UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DIRETORIA DE GRADUAÇÃO E EDUCAÇÃO PROFISSIONAL
DEPARTAMENTO DE COMPUTAÇÃO
ENGENHARIA DE SOFTWARE

LUIZ GUILHERME DEVIDE SPIRITO

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA GERENCIAMENTO E DESCOBERTA DE EVENTOS UTILIZANDO PWA

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2018

LUIZ GUILHERME DEVIDE SPIRITO

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA GERENCIAMENTO E DESCOBERTA DE EVENTOS UTILIZANDO PWA

Trabalho de Conclusão de Curso apresentada na Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de bacharel em Engenharia de Software

Orientador: Diogo Cezar Teixeira Batista

CORNÉLIO PROCÓPIO

2018

RESUMO

DEVIDE SPIRITO, Luiz Guilherme. Desenvolvimento de uma aplicação para gerenciamento e descoberta de eventos utilizando PWA. 22 f. Trabalho de Conclusão de Curso – Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2018.

Progressive Web App (PWA) é uma tecnologia que utiliza ferramentas do desenvolvimento *Web*, como o HTML, CSS e JavaScript, para desenvolver aplicações que funcionam como uma página *Web*, e como uma aplicação nativa de dispositivos móveis. Através desta tecnologia será desenvolvida uma aplicação que permite a criação, e descoberta de eventos no âmbito universitário. Estes eventos poderão ser descobertos por qualquer usuário próximo de sua criação, portanto ajudando assim em sua divulgação.

Palavras-chave: HTML, CSS, JavaScript, PWA, Eventos, Mobile, Web

SUMÁRIO

1 INTRODUÇÃO 4
1.1 OBJETIVOS 4
1.1.1 Objetivos Específicos 5
1.2 ORGANIZAÇÃO DO TEXTO 5
2 FUNDAMENTAÇÃO TEÓRICA 6
2.1 CONCEITOS DE DESENVOLVIMENTO WEB 6
2.2 APLICAÇÕES MÓVEIS 7
2.3 PWA 8
2.4 MANIFEST 9
2.5 SERVICE WORKER 11
2.6 TRABALHOS RELACIONADOS 15
2.6.1 Flipkart 15
2.6.2 Trivago 15
2.6.3 Facebook 15
3 PROPOSTA 17
4 CONCLUSÃO 20
REFERÊNCIAS 21

1 INTRODUÇÃO

Os eventos organizados nas universidades constituem grande parte da vida de um universitário. Estes eventos formam atividades realizadas pelos estudantes, podendo ou não serem obrigatórias. Eventos estes que podem variar de finalidade, servindo para aprendizado, como uma palestra ou para recreação, como um evento esportivo. Com a participação em eventos não obrigatórios, os estudantes tendem a ter uma maior satisfação com seu curso, uma facilidade no relacionamento entre pessoas e o desenvolvimento de valores altruísticos (FIOR; MERCURI, 2009).

No campus de Cornélio Procópio da Universidade Federal do Paraná, para um estudante ter conhecimento de um evento ele precisará ou ser convidado para o mesmo, ou de alguma maneira descobrir a sua realização, como por exemplo ao encontrar um cartaz de divulgação do evento. Sendo o foco desta aplicação, o de tornar a divulgação destes eventos mais ampla, alcançando deste modo um numero maior de pessoas. Tendo em vista a grande utilização de dispositivos móveis (BOUSHEHRINEJADMORADI et al., 2015).

Um dos métodos mais utilizado para a divulgação dos eventos é a utilização de *emails*, contudo esta abordagem pode acarretar alguns problemas, como: o estudante não ter o seu *email* cadastro, o mesmo estar errado, a não checagem do *email*, entre outros casos que impossibilitam a ciência do estudante. Outro método utilizado por outras universidades, como a UFF (Universidade Federal Fluminense) é de permitir a criação dos evento universitários por suas unidades, limitando a interação do estudante (UFF, 2018).

1.1 OBJETIVOS

O objetivo deste trabalho será o desenvolvimento de uma aplicação usando PWA(*Progressive Web App*). Sendo sua principal função, a de permitir que o usuário crie, gerencie e descubra eventos na sua proximidade. Sua primeira versão será funcional apenas no campus de Cornélio Procópio, da Universidade Tecnológica Federal do Paraná, devido ao tempo de desenvolvimento necessário para a implementação de novos campus.

1.1.1 OBJETIVOS ESPECÍFICOS

- Disponibilizar o mapa, através do Google Maps, com a localização atual do usuário, mostrando assim os eventos ao seu redor. Esse mapa, assim como toda a aplicação, estará em apenas uma página, tornando a aplicação de mais fácil compreensão para o usuário e focando apenas o necessário na tela.
- Ter o seu desenvolvimento em PWA irá garantir uma maior flexibilidade para o usuário, pois isso o permitirá acessar a aplicação tanto como um site, através de uma URL, tanto como em forma de aplicativo.
- O *layout* da aplicação deverá ser responsivo. Portanto deve-se comportar da mesma forma, independente do dispositivo em que esteja sendo usado.

1.2 ORGANIZAÇÃO DO TEXTO

Este trabalho apresenta a seguinte organização: o Capítulo 2 mostra as características de um PWA, e suas principais funções. Assim como é mostrado aplicações já existentes, que tenham funções parecidas com a deste trabalho. O Capítulo 3 mostra a proposta de desenvolvimento da aplicação. Já o Capítulo 4 mostra o que foi concluído com esse estudo.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 CONCEITOS DE DESENVOLVIMENTO WEB

O desenvolvimento web pode ser separado em duas partes. A primeira parte do *front-end*, e a segunda do *back-end* (WALES, 2014). O *front-end* é a parte na qual o usuário tem acesso, a interface de seu sistema, nesta camada temos o uso das tecnologias HTML (W3C, 2018b), CSS (W3C, 2018a) e JavaScript (ECMA, 2018). Já o back-end é o que controla as regras de negócio do sistema e o banco de dados. Segundo especialistas o back-end pode ser feito usando ASP, PHP e Java (WALES, 2014).

O HTML é uma linguagem de marcação que usa *tags* para definir elementos da página. Dessa forma temos a base da página, porém sem nenhuma estilização, apenas com os elementos e suas definições (W3C, 2018b).

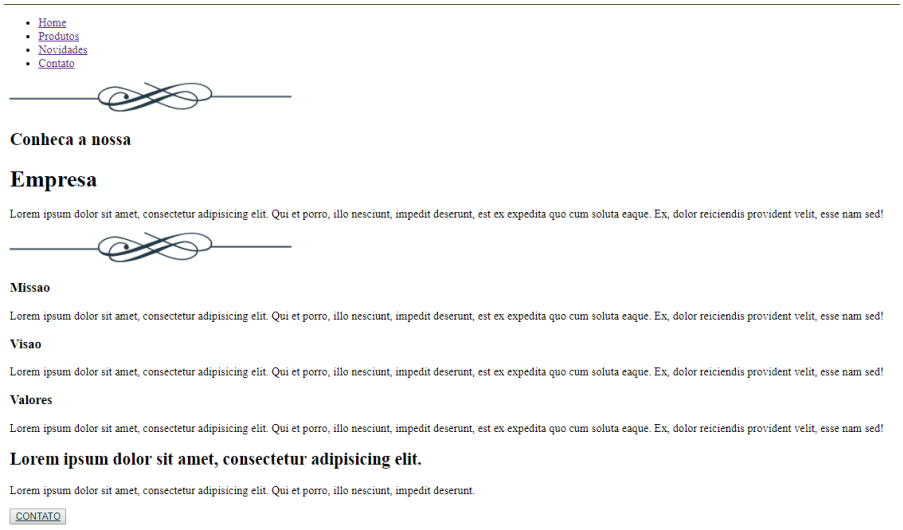

Figura 5: Exemplo de uma página apenas com o HTML5

Fonte: Autoria Própria

O CSS é utilizado para estilizar a página HTML, melhorando assim a experiência do usuário. Pois através dele podemos alterar fontes, definir cores, e alterar a posição de cada

elemento de nossa página (W3C, 2018a).

Figura 6: Exemplo de uma página com o HTML5 e CSS3

Fonte: Autoria Própria

O JavaScript é uma linguagem de programação interpretada, permitindo a execução de *scripts* do lado do cliente, sem a necessidade de passar pelo servidor. Esta função se torna muito útil ao se realizar tarefas que apenas interagem com o *front-end* (ECMA, 2018). É muito importante para a aplicação, pois através de *scripts* será possível fazer a ligação da aplicação com o servidor. Os principais arquivos responsáveis pelo envio do funcionamento da aplicação serão o *manifest.json* e o *Service Worker*. Estes arquivos serão mais detalhados na seção 2.4 e 2.5, respectivamente.

2.2 APLICAÇÕES MÓVEIS

Existem três grupos de aplicações móveis: aplicações nativas, aplicações *Web* e aplicações híbridas. Aplicações nativas são desenvolvidas para sistemas operacionais específicos, sendo a forma de desenvolvimento definida pela organização proprietária. Através da organização proprietária é fornecido o *Software Development Kit* (SDK), e uma *Integrated Development Environment* (IDE), possibilitando o desenvolvimento da aplicação para a plataforma (IBM, 2018). Ao se desenvolver de forma nativa, a aplicação estará disponível na loja de seu respectivo sistema operacional, e o seu desenvolver terá acesso a todas *Application Programming Interface* (API) do sistema operacional. Porém a desvantagem de seu uso é que a aplicação estará disponível em uma única plataforma, sendo necessário o seu desenvolvimento de outro modo para ser possível o uso em outras plataformas (MENEGASSI; ENDO, 2015).

Aplicações *Web* são desenvolvidas com recursos *Web*, como HTML, CSS e JavaScript, para executar diretamente no navegador e podem ter acesso a recursos como geolocalização, armazenamento local *offline* e componentes de *interface* (IBM, 2018). Apesar de não ficar disponível na loja, tem como principal vantagem a disponibilização para todas plataformas móveis. Como exemplo de aplicação *Web* temos o *Progressive Web App*(PWA), que será melhor detalhado na seção 2.3 (MENEGASSI; ENDO, 2015).

Aplicações híbridas juntam características das aplicações nativas e *Web*, sendo desenvolvida com o uso de HTML, CSS e JavaScript e tendo o suporte das APIs nativas. A aplicação passa ter duas partes, a nativa e a chamada *WebView*, responsável pela parte *Web*. Por se tratar de uma aplicação nativa, ela pode ser disponibilizada na loja da plataforma (MENEGASSI; ENDO, 2015).

2.3 PWA

Dados comparados entre os anos de 2015 e 2016, mostram que o número de downloads de aplicativos móveis diminui em 20% (LIMA, 2017). Esse fato, somado ao dado de que o usuário gasta 80% do seu tempo em apenas cinco aplicativos, que são: Facebook, YouTube, Google Maps, Pandora e Gmail (LIMA, 2017).

Uma aplicação nativa traz algumas desvantagens, como um custo de desenvolvimento maior, e até mesmo uma incerteza sobre a aprovação da mesma em ser publicadas nas principais lojas de aplicativos, assim como o fato de seu desenvolvimento ser exclusivo de uma plataforma (MADUREIRA, 2017).

É nesse momento que surge a opção do PWA, um modelo de desenvolvimento *Web*. Bastando assim apenas o desenvolvimento de tipo web, para ser ter um aplicativo funcional, assim como um site.

Mas para ser considerado um PWA, este aplicativo precisa obedecer algumas regras, que são (DEVELOPERS, 2018):

1. Progressivo: Deve funcionar para todo e qualquer tipo de usuário, independente do navegador utilizado.

2. Responsivo: O *layout* da página deve se comportar de forma funcional, independente do dispositivo usado. Como um celular, tablet, notebook.

3. Independente de conectividade: Uma aplicação PWA deve ter pelo menos uma funcionalidade disponível de forma *offline*, e aprimorada para funcionar em redes de baixa

qualidade. Um exemplo para isso é um site de notícias, onde mesmo estando offline, o usuário poderia ter acesso as noticias previamente carregas em seu último acesso.

4. Semelhante a aplicativos: Necessário que se pareça com aplicativos nativos, com interações e até emissões de notificações.

5. Atual: Não é necessário realizar o download de atualizações, o navegador irá atualizar a aplicação de forma automática.

6. Seguro: A sua página deve ser no formato HTTPS, mantendo assim a sua segurança. Este protocolo é implementado com uma camada a mais em relação ao HTTP, garantindo assim uma maior segurança á página.

7. Descobrível: Em seus manifestos e *Service Worker*, deverá ser definido que ele pode ser identificável como um aplicativo, permitindo assim a sua instalação. Este assunto será melhor explorado na seção 2.3.

8. Reenvolvente: Deve permitir o uso de notificações push. Permitindo aplicações mandarem notificações para o usuário, tanto na aplicação móvel, tanto na página *Web*.

9. Instalável: Permite que o usuário instale a aplicação, porém sem ser preciso acessar alguma loja de aplicativos, como a Play Store ou a App Store.

10. Linkável: Pode ser acessado por uma URL, podendo assim ser acessado via navegador, e aplicação. Quando solicitada, a instalação deve ser executada de forma simples.

Algumas destas regras terão um enfoque maior no desenvolvimento da aplicação, como por exemplo o desafio de se ter uma aplicação que é uma *Single Page Application*, isso quer dizer que a *interface* é formada por apenas uma página, responsiva em inúmeros dispositivos (ANALYTICS, 2018). Outro grande foco será a parte de instalação, onde o maior o objetivo é que a aplicação depois de instalada seja igual a uma aplicação nativa.

Para tornar estes objetivos possíveis, e definir a aplicação como um *PWA*, iremos utilizar dois arquivos: O *manifest.json* e o *Service Worker*, que através de *scripts* mandará as informações para o navegador.

2.4 MANIFEST

Para armazenar e transmitir informações no formato texto,é usado o modelo JSON (*JavaScript Object Notation*), sendo conhecido por sua simplicidade e capacidade de estruturar

informações de forma mais compacta(CORRÊA, 2017). Modelo responsável por um dos principais arquivos contidos em um *PWA*, o manifest.json. Este arquivo é responsável pelo envio de informações específicas da aplicação para o navegador.

No código 2.1 podemos ver um exemplo de um arquivo manifest.json, e suas principais regras (KINLAN, 2018):

Listing 2.1: Exemplo de um arquivo manifest.json

```
"name": "PWA Aplication",
"short_name": "A-PWA",
"theme_color": "#09ffla",
"background_color": "#000000",
"display": "fullscreen",
"scope": "/",
"start_url": "/pwa-aplication",
"lang": "pt-BR",
"orientation": "any",
"icons": [
        {
        "src": "/assets/img/icons/icone.png",
        "sizes": "512x512",
        "type": "image/png"
        }
]
```

- name: Esse será o nome da aplicação.
- short-name: O nome que irá aparecer no ícone do aplicativo.
- theme-color: Define a cor tema da aplicação, como por exemplo a cor da barra de ferramentas.
- background-color: Cor de fundo da aplicação, item obrigatório.
- display: Define de que modo a aplicação apresentada na tela.
- starturl: Com ela é possível saber se a página foi aberta via app.
- lang: Define em que língua o aplicativo será utilizada.
- orientation: Define se a aplicação será utilizada em telas na vertical ou na horizontal.
- icons: Define as dimensões da imagem a ser usada com ícone. Essa imagem deve ser do tipo PNG.

Após a criação do manifest, é necessário o adicioná-lo ao seu projeto. Para isso é necessário referenciá-lo no head do projeto, utilizando a *tag* link.

Listing 2.2: Exemplo da adição do arquivo manifest

```
<head>
        <link rel="manifest" href="/manifest.json">
</head>
```

2.5 SERVICE WORKER

O *Service Worker* é um arquivo JavaScript executado de forma paralela ao navegador. Sendo muito utilizado para o tratamento de solicitações de redes e gerenciamento de cache (GAUNT, 2018). Portanto é através dele que se cria uma experiência offline para o usuário.

Mesmo que de forma imparcial, a aplicação deve continuar funcional, isto será realizado através do uso de cache, que são dados salvos no dispositivo no momento em que havia uma conexão com a Internet (JOUPPI, 1990). Estes dados podem ser recuperados e mostrados ao usuário enquanto o mesmo estiver offline.

Existem algumas informações importantes sobre o Service Worker(JUSTEN, 2017).

- Sempre que houver alguma atualização no site, deve-se excluir o cache antigo. Isso evitará que o usuário esteja vendo conteúdo já apresentado.
- O nome, e a localização do arquivo *Service Worker* devem ser sempre iguais, pois caso sejam alterados podem gerar uma duplicação de *Service Worker*.

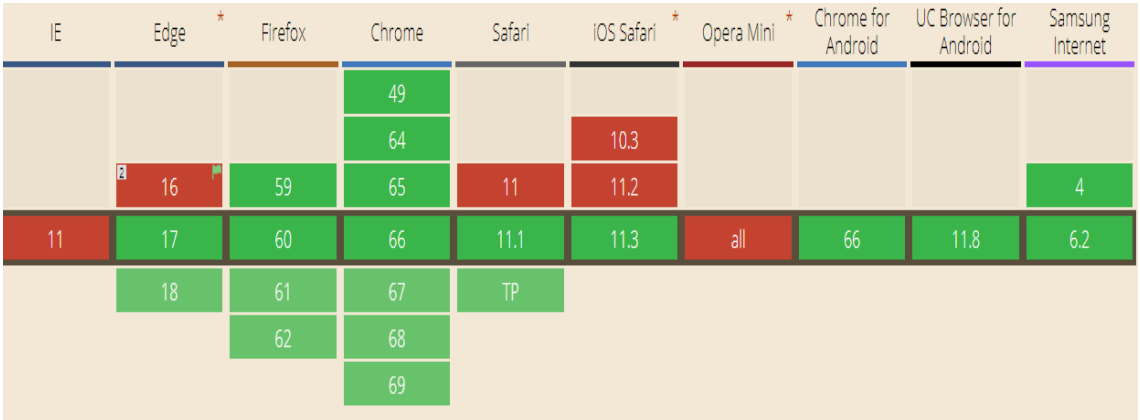

Figura 7: Exemplo de navegadores, e suas versões que suportam o *Service Worker*

Fonte: (USE, 2018)

O primeiro passo será verificação de que o navegador utilizado suporta o Service Worker. Atualmente navegadores como Chrome, FireFox e Opera se mostram bastante receptivos a esta tecnologia. Porém alguns, como o Safari e Edge, mostraram não estar preparados para ela (GAUNT, 2018).

O funcionamento de um Service Worker é realizado através de fases, que se comportam como o ciclo de vida dele. Esse ciclo é formado por 6 eventos, que são (JUSTEN, 2017):

- Install: Este evento só é chamado na primeira vez em que o *Service Worker* é registrado, porém caso haja alguma atualização no arquivo, ele será executado novamente.

Listing 2.3: Exemplo de um código implementando o evento *install*

```
const staticCacheName = 'luiz-devide-2018-05-08-12-35';
this.addEventListener('install', event => {
        this.skipWaiting();
        event.waitUntil(
                caches.open(staticCacheName)
                        .then(cache => {
                                return cache.addAll(filesToCache);
                })
        )
});
```

No código 2.3 podemos observar que é atribuído um nome a variável cache, e pra ele é passado o horário em que o site foi gerado. Desse modo teremos a informação de quando o site foi atualizado. Isso permite que o Service Worker atualize o cache. Desse modo sempre será salva no cache a informação mais recente.

- Activate: É executado apenas uma vez quando uma nova versão do Service Worker for instalada, e nenhuma outra versão antiga estiver rodando.

Listing 2.4: Exemplo de um código implementando o evento *activate*

```
this.addEventListener('activate', event => {
   event.waitUntil(
           caches.keys().then(cacheNames => {
                   return Promise.all(
                           cacheNames
                                   .filter(cacheName =>
                                   (cacheName.startsWith('luiz-devide')))
                                   .filter(cacheName =>
                                   (cacheName !== staticCacheName))
                                   .map(cacheName =>
```

```
                                        caches.delete(cacheName))
                    );
            })
    );
});
```

Uma das principais funções que podemos ver no código 2.4, é o de excluir arquivos antigos. Dessa forma garantimos que nosso usuário nunca estará vendo informações antigas.

- Fetch: Este evento é executado toda vez que a página for requisitada. Sendo um dos principais pela velocidade de carregamento de um conteúdo específico. Pois ele irá verificar se um arquivo já existe na cache, e caso não exista você poderá redirecionar o usuário para uma outra página, podendo ser até offline.

Listing 2.5: Exemplo de um codigo implementando o evento *fetch*

```
this.addEventListener("fetch", event => {
  event.respondWith(
        caches.match(event.request)
        .then(response => {
                return response || fetch(event.request);
        })
        .catch(() => {
                return caches.match('/offline/index.html');
        })
  )
});
```

- Message: Este evento é executado em situações específicas. Geralmente é uma mensagem criado pelo cliente, e lida pelo Service Worker.

Listing 2.6: Exemplo de um código implementando o evento *message*

```
//Enviando a mensagem para o cliente
client.postMessage({
        msg: "Exemplo de mensagem",
        url: event.request.url
});
//Recebendo a mensagem
        navigator.serviceWorker.addEventListener('message',event =>{
        console.log(event.data.msg, event.data.url);
});
```

Nesse caso o cliente utilizou um método chamado postMessage() para enviar a mensagem para o Service Worker.

- Sync: Este evento será executado sempre que necessário. Sua principal função será de sincronizar uma página, mesmo que o usuário não tenha internet para isso no momento. Portanto ele ficará tentando fazer o seu carregamento até que a página esteja disponível, e quando isso acontecer poderá ser enviado uma notificação para o usuário. Um exemplo disso, são as publicações offline do facebook. Onde mesmo sem conexão o usuário pode realizar uma postagem. E ao se conectar a uma rede, essa publicação estará online.

Listing 2.7: Exemplo de um codigo implementando o evento *sync*

```
self.addEventListener('sync', function(event) {
        if (event.tag == 'myFirstSync')
                event.waitUntil(doSomeStuff());
});
```

No código 2.7, o *doSomeStuff()* irá retornar uma *promise* indicando o sucesso ou falha de sua operação (ARCHIBALD, 2018).

- Push: Este evento é executado sempre que uma notificação for solicitada. É através dele que podemos criar notificações, tanto em dispositivos móveis, tanto em sistemas como o próprio Windows.

Listing 2.8: Exemplo de um codigo implementando o evento *push*

```
navigator.serviceWorker.ready
        .then(function (registration) {
                registration.pushManager.getSubscription()
                .then(function (subscription) {
                        if (subscription)
                          changePushStatus(true);
                        else
                          changePushStatus(false);
                })
                .catch(function (error) {
                  console.error('Error occurred while enabling push',error);
                });
        });
}
```

No código 2.8 é determinado se o navegador suporta ou não o *push*.

2.6 TRABALHOS RELACIONADOS

No momento já existem projetos que compartilham das mesmas ideias deste trabalho, porém de formas separadas. Portanto as seções 2.6.1, 2.6.2 e 2.6.3 detalharão o uso de outra

aplicações que compartilham o mesmo conceito da aplicação.

2.6.1 FLIPKART

Flipkart é um *e-commerce* indiano, considerado o caso de maior sucesso de uso do PWA (LIMA, 2017). Isso pode ser comprovado por alguns números ao se comparar a sua versão *Lite*, que usa o PWA, e a sua versão nativa. Onde a pricipais vantagens de seu uso foram (DEVELOPERS, 2016):

- Os usuários passarão a gastar três vezes mais tempo no site
- Houve um engajamento 40% maior dos usuários
- O consumo de dados foi três vezes menor

2.6.2 TRIVAGO

É uma aplicação para se realizar buscas de hotéis (TRIVAGO, 2018). Foi desenvolvida utilizando-se o conceito de *PWA*. Mantém um conceito minimalista, mostrando apenas o que é necessário na tela, algo pretendido com esse projeto.

Sua interface se resume a apenas um menu, para login, e uma barra de busca. Nela é possível digitar o nome de uma cidade, e será mostrado uma lista com os hotéis disponíveis.

Por se tratar de uma aplicação *PWA* podemos observar o uso de algumas de suas recomendações. Como por exemplo o fato de a aplicação ser instalável, a responsividade em diferentes tipos de aparelhos, e o uso do https para reforçar a segurança. Características essas que serão aproveitadas no projeto.

2.6.3 FACEBOOK

O Facebook é a maior rede social do planeta, tendo atualmente mais de 2,07 bilhões de usuários (ESTADÃO, 2017). Apesar de não ser um PWA, ele contém uma característica muito semelhante a do projeto.

Nele é possível realizar a criação de eventos. A criação pode representar qualquer tipo de evento, desde um show musical, até um jogo de futebol. Para isso basta entrar com o local onde ele irá acontecer, uma data de início e fim, e o horário em que ele irá acontecer (FACEBOOK, 2018).

Porém para saber da criação de um evento é necessário ou ser convidado ou o procurar de forma manual. Isso deixa o usuário com a sensação de que ele poderia ter o conhecimento de algum evento acontecendo ao seu redor (FACEBOOK, 2018).

3 PROPOSTA

Para o desenvolvimento da aplicação será usado o modelo incremental. Neste modelo o software será dividido em fases, onde em cada uma delas será definida uma funcionalidade (MOHAMMAD et al., 2010). Apos a implementação da fase o software precisa estar funcional, de modo que seja possível iniciar a próxima fase em sequencia. Este modelo de desenvolvimento é recomendado para projetos pequenos que não tenham muitos requisitos (MOHAMMAD et al., 2010).

O primeiro passo do desenvolvimento será o de construir o front-end da aplicação. Nessa fase teremos a interface do sistema. Por se tratar de um *Single Page Application*(SPA), a sua construção será bem minimalista. Contendo apenas um campo de busca, um menu de opções, um botão para adicionar um evento, tudo isso sobre o layout do mapa. Esse mapa será formado utilizando a API do Google Maps. No menu de opções será possível realizar o *login* e o *logout* do usuário, alem do gerenciamento de eventos já criados. A criação dos eventos será possível através do botão adicionar. Para completar a criação o usuário deverá fornecer o local ou em que sala ele irá ocorrer, o horário e a duração do mesmo.

Na segunda etapa será implementado o campo de busca, com ele será possível procurar salas, tendo acesso a eventos próximos a ele. Já no menu teremos algumas informações do usuários, locais recentes e a opção de criar um evento. Outro modo de se criar o evento e clicando no botão rápido existente.

A ultima etapa será a implementação do mapa, que deverá estar ligado ao sistema de posicionamento do dispositivo.

O mapa a ser usado irá sempre acompanhar o usuário, mostrando todos os eventos ao seu redor. Esse mapa será no formato 2D, e irá ocupar a tela inteira. A localização do usuário será captada através de seu GPS, que compartilhará essa informação com a API do Google Maps, mostrando assim a sua localização.

Quando os eventos disponíveis aparecem no mapa, será possível selecionar ele. Isso mostrará os dados de quem criou o evento, em que horário ele será realizado, e em que local ou

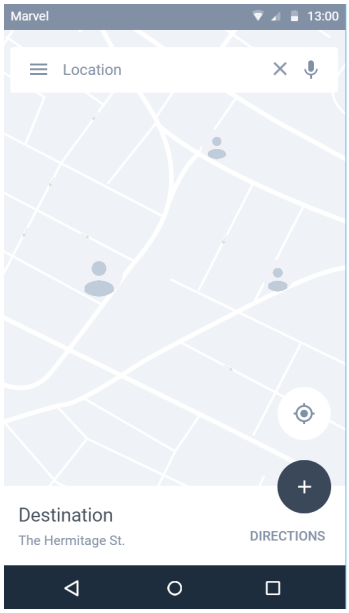

Figura 9: Protótipo de tela da aplicação

Fonte: Autoria Propria

sala ele irá ocorrer. Essas mesmas informações serão necessárias quando o usuário criar o seu evento.

Na etapa inicial do projeto, será desenvolvido que funcione apenas no campus de Cornélio Procópio da Universidade Tecnológica Federal do Paraná.

O cronograma compreende o período de fevereiro a dezembro de 2018.

Atividade 1: Definição do tema

Atividade 2: Estudo teórico do desenvolvimento e das ferramentas de um PWA

Atividade 3: Desenvolvimento de uma aplicação básica em PWA

Atividade 4: Leitura de artigos relacionados ao trabalho

Atividade 5: Entrega da proposta

Atividade 6: Correção da proposta

Atividade 7: Desenvolver o *front-end* do sistema

Atividade 8: Implementar o mapa de localização e suas funcionalidades

Atividade 9: Realizar os testes

Atividade 10: Entrega do trabalho final

Atividade 11: Correção do trabalho final

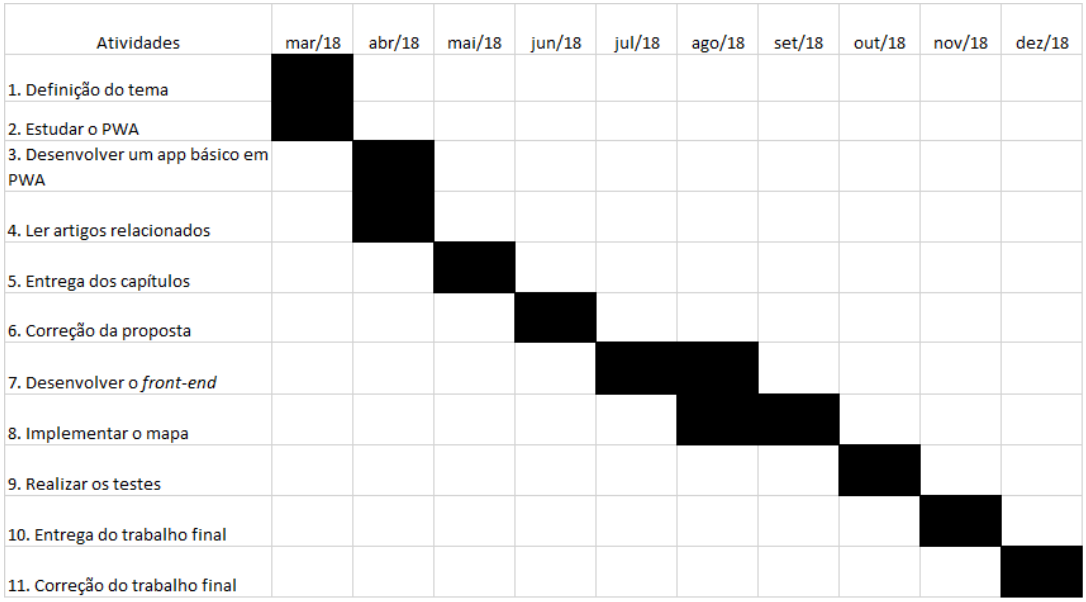

Figura 10: Cronograma

Fonte: Autoria Propria

4 CONCLUSÃO

Ao fim do desenvolvimento é esperado que haja um maior engajamento dos estudantes em relação aos eventos que acontecem no ambiente da universidade. Isto é esperado devido ao fato de a aplicação tornar a divulgação dos eventos mais ampla, tornando publico o convite para todos que estiverem próximos ao evento. Combinado com o uso do PWA torna a aplicação mais acessível para os usuários, podendo acessa-la por diversos tipos de dispositivos, aumentando assim o alcance de divulgação do evento.

REFERÊNCIAS

ANALYTICS, G. **Acompanhamento de aplicativo de página única**. 2018. Disponível em: <https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications?hl=pt-br>.

ARCHIBALD, J. **Introducing Background Sync**. 2018. Disponível em: <https://developers.google.com/web/updates/2015/12/background-sync>.

BOUSHEHRINEJADMORADI, N. et al. Testing cross-platform mobile app development frameworks (t). In: **2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2015. p. 441–451.

CORRÊA, E. **JSON Tutorial**. 2017. Disponível em: <https://www.devmedia.com.br/json-tutorial/25275>.

DEVELOPERS, G. **Flipkart triples time-on-site with Progressive Web App**. 2016. Disponível em: <https://developers.google.com/web/showcase/2016/flipkart>.

DEVELOPERS, G. **Progressive Web Apps**. 2018. Disponível em: <https://developers.google.com/web/progressive-web-apps/>.

ECMA. **ECMA**. 2018. Disponível em: <http://www.ecma-international.org/>.

ESTADÃO, L. **Facebook alcança 2,07 bilhões de usuários no mundo**. 2017. Disponível em: <http://link.estadao.com.br/noticias/empresas,facebook-alcanca-2-07-bilhoes-de-usuarios-no-mundo,70002069551>.

FACEBOOK. **Criação e edição de eventos**. 2018. Disponível em: <https://www.facebook.com/help/131325477007622/>.

FIOR, C.; MERCURI, E. Formação universitária e flexibilidade curricular : importância das atividades obrigatórias e não obrigatórias Camila Alves Fior. **Psic. da Ed**, p. 191–215, 2009. ISSN 1414-6975.

GAUNT, M. **Service Workers: uma Introdução**. 2018. Disponível em: <https://developers.google.com/web/fundamentals/primers/service-workers/?hl=pt-br>.

IBM. **Mobile approaches: Native, web or hybrid mobile application development**. 2018.

JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: **[1990] Proceedings. The 17th Annual International Symposium on Computer Architecture**. [S.l.: s.n.], 1990. p. 364–373.

JUSTEN, W. **Como fazer seu site funcionar offline com PWA**. 2017. Disponível em: <https://willianjusten.com.br/como-fazer-seu-site-funcionar-offline-com-pwa/>.

KINLAN, M. G. P. **O manifesto do aplicativo web**. 2018. Disponível em: <https://developers.google.com/web/fundamentals/web-app-manifest/?hl=pt-br>.

LIMA, M. **Introdução aos Progressive Web Apps**. 2017. Disponível em: <https://tableless.com.br/introducao-aos-progressive-web-apps/>.

MADUREIRA, D. **Aplicativo nativo, web App ou aplicativo híbrido?** 2017. Disponível em: <https://usemobile.com.br/aplicativo-nativo-web-hibrido/>.

MENEGASSI, A. A.; ENDO, A. T. Uma avaliação de testes automatizados para aplicações móveis híbridas. 2015.

MOHAMMAD, N.; MUNASSAR, A.; GOVARDHAN, A. A Comparison Between Five Models Of Software Engineering. **International Journal of Computer Science Issues**, v. 7, n. 5, p. 94 – 101, 2010. ISSN 09736107.

TRIVAGO. **Trivago**. 2018. Disponível em: <https://www.trivago.com.br/>.

UFF. **Como divulgar os eventos da minha unidade?** 2018. Disponível em: <http://www.uff.br/?q=faq/como-divulgar-os-eventos-da-minha-unidade>.

USE, C. I. **Service Workers**. 2018. Disponível em: <https://caniuse.com/#feat=serviceworkers>.

W3C. **Cascading Style Sheets**. 2018. Disponível em: <https://www.w3.org/Style/CSS/Overview.en.html>.

W3C. **HTML 5.2**. 2018. Disponível em: <https://www.w3.org/TR/html52/introduction.html#scope>.

WALES, M. **3 Web Dev Careers Decoded: Front-End vs Back-End vs Full Stack**. 2014. Disponível em: <https://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html>.