Progressive Web Apps: Uma nova abordagem no desenvolvimento de aplicações Web

Vinícius Mazzarolo[1], Roger Sá da Silva[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - *Campus* Avançado Veranópolis – Veranópolis – RS – Brasil

vinimazzarolo1@outlook.com, roger.silva@veranopolis.ifrs.edu.br

Resumo. *Os Progressive Web Apps (PWAs) surgem como uma nova forma de unificar o desenvolvimento e a distribuição de aplicativos. Essa nova proposta tem o potencial de amenizar o problema da necessidade de desenvolver um projeto para cada plataforma. Um dos principais questionamentos em relação aos PWAs refere-se à sua semelhança com aplicações nativas e se oferece uma experiência de utilização semelhante a estas. Diante desse contexto, o presente trabalho objetiva desenvolver um PWA e analisar o seu comportamento a partir do desenvolvimento de uma aplicação protótipo, ao implementar as principais funcionalidades disponíveis em sua arquitetura e verificar a viabilidade de utilização de um PWA em alternativa ao desenvolvimento de uma aplicação nativa. Para tanto, por meio do método de prototipação exploratório, utilizou-se o framework VueJS em conjunto com a biblioteca CapacitorJS para a construção da aplicação, validando-a nos ambientes Windows e Android. O protótipo desenvolvido foi capaz de utilizar os recursos dos dispositivos por meio dos navegadores. Portanto, mostra-se uma alternativa viável a aplicações nativas, uma vez que a experiência de utilização do usuário no PWA é bastante similar à de uma aplicação nativa.*

Abstract. *Progressive Web Apps (PWAs) emerge as a new way to unify application development and distribution. This new proposal has potential to mitigate the requirement to develop a project for each platform. One of the main questions regarding PWAs refers to its similarity to native applications and offers a similar user experience to these. Given this context, this work aims to develop a PWA and analyze its behavior from the development of a prototype application, by implementing main features available in its architecture and verifying feasibility of using a PWA as an alternative to the development of a native application. For this purpose, through exploratory prototyping method, VueJS framework was used together with CapacitorJS library to build the application, validating it in Windows and Android environments. The developed prototype was able to use the resources of devices through browsers. Thus, it is a viable alternative to native applications, since the user experience in PWA is very similar to a native application.*

1. Introdução

Os *Progressive Web Apps* (PWAs) surgem como uma nova forma de unificar o desenvolvimento e a distribuição de aplicativos, podendo ser utilizado pelos principais dispositivos que disponham de um *Web Browser*. Por muito tempo, determinadas funcionalidades eram acessíveis exclusivamente a partir de aplicações nativas, aquelas desenvolvidas na mesma linguagem de programação operante pelo sistema operacional. Com o advento dos PWAs em 2015, proposto pela Google, esse contexto passou por mudanças significativas.

Faz-se interessante analisar essa nova proposta de desenvolvimento, uma vez que

a unificação de desenvolvimento e distribuição pode amenizar um problema que é encontrado em cenários de aplicações nativas: a necessidade de desenvolver um projeto para cada plataforma. Como aponta Charland e LeRoux (2011) construir um aplicativo para cada plataforma é custoso e nem mesmo as gigantes do mercado de tecnologia são capazes de suportar todas as plataformas. Cada sistema operacional (Android, iOS, Windows, etc.) exige uma linguagem de programação específica, fazendo, assim, com que as empresas necessitem de profissionais especializados para cada plataforma.

Um dos principais questionamentos em relação aos PWAs, refere-se à sua semelhança com aplicações nativas. De acordo com Fortunato e Bernardino (2018), um PWA deve oferecer uma experiência de utilização semelhante a uma aplicação nativa. A intenção é que um usuário comum não tenha a capacidade de distinguir entre uma aplicação nativa ou web.

Trabalhos recentes sobre PWAs abordam em sua maioria a implementação e desenvolvimento de aplicações voltadas a cenários específicos, utilizando um PWA como a solução técnica, tais como: um sistema para monitorar a produção de óleo, de forma inteligente, utilizado pelos colaboradores da empresa [Nugroho et al. 2017]; uma aplicação no contexto educativo, que torna o processo de e-learning mais interativo para os alunos e que oferece as praticidades de um PWA [Wijaya e Abbas 2018]; e por fim, em caráter de comparação, abordou-se duas pesquisas distintas, possuindo métricas de desempenho, que possibilitam analisar qual abordagem apresenta melhor performance. [Linden 2020][Fransson e Driaguine 2017].

Nesse cenário, o presente trabalho objetiva desenvolver um PWA e analisar o seu comportamento a partir do desenvolvimento de uma aplicação protótipo. Mais especificamente, pretende-se implementar as principais funcionalidades disponíveis em sua arquitetura, tais como: sincronização *offline*, instalação, geolocalização, acesso à câmera, *notificações push* e *service workers*. Ainda, busca-se verificar a viabilidade de utilização de um PWA em alternativa ao desenvolvimento de uma aplicação nativa em ambientes Windows e Android.

O restante do trabalho está estruturado da seguinte forma: na seção 2, encontra-se a revisão da literatura, que sistematiza os principais conceitos e trabalhos relacionados sobre o tema proposto. Na sequência, na seção 3, são elencados os métodos aplicados no decorrer do trabalho. Por fim, na seção 4, são apresentados e discutidos os resultados obtidos durante o desenvolvimento e, na seção 5, são apresentadas as considerações finais.

2. Revisão da Literatura

O conceito de *Progressive Web Apps* (PWAs) teve início em 2015, apresentado oficialmente pela Google. Um PWA unifica a experiência de navegação em *laptops*, *tablets*, entre outros dispositivos que possuem variação nas dimensões de tela, pois seu *layout* comporta-se de maneira responsiva adequando-se ao espaço disponível em tela. Quanto ao seu acesso, não é necessário realizar a instalação nos *marketplaces* (App Store, Google Play, Microsoft Store, etc) para disponibilizar um PWA aos usuários, a instalação pode ocorrer por meio do navegador [Sharma et al. 2019].

Outra mudança causada por esse tipo de aplicação é a diminuição da necessidade

de mão de obra especializada em diferentes plataformas para o desenvolvimento nativo em cada uma delas. Em um cenário no qual necessita-se produzir uma versão para Android e iOS, sem a utilização do conceito de PWA, o resultado final será duas aplicações implementadas em linguagens de programação distintas, mas que executam as mesmas tarefas. Os PWAs surgem como uma solução para esse tipo de problema, unificando o desenvolvimento a partir da linguagem de programação Javascript, podendo ser utilizado multiplataforma posteriormente [Sharma et al. 2019].

Esse conceito também busca se beneficiar das vantagens de utilizar o que está disponível em determinado ambiente, ou seja, não é necessário que haja requisitos fixos: o PWA é capaz de manter o funcionamento geral da aplicação, com ou sem determinadas funcionalidades, conforme o ambiente. O termo "progressivo" é oriundo dessa capacidade do PWA de progressivamente se adaptar às características e funcionalidades de cada plataforma ou navegador, podendo vir a se tornar uma aplicação, ou ser apenas uma página *Web* [Tjarco 2019 apud Russel 2016].

Como encontrado na documentação da [MDN Web Docs 2021] os PWAs têm a capacidade de trabalhar com as APIs, originalmente exclusivas de aplicações nativas, a partir do navegador do usuário que está nativamente instalado no dispositivo.

Para uma aplicação ser considerada um PWA ela deve trabalhar sob um contexto seguro (utilizando o protocolo HTTPS), dispor de um ou mais *service workers* e conter um arquivo *manifest*. Nas subseções a seguir cada uma destas características são melhor exploradas. Abaixo, pode-se observar outras características que são de suma importância para uma aplicação ser classificada como um PWA [MDN Web Docs 2021]:

- Detectável - A aplicação deve ser encontrada em mecanismos de busca;
- Instalável - Para que possa estar disponível na tela inicial dos dispositivos;
- Compartilhável - Há a possibilidade de compartilhá-lo através de uma *url*;
- Independente de conexão - Deste modo, pode funcionar *offline*;
- Progressivo - Pode ser usado em um nível básico em navegadores mais antigos, mas totalmente funcional nos mais recentes;
- Interatividade - Para que possa enviar notificações;
- Responsivo - Tem potencial para ser utilizado em qualquer dispositivo com tela e navegador;
- Seguro - As conexões entre o usuário, o aplicativo e seu servidor são protegidos contra quaisquer terceiros que tentem obter acesso a dados confidenciais.

2.1. HTTPS

O protocolo HTTPS criptografa a transmissão de dados que ocorre na internet, entre cliente e servidor, através de um certificado de segurança SSL/TLS, instalado no servidor, de forma que a interação não possa ser vista por terceiros, pois as atividades permanecerão privadas entre o navegador e o servidor [Rodriguez 2018].

Pode-se dizer que o HTTPS é um ponto-chave para o fluxo de trabalho de um PWA, pois ele opera em conjunto com o mecanismo de permissões, que é solicitado sempre que uma API recebe uma chamada para ser utilizada [Kayce 2021].

O aplicativo da *web* deve ser servido por uma rede segura. Um site seguro não é

apenas uma prática recomendada, mas também estabelece o aplicativo *web* como um sistema confiável, especialmente se os usuários precisarem fazer transações. A maioria dos recursos relacionados a um PWA, como geolocalização e *service workers*, estão disponíveis apenas depois que o aplicativo é carregado usando HTTPS [MDN Web Docs 2021].

Portanto, considera-se o HTTPS um recurso fundamental, oferecendo segurança para os utilizadores e exigindo que os desenvolvedores sigam padrões modernos de desenvolvimento, ao mesmo tempo que é condição necessária para que o navegador possa se comunicar com funções nativas do sistema operacional.

2.2. *Service Worker*

Um *service worker* é, em essência, um arquivo Javascript que executa em paralelo ao navegador. Ele controla como as requisições de rede são gerenciadas, interceptando-as e, além disso, incluindo funcionalidades como notificações *push*, sincronização *offline*, dentre outras [Pande et al. 2018]. O *service worker* funciona da mesma forma que uma API, de forma assíncrona e orientado a eventos, por isso tem a capacidade de operar em segundo plano [Tjarco 2019].

Quando um *service worker* intercepta uma requisição realizada pela página, ele dispara um evento *fetch*, retornando uma resposta do servidor, ou do *cache* armazenado localmente. Portanto, por meio desse artifício, torna-se possível acessar os recursos nativos do sistema operacional em questão.

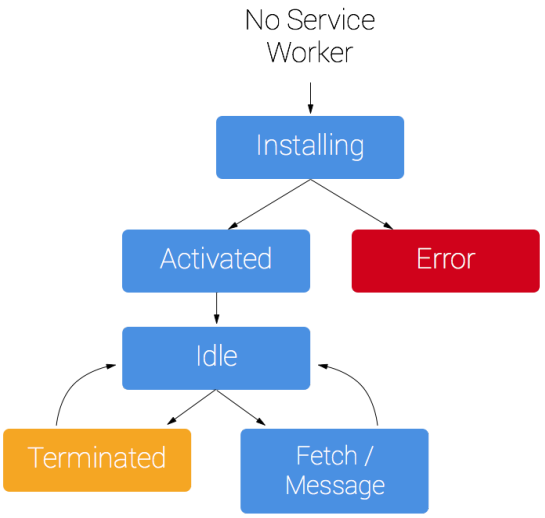

Figura 1. Ciclo de vida de um *service worker* [Gaunt 2021].

Conforme a Figura 1, pode-se observar o ciclo de vida de um *service worker.* Como visto anteriormente, entende-se que ele responde a eventos assíncronos, o que significa que sua execução ocorre em segundo plano, de acordo com o estado do *service worker*. O primeiro evento executado, ao acessar uma página *web*, será a sua instalação. Cada página pode instalar o seu próprio *service worker*. Se o seu código fonte estiver diferente, um novo será instalado em seu lugar. Após, em caso de sucesso na instalação, será realizada sua ativação. Ele será ativado se não houver nenhum outro ativo ou se a página for atualizada. Após, o *service worker* entrará em espera, escutando por eventos

(*fetch, sync, push)* que serão disparados pelos usuários. Percebe-se que nos processos finais, ele pode assumir dois estados: respondendo por requisições, ou encerrado, para economizar memória, sendo este um mecanismo de gerenciamento próprio, em situações onde ele não está sendo requisitado [Gaunt 2021].

2.3. Arquivo *manifest*

O arquivo *manifest* estabelece as configurações fundamentais e iniciais do PWA. Como apresentado pela MDN Web Docs (2021), é um arquivo no formato JavaScript Object Notation (JSON). Através dele, algumas configurações são pré-estabelecidas e o navegador obtém a informação de que a página em questão trata-se de um PWA, fornecendo todas as informações e recursos relacionados ao aplicativo, adaptando-se também aos sistemas operacionais. Neste arquivo configura-se o nome, a *url* da página inicial, os ícones, o modo de exibição, o logotipo, a *splash screen*, entre outros detalhes, que são necessários para "transformar" o site em um modelo de aplicativo [Břoušek 2017]. Em síntese, o navegador baseia-se na leitura desse arquivo para ter a informação se determinado site trata-se de um PWA.

2.4. Relação com os navegadores

Dado que um PWA é executado sob a custódia de um navegador e baseia-se nas APIs oferecidas por estes *softwares*, faz-se interessante analisar os principais navegadores em uso atualmente. Segundo levantamento da Statcounter Global Stats (2021), a partir do *market share* (fração do mercado que o navegador ocupa) no Brasil, destacam-se os seguintes navegadores e suas respectivas frações de mercado (dados de setembro de 2021):

- Google Chrome - 80,40%;
- Safari - 7,01%;
- Microsoft Edge - 3,15%;
- Opera - 3,06%;
- Samsung Internet - 3,00%.

Na Tabela 1 pode-se observar a relação entre estes cinco principais navegadores e a sua integração com algumas das APIs emergentes passíveis de serem exploradas pelos PWAs. Posteriormente, na seção de métodos, são abordados mais detalhes técnicos referentes a cada recurso.

É possível constatar que o processo de adoção das APIs pelos navegadores, mesmo alguns anos após o conceito de PWA ser introduzido, ainda não é completo, conforme mostra a Tabela 1. Por outro lado, com a possível ampliação de usuários e desenvolvedores que atuam no contexto dos PWAs, percebe-se que algumas organizações estão mais à frente na integração das APIs, impondo sua posição na evolução e discussão desse conjunto de tecnologias, sendo capaz de atingir mais pessoas com seu produto e/ou serviço [Can I Use 2021].

Tabela 1. Compatibilidade das APIs entre os navegadores. [Can I Use 2021]

	Chrome	Safari	Edge	Opera	Samsung Internet
Service Workers	X	X	X	X	X
Câmera	X	X	X	X	X
Geolocalização	X	X	X	X	X
Push	X		X	X	X
Background Sync	X		X	X	X
A2HS	X		X		X

2.5. Estado da arte e trabalhos relacionados

No primeiro trabalho, desenvolveu-se uma solução baseada em PWA para aplicar os conceitos de *smart farming*, uma maneira inteligente de produção agrícola que busca utilizar conceitos de IoT (Internet das Coisas), aplicando um monitoramento em tempo real na propriedade rural. A principal dificuldade era oferecer uma interface de interação para os usuários, uma vez que os dados já estavam à disposição, mas só podiam ser acessados pelos desenvolvedores. Considerando este problema, foi desenvolvido um sistema de informação, utilizando uma abordagem em PWA, para que os colaboradores pudessem acessar a aplicação tanto no dispositivo móvel quanto no *desktop*, oferecendo uma experiência de utilização nativa em ambos [Nugroho et al. 2017].

Em outro trabalho, desenvolveu-se uma solução para tornar o *e-learning* um processo de aprendizado mais amigável, através de animações e interações. Nesse contexto, partindo-se do princípio de que uma parcela expressiva de alunos não dispõe de muitos recursos de tecnologia e conexão, empregou-se um PWA para oferecer um produto que pode ser acessado com facilidade a partir de uma conexão precária com a internet e, mesmo assim, capaz de oferecer condições de utilização em um modo *offline*. Nesse caso, o PWA pode incorporar as inovações recentes no âmbito da *web* e de um aplicativo, podendo ser adicionado à tela inicial, funcionando em conexões de internet instáveis e entregando interação através de notificações [Wijaya e Abbas 2018].

Além destes, cabe destacar trabalhos de comparação de desempenho entre aplicações nativas e PWAs. Em um destes trabalhos realizou-se um estudo de desempenho no qual comparou-se o tempo de acesso às APIs entre um aplicativo nativo e um PWA. Os testes foram realizados em dois dispositivos distintos, porém, ambos executando o Android como sistema operacional. O estudo conclui que, no caso da API da câmera, o nativo apresenta vantagem no desempenho, sendo mais rápido, enquanto que na API de geolocalização, o PWA leva vantagem. Em relação às diferenças de desempenho, os autores apontam que no momento em que a pesquisa foi realizada, a API de câmera estava em versão experimental, o que pode explicar a diferença mais acentuada no desempenho. Enquanto que no caso da geolocalização, o recurso já existe há mais tempo, portanto se encontra em uma versão mais estável [Fransson e Driaguine 2017].

Em outro trabalho comparativo semelhante, utilizou-se a API de geolocalização em uma aplicação nativa e em um PWA a fim de determinar qual possui a melhor performance considerando-se o tempo para ser inicializada. O teste consistiu em inicializar o aplicativo trinta vezes. Ao final dos testes, o autor concluiu que a aplicação nativa tem um desempenho significativamente melhor em relação ao PWA, sendo que a maior latência do nativo foi 2950 milissegundos e o PWA 27700 milissegundos. Porém, observou-se que na aplicação nativa a variação de tempo, entre as 30 execuções, é maior do que em relação ao PWA, ocorrendo situações em que o PWA levou vantagem. Ainda sobre essa comparação, o autor conclui que é difícil determinar algo com os resultados, sendo que muitas vezes, eles são contraditórios, dadas as tecnologias recentes no contexto do PWA e as variações de tempo entre ambas as abordagens. [Linden 2020].

2.6. Prototipação

Como este trabalho se propõe a desenvolver um protótipo, cumpre abordar este tema nesta seção. Segundo Lichter (1994), os métodos de prototipagem são uma abordagem que afeta todo o processo de construção do *software*, baseado em uma visualização que pode ser exploratória, evolutiva ou experimental, envolvendo versões prematuras e experimentos até chegar ao produto final. É de importância também para estabelecer comunicação entre os envolvidos, pois oferece uma linguagem amigável entre usuários e desenvolvedores. O autor classifica três formas de prototipagem:

- Exploratória: Utilizado onde o problema não está bem definido, e as ideias iniciais são utilizadas como base para dar início ao processo;
- Experimental: A viabilidade técnica é discutida, os usuários e desenvolvedores comunicam-se, discutem os problemas e buscam alcançar uma solução;
- Evolutiva: É um processo contínuo e de adaptação, que evolui em conjunto com as novas versões do software, que são dadas a partir dos requisitos do usuário.

Em um contexto de prototipação, pode-se encontrar também a necessidade de validação do que está sendo desenvolvido. Os protótipos desenvolvidos para fins de elicitação de requisitos são usados posteriormente para validação desses requisitos. Assim, entende-se que, a partir da construção da solução, desencadeiam-se diversas alternativas observadas no decorrer do processo e que, posteriormente, poderão ser utilizadas a fim de validar o que foi produzido [Santos, et al. 2013].

3. Métodos

Para a implementação do PWA, de forma a possibilitar a análise de comportamento, utilizou-se o desenvolvimento de um protótipo segundo o método exploratório, dada a necessidade de se explorar, sem um escopo definido, as funcionalidades características destes aplicativos.

3.1. Modelagem

Nesse sentido, projetou-se um cenário no qual o usuário tem a necessidade de cadastrar um alerta em determinada geolocalização, permitindo-se informar um título para o alerta, uma descrição textual, uma imagem ilustrativa, a localização geográfica do acontecimento, e posteriormente, cadastrar o alerta. Em seguida, uma notificação *push* é

enviada para todos os demais usuários da aplicação que permitem receber notificações do aplicativo.

Com o intuito de representar as ações disponíveis na aplicação, utilizou-se um diagrama de caso de uso, conforme apresentado na Figura 2. Os diagramas de caso de uso são uma técnica diretamente ligada com a UML, que representa, em alto nível, os atores e suas interações com os componentes do sistema [Sommerville 2011]. Explorando-se as interações das entidades abordadas na Figura 2, é possível executar algumas das APIs emergentes dos navegadores, e consequentemente, observar seu comportamento em um PWA.

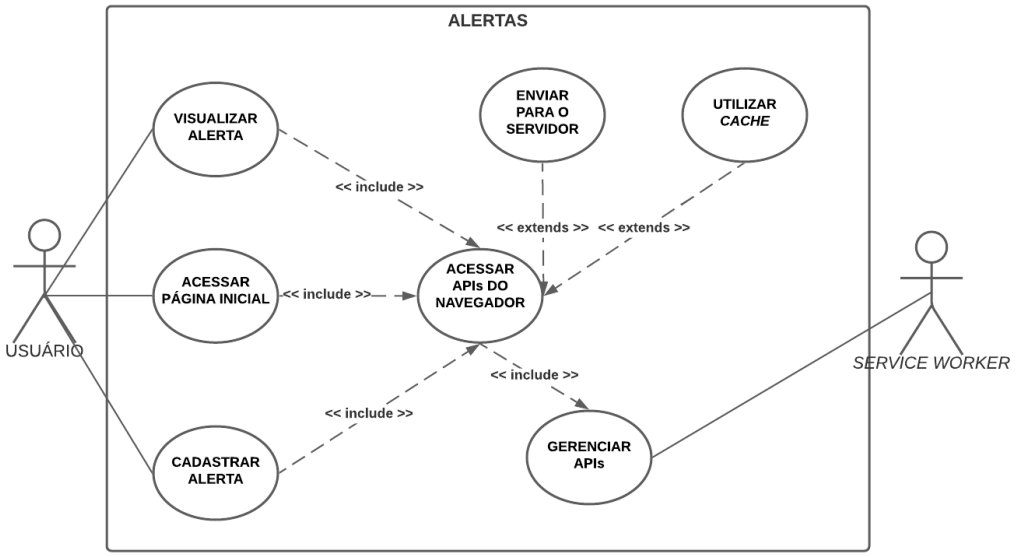

Figura 2. Diagrama de caso de uso [autor].

3.2. Ferramentas, linguagens e tecnologias

Para realizar a implementação do protótipo da aplicação é importante verificar as tecnologias disponíveis que possuem suporte para a construção de um PWA. Levando em consideração fatores como o tempo disponível para desenvolvimento, o conhecimento necessário e a complexidade inerente a esse tipo de aplicação, escolheu-se utilizar uma abordagem de desenvolvimento através de frameworks. Um *framework* estabelece um molde, abstrai complexidades, impõe organização no projeto e dispõe de implementações que resolvem problemas recorrentes. Sendo assim, o desenvolvedor segue padrões pré-estabelecidos, podendo entregar um produto com maior estabilidade, em uma janela de tempo que depende da sua adaptação ao framework [Sommerville 2011].

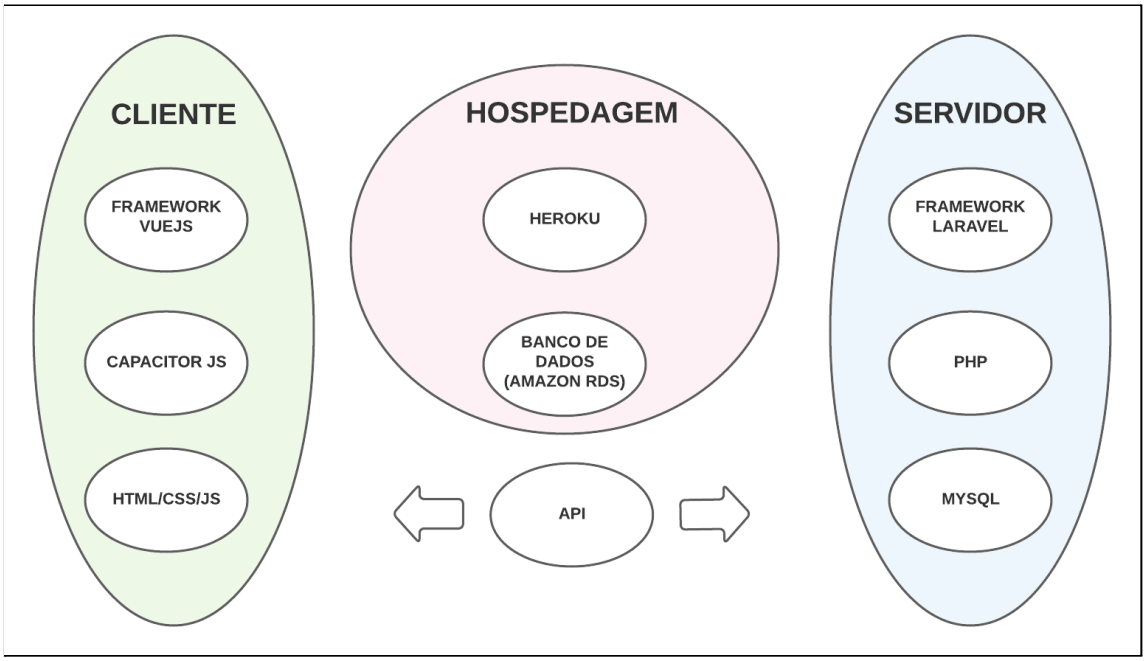

Figura 3. Relação entre as ferramentas, linguagens e serviços utilizados no desenvolvimento da aplicação [autor].

A utilização dos recursos observados na Tabela 1 foram empregados graças às evoluções dos navegadores web e da linguagem Javascript. A documentação oficial do CapacitorJS (2021) destaca que os aplicativos *web* conseguem acessar completamente as APIs nativas com *plugins*. Os *plugins* têm a capacidade de agrupar as operações nativas, entre distintas plataformas, centralizando tudo em uma única API, que é baseada em Javascript, sendo possível utilizar esses recursos de forma unificada, em uma única linguagem de programação. A partir disso, foram implementados os seguintes recursos:

- Acesso à câmera - funciona de forma que, se o usuário permitir o acesso, é possível utilizar os recursos da câmera do dispositivo, ou até mesmo, acessar a galeria de fotos/vídeos;
- Geolocalização - consegue rastrear o dispositivo, reunindo os dados de latitude, longitude, altitude, direção e velocidade do dispositivo, posteriormente podendo se comunicar com algum serviço de mapas. No protótipo em questão, após coletar os dados de geolocalização, utilizou-se o serviço Google Maps para interação;
- Notificações *Push* - oferece interatividade com o usuário, encarregando-se de notificá-lo quando um alerta é cadastrado. Esse recurso funciona quando o aplicativo está aberto, ocasião na qual o usuário recebe uma notificação customizada na tela do sistema, e também, quando o aplicativo está em segundo plano e a notificação é recebida através das características da interface do sistema operacional;
- *Background Sync* - este recurso possibilita que o usuário realize requisições, mesmo que sem conexão. As requisições de cadastro dos alertas são armazenadas em um banco de dados local provido pelo navegador. Enquanto isso, o *service worker* fica “escutando” a aplicação, esperando-a restabelecer a

conexão. No momento que o usuário possuir acesso à internet novamente, a requisição é enviada para o servidor e, então, o alerta é cadastrado.

- Instalação - possibilita instalar o aplicativo *web* no dispositivo através do navegador. O aplicativo ficará disponível na tela inicial do sistema operacional, oferecendo uma experiência de usabilidade muito similar a de uma aplicação nativa.

Os recursos citados anteriormente foram implementados utilizando o CapacitorJS, uma biblioteca construída em Javascript, que possui os métodos apropriados para realizar a comunicação entre o HTML, interface visual do usuário no navegador, e as APIs de cada dispositivo. Ressalta-se que, para todas as funcionalidades abordadas, não é necessário ter o aplicativo instalado. O protótipo em questão dispõe de uma arquitetura básica de comunicação entre servidor e cliente (Figura 3), por isso é necessário escolher uma linguagem de programação específica para cada contexto.

No lado do servidor, onde são aplicadas as regras de negócio, empregou-se o *framework* Laravel, baseado na linguagem de programação PHP. Para a persistência dos dados, optou-se pelo uso de um banco de dados de abordagem relacional, uma vez que a aplicação necessita realizar o vínculo entre os atores. Nesse sentido, escolheu-se o MySQL, tendo sua integração com o aplicativo em produção realizada através dos serviços da Amazon Web Services (AWS), nesse caso, o Amazon Relational Database Service (RDS).

Quanto ao cliente, que é responsável por gerenciar a experiência do usuário, utilizou-se o *framework* Quasar, que é baseado em Vue/Javascript e atende às especificações de um PWA, realizando também a integração com a biblioteca CapacitorJS.

Destaca-se que o ponto chave para a implementação de um PWA está na ferramenta que manipula o lado do cliente, pois é ela que gerencia a comunicação com as APIs do dispositivo do usuário. A parte do servidor, nesta implementação, mostra-se menos relevante pois apenas gerencia as requisições, ou seja, armazena e entrega os dados.

O desenvolvimento foi realizado em ambiente local, bem como os testes e as validações do protótipo, sendo utilizados os sistemas operacionais Android e Windows. Devido à indisponibilidade de dispositivos da Apple/iOS, a aplicação não foi validada neste ambiente.

A plataforma de hospedagem empregada foi o Heroku, uma vez que possui abstração quanto às configurações do *web server* e disponibiliza os serviços de hospedagem sob o protocolo HTTPS, essencial a um PWA. Esse cenário, inclusive, vai ao encontro da proposta de desenvolvimento do protótipo e das necessidades básicas de implantação.

4. Resultados

Ao longo da etapa de desenvolvimento da aplicação constatou-se na prática uma das principais características dos PWAs: a unificação da implementação em relação às plataformas, uma vez que foi possível desenvolver o lado cliente a partir de uma única linguagem de programação. Por meio do Javascript e de suas bibliotecas

complementares, diversas funcionalidades puderam ser implementadas. O aspecto da unificação também foi observado na distribuição, pois o aplicativo pôde ser distribuído para mais de uma plataforma através dos navegadores disponíveis em cada ambiente.

Ainda sob a ótica do desenvolvimento, as funcionalidades no PWA foram implementadas e validadas de acordo com o método exploratório de prototipação. A partir do diagrama de caso de uso da Figura 2, as ideias de implementação iniciais do projeto foram testadas de modo imediato, verificando se o comportamento de cada recurso ocorria como o esperado.

Em um segundo momento, já com o protótipo do aplicativo desenvolvido por completo, nos sistemas operacionais Android e Windows, foi possível constatar mais resultados que merecem ser abordados. A instalação ocorre como o esperado, com o aplicativo permanecendo na tela inicial do dispositivo como se fosse uma aplicação baixada diretamente de uma loja de aplicativos, conforme demonstra a Figura 4 (a). O acesso à câmera também ocorre pelo PWA, conforme mostra a Figura 4 (b), podendo também ser acessada a galeria do sistema operacional, desde que o usuário conceda as permissões necessárias.

Da mesma forma, as coordenadas geográficas nas quais o aparelho está situado, uma vez concedida a permissão, ficam disponíveis ao navegador que fornece tais dados ao aplicativo, conforme apresentado na Figura 4 (b). As notificações *push* são enviadas e recebidas a partir do momento em que um novo alerta é cadastrado. Destaca-se que o comportamento das notificações difere a depender da situação: se o aplicativo estiver aberto, a notificação será destacada dentro dele; caso contrário, a notificação será recebida em segundo plano através da seção de notificações do sistema operacional, conforme tela apresentada na Figura 4 (c). Salienta-se também que, neste caso, é necessário que haja permissão do usuário para receber as notificações pelo navegador. Por último, o *background sync* ocorre quando o usuário não dispõe de uma conexão de internet estável. Nesse caso, observa-se que os dados cadastrados de um novo alerta permanecem no mecanismo de armazenamento do navegador e, uma vez retomada a conexão, o alerta é lançado ao servidor e os usuários são notificados.

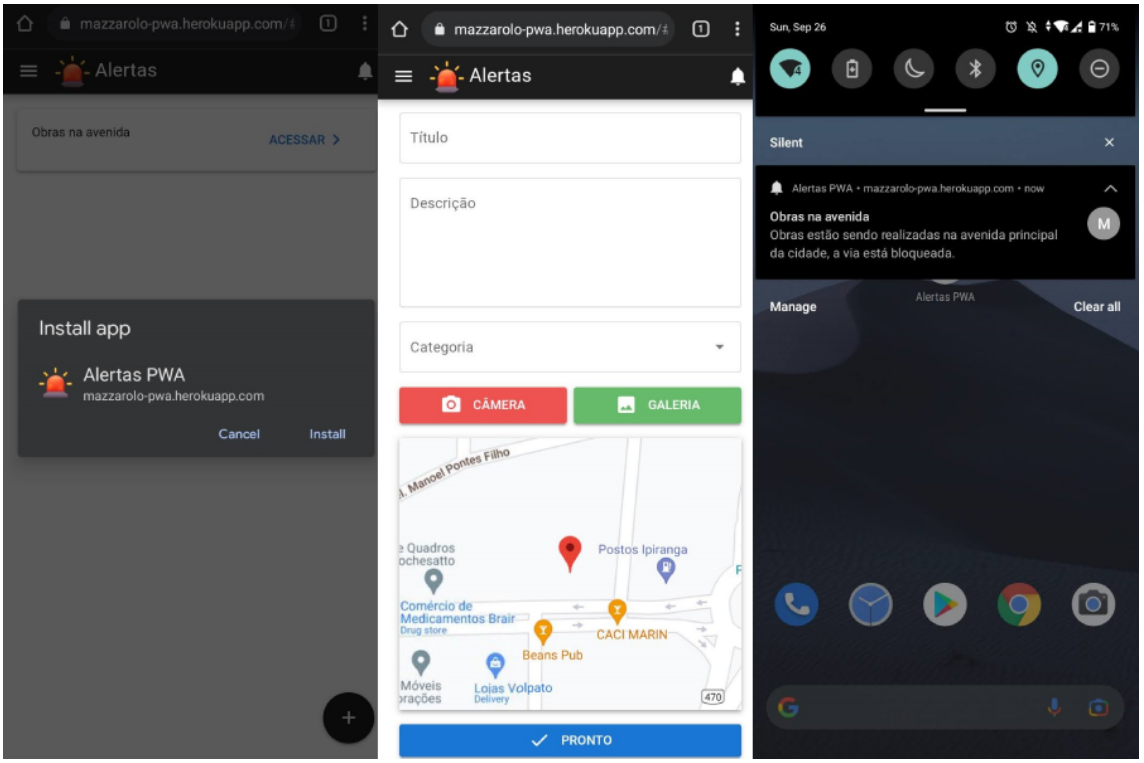

Figura 4. Telas do aplicativo mostrando as funcionalidades de instalação (a), acesso à câmera e geolocalizado do dispositivo (b) e recebimento de notificação no sistema operacional (c) [autor].

No que se refere ao ambiente de hospedagem, não relacionado diretamente ao lado do cliente e à implementação do PWA, o funcionamento ocorre fornecendo as funcionalidades necessárias. O servidor de hospedagem Heroku foi capaz de entregar um domínio protegido sob o protocolo HTTPS que comunica-se com o serviço de banco de dados RDS, da Amazon, possibilitando que ambos os lados da aplicação se comuniquem.

Um último ponto a ser observado é o relacionado à experiência de utilização do PWA. Foi possível verificar que a experiência de uso assemelha-se muito com a de uma aplicação nativa, pois o usuário é capaz de interagir com as APIs propostas através dos recursos implementados. Destaca-se que todas as funcionalidades são acessadas a partir do navegador do dispositivo. No entanto, o aplicativo é progressivo, ou seja, se um navegador não implementou determinada API, o sistema continua operando, porém com uma redução de recursos fornecidos ao usuário.

5. Considerações finais

O presente trabalho objetivou desenvolver uma aplicação protótipo, permitindo analisar o comportamento de um PWA, através de algumas de suas funcionalidades emergentes, dessa forma, pode-se compreender como essa abordagem funciona e quais suas perspectivas.

O protótipo do PWA desenvolvido mostrou-se uma alternativa viável a aplicações nativas, pois ele tem a capacidade de utilizar recursos dos dispositivos por

meio dos navegadores, recursos estes comumente disponíveis em aplicativos nativos. Ainda, a viabilidade também é constatada em relação à experiência de utilização do usuário no PWA, bastante similar à de uma aplicação nativa ao possibilitar o acesso a funcionalidades do dispositivo de maneira intuitiva. Além disso, é possível afirmar que há vantagens no uso do PWA, no sentido que ele possibilita lidar com situações nas quais não há uma conexão estável de internet, através da sua capacidade de realizar requisições mesmo offline por meio do seu armazenamento *cache*.

Faz-se interessante também, após a realização do trabalho, abordar possíveis cenários de utilização sob o viés das equipes de desenvolvimento. Um PWA se apresenta como uma boa solução quando a equipe tem a necessidade de distribuir o aplicativo em mais de uma plataforma. Embora exija a participação de profissionais com conhecimento em Javascript e seus frameworks derivados, é possível, através de uma base de código unificada, atender às mais diversas plataformas trabalhando com uma variedade interessante de ferramentas no contexto da web.

As principais dificuldades em relação ao desenvolvimento de um PWA são: a curva de aprendizado de todo o ecossistema das ferramentas que constituem a aplicação; a realização de testes, que são recorrentemente afetados pelas versões armazenadas em cache do aplicativo; adversidades gerais encontradas, que são distintas entre os ambientes de desenvolvimento e de produção; e prevenir erros de compatibilidade em função de fornecer uma experiência progressiva em todas as distribuições de navegadores.

Por fim, a partir do protótipo obteve-se uma noção sobre a viabilidade de implementar um PWA nos contextos abordados neste trabalho. Portanto, em trabalhos futuros, vislumbra-se ampliar os testes de uso em outros ambientes, sistemas operacionais e navegadores e desenvolver uma aplicação mais robusta e completa, aplicada na solução de uma situação-problema no cotidiano dos usuários.

Referências bibliográficas

Basques, K. (2020) “Why HTTPS matters”, https://web.dev/why-https-matters/, Outubro.

Břoušek, P. (2017). Evaluation and usage of Google Progressive Web Apps technology. 1–50.

Can I Use (2021), https://caniuse.com, Agosto.

Charland, A., Leroux, B. (2011). Mobile Application Development Web vs. Native. Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011, 54, 49–53. https://doi.org/10.1109/CloudCom.2011.113

Fortunato, D., & Bernardino, J. (2018). Progressive web apps: An alternative to the native mobile Apps | Progressive Web Apps: uma alternativa às Apps móveis nativas. Iberian Conference on Information Systems and Technologies, CISTI, 2018-June, 1–6.

Fransson, R., & Driaguine, A. (2017). Comparing Progressive Web Applications with Native Android Applications. Linnaeus University, Faculty of Technology, 1, 59.

Gaunt, M. Web Fundamentals (2021) “Introdução aos service workers”, https://developers.google.com/web/fundamentals/primers/service-workers, Agosto.

Lichter, H., Schneider-Hufschmidt, M., & Zullighoven, H. (1994). Prototyping in Industrial Software Projects-Bridging the Gap Between Theory and Practice. IEEE Transactions on Software Engineering, 20(11), 825–832. https://doi.org/10.1109/32.368126

Linden, O. (2020). Achieving native-like experience on the web with progressive web apps.

MDN Contributors (2021) “Progressive web apps (PWAs)”, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps, Agosto.

Nugroho, L. E., Pratama, A. G. H., Mustika, I. W., & Ferdiana, R. (2017). Development of monitoring system for smart farming using Progressive Web App. 2017 9th International Conference on Information Technology and Electrical Engineering, ICITEE 2017, 2018-Janua, 1–5. https://doi.org/10.1109/ICITEED.2017.8250513

Pande, N., Somani, A., Prasad Samal, S., & Kakkirala, V. (2018). Enhanced Web Application and Browsing Performance through Service-Worker Infusion Framework. Proceedings - 2018 IEEE International Conference on Web Services, ICWS 2018 - Part of the 2018 IEEE World Congress on Services, 195–202. https://doi.org/10.1109/ICWS.2018.00032

Rodriguez, M. (2018). HTTPS Everywhere: Industry Trends and the Need for Encryption. Serials Review, 44(2), 131–137. https://doi.org/10.1080/00987913.2018.1472478

Santos, F. S., Custódio, P., & Júnior, S. (2013). Validação de Requisitos Através da Prototipação de Software. 1, 13–14.

Sharma, V., Verma, R., Pathak, V., Paliwal, M., & Jain, P. (2019). Progressive Web App (PWA) - One Stop Solution for All Application Development Across All Platforms. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 1120–1122. https://doi.org/10.32628/cseit1952290

Sommervile, I. (2011). Engenharia de Software - 9a edição (Vol. 148).

Statcounter Global Stats (2021) “Browser Market Share Worldwide”, https://gs.statcounter.com/browser-market-share, Agosto.

Tjarco, K. (2019). Applicability of Progressive Web Apps in Mobile Development. June, 8–12.

Wijaya, H., & Abbas, R. A. (2018). Animation effectiveness for E-learning with progressive web APP approach: A narrative review. International Journal of Engineering and Technology(UAE), 7(4), 112–120. https://doi.org/10.14419/ijet.v7i4.11.20785