



FACULDADE LOURENÇO FILHO
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Pedro Cruz

PWA

Fortaleza, Ceará

2019

Pedro Cruz

PWA

Orientador: Prof Msc. Fred Viana

Co-Orientador:

Fortaleza, Ceará

2019

Lista de ilustrações

Figura 1 – Modelo Cliente-Servidor	12
Figura 2 – Modelo Multicamada	13
Figura 3 – Aplicação Monolítica	17
Figura 4 – Representação de módulos monolítico e micro serviços	19
Figura 5 – Motorola DynaTAC	22
Figura 6 – IBM Simon	22
Figura 7 – Nokia 9000	23
Figura 8 – Primeiro IPhone	24
Figura 9 – Ciclo de Vida do Service Worker	29
Figura 10 – Adicionar a tela inicial	30
Figura 11 – Telas Customizadas	33
Figura 12 – Tela Inicial	39
Figura 13 – Tela de Criação de Post	40
Figura 14 – Tela de Ajuda	40
Figura 15 – Adicionando App	41
Figura 16 – Tela Offline	45
Figura 17 – Habilitar Notificações	49
Figura 18 – Notificação Pré Configurada	53

Listas De Quadros

6.1 Arquivo Manifesto	42
6.2 Service Worker	43
6.3 Armazendo Post	44
6.4 Página offline customizada	46
6.5 Sincronização Em Background	47
6.6 Limpando Dados	48
6.7 Verificando suporte notificações	49
6.8 Pedindo permissão ao usuário	50
6.9 Configurando Push Notifications	50
6.10 Notificação Pré configurada	52
6.11 Notificação personalizada	53

Listas de abreviaturas e siglas

API Interface de Programação de Aplicações

CERN Organização Europeia para a Pesquisa Nuclear

CSS *Cascading Style Sheets*

DOM Modelo de Objeto de Documento

HTML Linguagem de Marcação de Hipertexto

HTTP Protocolo de Transferência de Hipertexto

JS JavaScript

JSON Notação de Objetos JavaScript

PWA *Progressive Web App*

SGBD Sistema Gerenciador de Banco de Dados

SO Sistema Operacional

UI *User Interface*

W3C *World Wide Web Consortium*

WWW *World Wide Web*

PDA Assistente Pessoal Digital

Sumário

1	INTRODUÇÃO	7
2	ARQUITETURAS WEB	8
2.1	História	8
2.2	Modelo de Multicamadas	10
2.2.1	Camada de Apresentação	14
2.2.2	Camada de Regra de Negócio	14
2.2.3	Camada de Dados	14
2.3	Vantagens	14
2.3.1	Clientes Leves	15
2.3.2	Facilidade de Redistribuição	15
2.3.3	Modularização	15
2.3.4	Economia de conexões no servidor	15
2.3.5	Independência de localização	16
2.3.6	Escalabilidade	16
2.4	Estilos Arquitetônicos	16
2.4.1	Arquitetura Monolítica	16
2.4.2	Arquitetura de Micro serviços	18
3	ARQUITETURA DOS DISPOSITIVOS MÓVEIS	21
3.1	Evolução dos Dispositivos Móveis	21
3.2	iOS	24
3.3	Android	25
4	CONHECENDO AS PROGRESSIVES WEB APPS	26
4.1	Características	27
4.1.1	Confiável	27
4.1.2	Rápido	29
4.1.3	Integração e Engajamento	29
4.2	Vantagens	31

4.2.1	Fácil Instalação e Atualização	31
4.2.2	Custos Reduzidos de Desenvolvimento e Suporte	31
4.2.3	Rápido, Leve e Seguro	32
4.2.4	Não necessita de nenhuma loja de aplicativos	32
4.2.5	Customizável	32
4.3	Desvantagens	33
4.3.1	Funcionalidades Limitadas	33
4.3.2	Limitações para o iOS	33
5	ANÁLISE COMPARATIVA ENTRE PWAS E APLICAÇÕES MÓVEIS NATIVAS	34
5.1	Aplicação Nativa	34
5.2	Comparativo PWA e Aplicação nativa	35
5.2.1	Criação da aplicação e Distribuição	35
5.2.2	Instalação	36
5.2.3	Engajamento do Usuário	36
5.2.4	Funcionamento Offline	37
6	ESTUDO DE CASO	38
6.1	Ferramentas Utilizadas	38
6.2	Conhecendo o Aplicativo	39
6.3	Configurações Iniciais Do Aplicativo com PWA	41
6.3.0.1	Arquivo de Manifesto	41
6.4	Recursos Avançados do Service Worker	44
6.4.1	Ações Customizadas	45
6.4.2	Sincronização de Dados em Background	46
6.4.3	Notificações	48
7	CONCLUSÃO	54
	REFERÊNCIAS	55

1 Introdução

O uso de dispositivos móveis ocupa grande parte de nossas vidas. Verificar os *smartphones* várias vezes por dia tornou-se uma rotina para a maioria de nós. Durante anos, a única maneira das empresas atingirem os usuários de dispositivos móveis era criando um aplicativo móvel nativo ou híbrido. Hoje, porém, com a bordagem de *Progressive Web App (PWA)*s tornou-se uma solução alternativa para empresas de qualquer tamanho para atrair usuários móveis ativos.

Diante deste cenário, é necessário avaliar se uma aplicação para dispositivos móveis, irá fazer uso de um aplicativo nativo, ou se uma abordagem com PWA é o suficiente para atender os requisitos.

Este trabalho tem como objetivo geral, abordar as PWAs, apresentando suas vantagens e desvantagens com relação a aplicativos nativos, com o intuito de servir como base para saber qual abordagem usar ao desenvolver uma aplicação. Como objetivos específicos têm-se: abordar conceitualmente e funcionalmente as PWAs, os impactos que elas podem causar no desenvolvimento como um todo, já que podemos está presenciando o fim de aplicativos nativos, e por fim configurar um ambiente de desenvolvimento onde será criado uma PWA, fazendo uso de tecnologias como IndexedDb, Firebase e JavaScript.

Esse trabalho está inicialmente organizado em 7 partes contando com essa seção, o capítulo 2 apresenta como funciona a Arquitetura WEB, neste tópico será falado sobre a evolução da *internet*, os modelos de multicamadas e os principais estilos arquitetônicos. No capítulo 3, é abordado a evolução dos dispositivos móveis, e seus principais sistemas operacionais disponíveis no mercado. A quarta parte desse trabalho, foca sobre as PWAs, suas características, vantagens e desvantagens. No capítulo 5, é feito um estudo comparativo entre as aplicações móveis nativas e as PWAs. A penúltima parte desse trabalho foca na configuração de um ambiente e desenvolvimento de uma aplicação PWAs. E o capítulo 7 é destinado às conclusões que foram conseguidas durante o desenvolvimento desse trabalho.

2 Arquiteturas WEB

Uma aplicação *web* é um *software* que roda no lado do cliente, ou seja, na máquina do próprio usuário, com essa abordagem é necessário apenas que o cliente possua um *browser*(navegador), independente de seu sistema operacional para acessar uma aplicação *web*. Com isso os desenvolvedores não precisam desenvolver diversas versões de uma mesma aplicação, para sistemas operacionais diferentes.

Tendo isso em vista, este capítulo visa apresentar um breve histórico da evolução da arquitetura *web*, apresentará o modelo de multicamadas e por fim demonstrará seus estilos arquitetônicos.

2.1 História

A Internet mudou completamente quando Tim Berners-Lee, a fim de solucionar os problemas de comunicações da Organização Europeia para a Pesquisa Nuclear (CERN), propôs que se usasse um sistema de comunicação em hipertexto. Esse sistema acabou agradando os gerentes do CERN, e no ano seguinte, ele foi implantado com o nome de *World Wide Web (WWW)*. Todos os serviços da *Internet* se renderam ao poder da *Web* e à linguagem Linguagem de Marcação de Hipertexto (HTML), que a sustenta. Até o serviço de correio eletrônico, campeão de tráfego na *Internet* por muitos anos, que por muito tempo exigia aplicações específicas, separadas do *browser*, hoje é lido dentro de um *browser*, através de páginas HTML (ROCHA, 1999).

A medida que o HTML foi ganhando fama, quem a usava não queria apenas uma simples apresentação de texto estruturada, queriam usar cores, imagens e técnicas de design mais avançado, suas páginas HTML acabavam ficando com vários códigos de estilos. Só que a manutenção seria o maior problema dessa abordagem, se um componente mudar de estilo em apenas uma página a alteração seria simples, mas se fosse em várias páginas, a manutenção já seria bastante penosa. Misturar estilo e estrutura não era mais interessante, e foi assim que em 1995, Håkon Wium Lie e Bert Bos apresentaram a proposta do *Cascading Style Sheets (CSS)* que logo foi apoiada pela *World Wide Web Consortium (W3C)*. A ideia geral era, utilizar HTML somente

para estruturar o *website* e a tarefa de apresentação fica com o CSS disposto em um arquivo separado com o sufixo .css ou no próprio HTML demarcado pelas tags <style> (PEREIRA, 2009).

O ano de 1995 é muito importante para a *internet*, a Netscape Communications apresenta o *Javascript*, uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe, mais conhecida como a linguagem de *script* para páginas *Web*, mas usada também em vários outros ambientes sem *browser* como *node.js*, *Apache CouchDB* e *Adobe Acrobat*. O *Javascript* é uma linguagem de script multi-paradigma, baseada em protótipo que é dinâmica, e suporta estilos de programação orientado a objetos, imperativo e funcional (MDN, 2018), e possibilitou que os desenvolvedores pudessem criar componentes dinâmicos, e assim possibilitando uma melhora na *interface* do usuário. Com o *javascript* a *internet* acabou se tornando mais performática e produtiva, porque os dados não precisavam mais ser enviados ao servidor para se gerar toda a página HTML. Juntos o *Javascript*, HTML e CSS são às três tecnologias mais populares para a produção de conteúdo na internet (DEVSARAN, 2016).

Já no ano de 1996 a Macromedia lançaria o *Flash*, que é uma plataforma multimídia de desenvolvimento para aplicações que contenham animações, áudio e vídeo, bastante utilizada na construção de anúncios publicitários e páginas *web* interativas. O *Flash* pode manipular vetores e gráficos para criar textos animados, desenhos, imagens e até *streaming* de áudio e vídeo pela *internet*. O *Flash* ganhou bastante popularidade entre os programadores e desenvolvedores *web* por permitir um desenvolvimento rápido de aplicações com alta qualidade e integração transparente com outras categorias de conteúdo (CIPOLI, 2018).

Com o *Flash* os programadores puderam dar ao usuário, uma experiência mais rica através de animações de áudio e vídeo. Além disso, as interações feitas no *Flash* eram todas no lado do cliente, não precisando se comunicar com o servidor. Era comum que os *sites* antes dos anos 2000, usasse conteúdo multimídia incorporado, o que por muitas vezes gerava uma poluição visual muito grande, e como consequência disso a popularidade do *Flash* acabou caindo, as páginas acabaram ganhando um visual padrão, o acesso do usuário não era mais interrompido por áudio, vídeos e anúncios inesperados, a medida que seu uso foi diminuindo a página ficava mais leve e o uso de

rede também (MARQUES, 2016).

Apesar de seu uso ir diminuindo gradualmente, o *Flash* por muitos anos foi usado para criação de jogos e aplicativos interativos para dispositivos móveis. No ano de 2008 o primeiro rascunho público do HTML5 é lançado pelo WHATWG (THOMS, 2017), essa tecnologia ganhou os holofotes, foi quando em 2010 o CEO da Apple Inc., Steve Jobs emitiu uma carta pública intitulada "Reflexões sobre o Adobe Flash", onde ele conclui que o desenvolvimento do HTML5 tornaria o *Flash* não mais necessário para exibir qualquer conteúdo *web*. Depois disso o fim do *Flash* era questão de tempo, muitos *browsers* já não o utilizavam mais, e em julho de 2017, em comunicado oficial a Adobe, anunciou que vai encerrar definitivamente o suporte para sua tecnologia Flash até 2020. A companhia disse precisar destes últimos anos para incentivar criadores de conteúdo e desenvolvedores a utilizarem novas plataformas, principalmente formatos de código aberto e que funcionem também em celulares e *tablets* (CANALTECH, 2017).

Em 1999, o conceito de aplicação *web* apareceu na linguagem Java. Mais tarde, em 2005, o Ajax foi apresentado por Jesse James Garrett em seu artigo "Ajax: uma nova abordagem para a aplicação Web". Esta nova técnica de desenvolvimento, possibilitou a criação de aplicações *web* assíncronas, ou seja, o fluxo do código não é interrompido até que a resposta seja obtida. Ao invés disso, após realizar a requisição, a resposta é obtida em um momento posterior, de forma independente, e tratada por uma função (chamada função de *callback*). Com isso era possível enviar dados para o servidor e recuperá-los sem interferir na navegação de uma página específica, não precisando baixar a página inteira (MARQUES, 2016).

2.2 Modelo de Multicamadas

Modelo de uma camada

Esse modelo foi fortemente usado durante os anos 1960 até meados dos anos 80, também chamado de sistemas centralizados ou de arquitetura uni processada, o modelo de uma camada era caracterizado por manter todos os recursos do sistema (banco de dados, regras de negócios e interfaces de usuário) em computadores de grande porte, os conhecidos mainframes. Os terminais clientes não possuíam recursos

de armazenamento ou processamento, sendo conhecidos como terminais burros ou mudos. Nesta arquitetura, o mainframe tinha a responsabilidade de realizar todas as tarefas e processamento (GRANATYR, 2007). As aplicações eram escritas em um único módulo ou camada monolítica, com programas e dados firmemente entrelaçados. Tal integração estreita entre programa e dados dificultava a evolução e reuso de componentes. Cada desenvolvedor de aplicação escolhia como estruturar e armazenar os dados e usava-se frequentemente técnicas para minimizar o caro armazenamento e manipulação em memória principal (MARTINS, 2012).

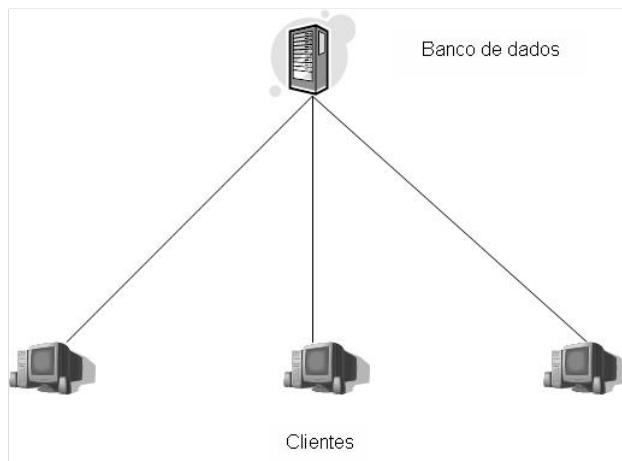
Modelo de duas camadas

Também conhecido como modelo cliente-servidor, seu uso se deu bastante durante os anos 1980 à 1990, durante essa época novas tecnologias foram desenvolvidas e adotadas como, *softwares* para gerenciamento de banco de dados relacionais (SGBD), a utilização de rede locais interligando microcomputadores departamentais, o início do paradigma da programação orientada a objetos, e a redução dos custos do hardware, tornando possível a massificação da computação pessoal (MARTINS, 2012). Também devido à grande expansão das redes de computadores, os métodos para desenvolvimento de *software* foram aos poucos evoluindo para uma arquitetura descentralizada, na qual não somente o servidor era responsável pelo processamento, mas as estações clientes também assumem parte desta tarefa. Dentro deste contexto que surgiu o modelo de duas camadas, justamente com o objetivo de dividir a carga de processamento entre o servidor e as máquinas clientes (GRANATYR, 2007). A Figura 1 é uma representação do modelo cliente-servidor.

Na camada do cliente, é onde o usuário interage com o sistema, ela é responsável por prover uma *User Interface* (UI) agradável e de fácil acesso para o usuário possa manipular o sistema. Mas apesar de prover a UI, a camada do cliente não se restringe a isso, regras de negócio também podem ser implementadas, assim diminuindo a complexidade no servidor. A camada do cliente, de acordo com a sua complexidade pode ser definida de duas formas segundo (MARTINS, 2012). Sendo elas:

- Cliente Gordo

Figura 1 – Modelo Cliente-Servidor



Fonte: (MARTINS, 2012)

- Maior complexidade de regras de negócio
- Menos processamento para o servidor
- Possivelmente mais tráfego na rede
- Cliente é mais sensível a mudanças
- Cliente Magro
 - Menor complexidade de regras de negócio
 - Mais processamento para o servidor
 - Menor tráfego na rede
 - Manutenção mais simples

Com o modelo de duas camadas foi possível que *softwares* de terceiros tivessem acesso ao banco de dados, ou seja, com isso os *softwares* poderiam ser diferentes para cada usuário ou setor, afim de melhor atender às suas necessidades. Outra vantagem é a questão do custo-benefício, as estações dos clientes são mais baratas do que um servidor (MARTINS, 2012).

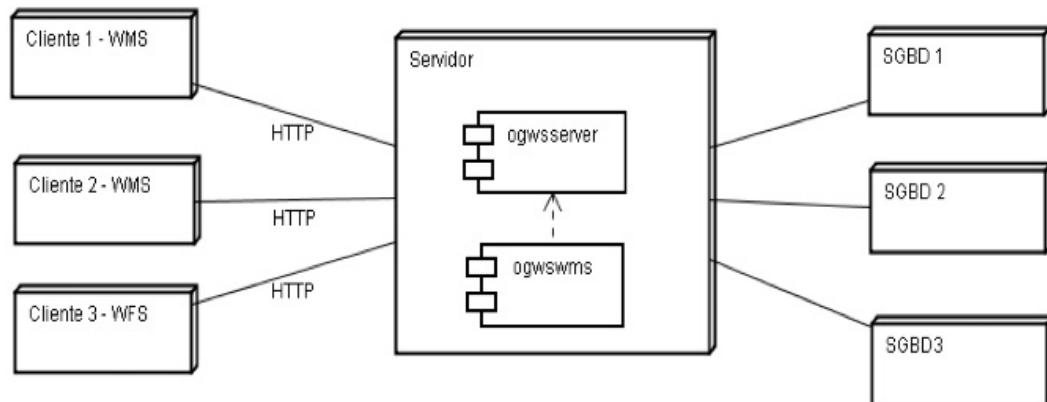
O modelo de duas camadas também pode apresentar algumas desvantagens, como sua aplicação é dividida em partes, acaba acarretando em um *software* mais complexo, com novos cenários a serem tratados. A comunicação do cliente com o servidor se dá por meio da rede, com isso dados sensíveis serão trafegados na rede e precisam de um cuidado maior com a criptografia (ROCHA, 1999).

Modelo de Multicamadas

Modelo de multicamadas ou cliente servidor de múltiplas camadas, se consolida a partir dos anos 90 como resposta a popularização da internet e melhoria das tecnologias de redes, sendo uma evolução do modelo de duas camadas. Ele tem como propósito que uma aplicação cliente não realizasse comunicação direta com o banco de dados, no meio do caminho haveria uma ou mais camadas, que elas sim se comunicariam com o banco de dados. A ideia básica é distribuir o processamento da aplicação em várias máquinas, evitando a sobrecarga sobre uma única camada, como ocorria no modelo cliente-servidor. Com a distribuição da carga de processamento em diversas máquinas é possível melhorar o desempenho e compartilhar recursos, utilizando-os como se fossem recursos locais, característica conhecida como transparência de uso (MARTINS, 2012). Com isso a aplicação pode ser dividida em pequenos pedaços, cada um com sua responsabilidade. Além dessas vantagens, há uma compensação no custo em relação ao desempenho, a possibilidade de aumento de escala e expansão da rede sem perda de qualidade, melhora na robustez em função da distribuição dos serviços em mais de uma máquina, e muitos outros benefícios (MARTINS, 2012).

Em uma aplicação que faz uso do modelo de multicamadas, faz-se necessário ao menos três camadas: camada de apresentação, camada de regras de negócio e a camada de dados. A Figura 2 apresenta o esquema de um sistema multicamada.

Figura 2 – Modelo Multicamada



Fonte: (MACEDO, 2012)

2.2.1 Camada de Apresentação

A camada de apresentação fica fisicamente localizada na estação cliente e é responsável por fazer a interação do usuário com o sistema. É uma camada bastante leve, que basicamente executa os tratamentos de telas e campos e geralmente acessa somente a segunda camada, a qual faz as requisições ao banco de dados e devolve o resultado. É também conhecida como cliente, regras de interface de usuário ou camada de interface (GRANATYR, 2007).

2.2.2 Camada de Regra de Negócio

Também conhecida como servidor de aplicação, lógica de negócios ou camada de acesso a dados, essa camada é a responsável por intermediar a comunicação entre a camada de apresentação com a camada de dados, só ela tem acesso a camada de dados. O servidor de aplicação é, geralmente, uma máquina dedicada e com elevados recursos de hardware, uma vez que nele é que ficam armazenados os métodos remotos (regras de negócios) e é realizado todo o seu tratamento e processamento. (GRANATYR, 2007).

2.2.3 Camada de Dados

Também chamada de camada de banco de dados, essa camada é onde se localiza o Sistema Gerenciador de Banco de Dados (SGBD), ela é responsável por receber requisições da camada de regra de negócio, interpretá-las e assim executá-las no banco de dados (GRANATYR, 2007).

2.3 Vantagens

O modelo de multicamadas apresentou diversas vantagens em relação ao modelo de duas camadas, são essas as que mais se destacam:

2.3.1 Clientes Leves

Diferentemente do modelo de duas camadas, onde a regra de negócio era dividida tanto na camada do cliente quanto na do servidor, aqui a camada intermediária é quem fica a cargo disso, na camada de apresentação será apenas para visualização de dados, além de possuir possíveis tratamentos de campos e telas (GRANATYR, 2007).

2.3.2 Facilidade de Redistribuição

As estações clientes acessam a mesma camada intermediária, sendo assim, quando houver novas implementações ou alterações nas regras de negócios, será refletido para todas as estações clientes (GRANATYR, 2007).

2.3.3 Modularização

A modularização refere-se a separar a lógica do negócio e regras de acesso ao banco de dados da camada de apresentação. Desta maneira, várias aplicações clientes podem compartilhar as mesmas regras, que ficam encapsuladas em uma camada de acesso comum. Assim sendo, as regras ficam centralizadas em um único local, ao contrário de em uma aplicação desenvolvida em duas camadas, na qual geralmente existe redundância nestas regras e uma mudança mesmo que pequena acarretará na redistribuição do aplicativo em cada estação cliente (GRANATYR, 2007).

2.3.4 Economia de conexões no servidor

Em um sistema com o modelo de duas camadas, as estações clientes se comunicavam diretamente ao servidor que se localizava o banco de dados, então para cada estação cliente conectada era uma conexão aberta com o banco de dados, sendo que o banco de dados possuí um limite para conexões, tendo isso em vista, no modelo de multicamadas isso não ocorre já que quem se conecta com o banco de dados é o servidor de aplicação, localizado em uma camada intermediária, e uma conexão realizada pelo servidor de aplicação é compartilhada para as estações clientes a ele conectado, sendo assim poderia se ter várias estações clientes requisitando recursos

do banco de dados, mas apenas uma conexão estaria aberta já que quem ficaria responsável pela conexão é o servidor de aplicação (GRANATYR, 2007).

2.3.5 Independência de localização

A localização não é um empecilho para a comunicação entre camadas, a estação cliente pode estar fisicamente distante para acessar as camadas intermediárias (GRANATYR, 2007).

2.3.6 Escalabilidade

No modelo de duas camadas, quando um grande número de estações clientes se conectam ao servidor, acaba ocorrendo uma grande perda de desempenho. Já com o modelo de multicamadas este problema pode ser contornado, já que é possível replicar a regra de negócio em servidores distintos através do balanceamento de carga, isso quer dizer que quando um servidor de aplicação estiver sobrecarregado, outro servidor é acionado para ajudar no controle de conexões, isso também pode ser usado quando um servidor de aplicação parar de funcionar (GRANATYR, 2007).

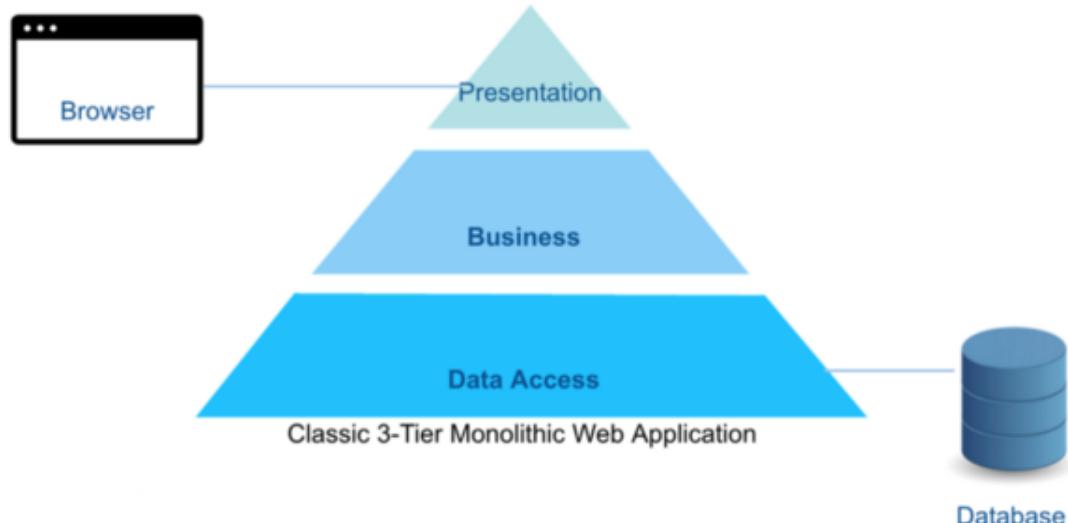
2.4 Estilos Arquitetônicos

Nesta seção será apresentados os estilos arquitetônicos de desenvolvimento de uma aplicação web.

2.4.1 Arquitetura Monolítica

Segundo (SANTOS, 2017), "Uma aplicação monolítica é aquele tipo de aplicação na qual toda a base de código está contida em um só lugar, ou seja, todas as funcionalidades estão definidas no mesmo bloco". Esse bloco geralmente se divide em três partes: apresentação, negócio e dados. A Figura 3 apresenta o modelo dessa arquitetura.

Figura 3 – Aplicação Monolítica



Fonte:(SANTOS, 2017)

Camadas

Apresentação

Camada responsável pela visualização ou interface, que será apresentada para o usuário. "Em uma aplicação web, esta camada contém as páginas HTML com JavaScript (JS) e CSS que serão renderizadas no *browser* de quem as acessar", (SANTOS, 2017).

Negócio

Camada que é responsável pela lógica da aplicação. Segundo (SANTOS, 2017) "Nesta camada geralmente se encontram todas as bases de código, chamadas, Interface de Programação de Aplicações (API)'s e literalmente toda a inteligência do sistema em questão".

Dados

Camada em que se encontram as classes encarregadas pela conexão com o SGBD ou outro tipo de sistema de armazenamento de dados (SANTOS, 2017).

Vantagens

- Fácil deploy da aplicação.
- Não há duplicidade de código.
- Aplicação é desenvolvida usando uma mesma tecnologia.
- Seu desenvolvimento tende a ser mais rápido, por ser uma arquitetura mais simples.
- Fluxo de deploy simples.

Desvantagens

- Ponto único de falha.
- Aumento de tamanho e complexidade ao longo do tempo.
- Falta de flexibilidade, já que usa uma mesma tecnologia.
- Baixa escalabilidade, tendo que copiar toda a aplicação para escalar horizontalmente.
- Alta dependência de componentes.
- Dificuldade de alterações em produção, qualquer mudança se faz necessário, a reinicialização de todo o sistema.
- Demora de aculturamento, um novo desenvolvedor pode ter dificuldades para entender o funcionamento de um componente.

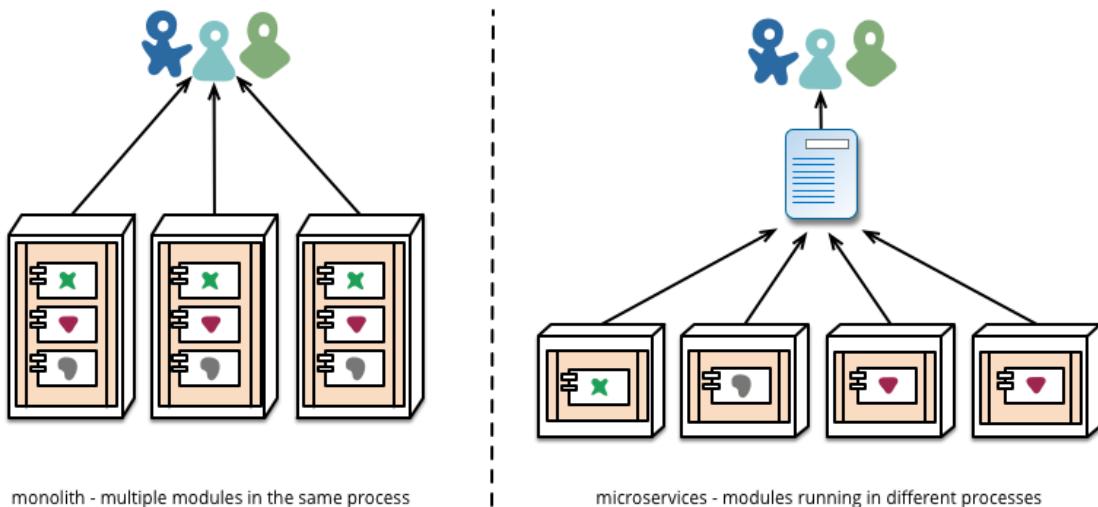
2.4.2 Arquitetura de Micro serviços

Uma arquitetura de micro serviços segundo (FOWLER, 2014):

"é uma abordagem que desenvolve uma aplicação única como uma suíte de pequenos serviços, cada um rodando em seu próprio processo e se comunicando com mecanismos leves, geralmente uma API de recurso Protocolo de Transferência de Hipertexto (HTTP)"

Desse jeito nós separamos uma aplicação monolítica em pequenas aplicações autônomas, ou seja, devem possuir um sistema de deploy automático e independente, além de que cada aplicação possui um conjunto de regras de negócio específico. A Figura 4 mostra a comparação entre a arquitetura monolítica e de micro serviços.

Figura 4 – Representação de módulos monolítico e micro serviços



Fonte: (FOWLER, 2014)

Vantagens

- Arquitetura individual simples.
- Sistemas totalmente independentes.
- Ausência de ponto de falha única.
- Fácil deploy da aplicação e testes unitários.
- Módulos podem usar tecnologias distintas.
- Serviços coesos e desacoplados.
- Facilidade de alterações em ambiente de produção, apenas o serviço específico é necessário ser reinicializado para refletir as alterações.
- Escalabilidade do sistema, serviços que sofrem de alta demanda podem ser replicados individualmente.

Desvantagens

- Desempenho prejudicado pela latência da rede e pelo custo de serialização e deserialização.
- Se a aplicação não for bem documentada, a arquitetura geral tende a ser tornar complexa.
- Falta de planejamento e má execução, a arquitetura pode se tornar uma grande bagunça.
- Repetição de código nos serviços.

3 Arquitetura dos Dispositivos Móveis

Nesse capítulo será abordado o histórico da evolução dos dispositivos móveis e apresentará os Sistema Operacional (SO)s mais famosos que são o IOS e Android, de modo a traçar um paralelo entre eles.

3.1 Evolução dos Dispositivos Móveis

Atualmente, os dispositivos móveis, são imprescindíveis na vida das pessoas. Quem imaginaria que no começo eles que tinham recursos muito limitados, poucas funcionalidades, se restringindo basicamente a serem calculadoras, e hoje, temos praticamente computadores pessoais em nossas mãos, tanto, em relação a funcionalidades quanto na questão de poder de processamento. A medida que, os dispositivos móveis foram evoluindo, novas funcionalidades foram aparecendo, calendário, rádio, jogos, mas foi quando a Apple lançou seu primeiro smartphone, o Iphone, com seu próprio SO o IOS, e a Google lançou o seu SO para smartphones, o Android, que os dispositivos móveis ganharam a fama de hoje. Com esses sistemas abriu-se um leque de oportunidades para empresas e programadores avulsos, à desenvolverem os mais diversos aplicativos.

O marco da telefonia móvel se deu no ano de 1973, quando dois engenheiros da Motorola, realizaram a primeira ligação da história feita a partir de um telefone móvel, o modelo era o então, DynaTAC como mostra a Figura 5. Com seus poucos mais de 30 cm de tamanho e peso de quase 1kg, o seu lançamento só ocorreu no ano de 1983, 10 anos após a demonstração (RIGUES, 2016).

A primeira geração de telefones móveis, teriam as mesmas características do DynaTAC, não sendo tão portáteis, mas era o primeiro passo e a tendência seria que esses aparelhos evoluíssem e aumentasse suas funcionalidades (JORDAO, 2009).

No ano de 1994, a IBM lança o Simon, um aparelho que reunia diversas funcionalidades, dentre elas Assistente Pessoal Digital (PDA) e Fax, e foi um grande passo

Figura 5 – Motorola DynaTAC



Fonte: (PINTEREST, 2019)

para o desenvolvimento dos dispositivos móveis, sendo o precursor da tela *touchscreen*, mas suas medidas ainda eram muito grandes e acabou não sendo tão popular, vendendo apenas 50 mil unidades (RIGUES, 2016). A figura Figura 6 mostra o Simon.

Figura 6 – IBM Simon



Fonte: (RIGUES, 2016)

Na figura Figura 7 é possível ver o Nokia Communicator 9000 que foi um aparelho que fez bastante sucesso, lançado em 1996, foi o primeiro dispositivo móvel comercializado com acesso a internet, além disso reunia diversas funcionalidades como, FAX, SMS, e-mail, agenda de contatos, calculadora, bloco de anotações e outras mais. Suas medidas ainda eram grandes, mas possuía um o recurso de flip, em que

tinha uma segunda tela e um teclado QWERTY (RIGUES, 2016).

Figura 7 – Nokia 9000



Fonte: (PRODNOTE, 2015)

Em janeiro de 2007, Steve Jobs em uma apresentação da Apple, lança um smartphone que mudaria tudo, o iPhone. Com design minimalista, o iPhone dispensou o uso de teclas físicas, tudo seria a base do toque na tela como mostra a Figura 8 (JORDAO, 2009).

Figura 8 – Primeiro iPhone



Fonte: (RIGUES, 2016)

A Apple desenvolveu um SO próprio para o dispositivo, o IOS, um sistema operacional de fácil interação para o usuário que o tornou bastante popular. Além disso, com o iPhone surgiram também os aplicativos, inicialmente vindos apenas de fábrica, e depois comprados pela loja virtual, a App Store (RIGUES, 2016).

Em 2008 surgiria um novo SO que entraria na concorrência com o IOS da Apple, o smartphone T-Mobile G1, era lançado com um SO desenvolvido pela Google, o Android, ele também trazia uma loja de aplicativos, a Android Market que depois passou a ser chamada como é hoje, Google Play. Com o tempo o Android ganhou espaço e hoje a maioria dos smartphones usam o SO da Google (MICROLINS, 2017).

3.2 iOS

Segundo (GARCIA, 2019) "O iOS é um sistema operacional desenvolvido pela Apple que pode ser encontrado no iPhone, iPad e iPod Touch da empresa, visto que os notebooks da empresa utilizando o MacOS e os relógios inteligentes o watchOS."

3.3 Android

De acordo com (BARROS, 2015) "Android é o sistema operacional móvel do Google. Presente em múltiplos aparelhos de diversas fabricantes, como Samsung, Motorola, LG, e Sony, é a plataforma mobile mais popular do mundo. É conhecido por ser baseado no núcleo do Linux, ter um código aberto e uma série de possibilidades de personalização."

4 Conhecendo as Progressives Web Apps

PWA é uma tecnologia emergente do Google, um conceito relativamente novo no mundo dos dispositivos móveis e da internet. De acordo com o Google Developers (GOOGLE DEVELOPERS, 2019b):

"As Progressive Web Apps fornecem uma experiência instalável, semelhante a um aplicativo, em computadores e dispositivos móveis que são criados e entregues diretamente pela Web. Eles são aplicativos da web que são rápidos e confiáveis. E o mais importante, são aplicativos da web que funcionam em qualquer navegador."

PWAs são desenvolvidas usando certas tecnologias e abordagens para criar aplicações que aproveitam os recursos dos dispositivos móveis nativos e de aplicativos da Web, essencialmente é uma mistura de aplicações web e mobiles nativas.

O Google definiu um *checklist* a ser seguido para se considerar uma aplicação como uma PWA são elas:

- Progressivo - Funciona para qualquer usuário, independentemente do navegador escolhido, pois é criado com aprimoramento progressivo como princípio fundamental.
- Responsivo - Se adéqua a qualquer formato: desktop, celular ou tablet.
- Independente de conectividade - Aprimorado com *service workers* para trabalhar *off-line* ou em redes de baixa qualidade.
- Semelhante a aplicativos - Parece com aplicativos para os usuários, com interações e navegação de estilo de aplicativos, pois é compilado no modelo de *shell* de aplicativo.
- Atual - Sempre atualizado graças ao processo de atualização do *service worker*.
- Seguro - Fornecido via HTTPS para evitar invasões e garantir que o conteúdo não seja adulterado.

- Descobrível - Pode ser identificado como “aplicativo” graças aos manifestos W3C e ao escopo de registro do *service worker*, que permitem que os mecanismos de pesquisa os encontrem.
- Reenvolvente - Facilita o reengajamento com recursos como notificações *push*.
- Instalável - Permite que os usuários “guardem” os aplicativos mais úteis em suas telas iniciais sem precisar acessar uma loja de aplicativos.
- Linkável - Compartilhe facilmente por URL, não requer instalação complexa.

4.1 Características

4.1.1 Confiável

Quando iniciado a partir da tela inicial do usuário, os *service workers* permitem que uma PWA seja carregado instantaneamente, independentemente do estado da rede (GOOGLE DEVELOPERS, 2019b).

Basicamente os *service workers* são *scripts* que o navegador roda por debaixo dos panos, separado de uma página Web. Possibilitando que recursos sejam acessados mesmo sem uma interação do usuário ou de uma página Web. O *service worker* possui hoje funcionalidades como *Push Notifications*, que é basicamente, uma notificação que o usuário recebe sem requisita-lá, e Sincronização em Segundo Plano, que é uma API que permite que a aplicação adie ações até que o usuário tenha uma conectividade com a internet estável. Isso é útil para garantir que uma ação feita pelo usuário, seja realmente realizada (GOOG DEVELOPERS, 2019a). O *service worker* é uma API interessante para os desenvolvedores já que permite configurar experiências off-line. Os *Service Workers* possuem algumas características importantes:

- É executado em uma *thread* separada do navegador, portanto, não possui acesso ao Modelo de Objeto de Documento (DOM) diretamente.
- É um *proxy* de rede programável, portanto, permite gerenciar as solicitações de rede da página.
- É encerrado quando ficar ocioso e reiniciado quando for necessário.

- Os *service workers* utilizam promessas para retornar os dados de suas funções.

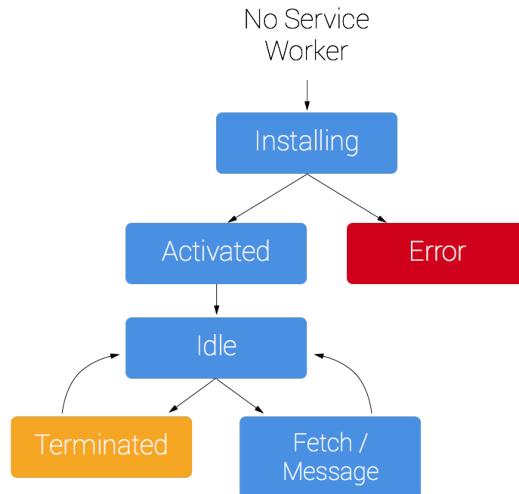
O *Service Worker* possui um ciclo de vida à parte da página da Web e funciona em uma estrutura já determinada. Entendendo como cada evento ocorre e suas respostas é o ideal para se ter a melhor forma de atualizar e guardar os arquivos. Primeiramente é necessário registrar o service worker na aplicação, isso pode ser feito via um arquivo JS, pode se adicionar uma verificação, antes do registro, para verificar se o navegador suporta o uso de *service worker*, após registrar o *service worker* o navegador inicia a etapa de instalação em segundo plano (JUSTEN, 2018).

Na etapa de instalação, é onde se normalmente é armazenado recursos estáticos em cache. Se todos os recursos forem salvos em cache corretamente, o service worker estará instalado. Se no momento de guardar os recursos, ocorrer alguma falha, a etapa de instalação não será finalizada corretamente, e portanto, o service worker não será ativado. Finalizando a etapa de instalação, é iniciada a fase de ativação, é nesse evento onde se gerencia o cache da aplicação e deleta coisas antigas de versões anteriores (GOOG DEVELOPERS, 2019a).

Após a etapa de ativação, o service worker gerenciará as páginas dentro do seu escopo, com exceção da página que registrou o service worker, onde ela só será controlada se for carregada novamente. Enquanto o service worker estiver controlando as páginas, ele poderá assumir dois estados: tratando de eventos de busca e mensagens que as páginas possam gerar, ou encerrado, para economizar memória do dispositivo (GOOG DEVELOPERS, 2019a).

A Figura 9 mostra uma versão minimalista do ciclo de vida do service worker, em sua primeira instalação.

Figura 9 – Ciclo de Vida do Service Worker



Fonte:(GOOG DE DEVELOPERS, 2019a)

4.1.2 Rápido

A maioria dos dados das PWAs é salvo no armazenamento do dispositivo no primeiro acesso. A próxima vez que o usuário acessá-la, a aplicação fará o download de poucos dados. Este é um recurso útil para pessoas que possuem conexão com a internet limitada. O aplicativo é mais confiável do que apenas um site e permite que você envie notificações para seus usuários, mesmo depois que o aplicativo for fechado. Uma vez armazenado em um dispositivo, leva muito menos tempo para ser reativo do que um site comum que precisa buscar e carregar tudo de novo (WILSON; MADSEN, 2010).

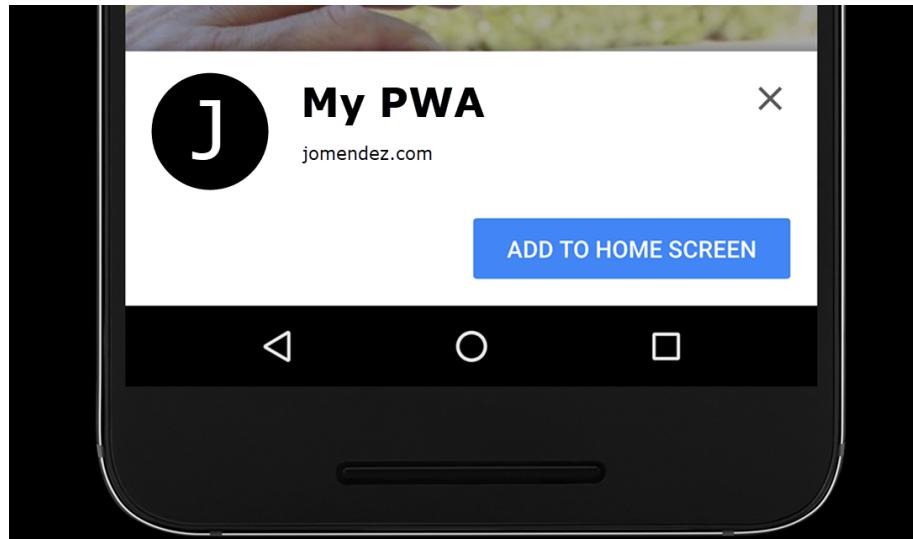
4.1.3 Integração e Engajamento

As PWAs são instaláveis e podem ser exibidos na tela inicial do usuário, sem a necessidade de uma loja de aplicativos, como Google Play Store ou Apple Store. Elas oferecem uma experiência imersiva em tela cheia com a ajuda do arquivo de manifesto do aplicativo web.

O Manifesto do Aplicativo Web, é um arquivo no formato Notação de Objetos JavaScript (JSON), que permite controlar como a aplicação irá aparecer e como ela será iniciada, ela conterá também informações relevantes a aplicação, assim fazendo

com que o navegador entenda que a aplicação é uma PWA e assim o navegador apresentará uma mensagem para o usuário para que se possa instalar o app na tela inicial do *smartphone* (GOOGLE DEVELOPERS, 2019a).

Figura 10 – Adicionar a tela inicial



Fonte:(JOMENDEZ, 2018)

Existem algumas configurações disponíveis no arquivo de manifesto, sendo algumas bem importantes como:

- background color - Define a cor de fundo esperada para a aplicação. Esse valor repete o que já está disponível no CSS do site, mas pode ser usado pelos navegadores para desenhar a cor de fundo de um atalho quando o manifesto está disponível antes de a folha de estilo ser carregada. Isso cria uma transição suave entre o carregamento da aplicação e o carregamento do conteúdo do site (MDN, 2019a).
- description - Fornece uma descrição geral do que a aplicação representa
- lang - Especifica o idioma principal para as propriedades, "name" e "short name"
- dir - Especifica a direção do texto principal para os membros "name", "short name" e "description". Juntamente com o membro "lang", ajuda na exibição correta dos idiomas da direita para a esquerda (MDN, 2019a).
- display - Define o modo de exibição da aplicação.

- icons - Especifica uma lista de arquivos de imagem que podem servir como ícones de aplicativo, configurando de acordo com a resolução de tela do dispositivo.
- related applications - Define uma lista de aplicativos nativos que podem ser instalados ou acessíveis via plataforma externa, como Google Play Store, esses aplicativos podem ser versões alternativas da aplicação PWA (MDN, 2019a).
- short name - Fornece um nome curto legível para o aplicativo. Isso é feito quando não há espaço suficiente para exibir o nome completo da aplicação, como as telas iniciais dos dispositivos.
- theme color - Define a cor do tema padrão para a aplicação. Isso às vezes afeta o modo como o sistema operacional o exibe.

4.2 Vantagens

4.2.1 Fácil Instalação e Atualização

Como já foi apresentado, para adicionar ou instalar uma aplicação PWA, os usuários simplesmente precisam abrir seu site progressivo em um navegador em seu dispositivo móvel. Em algum momento, eles serão solicitados a adicionar o aplicativo em sua tela inicial, ou os usuários podem adicionar o próprio PWA no menu do navegador. Bem simples em comparação a aplicativos nativos, em que o usuário necessita ir a uma plataforma de aplicativos e instalar (CODICA, 2019).

Quanto às atualizações, os usuários de aplicações PWA não precisam atualizar seu aplicativo toda vez que for lançada uma nova versão, eles sempre terão o mais novo.

4.2.2 Custos Reduzidos de Desenvolvimento e Suporte

Não precisa criar uma solução diferente para cada plataforma, pois o mesmo PWA funciona no *Android* e no *iOS* e cabe em qualquer dispositivo. Devido à atualização fácil e simultânea, existirá apenas uma versão do aplicativo circulando. O que significa que não se gastará custos extras suportando várias versões (CODICA, 2019).

4.2.3 Rápido, Leve e Seguro

Com o PWA implementado, a aplicação pode ser carregada em poucos segundos, o que é possível graças aos dados armazenados em cache pelos. Sendo menores em tamanho do que os aplicativos móveis nativos, as PWA são mais leves e eficientes e usam menos capacidade e dados do dispositivo. Ao mesmo tempo, eles fornecem uma experiência de usuário próxima à de um aplicativo nativo *Service Workers*. Além disso todas as aplicações PWA funcionam via HTTPS, o que significa segurança extra e nenhum acesso não autorizado aos dados (JUSTEN, 2018).

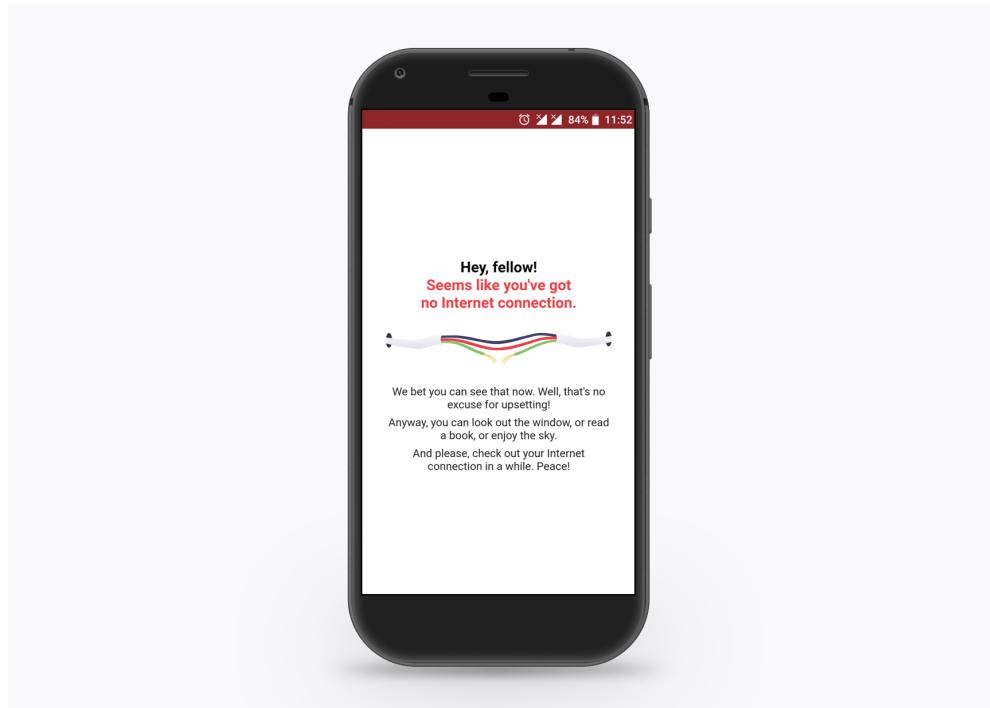
4.2.4 Não necessita de nenhuma loja de aplicativos

Para publicar uma aplicativo nativo para dispositivo móvel é necessário enviá-lo para alguma loja de aplicativos, como, Google Play Store ou Apple Store, e que ainda vai passar por um processo de análise do aplicativo. Com uma PWA, não há necessidade de aguardar o término do período de moderação e como já mencionado, para instalar a aplicação progressiva, os usuários só precisarão abrir o website e clicar em "Adicionar à tela inicial". Embora se comportando como um aplicativo nativo, a PWA ainda seria uma página da Web, clicável e compartilhável, e também é indexada pelo Google (CODICA, 2019).

4.2.5 Customizável

Se o usuário estiver sem conexão e quiser acessar novas páginas que não foram guardadas em cache, a aplicação não irá travar, mas mostrará uma mensagem personalizada, como na Figura 15:

Figura 11 – Telas Customizadas



Fonte:(CODICA, 2019)

4.3 Desvantagens

4.3.1 Funcionalidades Limitadas

As PWAs ainda são sites e ainda não suportam todas as funcionalidades que os aplicativos nativos podem oferecer. Elas têm acesso limitado a recursos dos dispositivos. Além disso, as PWAs podem oferecer um nível de notificação menos personalizado se comparado aos aplicativos nativos.

4.3.2 Limitações para o iOS

No momento, ainda há uma lacuna entre as PWAs para Android e iOS. Embora os PWAs estejam disponíveis para usuários do iOS, nem todos os recursos que funcionam em dispositivos Android são oferecidos para iOS, no entanto, existe um movimento na equipe por trás do iOS que aos poucos estão implementando recursos para serem usados via PWA, e provavelmente os usuários do iOS irão receber as mesmas funcionalidades das aplicações progressivas disponíveis no Android (FIRTMAN, 2018).

5 Análise comparativa entre PWAs e Aplicações Móveis Nativas

As PWAs provaram ser muito úteis, e foram apresentados diversos recursos disponíveis em dispositivos móveis, em que as PWAs dão suporte. No entanto, eles não estão aqui para tomar o lugar dos aplicativos nativos, mas para corrigir alguns problemas, como a compatibilidade entre plataformas e funcionamento em baixa conectividade e até mesmo offline.

Tendo isso em vista, esse capítulo irá abordar, o que são as aplicações para dispositivos móveis nativas, e uma comparação levando em consideração as vantagens e desvantagens entre desenvolver uma aplicação PWA e um aplicativo nativo.

5.1 Aplicação Nativa

Os aplicativos nativos são desenvolvidos para um dispositivo específico ou uma plataforma, como Android e iOS, e podem interagir e aproveitar os recursos fornecidos por esse dispositivo específico. Esses aplicativos oferecem acesso mais rápido a diferentes recursos do dispositivo, como câmera, microfone, agenda, localização e etc. Para usa-lós é necessário realizar a instalação do aplicativo no dispositivo desejado, normalmente os aplicativos se encontram nas lojas das plataformas, mas também é possível instalar via pacotes externos (CLUSTOX, 2018).

Ao desenvolver uma aplicação nativa, é necessário muitas vezes usar linguagens de programação específicas para a plataforma, elas permitem ao desenvolvedor ter acesso às funcionalidades de API e hardware do dispositivo. As linguagens de programação disponíveis para desenvolver aplicativos nativos, varia para cada plataforma (CLUSTOX, 2018). Abaixo alguns exemplos das linguagens de programação disponíveis:

- Java - Linguagem oficial do sistema operacional Android, usado para criar aplicativos nativos para a plataforma.

- Kotlin - Linguagem mais recente e semelhante ao Java, também para criar aplicativos nativos para Android.
- Objective-C - Linguagem principal para criar *software* para dispositivos iOS.
- Swift - Linguagem lançada da Apple para a criação de *software* para o iOS, propõe-se a ser mais simples de implementar do que o Objective-C.

5.2 Comparativo PWA e Aplicação nativa

5.2.1 Criação da aplicação e Distribuição

Aplicação Nativa

O desenvolvimento de uma aplicação móvel nativa para Android e iOS requer duas equipes, uma para cada sistema. Mesmo que os aplicativos de ambos os sistemas sejam desenvolvidos ao mesmo tempo, ainda demorará mais para garantir que a funcionalidade seja a mesma para os dois aplicativos. Tudo isso significa tempo e custos consideráveis necessários para criar um aplicativo.

O envio e aprovação via App Stores é uma parte separada da distribuição do aplicativo móvel nativo. O produto terá que passar por um período de moderação, pela equipe da plataforma, o que geralmente leva algum tempo. Para a Google Play Store, pode levar algumas horas, enquanto na Apple App Store pode levar de dois a quatro dias. Esse tempo de moderação, acaba impactando no atraso da distribuição da aplicação.

PWA

A criação de uma PWA requer apenas uma equipe de desenvolvimento da Web, pois na verdade é um site, embora com alguns recursos nativos dos dispositivos móveis. A validação pelas lojas não é necessária, pois se está criando um *website*. Não é necessário enviar a aplicação para nenhuma loja nem esperar que ele seja aprovado. Depois que a PWA é construída e publicada na Web, ela está pronto para ser usada.

5.2.2 Instalação

Aplicação Nativa

Basicamente o fluxo de instalação é, ir em uma loja de aplicativos, procurar a aplicação e realizar o download, e após isso, a instalação no dispositivo móvel, e por fim o aplicativo estará disponível para uso.

PWA

Com a PWA é necessário abrir um navegador, acessar o site da aplicação, irá ser apresentado uma mensagem, perguntando se quer adicionar a aplicação para a tela inicial do dispositivo, confirmando, a aplicação irá ser instalada.

5.2.3 Engajamento do Usuário

Uma das ferramentas de engajamento mais poderosas é a *Push Notifications*. São mensagens entregues por meio de um aplicativo instalado nos dispositivos, nos dispositivos móveis ou nos desktops dos usuários, para alertar seus usuários sobre novas chegadas de ações, vendas ou outras notícias.

Aplicação Nativa

Em aplicativos móveis nativos, a disponibilidade do recurso de notificações por *push* não depende do sistema operacional ou do modelo do dispositivo. Os usuários os receberão independentemente desses fatores.

PWA

Nas PWA, as *Push Notifications* também estão disponíveis, mas apenas para o Android. Isso é possível graças aos *service workers* eles podem enviar notificações quando uma aplicação PWA não está em execução.

5.2.4 Funcionamento Offline

Aplicação Nativa

Quando estamos falando sobre o modo offline do aplicativo nativo, assumimos que ele opera da mesma maneira que na conexão. O ponto é que um aplicativo nativo mostra o conteúdo e a funcionalidade que ele conseguiu armazenar em *cache* quando a conexão ainda estava lá. Isso está disponível devido ao armazenamento local e à sincronização suave de dados com a nuvem.

PWA

Nas aplicações PWA, os usuários também podem aproveitar o modo offline. Quando ativadas, as páginas mostram o conteúdo pré-carregado ou carregado, que é fornecido com os *service workers*. No entanto, o modo offline nas PWAs é um pouco mais lento em comparação a um aplicativo móvel nativo, pois ele é implementado de forma diferente. Ao mesmo tempo, a diferença entre os dois tipos de aplicativos não é tão drástica.

6 Estudo de Caso

Este capítulo visa demonstrar como desenvolver uma aplicação com suporte à PWA. Basicamente o app consiste em 3 telas. Será apresentada uma aplicação que utiliza alguns recursos de uma PWA, como, funcionamento *offline*, câmera do dispositivo, localização atual, notificações e *background sync*. Para isso foi utilizado algumas ferramentas como *Firebase*, *Workbox* e *IndexedDB*.

6.1 Ferramentas Utilizadas

Firebase

De acordo com (TREINAWEB, 2017) "O Firebase é uma plataforma do Google que contém várias ferramentas e uma excelente infraestrutura para ajudar desenvolvedores web e mobile a criar aplicações de alta qualidade e performance". Para o desenvolvimento do app, serão utilizados alguns recursos disponibilizados pelo firebase:

- **Realtime Database:** Banco de dados *NoSQL* hospedado em nuvem.
- **Storage:** Banco de dados utilizado para guardar arquivos de mídia, como imagens, vídeos e áudio.
- **Notifications:** Serviço de envio de notificações para usuários conectados.

IndexedDB

O *indexedDB* é um banco de dados disponibilizado pelo *browser*, segundo (MDN, 2019b) "*IndexedDB* é uma API para armazenamento *client-side* de quantidades significantes de informações e buscas com alta performance por índices". *IndexedDB* é a solução para grande porção de dados estruturados.

Workbox

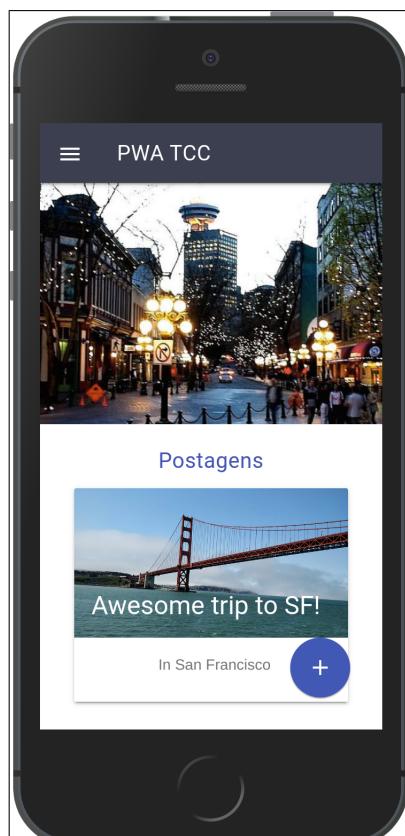
O *workbox* é uma ferramenta, que vai nos ajudar a manipular, de forma mais simples, o *service worker* da aplicação. Segundo (GOOGÉ DEVELOPERS, 2019b) "O *workbox* é um conjunto de bibliotecas e módulos que facilitam o armazenamento em cache e aproveitam ao máximo os recursos usados para criar uma PWA".

6.2 Conhecendo o Aplicativo

Para esse projeto foi desenvolvido um aplicativo que realiza postagens. Cada uma dessas postagem deve possuir além da imagem, uma descrição e a localização de onde foi realizado tal evento. O aplicativo desenvolvido têm três telas:

- **Tela Inicial:** É apresentação do aplicativo, ela reúne todas as postagens feitas pelos os usuários, e um botão de ação, para criar uma nova postagem no aplicativo. Como é visto na Figura 12.

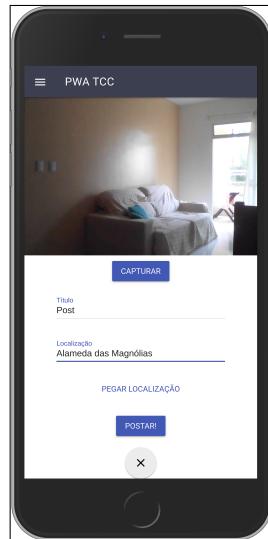
Figura 12 – Tela Inicial



Fonte: (Elaborado Pelo Autor, 2019)

- **Tela de Criação de Postagens:** A Figura 13 mostra como é a tela que possibilita ao usuário criar uma postagem.

Figura 13 – Tela de Criação de Post



Fonte: (Elaborado Pelo Autor, 2019)

- **Tela de Ajuda:** É uma página para auxiliar o usuário na navegação do aplicativo. Ainda existem botões de ação que simulariam o entrar em contato com a equipe de SAC, como mostrado na Figura 14.

Figura 14 – Tela de Ajuda



Fonte: (Elaborado Pelo Autor, 2019)

6.3 Configurações Inicias Do Aplicativo com PWA

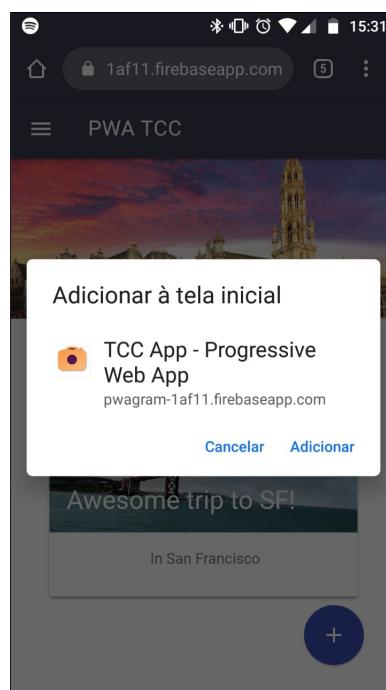
Adicionando suporte à PWA

Para o aplicativo desenvolvido usar recursos PWA, temos que adicionar um arquivo de manifesto e um arquivo de configuração do Service Worker.

6.3.0.1 Arquivo de Manifesto

O manifesto da nossa aplicação é configurado por meio de um arquivo de texto, onde possibilita-nos a adicionar informações relevantes sobre o nosso aplicativo. Com o arquivo de manifesto configurado nossa aplicação agora pode ser adicionada a tela inicial (MDN, 2019a). Abaixo a Figura 15 que ilustra isso:

Figura 15 – Adicionando App



Fonte: (Elaborado Pelo Autor, 2019)

E para exemplificar, no Quadro 6.1 abaixo é mostrado como o manifesto da aplicação ficou configurado:

Quadro 6.1 – Arquivo Manifesto

```

1  {
2      "name" : "TCC App – Progressive Web App",
3      "short_name" : "PWA TCC",
4      "icons" : [
5          {
6              "src" : "/src/images/icons/app-icon-48x48.png",
7              "type" : "image/png",
8              "sizes" : "48x48"
9          },
10         ...
11     ],
12     "start_url" : "/index.html",
13     "scope" : ".",
14     "display" : "standalone",
15     "orientation" : "portrait-primary",
16     "background_color" : "#fff",
17     "theme_color" : "#3c3f50",
18     "description" : "A app, implementing a lot of PWA features.",
19     "dir" : "ltr",
20     "lang" : "en-US"
21 }

```

Fonte: (Elaborado Pelo Autor, 2019)

Service Worker

Como o service worker é um script do navegador que executa em segundo plano (GOOGÉ DEVELOPERS, 2019a). No primeiro momento vamos apenas configura-lo para fazer cache de alguns recursos úteis a nossa aplicação, como, arquivos javascript, css, imagens e páginas html do aplicativo. Para configurar o service worker, será criado um arquivo javascript, que utilizará uma biblioteca externa chamada *workbox*. O Workbox nós ajudará a configurar o service worker de maneira mais fácil, já que ele encapsula, muito código para nós (GOOGÉ DEVELOPERS, 2019b). A seguir, o Quadro 6.2 mostra a configuração inicial do service worker.

Quadro 6.2 – Service Worker

```

1  workboxSW.router.registerRoute (:googleapis|gstatic)\.com.
2    ,workboxSW.strategies.staleWhileRevalidate ({
3      cacheName: 'google-fonts',
4        cacheExpiration: {
5          maxEntries: 3,
6          maxAgeSeconds: 60 * 60 * 24 * 30
7        }
8      }));
9
10
11  workboxSW.router.registerRoute ('https://cdnjs.cloudflare.com/ajax/libs←
12    /material-design-lite/1.3.0/material.indigo-pink.min.css', ←
13    workboxSW.strategies.staleWhileRevalidate ({
14      cacheName: 'material-css'
15    }));
16
17
18  workboxSW.precache ([{
19    "url": "favicon.ico",
20    "revision": "2cab47d9e04d664d93c8d91aec59e812"
21  },
22  {
23    "url": "index.html",
24    "revision": "df4415e7f962b5b6025142af8109f6d2"
25  },
26  {
27    "url": "manifest.json",
28    "revision": "cbd4ba1a23f99eb419b91b68d7591d3c"
29  },
30  {
31    "url": "offline.html",
32    "revision": "89c91dd62199aa07afc6b6d1ca0db9d1"
33  },
34  ...
35 });

```

Fonte: (Elaborado Pelo Autor, 2019)

6.4 Recursos Avançados do Service Worker

Cache das Postagens

Com o service worker configurado, um recurso importante do aplicativo é a listagem de postagens, é possível configurar o service worker, para fazer o cache das postagens a medida que forem sendo carregadas, assim tornando o aplicativo funcional mesmo em modo offline. No arquivo de configuração do service worker, criaremos um interceptador para a rota de listagem de Postagens e faremos o tratamento para guardar os dados em cache quando necessário. O Quadro 6.3 mostra o código necessário para que isso seja possível:

Quadro 6.3 – Armazendo Post

```
workboxSW.router.registerRoute('https://${FIREBASE_URL}/posts.json', ←
  function(args) {
2   return fetch(args.event.request)
    .then(function(res) {
4     var clonedRes = res.clone();
      clearAllData('posts')
6     .then(function() {
      return clonedRes.json();
8   })
     .then(function(data) {
10    for (var key in data) {
      writeData('posts', data[key])
12   }
     });
14   return res;
    });
16 }) ;
```

Fonte: (Elaborado Pelo Autor, 2019)

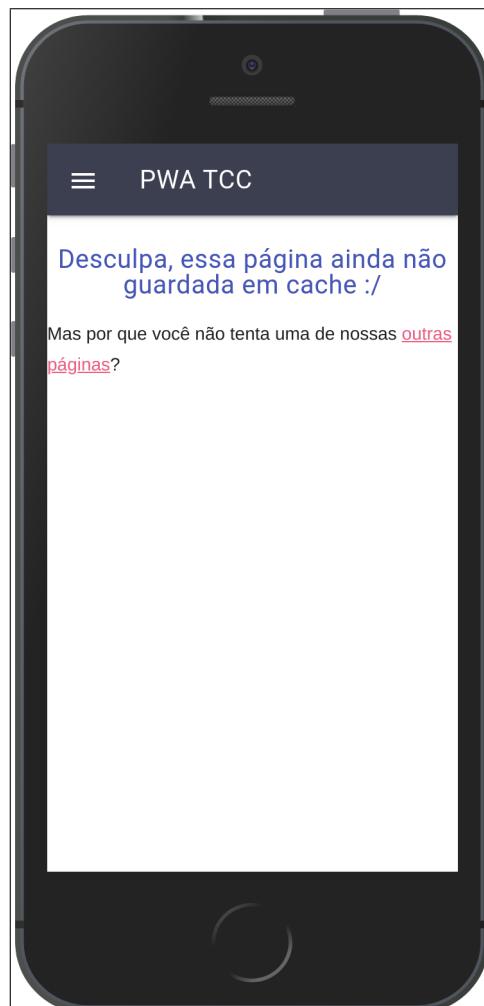
Na configuração acima, os dados retornados da requisição, são guardados no IndexedDB, na tabela 'posts'. Quando a requisição é retornada, os dados da tabela

'posts' são limpos, para então ser salvos, afim de manter-mos sempre dados validos no IndexedDB.

Ações Customizadas

O service worker nos permite customizar ações de tratamento, quando nossa aplicação estiver offline, e um recurso interessante que será adicionado, é a exibição de uma página customizada, quando o usuário tenta acessar um recurso offline e que ainda não foi cacheado pelo service worker, também conhecida como *fallback page* como mostra a Figura 16.

Figura 16 – Tela Offline



Fonte: (Elaborado Pelo Autor, 2019)

Abaixo o Quadro 6.4 mostra configuração do service worker, para exibir uma página customizada:

Quadro 6.4 – Página offline customizada

```

workboxSW.router.registerRoute(function (routeData) {
2   return (routeData.event.request.headers.get('accept').includes('text/←
      html')) ;
3   }, function(args) {
4     return caches.match(args.event.request)
5     .then(function (response) {
6       if (response) {
7         return response;
8       } else {
9         return fetch(args.event.request)
10        .then(function (res) {
11          return caches.open('dynamic')
12          .then(function (cache) {
13            cache.put(args.event.request.url, res.clone());
14          return res;
15        })
16      })
17      .catch(function (err) {
18        return caches.match('/offline.html')
19        .then(function (res) {
20          return res;
21        })
22      });
23    }
24  })
25);

```

Fonte: (Elaborado Pelo Autor, 2019)

Sincronização de Dados em Background

Um recurso interessante que o aplicativo possuí é o envio de postagens, mesmo o usuário estando offline, o service worker, possui um evento de sync, onde nele é possível verificar se o aplicativo está com acesso a internet e realizar uma determinada ação (GOOG DEVELOPERS, 2019a).

Ao cadastrar uma postagem, será salvo no IndexedDB, o body da requisição para o

servidor na tabela 'sync-posts', após isso será registrado um evento 'sync', com a tag 'sync-new-posts' no service worker, para que ele trate-o logo mais. Abaixo o Quadro 6.5 com o código das configurações:

Quadro 6.5 – Sícronização Em Background

```

1  if ('serviceWorker' in navigator && 'SyncManager' in window) {
2    navigator.serviceWorker.ready
3      .then(function (sw) {
4        var post = {
5          id: new Date().toISOString(),
6          title: titleInput.value,
7          location: locationInput.value,
8          picture: picture,
9          rawLocation: fetchedLocation
10         };
11        writeData('sync-posts', post)
12          .then(function () {
13            return sw.sync.register('sync-new-posts');
14          })
15          .then(function () {
16            var snackbarsContainer = document.querySelector('#confirmation-toast');
17            var data = {message: 'Seus post foi salvo para a sincronizaacao !'};
18            snackbarsContainer.MaterialSnackbar.showSnackbar(data);
19          })
20          .catch(function (err) {
21            console.log(err);
22          });
23        });
24      } else {
25        sendData();
26      }

```

Fonte: (Elaborado Pelo Autor, 2019)

Para o service worker tratar o evento que acabou de ser transmitido, é necessário implementar uma função. Para isso verificaremos se o evento transmitido terá a tag 'sync-new-posts', que foi passada previamente, e assim podemos tratar os dados da forma que quisermos, nesse caso, vai ser chamada os dados do IndexedDB, referente a

tabela 'sync-posts' e enviaremos os dados para o servidor Firebase, e por fim, limpamos os dados da tabela 'sync-posts', que acabaram de ser salvos. A seguir o código da função implementada é mostrada no Quadro 6.6:

Quadro 6.6 – Limpando Dados

```
self.addEventListener('sync', function(event) {  
  2  console.log('[ Service Worker] Background syncing', event);  
  3  if (event.tag === 'sync-new-posts') {  
  4    console.log('[ Service Worker] Syncing new Posts');  
  5    event.waitUntil(  
  6      readAllData('sync-posts')  
  7        .then(function(data) {  
  8          for (var dt of data) {  
  9            var postData = new FormData();  
10            postData.append('id', dt.id);  
11            postData.append('title', dt.title);  
12            postData.append('location', dt.location);  
13            postData.append('rawLocationLat', dt.rawLocation.lat);  
14            postData.append('rawLocationLng', dt.rawLocation.lng);  
15            postData.append('file', dt.picture, dt.id + '.png');  
16            fetch(`https://${FIREBASE_FUNCTION_URL}/storepostData`, {  
17              method: 'POST'  
18            body: postData  
19            })  
20            .then(function(res) {  
21              console.log('Sent data', res);  
22              if (res.ok) {  
23                res.json()  
24                .then(function(resData) {  
25                  deleteItemFromData('sync-posts', resData.id);  
26                })  
27              }  
28            })  
29          }  
30        })  
31      })  
32    })  
33  })  
34});
```

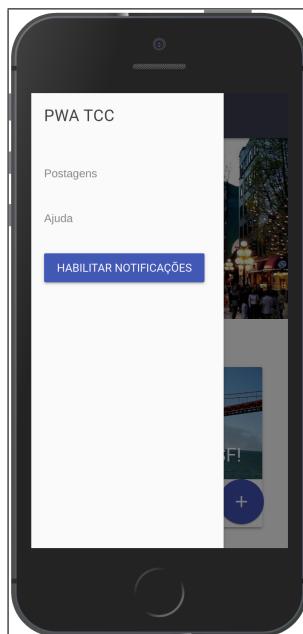
Fonte: (Elaborado Pelo Autor, 2019)

Notificações

Notificações é uma ferramenta importante para informar o usuário sobre um determinado evento, tendo isso em vista, o aplicativo desenvolvido vai enviar notificações

para os usuários quando for criada uma nova postagem. Para a criação do fluxo de notificação foi usado o Firebase e configurações no arquivo do service worker. Em um primeiro momento é necessário verificar se o usuário deseja receber notificações, por isso, o aplicativo possui um botão que habilita as notificações. Como mostra a figura Figura 17.

Figura 17 – Habilitar Notificações



Fonte: (Elaborado Pelo Autor, 2019)

Este botão só estará disponível se o browser possuir suporte as notificações, para isso foi adicionado uma verificação, em que esconde o botão caso as notificações não estiverem disponíveis. O Quadro 6.7 mostra a lógica para que isso aconteça.

Quadro 6.7 – Verificando suporte notificações

```

1 if ('Notification' in window && 'serviceWorker' in navigator) {
2   for (var i = 0; i < enableNotificationsButtons.length; i++) {
3     enableNotificationsButtons[i].style.display = 'inline-block';
4     enableNotificationsButtons[i].addEventListener('click', ←
5       askForNotificationPermission);
6   }

```

Fonte: (Elaborado Pelo Autor, 2019)

O browser tendo suporte as notificações, é necessário a permissão do usuário para podermos utilizá-las. Como mostra o Quadro 6.8.

Quadro 6.8 – Pedindo permissão ao usuário

```

1 function askForNotificationPermission() {
2   Notification.requestPermission(function(result) {
3     console.log('User Choice', result);
4     if (result !== 'granted') {
5       console.log('No notification permission granted!');
6     } else {
7       configurePushSub();
8     }
9   });
10 }

```

Fonte: (Elaborado Pelo Autor, 2019)

O usuário dando permissão para o aplicativo usar as notificações, podemos configurá-las e criar uma inscrição no Firebase, essa inscrição vai auxiliar no envio das notificações, como uma identificação para o dispositivo do usuário e enviar a notificação para eles. Para essa ação, foi criada uma função que verifica se o dispositivo já possui uma inscrição, senão, cria uma notificação no Firebase.

Quadro 6.9 – Configurando Push Notifications

```

1 function configurePushSub() {
2   if (!('serviceWorker' in navigator)) return;
3   var reg;
4   navigator.serviceWorker.ready
5     .then(function(swreg) {
6       reg = swreg;
7       return swreg.pushManager.getSubscription();
8     })
9     .then(function(sub) {
10       if (sub === null) {
11         // Create a new subscription
12         var convertedVapidPublicKey = urlBase64ToUint8Array(PUBLIC_KEY);
13         return reg.pushManager.subscribe({
14           userVisibleOnly: true,

```

```
    applicationServerKey: convertedVapidPublicKey
16  });
17 }
18 })
19 .then(function(newSub) {
20   return fetch(`https://${FIREBASE_URL}/subscriptions.json`, {
21     method: 'POST',
22     headers: {
23       'Content-Type': 'application/json',
24       'Accept': 'application/json'
25     },
26     body: JSON.stringify(newSub)
27   })
28 })
29 .then(function(res) {
30   if (res.ok) {
31     displayConfirmNotification();
32   }
33 })
34 .catch(function(err) {
35   console.log(err);
36 });
37 }
```

Fonte: (Elaborado Pelo Autor, 2019)

No retorno da requisição para o Firebase, se tudo ocorrer bem, é chamada uma função que exibe para o usuário uma notificação, pré configurada.

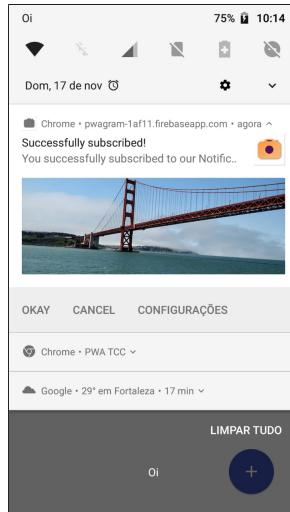
Quadro 6.10 – Notificação Pré configurada

```
1 function displayConfirmNotification() {
2   if ('serviceWorker' in navigator) {
3     var options = {
4       body: 'You successfully subscribed to our Notification service!',
5       icon: '/src/images/icons/app-icon-96x96.png',
6       image: '/src/images/sf-boat.jpg',
7       dir: 'ltr',
8       lang: 'en-US', // BCP 47,
9       vibrate: [100, 50, 200],
10      badge: '/src/images/icons/app-icon-96x96.png',
11      tag: 'confirm-notification',
12      renotify: true,
13      actions: [
14        { action: 'confirm', title: 'Okay', icon: '/src/images/icons/app-icon-96x96.png' },
15        { action: 'cancel', title: 'Cancel', icon: '/src/images/icons/app-icon-96x96.png' }
16      ]
17    };
18    navigator.serviceWorker.ready
19    .then(function(swreg) {
20      swreg.showNotification('Successfully subscribed!', options);
21    });
22  }
23}
```

Fonte: (Elaborado Pelo Autor, 2019)

A Figura 18, mostra o exemplo, de como será exibido a notificação:

Figura 18 – Notificação Pré Configurada



Fonte: (Elaborado Pelo Autor, 2019)

Por fim, para exibir as notificações, é necessário, configurar o service worker. O service worker possui o evento 'push', onde configuramos uma função para o tratamento das notificações.

Quadro 6.11 – Notificação personalizada

```

1 self.addEventListener('push', function(event) {
2   console.log('Push Notification received', event);
3   var data = {title: 'New!', content: 'Something new happened!', openUrl←
4     : '/'};
5   if (event.data) data = JSON.parse(event.data.text());
6   var options = {
7     body: data.content,
8     icon: '/src/images/icons/app-icon-96x96.png',
9     badge: '/src/images/icons/app-icon-96x96.png',
10    data: {
11      url: data.openUrl
12    }
13  event.waitUntil(self.registration.showNotification(data.title, options←
14    ));

```

Fonte: (Elaborado Pelo Autor, 2019)

7 Conclusão

Referências

- BARROS, T. *Android*. 2015. Disponível em: <<https://www.techtudo.com.br/tudo-sobre/android.html>>.
- CANALTECH. *Adobe anuncia o fim do Flash para 2020*. 2017. CanalTech. Disponível em: <<https://canaltech.com.br/software/adobe-anuncia-o-fim-do-flash-para-2020-97778/>>. Acesso em: 31 jul 2018.
- CIPOLI, P. *O que é o Adobe Flash?* 2018. CanalTech. Disponível em: <<https://canaltech.com.br/software/O-que-e-o-Adobe-Flash/>>. Acesso em: 31 jul 2018.
- CLUSTOX. *Aplicação Nativa*. 2018. Disponível em: <<https://www.clustox.com/blog/native-app-vs-pwa-comparative-study>>.
- CODICA. *PWA Benefits*. 2019. Disponível em: <<https://www.codica.com/blog/what-are-pwas-and-their-benefits/>>.
- DEVSARAN. *From History of Web Application Development*. 2016. Disponível em: <<https://www.devsaran.com/blog/history-web-application-development>>. Acesso em: 31 jul 2018.
- FIRTMAN, M. *PWA iOS*. 2018. Disponível em: <<https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7>>.
- FOWLER, M. *Microservices*. 2014. Disponível em: <<https://martinfowler.com/articles/microservices.html>>.
- GARCIA, R. *iOS*. 2019. Disponível em: <<https://www.apptuts.com.br/tutorial/ipad/o-que-e-ios/>>.
- GOOGLE DEVELOPERS. *Service Workers*. 2019. Disponível em: <<https://developers.google.com/web/fundamentals/primers/service-workers>>.
- GOOGLE DEVELOPERS. *Workbox*. 2019. Disponível em: <<https://developers.google.com/web/tools/workbox>>.
- GOOGLE DEVELOPERS. *Manifesto*. 2019. Disponível em: <<https://developers.google.com/web/fundamentals/web-app-manifest>>.
- GOOGLE DEVELOPERS. *PWA: Definição*. 2019. Disponível em: <<https://codelabs.developers.google.com/codelabs/your-first-pwapp>>.
- GRANATYR, J. *Introdução ao Modelo Multicamadas*. 2007. Disponível em: <<https://www.devmedia.com.br/introducao-ao-modelo-multicamadas/5541>>.
- JOMENDEZ. *PWA Add Home Screen*. 2018. Disponível em: <<http://www.jomendez.com/2018/06/05/add-home-screen-pwas>>.
- JORDAO, F. *História Mobile 2*. 2009. Disponível em: <<https://www.tecmundo.com.br/celular/2140-historia-a-evolucao-do-celular.htm>>.

- JUSTEN, W. *PWA Overview*. 2018. Disponível em: <<https://willianjusten.com.br/como-fazer-seu-site-funcionar-offline-com-pwa/>>.
- MACEDO, D. *Arquitetura de aplicações*. 2012. Disponível em: <<https://www.diegomacedo.com.br/arquitetura-de-aplicacoes-em-2-3-4-ou-n-camadas/>>.
- MARQUES, R. *Ajax com jQuery: Trabalhando com requisições assíncronas*. 2016. Disponível em: <<https://www.devmedia.com.br/ajax-com-jquery-trabalhando-com-requisicoes-assincronas/37141>>.
- MARTINS, C. *Aplicações corporativas multicamadas - Partes 1 e 2*. 2012. Disponível em: <<https://www.devmedia.com.br/aplicacoes-corporativas-multicamadas-partes-1-e-2/26545>>.
- MDN. *Web Docs - JavaScript*. 2018. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>. Acesso em: 31 jul 2018.
- MDN. *Arquivo de Manifesto*. 2019. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/Manifest>>.
- MDN. *IndexedDB*. 2019. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/IndexedDB>>.
- MICROLINS. *História Android*. 2017. Disponível em: <<https://www.microlins.com.br/noticias/tecnologia/linha-do-tempo-a-evolucao-dos-celulares>>.
- PEREIRA, A. *A origem do CSS, um pouco da história*. 2009. Disponível em: <<https://www.devmedia.com.br/a-origem-do-css-um-pouco-da-historia/15195>>.
- PINTEREST. *Motorola Dynatac*. 2019. Disponível em: <<https://br.pinterest.com/pin/346636502552324402/>>.
- PRODNOTE. *Nokia 9000*. 2015. Disponível em: <<https://prodnote.wordpress.com/2015/03/12/nokia-9000-communicator-throwback/>>.
- RIGUES, R. *História Mobile*. 2016. Disponível em: <<https://blog.meuquantum.com.br/veja-evolucao-do-smartphone/>>.
- ROCHA, H. L. S. da. *Arquitetura WEB*. [S.I.]: EP:SF-Rio/D99, 1999.
- SANTOS, L. *Microserviços: dos grandes monólitos às pequenas rotas*. 2017. Disponível em: <<https://medium.com/trainingcenter/microservicos-dos-grandes-monolitos-as-pequenas-rotas-adb70303b6a3>>.
- THOMS, N. *A look back at HTML5*. 2017. FastHosts. Disponível em: <<https://www.fasthosts.co.uk/blog/websites/look-back-html5>>.
- TREINAWEB. *Firebase*. 2017. Disponível em: <<https://www.treinaweb.com.br/blog/firebase-descubra-no-que-esta-plataforma-pode-te-ajudar/>>.
- WILSON, P.; MADSEN, L. *The Memoir Class for Configurable Typesetting - User Guide*. Normandy Park, WA, 2010. Disponível em: <<http://mirrors.ctan.org/macros/latex/contrib/memoir/memman.pdf>>. Acesso em: 19 dez. 2012.