

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



TERMO DE APROVAÇÃO

Desenvolvimento de uma aplicação para gerenciamento e descoberta de eventos utilizando
PWA

por

Luiz Guilherme Devidé Spirito

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de “Engenheiro em Engenharia de Software” e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Software da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio, XX/XX/XXXX

Prof. Titulação, Nome Professor Orientador
Universidade Tecnológica Federal do Paraná

Prof. Titulação, Nome professor membro da
banca
Universidade Tecnológica Federal do Paraná

Prof. Titulação, Nome professor membro da
banca
Universidade Tecnológica Federal do Paraná

RESUMO

DEVIDE SPIRITO, Luiz Guilherme. Desenvolvimento de uma aplicação para gerenciamento e descoberta de eventos utilizando PWA. 23 f. Trabalho de Conclusão de Curso – Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2018.

Será desenvolvido uma aplicação *Web*, de forma a ser considerada um PWA. Para isso serão utilizadas tecnologias como: HTML, CSS e JavaScript. O principal objetivo da aplicação é permitir que o usuário crie e gerencie eventos em um campus universitário.

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

SUMÁRIO

1	INTRODUÇÃO	5
1.1	MOTIVACIONAL	5
1.2	OBJETIVOS	6
1.2.1	Objetivo Geral	6
1.2.2	Objetivos Específicos	6
1.3	ORGANIZAÇÃO DO TEXTO	6
2	FUNDAMENTAÇÃO TEÓRICA	7
2.1	CONCEITOS DE DESENVOLVIMENTO WEB	7
2.2	APLICAÇÕES MÓVEIS	8
2.3	PWA	9
2.4	MANIFEST	11
2.5	SERVICE WORKER	12
2.6	TRABALHOS RELACIONADOS	16
2.6.1	Trivago	16
2.6.2	Facebook	16
3	PROPOSTA	18
4	CONCLUSÃO	21
	REFERÊNCIAS	22

1 INTRODUÇÃO

Em 1990, Tim Berners-Lee desenvolveu o primeiro navegador *web* da história, o WorldWideWeb. No final deste mesmo ano Lee criou o servidor, responsável por receber e gerenciar as informações, e as primeiras paginas *Web*. Este feito marcou a história como o primeiro serviço publicado na Internet (BERNERS-LEE, 1996). Em 1993, a empresa detentora desta tecnologia, a CERN, declarou que a Internet seria liberada para todos. Acarretando no aparecimento de novas tecnologias, como o JavaScript em 1995 e o CSS em 1995(CERN, 2018).

A união do HTML, CSS e JavaScript formam o pilar do desenvolvimento *Web*. Porém essas mesma tecnologias sofreram algumas melhorias, atualmente no desenvolvimento *Web* é utilizado o HTML5(W3C, 2018b) e CSS3(W3C, 2018a). Tecnologias essas que serão muito importante no desenvolvimento desta aplicação.

1.1 MOTIVACIONAL

Em universidades podem estar acontecendo vários eventos ao mesmo instante. Porém sem a devida divulgação fica difícil para os estudantes, ou frequentadores destes locais, saberem onde eles estão acontecendo, ou até mesmo de sua existência. O método mais utilizado para a divulgação dos eventos é a utilização de *emails*, porém se mostrando não muito eficaz. Essa falta de eficiência pode acontecer por alguns fatores: o estudante não ter o seu *email* cadastrado, ou o mesmo estar errado, a não checagem do *email*, entre outros fatores.

Um exemplo desse problema é a própria apresentação de um TCC. Na qual um aluno que se interessa por tema X, está próximo a apresentação, porém por falta de divulgação o mesmo acabar por não comparecer na mesma, perdendo assim uma oportunidade de aumentar o seu conhecimento.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O principal objetivo desse trabalho será o desenvolvimento de uma aplicação usando PWA(*Progressive Web App*) para isso. 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.

1.2.2 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 todo sistema, 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.

1.3 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 sendo a do *front-end*, e a segunda do *back-end*. O *front-end* é a parte na qual o usuário tem acesso, a interface de seu sistema. Nessa camada temos o uso das tecnologias HTML, CSS e JavaScript. Já o *back-end* é o que controla as regras de negócio do sistema e o banco de dados. Geralmente programado 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.

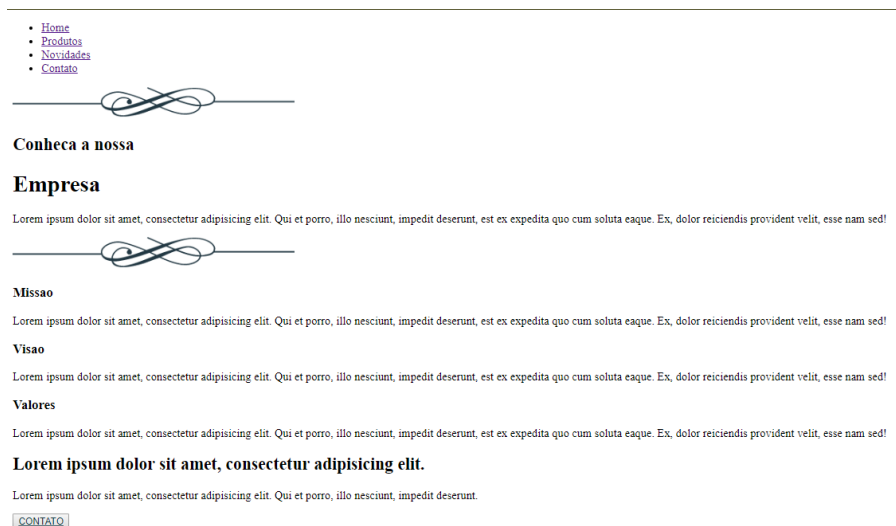


Figura 5: Exemplo de uma pagina apenas com o HTML5

Fonte: Autoria Propria

É nesse momento em que o CSS3 será utilizado. Ele será o responsável pela estilização da página. Pois através dele podemos alterar fontes, definir cores, e alterar a posição de cada elemento de nossa página.

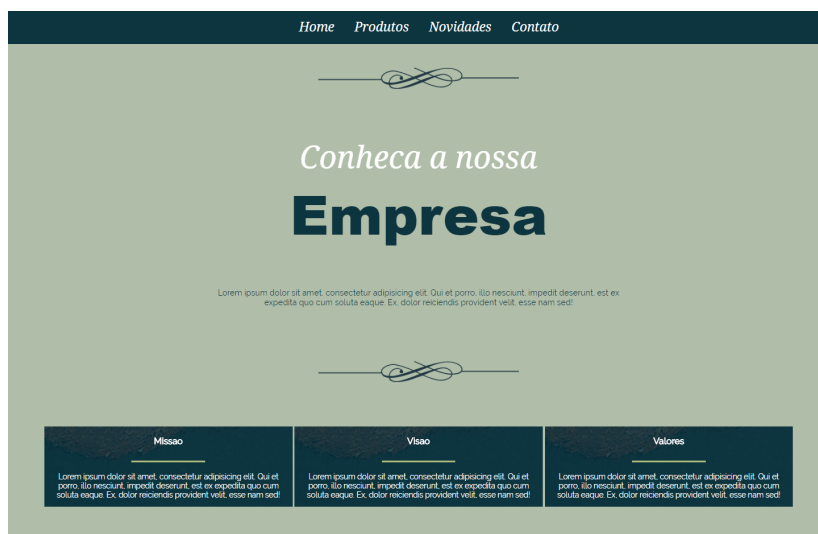


Figura 6: Exemplo de uma página com o HTML5 e CSS3

Fonte: Autoria Propria

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*. Será 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 seções 2.4 e 2.5, respectivamente.

2.2 APLICAÇÕES MÓVEIS

Existem três grupos de aplicações moveis: 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. 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. Podem ter acesso a recursos como geolocalização, armazenamento local *offline* e componentes de *interface*. Apesar de não ficar disponível na loja, tem como principal vantagem a disponibilização para todas plataformas moveis. Como exemplo de aplicação *Web* temos o *Progressive Web App*(PWA), que será melhor detalhado na sessão 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.

2.3 PWA

Dados comparados entre os anos de 2015 e 2016, mostram que o número de downloads de aplicativos móveis diminui em 20%. 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). Isso torna o mercado hostil para novas aplicações.

Devido a esses números algumas empresas se sentem receosas ao desenvolver uma aplicação. Principalmente se ela for feita de forma nativa, onde será necessário o uso de pelo menos duas equipes de desenvolvimento, uma para sistemas Android, e outra para sistemas iOS.

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 notícias 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 sessão 2.3.
8. Reenvolvente: Deve permitir o uso de notificações push. Permitindo aplicações mandarem notificações para o usuário, tanto na aplicação móvel, tanto na pagina *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. 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 atraves 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 (CORREIA, 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 o exemplo de um arquivo `manifest.json`, e quais suas principais regras (KINLAN, 2018):

Listing 2.1: Exemplo de um arquivo `manifest.json`

```
{
  "name": "PWA Application",
  "short_name": "A-PWA",
  "theme_color": "#09ff1a",
  "background_color": "#000000",
  "display": "fullscreen",
  "scope": "/",
  "start_url": "/pwa-application",
  "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 adicionar em seu projeto. Para isso é necessário fazer a sua chamada 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 podemos garantir uma experiencia offline para o usuário. Mesmo que de forma imparcial, a aplicação deve continuar funcional. Isso é possível através do uso de cache, que são dados salvos no dispositivo no momento em que havia uma conexão com a Internet. 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*.

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

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49						
			64		10.3				
	16	59	65	11	11.2				4
11	17	60	66	11.1	11.3	all	66	11.8	6.2
	18	61	67	TP					
		62	68						
			69						

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

Fonte: Adaptado de CanIUseIt(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:

- **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 código 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 código 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 código 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. Por esse motivo serão detalhados projetos desenvolvidos com conceitos almejados no desenvolvimento deste projeto.

2.6.1 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.

2.6.2 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 (FACEBOOK, 2018). 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.

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.

3 PROPOSTA

Para o desenvolvimento do software será usado o modelo incremental. Nesse modelo o software sempre deve estar funcional no fim de cada fase. E após isso o mesmo deve ser incrementado com novas funções.

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, além 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.

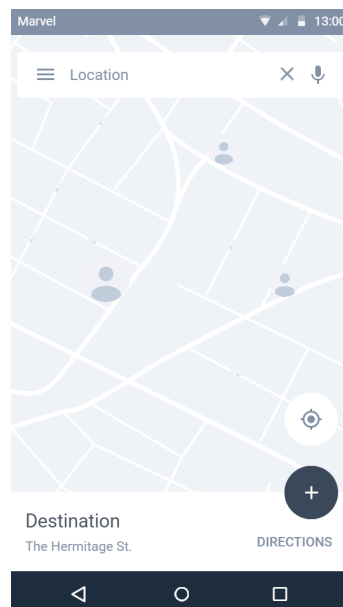


Figura 9: Protótipo de tela da aplicação

Fonte: Autoria Propria

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

Na etapa inicial do projeto, iremos cobrir apenas o 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

Atividades	mar/18	abr/18	mai/18	jun/18	jul/18	ago/18	set/18	out/18	nov/18	dez/18
1. Definição do tema										
2. Estudar o PWA										
3. Desenvolver um app básico em PWA										
4. Ler artigos relacionados										
5. Entrega dos capítulos										
6. Correção da proposta										
7. Desenvolver o <i>front-end</i>										
8. Implementar o mapa										
9. Realizar os testes										
10. Entrega do trabalho final										
11. Correção do trabalho final										

Figura 10: Cronograma

Fonte: Autoria Propria

4 CONCLUSÃO

O desenvolvimento de uma aplicação *Web*, no caso o PWA, possibilitou diferenciar suas particularidades com a de uma aplicação nativa. Evidenciando suas vantagens e desvantagens em relação aos outros modelos de desenvolvimento. Em relação ao tema foi concluído que no momento existem poucas formas de divulgação de eventos, principalmente na forma de aplicação.

REFERÊNCIAS

- ARCHIBALD, J. **Introducing Background Sync**. 2018. Disponível em: <<https://developers.google.com/web/updates/2015/12/background-sync>>.
- BERNERS-LEE, T. **The World Wide Web: Past, Present and Future**. 1996. Disponível em: <<https://www.w3.org/People/Berners-Lee/1996/ppf.html>>.
- BRASIL, G. do. **Ensino superior tem 8,05 milhões de alunos matriculados em 2016**. 2017. Disponível em: <<http://www.brasil.gov.br/educacao/2017/08/ensino-superior-tem-8-05-milhoes-de-alunos-matriculados-em-2016>>.
- CERN. **The birth of the web**. 2018. Disponível em: <<https://home.cern/topics/birth-web>>.
- CORRÊA, E. **JSON Tutorial**. 2017. Disponível em: <<https://www.devmedia.com.br/json-tutorial/25275>>.
- DEVELOPERS, G. **Progressive Web Apps**. 2018. Disponível em: <<https://developers.google.com/web/progressive-web-apps/>>.
- 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/>>.
- GAUNT, M. **Service Workers: uma Introdução**. 2018. Disponível em: <<https://developers.google.com/web/fundamentals/primers/service-workers/?hl=pt-br>>.
- 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.
- TRIVAGO. **Trivago**. 2018. Disponível em: <<https://www.trivago.com.br/>>.
- 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>>.