhecido padrão de qualidade, legibilidade e manutenibilidade.

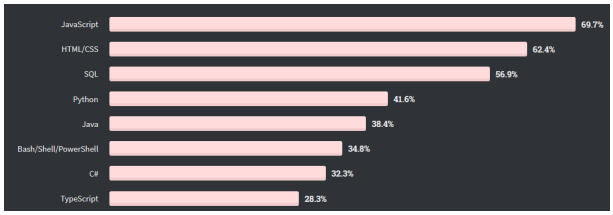

Figura 1: Pesquisa desenvolvida, em 2020 pelo Stack OverFlow, sobre as linguagens mais populares entre os desenvolvedores

Uma outra tendência é a utilização da Web através de dispositivos móveis. Um estudo do Statista [6] indica que cerca de 92,6% do uso da Web ocorre apenas nesses dispositivos. Na mesma direção, uma pesquisa do Comitê Gestor da Internet no Brasil [5], apresentou que cerca de 58% da população acessa a internet apenas pelo celular. Ademais, a revista Panorama Mobile Time/Opinion Box [4], em 2020, apontou que cerca de 81 dos aplicativos (App) utilizados mais vezes por brasileiros ao longo do dia são redes sociais como WhatsApp, Instagram e Facebook. Apenas 19 são aplicações de outro contexto. Além disso, a pesquisa demonstrou que dos 1.980 entrevistados, 48% instalou um aplicativo nas últimas 24 horas e 44% afirmaram ter desinstalado um aplicativo nesse mesmo intervalo de tempo. Sendo assim, é válido concluir que muitos usuários relutam ao fazer downloads de aplicativos e principalmente mantê-los em seus smartphones. Isso motivou uma demanda por aplicações independentes de conexão, leves e instaláveis que corresponde a metodologia de desenvolvimento estudada no presente artigo.

A combinação de mobilidade e tecnologia, está diretamente relacionada aos aplicativos nativos, aqueles que são baixados pelas lojas App Store[1] ou Google Play Store [2]. Porém, o grande tempo de desenvolvimento, a busca por mão de obra especializada em desenvolvimento móvel, o alto custo para empresas desenvolverem e publicarem nas lojas e a baixa aceitação por usuários que não pretendem lotar seus celulares de aplicativos inúteis, indicam vantagens para a adoção dos Progressive Web Apps (PWA) [3].

O PWA é uma metodologia de desenvolvimento que utiliza soluções digitais que apresentam características de aplicações nativas, mas consumem menos dados e não possuem a necessidade de serem baixadas. Foi introduzida por Alex Russell [10], engenheiro da Google, em 2015, como uma solução desenhada para favorecer a experiência do usuário, com o objetivo de facilitar o acesso ao mesmo tempo que leva o usuário a se sentir em um aplicativo. Sendo essa, uma proposta para o aperfeiçoamento das tecnologias Web e uma evolução híbrida entre sites e aplicativos mobile.

O intuito do PWA é combinar recursos oferecidos pelos navegadores mais modernos, junto as vantagens de uso de um celular. Logo, essa metodologia oferece um desenvolvimento menos burocrático e mais aberto que as aplicações nativas, sendo permitido menores gastos para serem desenvolvidos, o que os tornam mais fáceis de serem difundidos e reterem uma maior quantidade de usuários, como é apresentado na comparação da Figura 2, realizada por Alexandre Plennevaux em um estudo publicado no dev.to[8]. Ademais, o PWA busca reduzir problemas relacionados a aplicações nativas, tais como a exigência de grande velocidade de internet, necessidade de constante atualização, longa manutenção e utilização de mais recursos de processamento e armazenamento dos dispositivos. Assim, sendo considerada uma solução para a demanda que busca por aplicações econômicas e independentes de conexão.

Figura 2: Pesquisa compara a retenção de usuários entre um App nativo e um PWA

O desenvolvimento do PWA depende basicamente do conhecimento de tecnologias Web (HTML, CSS e Javascript) em conjunto com frameworks. Entretanto, existe uma vasta quantidade de frameworks que auxiliam desenvolvedores na criação de uma PWA. De acordo com o site The State of Javascript [11], em 2018, as tecnologias mais utilizadas para desenvolvimento Web são os frameworks Angular, Vue e a biblioteca React. A escolha de qual tecnologia usar não deve se basear apenas a afinidades pessoais ou ao *hype* da comunidade. Desta forma, consideramos importante que esta decisão seja baseada em parâmetros técnicos e nas várias características que possam atender aos requisitos do software que será desenvolvido.

Nesse contexto, esse trabalho tem por objetivo ajudar os desenvolvedores de aplicações Web/PWA a escolher, de maneira assertiva, entre as três tecnologias mais utilizadas no mercado (Angular, Vue e React) para a implementação de um software, desconsiderando a sua experiência prévia e focando no desempenho para a criação de aplicações complexas e escaláveis, a depender dos critérios que julgarem mais importantes. Será observado como cada uma das três tecnologias implementam componentes, interceptores, filtros, *services workers* e consomem APIs, do ponto de vista de alguém que possui média experiência com desenvolvimento Web/PWA. Serão considerados como parâmetros de avaliação a curva de aprendizado, a adoção ao Android e iOS, a performance de renderização, o tamanho do pacote de código gerado e a facilidade de implementação.

2 FUNDAMENTAÇÃO TEÓRICA

Nessa seção, serão explicados os conceitos básicos de cada framework junto as funcionalidades e arquitetura oferecidas, por cada um. Além disso, serão apresentados como os PWAs e os Services Workers funcionam.

2.1 Angular

Angular[3] é um framework de código aberto criado em 2009 por Misko Hevery e Adam Abron e mantido pela Google. Sua criação surgiu com intuito de otimizar e refatorar um código de 17 mil linhas em 3 semanas, resultando assim em um código de apenas 1.500 linhas que foi o pontapé para sua primeira versão, Angular.js.

Como framework, Angular busca o desenvolvimento de aplicações simples e otimizadas, assim criando aplicativos modernos, dinâmicos e escaláveis. Seu uso, permite o trabalho com componentes e módulos que utilizam classes para que seja possível encapsular as informações e implementar a aplicação de forma adequada. Sua estrutura utiliza uma arquitetura baseada no Model-View-Controller(MVC), e faz uso de uma funcionalidade chamada *Two-Way data binding* para a atualização simultânea dos *models* e *views*.

Ademais, Angular utiliza o conceito de *Single-Page-Application* (SPA), na qual, a atualização dos componentes é feita dinamicamente e apenas referente a solicitação em questão, sem o carregamento total da página. Consequentemente, reduzindo a quantidade de dados trafegados, tempo de processamento e eliminando a necessidade de recarregar a página inteira sempre que qualquer requisição for feita.

Por ser configurado como um framework, Angular já vem com bibliotecas prontas para todas as necessidades do *client-side* como:

- ***Router***: Biblioteca que permite a navegação na aplicação sem a necessidade do recarregamento da página.

[1] https://www.apple.com/br/app-store/
[2] https://play.google.com/store
[3] https://angular.io/

- ***HTTP***: Biblioteca que permite que uma aplicação frontend se comunique com o Backend por meio do protocolo HTTP.
- ***Forms***: Biblioteca que permite a sincronização com o modelo da aplicação (*two-way data-binding*) e o uso de formulários reativos.

Por fim, o próprio Angular oferece a biblioteca @angular/PWA [4] que permite transformar uma aplicação Web em um PWA, possibilitando a construção de aplicações Web instaláveis.

2.2 Vue

Vue [5] é um framework reativo baseado em Javascript, lançado em Fevereiro de 2014 por Evan You, com intuito de criar uma ferramenta completa e ágil para lidar com grandes *User Interfaces* (UIs). Por ser caracterizado como progressivo focado na camada visual e com fácil integração com outras bibliotecas, ele proporciona a construção de aplicações de larga escala compostas por pequenos componentes, auto-contidos e frequentemente reutilizáveis.

Embora não seja estritamente associado ao padrão Model-View-ViewModel (MVVM), a arquitetura do Vue foi parcialmente inspirada nele. O design de seus componentes é estruturado em um único arquivo (.vue) para facilitar o encapsulamento e manutenção. Vue faz uso da *virtualDOM* para renderização dos dados, com intuito de obter um aumento do desempenho e descartar atualizações desnecessárias. Ademais, o framework utiliza *hooks* para criar um ciclo de vida definido para seus componentes junto às diretivas integradas ao HTML, assim permitindo uma forma dinâmica de interpolação, que proporciona uma maior flexibilidade à linguagem de marcação.

Grandes aplicações podem crescer em complexidade, principalmente devido aos múltiplos dados de estado espalhados entre diversos componentes e as interações entre eles. Por isso, Vue oferece o VUEX [6], uma biblioteca de gerenciamento de estado inspirada em Elm que se integra ao vue-devtools, e permite viajar pelo histórico de mudanças de estado sem precisar de nenhuma configuração adicional.

Por fim, uma das características mais distintas do Vue é seu sistema de reatividade não obstrutivo, na qual, os dados são objetos Javascript puros que se modificam apenas quando a camada visual é atualizada. Isto torna o gerenciamento de estado simples e intuitivo, o que acaba tornando Vue uma das ferramentas frontend favoritas por gigantes como Adobe, Alibaba, Gitlab e Xiaomi.

2.3 React

Desenvolvido por Jordan Walke, um engenheiro de software do Facebook, e lançado em 2013 como uma ferramenta Javascript de código aberto, React[7] é considerada uma das bibliotecas para desenvolvimento Web mais utilizadas no mundo. Mantida pelo Facebook, junto a uma comunidade de desenvolvedores individuais, React pode ser vista como uma biblioteca declarativa na qual o desenvolvedor deve focar mais no resultado do que na forma como ele é atingido.

A tecnologia utiliza JSX, uma combinação de Javascript e HTML, que simplifica toda a estrutura de codificação escrita de um site, para a construção de seus componentes. Ademais, o uso dessa extensão facilita a renderização de múltiplas funções, deixando o código mais fácil e legível para o desenvolvedor. Uma outra vantagem da biblioteca está na construção de um *VirtualDOM* que é hospedado na memória, e tem como resultado um DOM que ao ser atualizado constantemente não interrompe a velocidade da aplicação.

Um dos principais pontos do React está na sua arquitetura própria denominada Flux, que permite que os dados fluam em um sentido unidirecional. Por exemplo, imagine uma árvore de componentes como uma cascata de propriedades (props), o *state* de cada componente é como uma fonte de água adicional que o une em um ponto arbitrário, mas que flui para baixo. A biblioteca adota o fato de que a lógica de renderização é inerentemente acoplada com outras lógicas de UI, na qual o Flux é dividido em Stores, Dispatcher, Views e Actions. Em resumo, as actions passam dados para o dispatcher, que envia esses dados para os stores registrados que por fim, emite eventos para que as views sejam renderizadas.

Por fim, por possuir uma arquitetura própria, ser otimizado e ter vasta comunidade, o React tem sido usado por grandes companhias ao redor do mundo, algumas delas: Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay e Instagram. Sendo essa uma prova de que a biblioteca tem um número de vantagens que ultrapassam certos critérios de comparação com seus competidores.

2.4 PWA e Service Workers

A construção de um aplicativo nativo pode ser uma solução bem cara para algumas empresas, além de que alguns produtos simplesmente não fazem sentido ter um app nativo. Por isso, a metodologia do PWA combina um conjunto de técnicas de desenvolvimento Web junto a funcionalidades que estão disponíveis em apps nativos, com objetivo de criar aplicações leves, otimizadas, independentes de conexão e que possuam uma boa retenção dos usuários.

Principais Características de um PWA:

- **Progressivo**: Pode ser acessado por qualquer usuário independente do dispositivo ou *browser*.
- **Responsivo**: Criado para diferentes dispositivos e tamanhos de tela.
- **Conexão**: Possuir funcionalidades que atuem mesmo se o usuário estiver offline.
- **Seguro**: Utiliza o protocolo *HTTPS*.
- **Instalável**: É possível ser adicionado a tela principal do smartphone com apenas um clique.

Visto como o complemento central para a construção de uma aplicação web utilizando a metodologia PWA, o Service Worker (SW) pode ser definido como um pequeno programa em Javascript que executa em segundo plano e pode interceptar qualquer solicitação HTTP. O seu uso permite manipular essas solicitações e implementar um cache para o aplicativo. Em resumo, o SW funciona como um proxy da rede que intercepta todas as solicitações e decide como respondê-las a partir de uma consulta ao Cache Storage.

Benefícios do Service Worker:

- Permite uma experiência similar a um aplicativo nativo.
- Podem ser usados na maioria dos navegadores ou dispositivos moveis.

[4] https://angular.io/guide/service-worker-getting-started
[5] https://br.vuejs.org/
[6] https://vuex.vuejs.org/ptbr
[7] https://pt-br.Reactjs.org/

- Permite turbinar o desempenho do aplicativo, esteja o dispositivo conectado ou não à Internet.
- Programa responsável por manter uma das principais características do PWA, funcionamento offline.

Ademais, o Service Worker carrega consigo um arquivo denominado manifest.json que contém as informações do aplicativo, descrevendo os recursos de cache assim como implementando funções básicas de um PWA como seu título, ícones, fonte, tela de inicio e paleta de cores.

Por razões de segurança, o Service Worker não pode receber solicitações de outros aplicativos da Web em execução no mesmo navegador. Além disso, ele só funciona em HTTPS. Seu registro só é acionado quando toda a página é carregada, incluindo seus recursos vinculados como imagens, CSS e Javascript. E assim como quase todas as APIs relacionadas ao uso do PWA, as fases do ciclo de vida da API do Service Worker são baseadas em promessas.

Por fim, a filosofia central da especificação do Service Worker é colocar recursos de proxying de rede nas mãos dos desenvolvedores, para que possam implementar muitos casos e padrões de uso em diferentes PWAs, em vez de fornecer apenas um conjunto de padrões off-line predefinidos.

3 ESTUDO EMPÍRICO

O objetivo desse estudo é comparar três tecnologias com a finalidade de auxiliar desenvolvedores de aplicações Web/PWA a escolher mais facilmente entre as tecnologias disponíveis, dependendo dos critérios que julgarem mais importantes em cada caso. Para isso, foi executada uma metodologia onde uma única especificação baseada em User Stories foi implementada pelo primeiro autor deste trabalho utilizando três tecnologias: dois frameworks (Angular e Vue) e a biblioteca React. O sistema escolhido consiste em um site de promoções com funcionalidades como postagem, aprovação, login, cadastro, *like* e visualizações, que simula uma pequena rede social e faz uso de um CRUD (acrônimo comumente utilizado para definir as quatro operações básicas usadas em Banco de Dados Relacionais), conectadas a uma API REST em Spring Boot[8]. A especificação do sistema exemplo consiste de seis User Stories:

- **User Story 1**: Eu como desenvolvedor gostaria de implementar o componente de *Header* junto às páginas de Login e Cadastro.
- **User Story 2**: Eu como desenvolvedor gostaria de implementar a página *Home* e Erro 404 junto ao componente de *Loading*.
- **User Story 3**: Eu como desenvolvedor gostaria de implementar as páginas de cadastro de promoção e admin junto a configuração das rotas da aplicação.
- **User Story 4**: Eu como desenvolvedor gostaria de implementar a página de aprovação do Admin e *Promotion* junto aos componentes de *Like* e *Back-page*.
- **User Story 5**: Eu como desenvolvedor gostaria de implementar os *services* da minha aplicação junto aos filtros e guardas.
- **User Story 6**: Eu como desenvolvedor gostaria de configurar o arquivo manifest.json junto ao service worker para transformar minha aplicação em um PWA .

[8]https://spring.io/projects/spring-boot

Durante o desenvolvimento, foram coletadas métricas com a finalidade de realizar uma análise objetiva das tecnologias:

- **Tempo de implementação**: Tempo médio gasto para a implementação de cada user story. Esta métrica tem a finalidade de realizar uma análise sobre a média de tempo que cada tecnologia gastou na implementação da mesma funcionalidade.
- **Número de linhas de código (LOC)**: Medida usada para avaliar o trabalho de um software, através da contagem do número de linhas no texto do código fonte do programa. Com essa métrica, normalmente, é possível prever a quantidade de esforço que foi necessário para desenvolver o programa, bem como a estimativa de produtividade de programação, uma vez que o software é produzido.
- **Analises com o Lighthouse**: O Lighthouse[9] é uma ferramenta de código aberto que aprimora e analisa a qualidade de APPs da Web. Tem como objetivo simular diversas situações que podem afetar a *User Experience*, realizando um série de testes que apresentam falhas como indicadores para possíveis melhorias, em um relatório final.
- **Tamanho do pacote de arquivos**: Quando um site é carregado o navegador realiza o download dos arquivos necessários para a renderização como os de extensão .HTML, .CSS e .JS além das fontes, imagens, etc. Para analisar o tamanho de arquivos, foi gerado a versão build de cada projeto e foi verificado através das propriedades dos arquivos o tamanho dos pacotes.
- **Número de vezes que o programador precisou recorrer à documentação e páginas de ajuda**: Para promover um bom desenvolvimento, normalmente, uma tecnologia precisa ter uma documentação bem elaborada e uma comunidade ativa que auxilie os programadores na solução de problemas. Um dos focos dessa métrica foi computar o número de vezes que o programador precisou recorrer à documentação oficial e sites como Stack OverFlow [10] para sanar dúvidas e buscar possíveis soluções para os impedimentos encontrados no desenvolvimento.

Complementarmente, foram realizadas avaliações subjetivas sobre a experiência no desenvolvimento, que auxiliaram a medir a curva de aprendizado.

- **Adoção entre o Android e iOS**: Os PWAS são aplicativos comumente construídos para Web mas que possuem *features* diferentes em cada sistema móvel. Portanto, foram analisadas as diferenças de comportamento apresentadas por cada PWA nos sistemas Android e iOS.
- **Curva de aprendizado**: Por apresentarem diferentes arquiteturas, as tecnologias tiveram suas arquiteturas analisadas para medir a curva de aprendizado junto a legibilidade de código na montagem dos componentes.

Por fim, todo o desenvolvimento das aplicações levou cerca de 6 semanas.

[9]https://developers.google.com/web/tools/Lighthouse?hl=pt-br
[10]https://pt.stackoverflow.com/

4 RESULTADOS E DISCUSSÕES

Nesta seção, serão apresentados os resultados do estudo junto as discussões sobre as análises feitas, a partir de cada user story. O tempo de implementação do sistema, nas três tecnologias, teve uma duração total de 40 horas e 12 minutos.

As três aplicações desenvolvidas (Angular[11], Vue[12] e React[13]) e o repositório com os projetos[14] estão disponíveis para análise. A tabela abaixo apresenta 3 das 5 métricas objetivas computadas durante o estudo.

Angular	LOC	Tempo(min)	Acessos a Doc.
US 1	567	158	2
US 2	418	216	14
US 3	530	144	17
US 4	384	126	14
US 5	321	115	18
US 6	100	224	29
Vue			
US 1	410	113	7
US 2	425	138	16
US 3	585	121	18
US 4	366	113	7
US 5	130	82	19
US 6	79	104	15
React			
US 1	496	107	5
US 2	403	163	22
US 3	441	109	19
US 4	268	127	5
US 5	130	76	7
US 6	143	176	17

Antes da implementação da primeira User Story foram realizadas as configurações de ambiente necessárias para a estruturação do projeto. No primeiro momento foi feita a instalação dos frameworks via *Command-Line Interface* (CLI) junto as primeiras configurações e o download de bibliotecas necessárias.

4.1 US 1

O desenvolvimento da primeira aplicação foi iniciado com Angular, na qual, por ser um framework, já oferecia toda a estrutura prévia para o desenvolvimento, como a biblioteca de rotas e HTTP. Em um primeiro momento, o protótipo da aplicação auxiliou ao desenvolvedor na construção da UI. Como proposto anteriormente, foram criadas nessas primeiras user stories o componente de *header*, junto as páginas de login e cadastro. Oferecendo assim a parte de registro e autenticação, que são funcionalidades básicas e essenciais para a construção da aplicação proposta.

Em Angular: O desenvolvimento do *Header* foi facilitado graças às diretivas HTML presentes no framework, como pode ser visto na Figura 3. Essas diretivas permitem que o HTML implemente estruturas básicas como condicionais e laços que auxiliam na reatividade e no funcionamento da aplicação, de acordo com a interação

```
<header class="myHeader">
  <div>
    <img class="myLogo" src="../../../assets/logo.svg" alt="promotion-logo">
  </div>
  <div class="buttons" *ngIf="logged && admin">
    <button id="span-btn" (click)="goToHome()" class=" mobile">
      <span class="material-icons-outlined">
      cottage
      </span></button>
    <button id="span-btn" (click)="approvePromotion()" class=" mobile">
      <span class="material-icons-outlined">
      task_alt
      </span></button>
    <button (click)="logOut()" class=" mobile">
      <span class="material-icons-outlined">
      logout
      </span></button>
  </div>
  <div class="buttons" *ngIf="logged && !admin">
    <button id="span-btn" (click)="goToHome()" class=" mobile">
      <span class="material-icons-outlined">
      cottage
      </span></button>
    <button id="span-btn" (click)="addPromotion()" class=" mobile">
      <span class="material-icons-outlined">
      add
      </span></button>
    <button (click)="logOut()" class=" mobile"><span class="material-icons-outlined">
      logout
      </span></button>
  </div>
  <div *ngIf="!logged && !admin">
    <button (click)="goToLogin()" class="btn">Login</button>
  </div>
</header>
```

Figura 3: HTML do componente de Header em Angular

do usuário. Por possuir apenas a logo e um menu, o header foi algo simples de ser implementado e projetado. Havendo trabalho apenas no uso da biblioteca rxjs [15] que permite implementar o padrão *observable/subject* para o menu se comunicar com a página de login e ser exibido de acordo com o usuário autenticado. Outro passo tomado na US, foi a criação da tela de login, componente essencial para a autenticação do usuário. Uma das funcionalidades oferecidas pelo framework foi a biblioteca de formulários que permite a construção de formulários reativos e que possuam validação, o que facilitou bastante no processo de implementação. Além disso, a biblioteca de rotas ajudou na implementação da lógica de negócio e no gerenciamento dos componentes. Por fim, o componente de cadastro foi implementado em um componente separado, que foi integrado a página de login utilizando a mesma lógica de formulário, validação, estilo e HTML propostos anteriormente.

Em Vue: Por possuir a estrutura de um framework, o uso das diretivas em conjunto ao arquivo único (.vue) permitiu um maior encapsulamento e legibilidade do código. A implementação em Vue se diferenciou de Angular no uso da biblioteca de gerenciamento de estados chamada VUEX, que possui um estrutura dividida em estados, mutações, ações e módulos e permite um gerenciamento global de variáveis que são inerentes ao contexto global da aplicação. Além disso, Vue oferece uma biblioteca própria de rotas que auxiliou na implementação dos componentes citados. Por fim, a implementação da tela de login junto ao componente de cadastro se difere no uso de uma biblioteca externa chamada VeeValidate que auxiliou o desenvolvedor na validação dos formulários junto ao uso da diretiva v-model, que utilizar interligações de mão dupla

[11] https://xenodochial-hodgkin-259383.netlify.app
[12] https://affectionate-ritchie-f371c6.netlify.app
[13] https://suspicious-booth-1b4f41.netlify.app
[14] https://github.com/almirgon/tcc
[15] https://angular.io/guide/rx-library

(*two-way binding*) entre os dados e elementos *input*, para buscar uma maneira correta de atualizar cada elemento com base no tipo de entrada.

```
<header className={styles.myHeader}>
  <div>
    <img onClick={home} className={styles.myLogo} src={logo} alt="promotion-logo" />
  </div>
  {authorized && !admin && <div className={styles.buttons}>
    <button onClick={home} id="span-btn" className={"mobile"}><Home/></button>
    <button onClick={addPromotion} id="span-btn" className={"mobile"}><AddPromotion/></button>
    <button onClick={logOut} id="span-btn" className={"mobile"}><Logout/></button>
  </div>}
  {authorized && admin && <div className={styles.buttons}>
  <button onClick={home} id="span-btn" className={"mobile"}><Home/></button>
  <button onClick={goToAdmin} id="span-btn" className={"mobile"}><Check/></button>
  <button onClick={logOut} id="span-btn" className={"mobile"}><Logout/></button>
  </div>}
  {!authorized && !admin && <div>
    <button onClick={login} className={"btn"}>Login</button>
  </div>}
</header>
```

Figura 4: Retorno em JSX do componente Header em React

Em React: Por ser considerada uma biblioteca, a instalação via CLI foi realizada em conjunto com bibliotecas essenciais como React-Router-Dom, Unform, Yup e Axios para cuidar das rotas, formulários, validação e comunicação com APIs via HTTP, respectivamente. Em suas últimas versões, React vem sendo mantido por componentes funcionais em conjunto aos React hooks que são funções que permitem o desenvolvedor conectar os recursos de estado e ciclo de vida aos componentes funcionais. A construção do componente *Header* utilizou o hook *useContext* que cria um consumidor para componentes React junto a implementação de um contexto para a aplicação. Ademais, foi utilizado a função *useHistory* para realizar o gerenciamento de rotas da aplicação. Porém, o uso de JSX faz com que React não possua diretivas HTML, utilizando assim a lógica através de ternários que junto a expressões booleanas permitem o aparecimento ou não de certos elementos em tela. Essa abordagem, citada anteriormente, pode ser vista na Figura 4, uma das suas principais vantagens está na facilidade da leitura do código, já que mantém o Javascript junto à linguagem de marcação. Por fim, a página de login junto ao componente de cadastro foram criados utilizando a biblioteca Unform[16] que auxilia na construção de formulários reativos e utiliza a biblioteca Yup[17] para a validação dos dados. Entre as 3 tecnologias, o desenvolvedor achou mais fácil a implementação por React, em comparação com as funcionalidades que os outros 2 frameworks ofereceram na criação das telas correspondentes.

4.2 US 2

A segunda *User Story* corresponde ao desenvolvimento de páginas fundamentais para a aplicação, como a *Home* que possui os *cards* com as promoções ativas que levam o usuário a acessar a principal funcionalidade do sistema, visualizar promoções. Ademais, foram criados componentes como a página de erro 404 que é renderizada quando o usuário tenta acessar uma rota inexistente e o componente de *loading* que é exibido enquanto uma página não é totalmente carregada.

Em Angular: O desenvolvimento da página *Home* foi feito durante 3 horas e 36 minutos, sendo eleito o componente mais demorado da aplicação por exigir grande tempo de estilização do CSS e estruturação do HTML. Foram utilizados alguns métodos fornecidos pelo framework como o ngOnInit, que pode ser considerado um gancho de ciclo de vida do componente em Angular para indicar que o framework terminou de construí-lo. Junto a uma estruturação do componente de *Like* que utilizou o sistema de propriedades e eventos que serão detalhados adiante. A criação dos componentes de Error 404 e *Loading* foram consideradas simples pelo desenvolvedor, já que utilizaram apenas HTML e CSS, não exigindo uma forte implementação do TypeScript.

Em Vue: O desenvolvimento da página *Home* em Vue foi a considerada a mais fácil pelo desenvolvedor, já que o mesmo possuía em mente a estrutura do HTML e CSS quase prontas, sendo necessária apenas mudanças na estruturação do Javascript. Na implementação, foi utilizada a função created() que assim como a ngOnInit representa uma parte do ciclo de vida do componente. Ademais, a Criação dos componentes de Erro 404 e *Loading* foram reaproveitados de Angular, mudando apenas a estrutura e a imagem de animação.

Em React: O desenvolvimento da página *Home* em React foi um pouco mais dificultoso que nas tecnologias anteriores, já que por mais que o desenvolvedor possuísse uma estrutura pré-definida, surgiram problemas de CSS durante a implementação, exigindo assim mais tempo para o desenvolvimento da página. Além disso, o desenvolvedor sentiu dificuldades na falta das diretivas HTML para utilizar laços e condicionais. Dificuldades essas que foram contornadas ao recorrer a fóruns como o Stack Overflow. Por fim, a Criação dos componentes de Erro 404 e *Loading* também foram reaproveitados dos frameworks, mudando apenas sua estrutura e imagem de animação.

4.3 US 3

A terceira *User Story* corresponde ao desenvolvimento de páginas de cadastro de promoção e admin, junto a configuração das rotas. As páginas de cadastro de promoção possuem um guarda na qual apenas um usuário cliente cadastrado na aplicação consegue acessá-la. Enquanto as rotas, são arquivos essenciais para a manutenção do padrão SPA, porque permitem que uma página não seja recarregada e renderizada do zero toda vez que um componente novo é exibido em tela. Ademais, nessa US foi instalada a biblioteca sweetAlert2[18], nas 3 tecnologias, com o intuito de facilitar na criação de alertas e notificações que auxiliassem na interface e experiência do usuário.

Em Angular: O desenvolvimento da página de cadastro foi facilitado graças à estrutura prévia dos formulários da página de login e cadastro junto às validações disponíveis. A página de administrador segue uma lista de cards na qual o admin pode verificar, acessar ou aprovar uma promoção enviada por um usuário cliente, possuindo um guarda que permite o acesso apenas do usuário que tenha a regra admin. Por fim, foi configurado o arquivo app-routing que corresponde a uma lista de objetos com atributos que especificam características e permite a construção das rotas de maneira simples e direta. Sendo papel do próprio framework, injetar essa funcionalidade na aplicação.

[16] https://unform.dev/
[17] https://www.npmjs.com/package/yup
[18] https://sweetalert2.github.io/

Em Vue: Assim como em Angular, os formulários anteriores junto ao uso da biblioteca VeeValidate auxiliou na implementação das páginas de cadastro de promoção e admin, possuindo alteração apenas das estruturas para o framework correspondente. Como visto na tecnologia anterior, Vue também já vem com uma biblioteca de gerenciamento de rotas instalada, o que facilita na implementação e estruturação dos componentes com suas respectivas rotas. Para o desenvolvedor, Vue pode ser eleita como a tecnologia mais fácil na implementação dessa US.

Em React: O desenvolvimento dos componentes em React apresentaram diferenças estruturais apenas na implementação do jsx e no uso de hooks como o useEffect que é responsável por ser executado toda vez que uma renderização estiver disponível em tela. Em contrapartida, a configuração das rotas em React foi feita do zero pelo desenvolvedor, que utilizou a biblioteca React-Router-Dom para criar funções básicas que fornecem a estruturação e fazem uso dos princípios de SPA. Do ponto de vista de desenvolvimento, o fato de não possuir uma implementação pronta atrasou o tempo de implementação, mas também fez com que o desenvolvedor se aproximasse do Javascript puro, o que acarreta em algumas vantagens para a tecnologia.

4.4 US 4

A quarta *User Story* corresponde ao desenvolvimento da página de aprovação de admin, que utiliza o conceito de rotas filhas para o acesso de um rota específica. Junto à criação de componentes básicos como *Like*, que faz o uso do *EventEmitter* que é uma abstração de Angular que permite emitir eventos personalizados e o componente de *Back* que auxilia na UI do usuário que acessa o PWA em dispositivos moveis.

Em Angular: O desenvolvimento da página de aprovação de admin foi mais uma vez facilitado pelos formulários já existentes, havendo modificações apenas na rota proposta. Foi utilizado o id da promoção como parâmetro da rota que pode ser pego graças a classe ActivatedRoute que permite essa interpolação entre componentes e o compartilhamento de parâmetros. Os componentes de *Like* e *Back* foram feitos usando ícones do Material Design [19], sendo o primeiro implementado com uso de props que permitem o compartilhamento de informações entre um componente pai e filho junto ao uso de eventos.

Em Vue: Assim como em Angular, os formulários anteriores junto ao uso da biblioteca VeeValidate facilitou a implementação das páginas propostas. Nesse framework, também foi utilizado o conceitos de props e eventos para a implementação do componente *Like*, enquanto o componente de *Back* foi criado com a mesma estruturação, se diferenciando apenas na sua paleta de cores.

Em React: O desenvolvimento dos componentes em React apresentou diferenças estruturais apenas na implementação do JSX e no uso de hooks. A implementação dos componentes de Like foi um pouco mais complexa porque em React só foi possível fazer uso das props nativamente, não havendo o conceito de eventos que os frameworks anteriores disponibilizavam. Para contornar a situação, foi utilizado o hook useContext para realizar o gerenciamento de estado do componente de *Like* que corresponde a promoção apresentada. Por fim, o componente de *Back* também só teve alteração na sua paleta de cores.

4.5 US 5

A quinta *User Story* corresponde ao desenvolvimento dos *services* que são arquivos Javascript utilizados para organizar e compartilhar estados de objetos e as regras de negócio da aplicação, junto aos *pipes* que são funções que se comportam como filtros na transformação e formatação dos dados modelo da aplicação.

Em Angular: Por possuir uma biblioteca de requisições HTTP (@angular/common/http), o desenvolvimento em Angular foi considerado o mais simples pelo desenvolvedor já que apenas foram criados os métodos necessários para a comunicação com a API, sendo papel do framework oferecer todo o tratamento e estrutura necessárias para o tratamento das respostas. A implementação de filtros em Angular também foi facilitada pela biblioteca @angular/-core que permite o desenvolvimento e o uso desse pipe no contexto global da aplicação.

Em Vue: Como Vue não possui nativamente um biblioteca para realizar requisições, e como o desenvolvedor optou por não utilizar funções nativas em Javascript, foi instalada a biblioteca axios[20] que representa um cliente HTTP baseado em *promises* para fazer requisições que possue vantagens como intercepção de requisições e respostas e a transformação dos dados em JSON automaticamente, auxiliando assim na estruturação dos serviços da aplicação. Por fim, a implementação dos filtros em Vue segue uma estrutura de arquivo único que é declarado no main da aplicação e é disponibilizado em contexto geral.

Em React: Assim como Vue, React não possui uma biblioteca nativa e foi reutilizada o axios, permitindo o reuso das funções básicas. Já os filtros foram criados como funções em arquivos js, que são chamados no jsx. Pela visão do desenvolvedor, React foi o mais complexo na implementação de tal funcionalidade já que apenas utilizou o Javascript puro, não possuindo nenhum auxílio da biblioteca para tais funcionalidades.

4.6 US 6

A sexta e última *User Story* corresponde aos toques finais da aplicação Web para a sua transformação em um PWA. A partir da configuração do arquivo *manifest* junto a configuração de uso do *Service Worker*.

Em Angular: Mesmo possuindo uma biblioteca própria junto a exemplos em sua documentação, a implementação da aplicação em Angular durou 3 horas e 44 minutos, se tornando a tecnologia mais demorada nesse processo. O baixo conhecimento do desenvolvedor acarretou nas buscas em fóruns e vídeos por possíveis soluções para a implementação do serviço. No primeiro momento, foram encontradas soluções vagas e pouco detalhadas para a configuração do arquivo principal, assim tomando muito tempo de desenvolvimento. Além disso, a estrutura oferecida pelo arquivo ngsw-config se tornou confusa na declaração das urls que correspondem a API que está sendo consumida pelo PWA. Por fim, após várias pesquisas, o desenvolvedor conseguiu encontrar uma solução e implementar o método, realizando posteriormente testes básicos para a verificação

[19] https://material.io/

[20] https://axios-http.com

dos conceitos do PWA, obtendo sucesso e se destacando entre as tecnologias comparadas.

Em Vue: Possui uma documentação oficial escassa em relação a construção de uma aplicação Web, utilizando a metodologia PWA. O pouco de base oferecida nas buscas durante a implantação do SW em Angular fez com que o desenvolvedor buscasse em vídeos e fóruns a possível configuração exigida para o funcionamento dos métodos básicos propostos por um PWA. A solução foi encontrada em um vídeo no Youtube, que utilizou a biblioteca Workbox[21] que faz uso de um conjunto de módulos do node que simplifica o processo de cache em uma aplicação, assim agilizando o trabalho na criação de uma Progressive Web App. Por fim, não foi possível configurar as telas iniciais (*splash screen*) que são apresentadas no primeiro instante que um aplicativo é iniciado.

Em React: Assim como em Vue, React não possui uma documentação oficial muito animadora quanto a configuração dos arquivos *manifest* e SW. Porém na instalação da biblioteca já é possível utilizar as funções do Workbox. Sendo assim, o desenvolvedor utilizou dos conhecimentos garantidos em Vue junto a pesquisas na documentação do Workbox para realizar a configuração do service worker da aplicação em React, que apresentou as mesmas limitações de Vue.

4.7 Analises com o Lighthouse

Nesta seção, serão apresentados e analisados os resultados dos relatórios gerados pelo Lighthouse.

O Lighthouse é uma ferramenta automatizada de código aberto que busca aprimorar a qualidade de APPs da web. Para utilizar a ferramenta, o desenvolvedor pode adicionar uma extensão do Google Chrome ou instalar a ferramenta via NodeJs, utilizando o comando *Lighthouse sua-url* para que a mesma realize uma série de testes na página e gere um relatório final que apresenta métricas como desempenho, acessibilidade, boas práticas, otimização para mecanismos de busca (SEO) e recursos de um PWA. As pontuações dessas métricas são classificadas de acordo com estes intervalos: 0 a 49 (Vermelho) Ruim; 50 a 89 (Amarelo) Precisa de melhorias; 90 a 100 (Verde) Bom. Abaixo serão detalhadas as métricas utilizadas pelo Lighthouse:

- **Desempenho**: Primeira métrica a ser analisada, utilizada indicadores como: Índices de velocidade, como a rapidez que um conteúdo é preenchido de forma visível na pagina; Tempo de interação, que mede a quantidade de tempo para que a página se torne totalmente interativa; e mudança de layout cumulativa, que mede o movimento dos elementos visíveis na janela de exibição.
- **Acessibilidade**: A métrica de acessibilidade analisa a construção da página e busca por melhorias na acessibilidade da aplicação. Utiliza um subconjunto de indicadores como: Botões com títulos acessíveis, imagens que possuam o atributo alt (texto alternativo que é utilizado para descrever o conteúdo de uma imagem), elemento HTML que possui um valor no lang (atributo que ajuda a definir o idioma de um elemento) e analisa se os elementos aparecem em ordem decrescente sequencial no documento.
- **Boas práticas**: Essa métrica busca verificar se a aplicação está seguindo as boas praticas de desenvolvimento web, analisando pontos como o uso de HTTPS, o não uso de APIs obsoletas, a exibição de imagens em proporção e resolução adequadas e a verificação do não uso de bibliotecas frontend com vulnerabilidades de segurança conhecidas.
- **SEO**: Essa métrica busca garantir que a aplicação esteja seguindo os conselhos básicos de otimização para os mecanismo de busca. Para realizar a pontuação, utiliza indicadores como: Links rastreáveis, textos com fontes legíveis, o não uso de *plugins* e a verificação das dimensões dos elementos interativos, para que sejam fáceis ao toque das telas touch.
- **PWA**: A ultima métrica busca verificar os aspectos que validam uma aplicação web como um Progressive Web App. Analisando pontos como: A existência de um arquivo manifest junto a um service worker que atenda aos requisitos de instabilidade; Redirecionamento do trafego HTTP para HTTPS e por fim verifica se o conteúdo está dimensionado e otimizado corretamente para dispositivos moveis.

Por possuir uma experiência mediana com desenvolvimento Web/PWA o programador utilizou o Lighthouse para auditar a página das 3 aplicações na busca por indicadores que possam medir algumas métricas para possíveis aprimoramentos do aplicativo. Os resultados gerados no relatório explanaram problemas de implementação junto ao desempenho de cada tecnologia, que foi desenvolvida com as mesmas funcionalidades e emuladas a partir de um dispositivo móvel. Os resultados gerados serão discutidos a seguir:

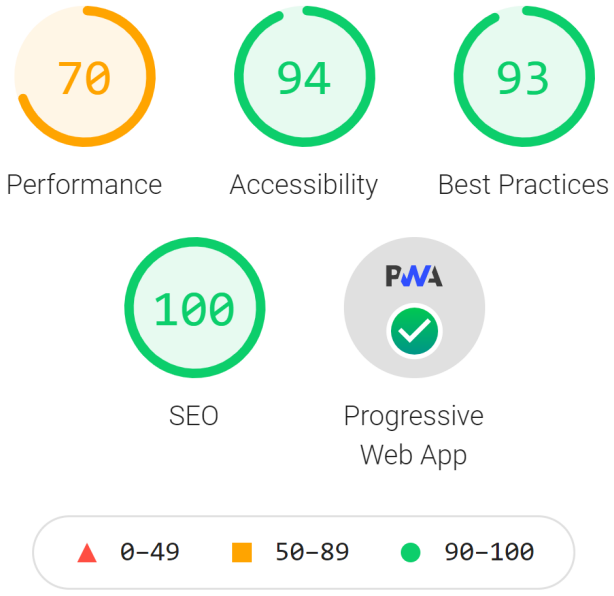

Figura 5: Resultados do Lighthouse da aplicação em Angular

Em Angular[22]: Como visto na Figura 5, Angular obteve um desempenho de 70 pontos, sendo configurada como a aplicação que apresentou o menor desempenho entre as três tecnologias. Entre

[21] https://developers.google.com/web/tools/workbox

[22] https://bit.ly/3hE0zhB

as falhas de implementação foram encontrados índices de velocidade, interação e exibição de conteúdos com um tempo médio de 2 segundos maiores que o normal. Além disso, para melhorias de desempenho foi recomendado pela ferramenta a redução de CSS e Javascript não utilizados. No quesito acessibilidade, a implementação em Angular falhou apenas no baixo contraste entre as cores de fundo e o primeiro plano, já que essa diferença de cores pode dificultar a leitura dos conteúdos por boa parte dos usuários. Na métrica de boas praticas foram apresentadas como falhas, erros não resolvidos no console e a falta de mapas de origem primaria em arquivos Javascript. Por fim, a implementação em Angular recebeu nota máxima (100) nas verificações de SEO junto aos aspectos de uso do PWA.

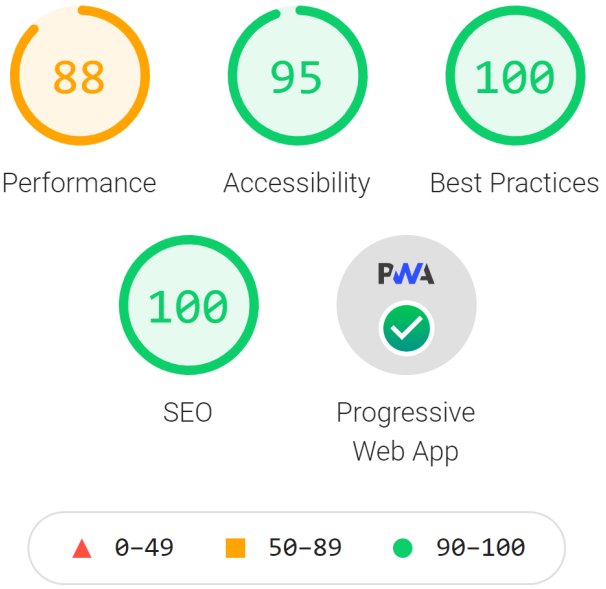

Figura 6: Resultados do Lighthouse da aplicação em Vue

Em Vue[23]: Como visto na Figura 6, Vue ocupou o primeiro lugar em desempenho, entre as 3 aplicações, obtendo 88 pontos de performance. Acabou não atingindo a nota máxima por apresentar falhas de desempenho apenas na exibição do primeiro e maior conteúdo (texto ou imagem) junto ao pedido de redução do Javascript não utilizado. Na métrica de acessibilidade, Vue obteve 95 pontos, apresentado o mesmo problema de legibilidade do contraste entre as cores padrões e de fundo. Por fim, garantiu a nota máxima (100) nas categorias de boas praticas, SEO e verificações das validades do PWA. Se consolidando como a implementação com o melhor resultado, pelo Lighthouse.

Em React[24]: Como visto na Figura 7, React ocupou o pódio com segundo melhor desempenho (83 pontos). Na métrica de desempenho foram apresentados problemas no índice de velocidade que mostra o conteúdo visível, mudança de layout cumulativa e na redução do Javascript não utilizado. Em acessibilidade, React apresentou o mesmo problema do contraste que as implementações anteriores. Ademais, ganhou nota máxima em boas praticas, mas perdeu

[23]https://bit.ly/3EittO2
[24]https://bit.ly/3nwav0u

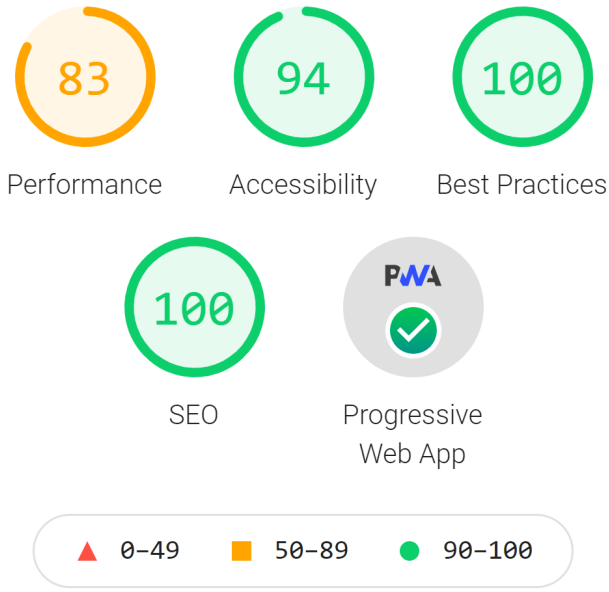

Figura 7: Resultados do Lighthouse da aplicação em React

8 pontos em SEO por não permitir o rastreamento por páginas de pesquisa. Por fim, React cumpriu todos os requisitos para ser configurado como uma aplicação PWA.

4.8 Adoção entre Android e iOS

Sendo considerado um navegador importante para os dispositivos Apple, o Safari se torna ainda mais relevante com seu motor de renderização chamado de WebKit. O Webkit é utilizado por todos os navegadores do ecossistema Apple a exemplo do Chrome. Como vantagens desse processo, existem a facilidade de testes junto a uma experiência consistente do usuário, independente do navegador utilizado. Entretanto, como consequência se tem a desatualização de todos os navegadores do sistema se o WebKit não implementar alguma funcionalidade durante o lançamento de uma nova versão do iOS.

Aplicações PWA começaram a ter suporte em iPhones apenas a partir do iOS 11.3, lançado em 2018. E mesmo ainda não possuindo funcionalidades como notificações ou pop-ups de instalação, os dispositivos Apple implementam a maioria dos seus recursos básicos. Porém, o fato do usuário ficar retido ao navegador Safari para a instalação na tela inicial junto a falta de um holofote da empresa para uma maior adoção dessa metodologia em seu ecossistema, faz com que o desempenho de PWAs no iOS seja inferior ao Android.

Enquanto isso, os dispositivos Android oferecem todo o leque de possibilidades que um PWA dispõe como: Tela cheia, notificações nativas, banners de instalação em qualquer navegador e acesso a mais recursos de hardware como bluetooth e controles de câmera. Como conclusão, pode ser ressaltar que a falta de notificações em dispositivos iOS é um argumento que pode ser utilizado para a vitória do Android. Porém, o aumento dos recursos e o suporte oferecido por empresas como Google, Apple e Microsoft ao longo dos anos vem priorizando e melhorando, mais do que nunca, os mecanismos do PWA. Por fim, os sistemas operacionais de ambas

empresas implementaram e permitiram o uso, de maneira igualitária, das funcionalidades básicas do PWA oferecidas pelo projeto desenvolvido.

5 GUIDELINES PARA DESENVOLVIMENTO DE PWA

Por se tratar de um problema multiobjetivo e pelas tecnologias estudadas apresentarem o mesmo paradigma, não foi possível a escolha de uma única tecnologia como vencedora do estudo comparativo. Contudo, foi observado que a escolha do desenvolvedor depende diretamente das características especiais de uma tecnologia junto a um *guideline* definido pelo escopo do sistema que será implementado. Abaixo serão apresentadas as vantagens de cada tecnologia de acordo com os *guidelines* que cada projeto possa seguir.

Foi analisado, pelo desenvolvedor, que para a implementação de um projeto corporativo que utilize soluções prontas na busca por produtividade, uma maior escalabilidade oferecida pelo TypeScrypt junto a um sistema de rotas, filtros, diretivas, eventos e formulários reativos bem estruturados e conectados, é indicado o uso de Angular. O framework oferece bibliotecas nativas que irão auxiliar o desenvolvedor na montagem de componentes, sem necessitar de bibliotecas externas, já que as estruturas oferecidas pela mesma auxiliam na criação de formulários reativos, validação e chamadas HTTP. Em contrapartida, Angular peca na falta de uma biblioteca de gerenciamento de estado ficando como responsabilidade para o desenvolvedor aprender a biblioteca rxjs, sendo essa o mais próximo de um contexto Global oferecido por Angular. Ademais, por ser um framework completo e possuir um maior pacote de arquivos, seu build foi o de maior tamanho entre as 3 tecnologias e seu desempenho foi o menor entre as aplicações analisadas pelo Lighthouse. Por fim, na visão do programador, Angular apresenta ser uma boa alternativa para o primeiro contato com o desenvolvimento web frontend, para aqueles que possuem um boa base de orientação a objetos e também por possuir toda uma estrutura encapsulada que oferece quase todas as funcionalidades básicas junto a uma grande comunidade que auxilia o desenvolvedor, mas camufla o fato do framework possuir uma documentação extensa e não tão amigável.

Todavia, se o desenvolvedor está buscando implementar uma aplicação que seja adaptável, utilize o melhor dos dois mundos (Angular e React) e que foque na User Interface junto ao uso de bibliotecas externas que utilizam um maior encapsulamento, facilidade de manutenção, reuso de código e componentes mais acoplados é indicado o uso do framework Vue. Ademais, algumas outras vantagens de Vue estão em seus arquivos únicos que permitem uma maior legibilidade de código, o uso de diretivas para manipulação da DOM e o fato de ser bem reconhecida por boa parte do mercado. Contudo, a principal vantagem para escolha de VUE está na biblioteca de estados VUEX, que irá auxiliar o desenvolvedor no gerenciamento de estado de toda a aplicação já que a mesma permite que vários componentes compartilhem um estado comum. Por fim, Vue possui uma boa comunidade que auxilia na resolução de dúvidas, seu código se caracterizou como o mais legível entre as 3 aplicações, além de possuir a menor, mais direta e detalhada documentação entre as três tecnologias.

Por fim, se o sistema que será implementado busca um desenvolvimento mais próximo do Javascript com componentes compartilháveis e personalizáveis junto ao uso de bibliotecas e pacotes externos, com menor LOC, menor tamanho de arquivos e uma fortalecida comunidade que auxilie na resolução de duvidas, sugerimos ao desenvolvedor utilizar o React. As vantagens da arquitetura própria (Flux) junto ao uso de componentes funcionais e os React Hooks, vieram para estabelecer novas regras no desenvolvimento web. Trazendo um código fácil, legível e rápido de ser implementado sem perder a essência do Javascript puro. Além disso, o build e os componentes em si apresentaram o menor tamanho entre a tecnologias estudadas. Ademais, o uso de hooks como o *useContext* que permite criar um contexto geral para a aplicação e auxiliar no compartilhamento de estado entre componentes, sem o uso de uma biblioteca externa, trás pontos positivos para o uso de React. Entretanto, mesmo sendo uma biblioteca que está em alta no mercado, seu domínio exige do programador um bom conhecimento do Javascript puro o que poderia causar divisões na implementação e na escolha da tecnologia.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresenta um estudo comparativo entre 2 frameworks (Angular e Vue) junto a biblioteca React para a criação de um sistema de promoções que consome uma API Rest e fornece funções básicas de um aplicativo PWA como cadastro, autenticação, postagem e deleção. O objetivo do estudo foi realizar uma comparação sob pontos como arquitetura, facilidade de aprendizado, tamanho de código e tempo de implementação, em cada uma das tecnologias, com o intuito de guiar desenvolvedores WEB na escolha mais assertiva de uma tecnologia para o desenvolvimento de um PWA.

Como solução do estudo, foram criados *guidelines* a partir dos resultados obtidos com intuito de auxiliar os desenvolvedores na escolha de uma tecnologia que busque um bom desempenho a partir de pontos como menor LOC, maior encapsulamento, menor tempo de implementação e melhor comunidade e documentação. Prezando pelo apego do desenvolvedor aos conceitos que elas trazem e os problemas que elas resolvem, ao invés da tecnologia, já que uma boa base do Javascript irá auxiliar o programador a utilizar qualquer framework atual e futuro.

De modo geral, espera-se que as observações realizadas possam auxiliar, em trabalhos futuros, na construção de um guia definitivo para programadores PWA/Web, apresentando detalhadamente funções, vantagens e desvantagens das tecnologias citadas junto a outras tecnologias e métricas. Visando um aprofundamento do estudo proposto, junto a abertura de diálogos para novas discussões e comparações.

REFERÊNCIAS

[1] 2020. *Framework: saiba como usar e quais são os mais populares.* Technical Report. Retrieved 2021/08/21, from https://blog.revelo.com.br/o-que-e-framework-exemplos-e-aplicacoes/.

[2] 2020. *TIOBE Index. Very Long Term History.* Technical Report. Retrieved 2021/08/29, from https://www.tiobe.com/tiobe-index/programming-languages-definition/.

[3] Ana Antunes. 2020. *PWA: O que são Progressive Web Apps e por que usar?* Technical Report. Retrieved 2021/08/24, from https://gobacklog.com/blog/progressive-web-apps/.

[4] Panorama Mobile Time/Opinion Box. 2020. Uso de Apps no Brasil. Dynatrace, 5–12.

[5] Centro Regional de Estudos para o Desenvolvimento da Sociedade da Informação. 2020. Pesquisa sobre o Uso das Tecnologias de Informação e Comunicação nos Domicílios Brasileiros – TIC Domicílios 2019. Cetic.br, 9.
[6] Joseph Johnson. 2021. Global digital population as of January 2021 (in billions). *Statista Research Analysis* (2021).
[7] Stack Overflow. 2020. *2020 Developer Survey.* Technical Report. Retrieved 2021/09/05, https://insights.stackoverflow.com/survey/2020most-popular-technologies.
[8] Alexandre Plennevaux. 2017. *The easy way to turn a website into a Progressive Web App.* Technical Report. Retrieved 2021/10/21 https://dev.to/becodeorg/the-easy-way-to-turn-a-website-into-a-progressive-web-app-77g.
[9] Ugo Roveda. 2020. *JavaScript: o que é, para que serve e como funciona o JS?.* Technical Report. Retrieved 2020/10/28, from https://kenzie.com.br/blog/javascript/.
[10] Alex Russell. 2015. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul.* Technical Report. Retrieved 2021/08/21, from https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.
[11] StateOfJS. 2018. *The State of JavaScript.* Technical Report. Retrieved 2021/08/21, from https://2018.stateofjs.com/front-end-frameworks/overview/.