

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. 26 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 ao mesmo tempo 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 a descoberta de eventos no âmbito universitário. Estes eventos poderão ser descobertos por qualquer usuário próximo de sua criação, 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	7
2.4	MANIFEST	9
2.5	SERVICE WORKER	10
2.6	ARQUITETURA	14
2.7	LIMITAÇÕES	15
2.8	TRABALHOS RELACIONADOS	16
2.8.1	Flipkart	16
2.8.2	Trivago	16
2.8.3	Facebook	17
3	PROPOSTA	19
3.1	APLICAÇÃO	19
3.2	MODELO DE DESENVOLVIMENTO	22
3.3	CRONOGRAMA	22
4	CONCLUSÃO	24
	REFERÊNCIAS	25

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 tornar a divulgação destes eventos mais ampla, alcançando deste modo um número maior de pessoas.

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 *e-mail* cadastro, o mesmo estar errado, a não checagem do *e-mail*, 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 eventos 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*), tecnologia esta que será melhor detalhada na seção 2.3. 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. O Capítulo 3 mostra a proposta de desenvolvimento da aplicação e o modelo de desenvolvimento junto com o cronograma. Já o Capítulo 4 mostra o que foi concluído com esse estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentado as tecnologias utilizadas no desenvolvimento da aplicação. Na seção 2.1 será mostrado os o conceito de desenvolvimento *Web* atual, na seção 2.2 será detalhado os tipos de desenvolvimento para aplicações móveis. O *PWA*, assim como suas tecnologias será mais detalhado nas seções 2.3, 2.4, 2.5. Na seção 2.6 será detalhado em qual arquitetura a aplicação será desenvolvida. Na seção 2.7 será mostrado algumas das limitações existentes em um *PWA*, e na seção 2.8 serão detalhados trabalhos relacionados com a aplicação.

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 é usado as 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 desenvolvido 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).

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).

O Java Script 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 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 desenvolvedor 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 Java Script, para executar diretamente no navegador e podem ter acesso a recursos como geolocalização, armazenamento local *off-line* 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* existe 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 Java Script 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

O PWA é um modelo de desenvolvimento *Web*. Com as suas tecnologias é possível desenvolver uma aplicação que funcione como uma página *Web*, e uma aplicação móvel, que pode ser instalada.

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

1. Progressivo: Deve funcionar para todo e qualquer tipo de usuário, independentemente 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 *off-line*, e aprimorada para funcionar em redes de baixa qualidade. Um exemplo para isso é um site de notícias, onde mesmo estando *off-line*, o usuário poderia ter acesso as notícias previamente carregadas 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 (DEVELOPERS, 2018), mantendo assim a sua segurança. Este protocolo é implementado com uma camada a mais em relação ao HTTP, garantindo assim uma maior segurança a 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 2.5.
8. Reenvolvente: Deve permitir o uso de notificações *push*. Portanto será possível realizar notificações independente do dispositivo em que o usuário esteja usando.
9. Instalável: Permite que o usuário instale a aplicação, porém sem ser preciso acessar alguma loja de aplicativos, como a Play Store (GOOGLE, 2018) ou a App Store (APPLE, 2018). A instalação aparecerá para o usuário em forma de notificação, que decidirá se deseja ou instalar a aplicação.
10. Acessível via link: 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*, serão utilizados 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 é exemplificado um arquivo *manifest.json*, e suas principais regras (KINLAN, 2018):

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

```
"name": "PWA Aplicação",
"short_name": "A-PWA",
"theme_color": "#09ffla",
"background_color": "#000000",
"display": "fullscreen",
"scope": "/",
"start_url": "/pwa-aplicação",
"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 utilizado.
- **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 Java Script 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 *off-line* 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 (JOUPI, 1990). Estes dados podem ser recuperados e mostrados ao usuário enquanto o mesmo estiver *off-line*. 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*.

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 5: 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). Na figura 5 é mostrado alguns navegadores *Web*, assim como suas versões, e se oferecem suporte ao *Service Worker*.

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 é atribuído um nome à variável *cache*, e pra ele é passado o horário em que o site foi gerado, e nas linhas seguintes é definido quais arquivos serão salvos no *cache*. 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))
      );
    })
  );
});
```

A função apresentada no código 2.4 tem o objetivo de excluir arquivos cache que estão desatualizados. Desta forma garantindo que o 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, caso não exista será possível redirecionar o usuário para uma outra página, que pode ser uma página *off-line* previamente carregada.

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 criada 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 *off-line* do Facebook (FACEBOOK, 2018). 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 exemplo 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 é possível 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 ARQUITETURA

O sistema será desenvolvido seguindo o modelo da arquitetura *MVC* (*Model-View-Controller*). Nesta arquitetura a aplicação é dividida em três camadas: a camada de interação do usuário (*View*), a camada de manipulação de dados (*Model*) e a camada de controle (*Controller*)(DEVELOPERS, 2017).

O *Model* é o responsável manipulação de dados, realizando a leitura, escrita e validação. O *View* é a camada responsável por realizar a interação com o usuário, apenas exibi os dados, no caso da aplicação a exibição será feita pelo *html*. Por último existe a *Controller*, responsável por receber as requisições do usuário, controlando qual *View* será mostrada (DEVELOPERS, 2017).

Na figura 6 é mostrado um exemplo de como as camadas se comunicam. Em um caso de *login* de usuário, a *View* irá requisitar os dados com a *Controller*, que por sua vez irá verificar com a *Model* se os dados são validos. Com a validação da *Model*, a *Controller* irá enviar os dados para a *View*, que será responsável por carregar o perfil do usuário. Portanto, toda camada terá o seu papel na execução da aplicação.

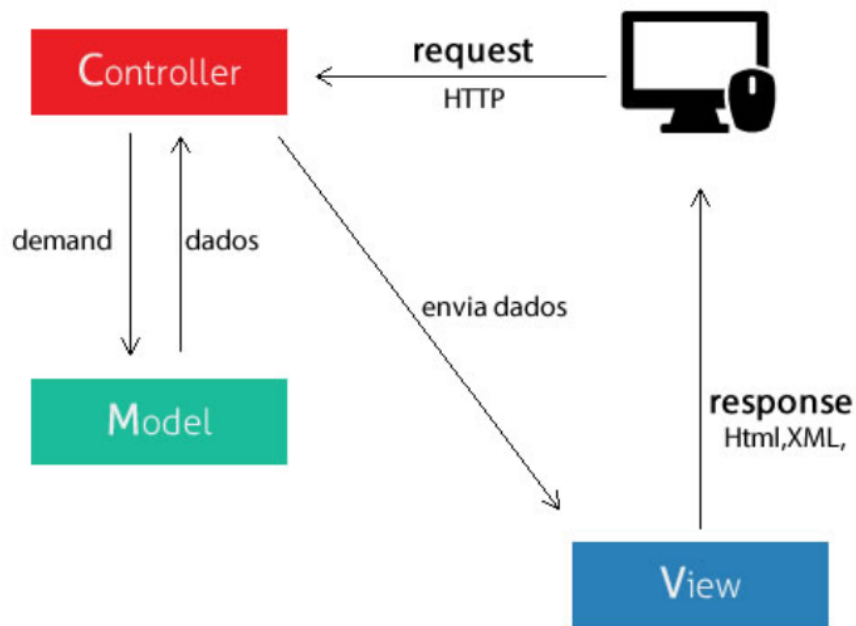


Figura 6: Exemplo de uma arquitetura MVC

Fonte: Adaptado de (RAMOS, 2015)

2.7 LIMITAÇÕES

Ao se comparar o *PWA* com uma aplicação nativa, o *PWA* irá apresentar algumas limitações, que são (XAVIER, 2017):

- A aplicação não estará disponível nas lojas de aplicativos, podendo não alcançar o grande público.
- A aplicação *PWA* não tem acesso a todas funções dos dispositivos moveis, como o *Bluetooth* e o leitor de impressão digital.
- Não são todas as versões de navegadores que suportam o *PWA*.

Porém as maiores limitações para a aplicação *PWA* acontece no sistema iOS, isso ocorre devido à falta de suporte oferecido pela Apple, que só foi adicionar um suporte parcial para o *PWA* na versão 11.3 do iOS. Esta falta de suporte acaba por gerar novas limitações em sistemas iOS, que são (MICROSOFTBLOG, 2018):

- Não existe suporte para notificações *push*.

- Não tem acesso a informações pessoais, como a agenda de contatos e a localização do usuário.
- Caso o usuário deixe de usar a aplicação por determinado tempo, a mesma será excluída do sistema. Porém o ícone continuará disponível na tela inicial, e caso o usuário volte a utilizá-lo será realizado o *download* da aplicação novamente.

2.8 TRABALHOS RELACIONADOS

No momento já existem projetos que compartilham das mesmas ideias deste trabalho, porém de formas separadas. Portanto as seções 2.7.1, 2.7.2 e 2.7.3 detalharão o uso de outras aplicações que compartilham o mesmo conceito da aplicação proposta.

2.8.1 FLIPKART

Flipkart é um *e-commerce* indiano, sendo considera em 2015 o maior site do segmento (DEVELOPERS, 2016). 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 principais 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
- Uma taxa de 70% dos usuários utilizaram a aplicação após a instalarem.

Dados mostravam que 63% dos usuários do *Flipkart* acessavam o site via 2G, por este motivo era necessário que o sistema fosse rápido (LITE, 2015). Para diminuir o tempo de carregamento foram usados *services workers*, possibilitando ao usuário que buscasse os produtos de forma *offline* (LITE, 2015).

2.8.2 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 este 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* é possível observar o uso de algumas de suas recomendações. Como por exemplo o fato da aplicação ser instalável, a responsividade em diferentes tipos de aparelhos, e o uso do *https* para reforçar a segurança. A instalação irá ocorrer de forma bem mais simples para o usuário, pois quando ele acessar a página via navegador irá aparecer uma notificação perguntando se ele deseja instalar a aplicação. Um exemplo disto pode ser visto na figura 7.

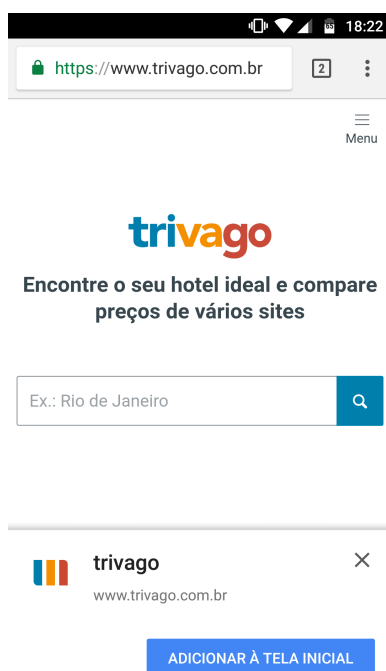


Figura 7: Exemplo da notificação de instalação

Fonte: Autoria Propria

2.8.3 FACEBOOK

O Facebook é a maior rede social do planeta, tendo atualmente aproximadamente 2,1 bilhões de usuários (HOOTSUITE, 2018). Apesar de não ser um PWA, ele contém uma característica muito semelhante à 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

Neste capítulo será apresentado detalhes e funcionamento esperado para a ferramenta proposta. Na seção 3.1 será mostrado as funcionalidades esperadas e seus protótipos de tela, na seção 3.2 será detalhado o método de desenvolvimento da aplicação e na seção 3.3 está o cronograma do desenvolvimento.

3.1 APLICAÇÃO

O primeiro passo do desenvolvimento será construir o *front-end* da aplicação, que se trata da *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. Um protótipo de tela com o exemplo desta página pode ser visto na figura 9.

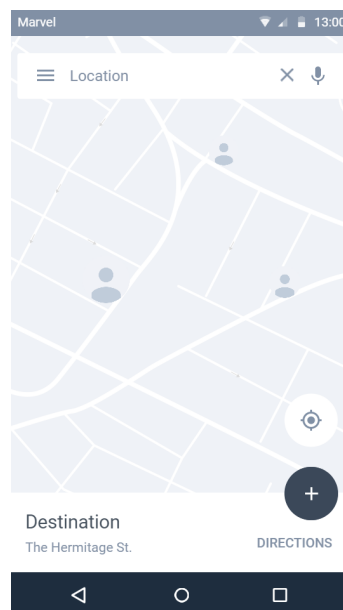


Figura 9: Protótipo de tela da aplicação

Fonte: Autoria Propria

Este 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. O exemplo do *layout* do menu pode ser visto na imagem 10, e o cadastro de eventos na imagem 11.

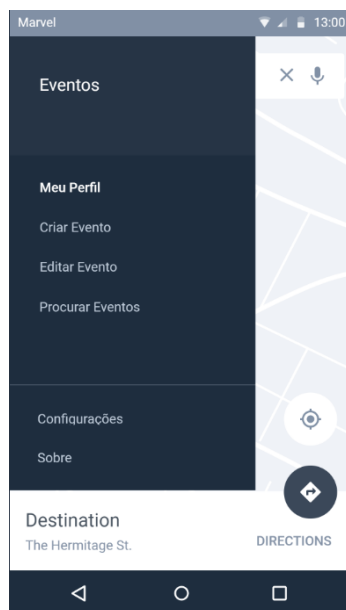


Figura 10: Protótipo de tela do menu da aplicação

Fonte: Autoria Propria

Na segunda etapa será implementado o campo de busca, com ele será possível procurar locais, tendo acesso a eventos próximos a ele. Já no menu teremos algumas informações dos usuários, locais recentes e a opção de criar um evento. Outro modo de se criar o evento e clicando no botão rápido existente. Para a criação dos eventos será necessário adicionar algumas informações, como o local, o nome, e a localização em que o evento irá ocorrer, e o usuário precisará estar logado no sistema. Este *login* será realizado utilizando uma conta Google, graças a sua *API*. Na imagem 12 consta o diagrama de caso de uso desta funcionalidade.

A última 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 selecioná-lo. Isso



Marvel 13:00

Nome do Evento

Localização

Horário

Observações

Criar

1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123 , .

Figura 11: Protótipo de tela do cadastro do evento

Fonte: Autoria Propria

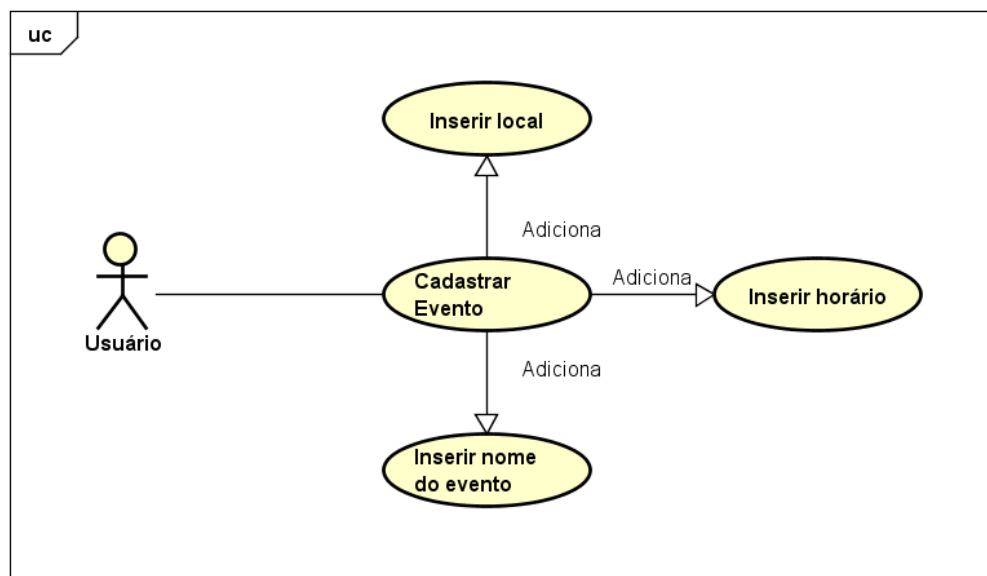


Figura 12: Diagrama de Caso de Uso

Fonte: Autoria Propria

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, a aplicação irá funcionar apenas no campus de Cornélio

Procópio da Universidade Tecnológica Federal do Paraná.

3.2 MODELO DE DESENVOLVIMENTO

Para o desenvolvimento da aplicação será usado o modelo incremental. Neste modelo é desenvolvido uma versão inicial do *software*, e a partir dela será adicionado novas funcionalidades até que o *software* esteja finalizado (SOMMERVILLE, 2003).

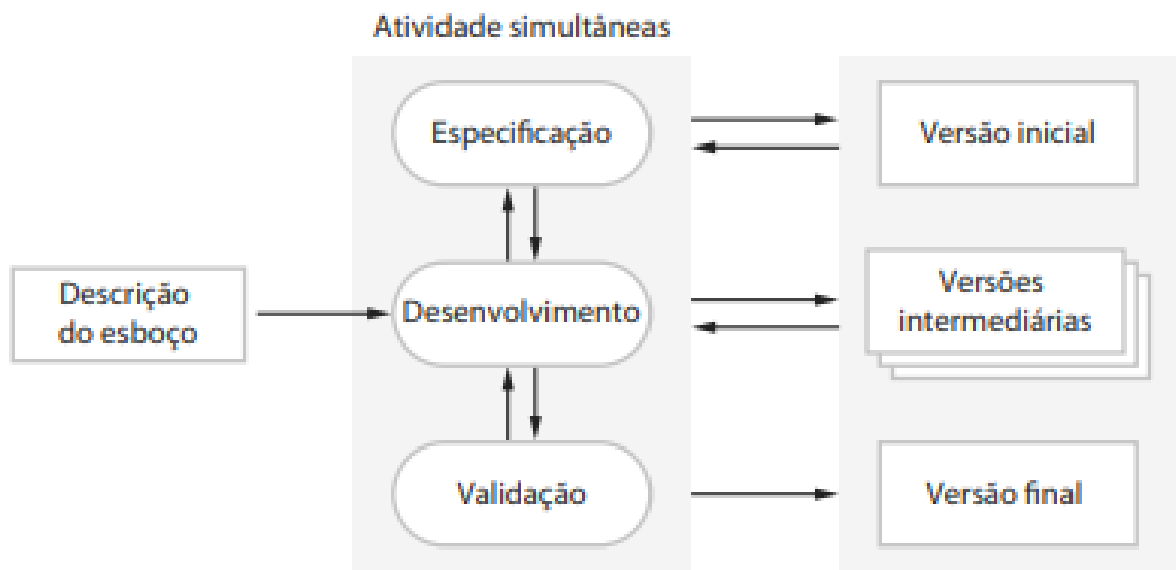


Figura 13: Modelo Incremental

Fonte: (SOMMERVILLE, 2003)

Na figura 13 é dado um exemplo de como o modelo incremental funciona. No exemplo existem atividades acontecendo de forma simultânea, no caso: especificação, desenvolvimento e validação. Durante as atividades são geradas versões do programa, que acabam sofrendo modificações, e assim são incrementadas para uma versão melhor. Portanto ao fim de cada processo o *software* terá a sua versão melhorada, até que a sua versão final seja concluída.

3.3 CRONOGRAMA

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 14: 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 público o convite para todos que estiverem próximos ao evento. Combinado com o uso do PWA torna a aplicação mais acessível para os usuários, podendo acessá-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>>.
- APPLE. **App Store**. 2018. Disponível em: <<https://www.apple.com/br/ios/app-store/>>.
- ARCHIBALD, J. **Introducing Background Sync**. 2018. Disponível em: <<https://developers.google.com/web/updates/2015/12/background-sync>>.
- 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. **MVC Architecture**. 2017. Disponível em: <https://developer.chrome.com/apps/app_frameworks>.
- 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/>>.
- 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>>.
- GOOGLE. **Play Store**. 2018. Disponível em: <<https://play.google.com/store?hl=pt>>.
- HOOTSUITE. **Active users of key global social platforms**. 2018. Disponível em: <<https://hootsuite.com/pt/>>.
- 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>>.

LITE, D. F. New Progressive Web App helps Flipkart boost conversions 70%. 2015.

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

MICROSOFTBLOG. **iOS da Apple ganha suporte aos PWA**. 2018. Disponível em: <<https://www.windowsteam.com.br/ios-da-apple-ganha-suporte-aos-pwa/>>.

RAMOS, A. **O que é MVC?** 2015. Disponível em: <<https://tableless.com.br/mvc-afinal-e-o-que/>>.

SOMMERVILLE, I. **Engenharia de Software**. [S.l.: s.n.], 2003. 1–6 p. ISBN 9788579361081.

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>>.

XAVIER, L. **Vamos falar sobre os Progressive Web Apps, os PWAS?** 2017. Disponível em: <<http://www.proxxima.com.br/home/proxxima/how-to/2017/07/05/vamos-falar-sobre-os-progressive-web-apps-os-pwas.html>>.