Offline-first Multidatabase Progressive Web Apps com PouchDB/CouchDB

Matheus Dal'Pizzol* Anubis Graciela de Moraes Rossetto †

2018

Resumo

Este artigo trata do desenvolvimento de *Progressive Web Apps* (PWAs) valendo-se da abordagem *offline-first*. São discutidos os problemas apresentados por aplicativos móveis nativos e as soluções a eles oferecidas pelas PWAs. Discorre-se a respeito dos mecanismos que viabilizam a execução de aplicações Web em um cenário *offline* e sua integração com a interface nativa dos dispositivos móveis, como os Service Workers e o Web App Manifest. A escolha do PouchDB para o armazenamento local das informações é justificada a partir da análise de ferramentas alternativas à tarefa. Estratégias são apresentadas para a sincronização e seguridade de múltiplos bancos de dados para cada usuário em um SGBD CouchDB. Estas estratégias são implementadas em uma aplicação para o auxílio do estudo de instrumentos musicais que tem por objetivo registrar as seções de prática do estudante para análise, referência futura e visualização do progresso do aprendizado do usuário em seu instrumento. Os resultados obtidos são comentados bem como o futuro das PWAs.

Palavras-chave: Progressive Web Apps. Offline-first. PouchDB. CouchDB. Aplicativos Móveis.

Introdução

Russel e Berriman (2015) cunharam o termo *Progressive Web App* (PWA) com o objetivo de denominar websites que "tomaram as vitaminas certas" (RUSSEL, 2015) para entregar aos usuários uma experiência mais próxima à oferecida por aplicativos móveis nativos enquanto tiram proveito das vantagens clássicas da plataforma Web. Intencionalmente ou não, o conceito de PWA trouxe à luz o que fora, na verdade, a ideia original para a execução de códigos de terceiros em dispositivos móveis quando

*<matheusdalpizzol@gmail.com>
†<anubisrossetto@gmail.com>

do lançamento do iPhone pela Apple, em 2007. A visão original da empresa não contemplava a instalação direta de códigos de terceiros no aparelho. O objetivo era que os aplicativos não fossem mais que aplicações web tradicionais, simplesmente apresentadas de forma distinta daquela consagrada nos navegadores (9TO5MAC, 2011).

Muitos aplicativos nativos para dispositivos móveis destinam-se à solução de problemas e usos pontuais. Em função desta pontualidade, vários desses aplicativos acabam ignorados por apresentarem um custo de oportunidade maior que o de soluções alternativas. De fato, estima-se que 20% dos usuários interessados em um determinado aplicativo desistem da sua instalação a cada etapa deste processo (CSELLE, 2012) e que 60% dos aplicativos ofertados em *marketplaces* nunca foram sequer baixados (THYGESEN, 2013). Em outras palavras, do ponto de vista do usuário, o custo de download e o engajamento exigidos por aplicativos nativos para sua instalação permanente no dispositivo é maior que o de soluções que não envolvam o aplicativo. Ainda, aplicativos nativos não são uma proposição barata do ponto de vista dos desenvolvedores, dado que o suporte à diversas plataformas, dispositivos e versões aumentam seus custos e sua distribuição em *marketplaces* depende de processos burocráticos e tediosos (KUMAR, 2016a). Assim, por facilitar usos descompromissados e pontuais, para a maioria dos casos em que se deseja uma interação que não exija um comprometimento demasiado do usuário, a plataforma Web é mais indicada (LOPES, 2016).

O funcionamento centralizado, por sua vez, é um fator frustrante comum a aplicativos móveis nativos e aplicações Web tradicionais, por depender de conexão estável com a Internet e da Nuvem como *Single Source Of Truth* (SSOT) das informações que manipulam. Não raro, essa característica impede por completo o acesso e a utilização da aplicação. Esses aplicativos também apresentam performance insatisfatória quando operados sob conexões lentas, como 2G e 3G e a observância desses fatores tem profundas implicações no design da camada de comunicação de uma aplicação (PRASAD, 2011). Ao contrário da ideia difundida de que conectividade ubíqua é uma realidade próxima a ser atingida pelo avanço contínuo em infraestrutura, Alex Feyerke afirma que

> Não tendo conexão de dados com frequência, mesmo nas cidades mais ricas e desenvolvidas do mundo, nos levou a concluir que não, o problema da conectividade/largura de banda não irá simplesmente resolver-se por conta própria em nenhum momento do futuro próximo." (FEYERKE, 2013, tradução nossa)

O desenvolvimento de PWAs que mesclem o custo de oportunidade de aplicações Web tradicionais com o engajamento e a imersividade de aplicativos nativos aumenta a atratividade de soluções móveis antes ignoradas por esses fatores e reduz seus custos de distribuição e uso. Por sua vez, a capacitação *offline* de PWAs oferece grandes benefícios e conveniência aos usuários em cenários em que o sinal de Internet seja lento, intermitente ou inexistente (DA-14, 2017). O usuário dispõe de uma interface mais rápida, dado que as operações de escrita e leitura são realizadas localmente, reduzindo drasticamente o tempo de resposta. Também há economia de

bateria, já que a diminuição de requisições externas reduz os ciclos de processamento e, em cenários de desastre, a aplicação ainda é operável até que a conexão seja restabelecida para a sincronização dos dados com os servidores (HOLT, 2015).

Tomar consciência desse fato desde o início de um novo projeto é de suma importância devido ao enorme impacto exercido na arquitetura e desenvolvimento das aplicações. Dado que o *workflow offline-first* é essencialmente diferente do *workflow* de aplicações que assumem a existência permanente de conexões rápidas e confiáveis, o funcionamento *offline-first* dificilmente pode ser implementado incrementalmente, em um momento futuro, sem que haja a reimplementação de grande parte do código (PRASAD, 2011).

Ademais, é preciso entender que o cenário *offline* não é um erro - como comumente é tratado pelos desenvolvedores -, mas um fato da vida mobile (FEYERKE, 2014) e que uma abordagem *offline-first* é a única maneira de se obter uma experiência ubíqua, ou seja, permitindo a operação da aplicação em qualquer cenário de conectividade (HOLT, 2015). Ainda, segundo Alex Feyerke

> Nós não podemos continuar construindo aplicativos com o *mindset desktop* de conectividade rápida e permanente, onde uma desconexão temporária ou um serviço lento são tratados como um problema e comunicados como um erro (FEYERKE, 2013, tradução nossa)

Dessa forma, o presente estudo objetiva o desenvolvimento de uma PWA capaz de operar completamente *offline*, realizando a replicação das informações geridas para a Nuvem apenas quando houver conexão estável com a Internet. Para isso, visa-se realizar o cacheamento do *app shell* utilizando Service Workers; integrar a aplicação à interface do dispositivo do usuário utilizando um Web App Manifest; realizar o armazenamento de informações primariamente no dispositivo do usuário; e realizar a sincronização de informações entre a Nuvem e os dispositivos de usuários.

Perseguindo esses objetivos, este trabalho apresenta a implementação de uma *Single Page Application* (SPA) com o uso do *framework* de interfaces Vue.js (VUE.JS, 2017), a biblioteca de componentes Vuetify (VUETIFY, 2017) e o *framework* de aplicação Nuxt.js (NUXT, 2017). A persistência dos dados a nível local se dá com o uso do PouchDB (POUCHDB, 2017) enquanto seu mecanismo de sincronização envia os dados a um servidor CouchDB. Operações restritas a administradores do CouchDB (COUCHDB, 2017) são realizadas através de um servidor intermediário implementado em Express.js (EXPRESS, 2017).

O presente estudo trata, em sua primeira parte, sobre a visão original para aplicativos móveis, o surgimento do conceito de PWAs, a simulação da experiência nativa em aplicações Web e a capacitação *offline* dessas aplicações. Em seguida, descreve a finalidade da aplicação desenvolvida e como diferentes problemas relacionados à sincronização de múltiplos bancos de dados são abordados. Por fim, apresenta os resultados obtidos durante o desenvolvimento e tece comentários sobre possíveis melhorias e o futuro dos conceitos analisados.

1 Referencial teórico

Nesta seção são discutidos os conceitos envolvidos no desenvolvimento de PWAs e o resgate da visão original sobre aplicativos móveis. Em seguida, são apresentados os mecanismos que viabilizam a execução de uma aplicação Web *offline* e sua integração com a interface nativa dos dispositivos, bem como a imitação dessas interfaces na plataforma Web. Por fim, são expostas as ideias fundamentais para aplicações *offline-first* e para sincronização de dados distribuídos nessas aplicações utilizando-se PouchDB e CouchDB.

1.1 A visão original para aplicativos móveis e as PWAs

Ao apresentar o iPhone ao mundo, em 2007, Steve Jobs declarou que os desenvolvedores já, à época, dispunham de tudo que precisavam para criar aplicativos para o novo dispositivo: bastava valer-se dos padrões modernos da Web. Sua visão original não contemplava a instalação direta de códigos de terceiros no aparelho. O objetivo era que os aplicativos não fossem mais que aplicações Web tradicionais, simplesmente apresentadas de forma distinta daquela consagrada nos navegadores (9TO5MAC, 2011).

A afirmação foi ampla e estranhamente ignorada e, em 2008, em resposta aos desenvolvedores, a Apple desviou-se da ideia inicial, dando origem à AppStore (9TO5MAC, 2011). Esse acontecimento abriria as portas do prolífero mercado *mobile* dominado pelo iOS e, mais tarde, pelo Android, do Google. Essa reviravolta criou oportunidades na mesma medida em que criou problemas. Como exposto por Alex Russel,

> Ao invés de clicar um link para acessar o conteúdo que você está procurando, esses sistemas fazem das lojas os mediadores das aplicações que, por sua vez, mediam e facilitam a descoberta de conteúdo. O processo de "hibridização" gera aplicações que não vivem mais nas ou com as premissas da Web. Como desenvolvemos para todas essas lojas de uma vez só? [...] Como a obrigatoriedade de empacotamento prévio altera seus pressupostos e infraestrutura? Como a indexação para buscas funciona? É uma troca profunda que favorece o suporte *offline* e as lojas em detrimento da rápida iteração e "linkabilidade". (RUSSEL, 2015, tradução nossa)

Visando alguns desses problemas e aproveitando as melhorias entregues pelas tecnologias Web a partir de 2010, Alex Russel e Frances Berriman definiram o conceito de Progressive Web Apps, enumerando as características necessárias à uma aplicação web para que se mantenham as características da Web enquanto se entrega uma experiência de usuário mais próxima à dos aplicativos nativos. Essas aplicações não são empacotadas e não são distribuídas através de lojas, sendo apenas websites aos quais são adicionadas funcionalidades até então presentes apenas em aplicativos nativos (RUSSEL, 2015). As características enumeradas são as seguintes:

- **Progressiva:** deve implementar melhoramento progressivo, possibilitando sua operação independentemente das funcionalidades suportadas ou não pelo navegador;
- **Responsiva:** deve funcionar igualmente em qualquer tamanho de tela;
- **Independente de conectividade:** deve ser melhorada progressivamente com Service Workers para que possa operar *offline*;
- **App-like:** deve adotar um modelo de aplicação Shell + Conteúdo para criar navegações e interações que simulem as de um aplicativo nativo;
- **Atualizada:** deve atualizar-se automática e transparentemente por meio de Service Workers;
- **Segura:** devem ser servidas através de HTTPS para garantir a integridade das informações;
- **Descobrível:** é identificável através de manifestos W3C permitindo que mecanismos de busca a encontrem;
- **Reengajável:** pode acessar as interfaces de reengajamento do sistema operacional, por exemplo, as notificações Push;
- **Instalável:** deve poder ser adicionada à *homescreen* do dispositivo através de diálogos do navegador, permitindo que o *app* seja mantido sem a necessidade de uma *app store*;
- **Linkável:** deve aproveitar-se do poder social das URLs para compartilhar e ser compartilhada.

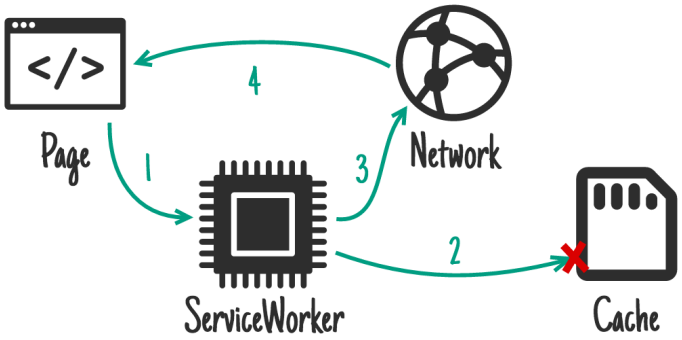
1.2 Operação *offline* com Service Workers

O Service Worker é um tipo específico de Web Worker, tratando-se assim de uma API JavaScript executada à parte do contexto de execução da aplicação principal e, portanto, sem acesso ao Documento Object Model (DOM). Em função disso, os Service Workers são recomendados para a execução de tarefas em *background*, que podem rodar mesmo quando a aplicação estiver *offline*, dando total controle sobre a experiência do usuário aos desenvolvedores (GAUNT, 2017). Uma de suas principais características é ser um *proxy* de rede programável, funcionando como um intermediário entre as requisições feitas pela aplicação principal e a rede, permitindo, entre outras coisas, o cacheamento de recursos especificados e endereçáveis por URLs (GAUNT, 2017). O esquema é ilustrado na Figura 1.

Quando 1) requisições feitas pela aplicação são interceptadas pelo Service Worker, 2) é verificada a presença do recurso solicitado no *cache* do navegador. Caso o recurso não exista no *cache*, 3) o Service Worker repassa a requisição para a rede que 4) envia o recurso à aplicação.

Com isso considerado, os Service Workers servem aos propósitos das PWAs em diversos aspectos:

Figura 1 – Service Worker para busca primária no *cache* e *fallback* para a rede



Fonte: (ARCHIBALD, 2017)

- Eles permitem que a aplicação seja instalável, podendo efetuar o cacheamento da totalidade do *app shell* (os arquivos estáticos responsáveis pela interface do usuário), permitindo o seu carregamento mesmo em um cenário *offline*;
- permitem que a aplicação seja automática e transparentemente atualizada, visto que toda vez que a aplicação é acessada, o navegador tenta identificar se o arquivo do Service Worker sofreu alterações, efetuando o download do novo arquivo em caso afirmativo. Entretanto, o antigo Service Worker permanece em execução até que a aplicação seja encerrada e iniciada novamente, quando, então, o novo Service Worker toma o controle;
- permitem que a aplicação seja reengajável, possibilitando o recebimento de notificações Push, exatamente como as notificações de aplicativos móveis nativos;
- permitem que a aplicação seja independente de conectividade, possibilitando a sincronização de dados com servidores em segundo plano. Assim, a aplicação pode funcionar *offline*, utilizando a memória do próprio dispositivo para o armazenamento das informações até que haja conexão para que, então, essas informações sejam enviadas para a Nuvem.

1.3 Simulação da experiência nativa

Quando o assunto é a experiência do usuário, uma aplicação Web tradicional peca em diversos aspectos se comparada a aplicativos nativos. Para utilizá-la, o usuário precisa abrir o navegador; digitar o endereço do *site*; ver uma tela branca enquanto os recursos são carregados pela rede; ter a barra de navegador ocupando espaço na interface. Todos esses são pequenos detalhes que, por mais insignificantes que pareçam, refletem em frustração e abandono dos usuários. É para sanar esses problemas que, em combinação com os Service Workers, hoje dispomos do manifesto dos aplicativos Web, ou Web App Manifest. Segundo Matt Gaunt e Paul Kinlan,

> O manifesto dos aplicativos Web é um arquivo JSON que permite controlar como o aplicativo Web ou *site* é exibido para o usuário

> em áreas que normalmente se espera ver aplicativos nativos (por exemplo, a tela inicial de um dispositivo), como definir o que o usuário pode inicializar e o visual durante a inicialização. (GAUNT; KINLAN, 2017)

Com algumas poucas propriedades, o Web App Manifest permite que integremos nossa aplicação Web à interface nativa do dispositivo, incluindo a adição de um ícone de inicialização à *homescreen* deste, provendo ao usuário uma experiência praticamente indistinguível daquela fornecida por aplicativos nativos (LOPES, 2016).

Para que uma aplicação Web se comporte de maneira visualmente mais próxima de um aplicativo nativo, recomenda-se que ela seja construída na forma de uma Single Page Application (SPA) (LOPES, 2016). SPAs são aplicações Web que, através do uso de requisições assíncronas - popularmente conhecidas pela sigla AJAX (Assynchronous JavaScript and XML) -, permitem que novos dados sejam recebidos de servidores Web e apresentados na página já renderizada no navegador do usuário, sem que ele seja obrigado a recarregar a página completamente para visualizar as novas informações. Combinando essa técnica às modernas propriedades CSS (Cascading Style Sheets) de animações e transições, é possível realizar mudanças e trocas suaves entre diferentes componentes da interface gráfica da aplicação, imitando - por vezes de modo muito fiel - a interface gráfica nativa do dispositivo.

Algumas ferramentas auxiliam na obtenção de uma aplicação que disponha de todos esses benefícios, como no caso do Vue.js, Vue-Router (VUE-ROUTER, 2017) e Vuetify, além do *bundler* Webpack (WEBPACK, 2017) e do *framework* de aplicação Nuxt.js.

Enquanto o Vue.js pode ser descrito como um *framework* para a construção de interfaces de usuários, o Vue-Router permite a especificação de componentes Vue.js a serem exibidos quando a URL do navegador corresponder à URL configurada para o componente em questão. Essa configuração pode, então, ser combinada com o empacotamento e separação de código (Code Splitting) realizados pelo Webpack. Assim, cada conjunto de componentes definidos para uma determinada URL pode ser empacotado em um único arquivo JavaScript a ser carregado sob demanda, quando houver a correspondência da URL acessada no navegador. Dessa forma, a obtenção de uma SPA é facilitada. Já o Nuxt.js pode ser descrito como um *framework* de aplicação, mas no fundo, ele consiste de uma série de configurações para Webpack que atendem diversas finalidades, incluindo o preprocessamento de CSS, a compilação de Single File Components do Vue.js, a realização do Code Splitting e a organização do código fonte.

Para completar a simulação da experiência nativa, o Vuetify oferece componentes Vue.js que utilizam o Material Design (MATERIAL-DESIGN, 2018), do Google, a mesma linguagem visual utilizada pelos dispositivos Android, possibilitando a criação de interfaces extremamente fiéis ao visual e à experiência desses dispositivos.

1.4 Informação *offline* e distribuída com PouchDB e CouchDB

Apesar dos Service Workers possibilitarem a operação *offline* do *app shell* e a sincronização de informações em *background*, eles não são diretamente responsáveis pelo armazenamento local das informações geridas pela aplicação. Além disso, a gestão de informações descentralizadas (replicadas e sincronizadas entre dispositivos e servidores) envolve uma série de problemas complexos e de difícil resolução. Por exemplo, o desenvolvimento de algoritmos de resolução de conflitos e revisões e a prevenção de perda de dados são requisitos que precisam ser implementados desde o início do desenvolvimento da aplicação (MARTYNUS, 2017).

Dentre as diversas opções para o armazenamento local de informações estão a Local Storage, a Session Storage e o WebSQL. Todos esses, porém, apresentam problemas como pouco suporte pelos navegadores, limitações graves de espaço em memória, incompatibilidade com Web Workers e funcionamento síncrono bloqueante (OSMANI; COHEN, 2017).

A recomendação atual para o armazenamento local das informações, então, é o IndexedDB (OSMANI; COHEN, 2017). Essa especificação dispõe de bastante espaço na memória, suporta transações, pode ser usada com segurança em diversas abas do navegador simultaneamente e é suportada por todos os navegadores modernos (TABALIN, 2017). No entanto, a complexidade de algumas funcionalidades oferecidas pelo IndexedDB implica na complexidade de sua API que, somada à inexistência de suporte ao padrão Promises/A+, dificulta sua utilização pelos desenvolvedores (OSMANI, 2016). A API do IndexedDB é de tão baixo nível que até mesmo a documentação da Mozilla Developer Network (MDN) alerta os desenvolvedores a esse respeito e recomenda o uso de uma biblioteca *wrapper* para facilitar sua manipulação (SMITH, 2016).

No entanto, quase a totalidade dessas bibliotecas, à notória exceção do PouchDB, preocupa-se exclusivamente com o armazenamento local, desconsiderando a possível inserção da aplicação em um contexto distribuído e delegando ao desenvolvedor o tratamento da coesão, replicação e sincronização das bases de dados.

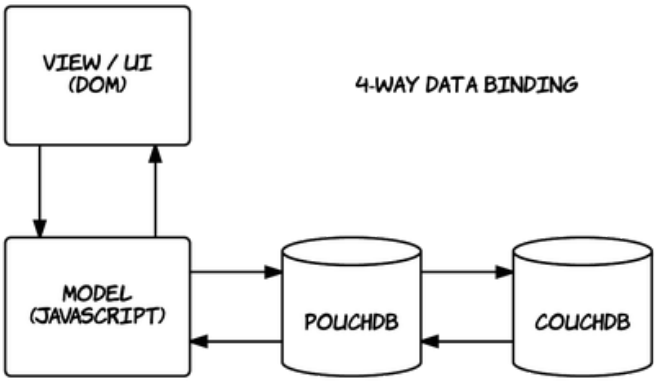
O PouchDB dá diversos passos além da simples facilitação de manipulação do IndexedDB. Abstraindo as diferenças entre o IndexedDB e o WebSQL em uma única API, o PouchDB amplia a capacidade de atuação da base de código, utilizando o IndexedDB como escolha primária de armazenamento e mudando para o WebSQL em navegadores sem suporte ao IndexedDB (BROWN, 2014).

Sendo uma reimplementação do CouchDB em JavaScript, o PouchDB também implementa o protocolo de replicação do CouchDB, permitindo que as informações salvas mesmo *offline* sejam replicadas e sincronizadas de maneira consistente com a Nuvem, tão logo haja conexão disponível (POUCHDB, 2017).

> O super poder do CouchDB é a sincronização. Às vezes eu tento explicar às pessoas dizendo que 'o CouchDB não é um banco de dados, mas um motor de sincronização'. Ele é uma forma eficiente de transferir dados de um lugar a outro, enquanto inteligentemente gerencia conflitos e revisões. É muito semelhante ao Git (LAWSON, 2017, tradução nossa).

Na Figura 2, podemos verificar a arquitetura de vinculação de dados em quatro vias obtida com o uso do PouchDB e do CouchDB em conjunto, ou 4 Way Data Binding. Os dados obtidos ou apresentados através da interface do navegador são vinculados a um modelo JavaScript que por sua vez referencia os dados persistidos no mecanismo de armazenamento determinado pelo PouchDB para, enfim, serem sincronizados com um servidor CouchDB remoto.

Figura 2 – 4 Way Data Binding



Fonte: (KUMAR, 2016b)

2 Desenvolvimento

Esta seção discorre sobre a finalidade da aplicação implementada no estudo dos conceitos apresentados e expõe a forma com que os Service Workers e o Web App Manifest foram à ela adicionados. A subseção sobre segurança e arquitetura apresenta as soluções encontradas para problemas gerados pelas limitações das ferramentas utilizadas, indo desde a criação de bancos de dados isolados para cada usuário, sua autenticação e autorização até a criação programática e sincronização de novos bancos no lado servidor por parte desses usuários. A seção é finalizada com a elucidação do mecanismo utilizado para contornar o problema do limite de requisições paralelas em navegadores Web e manter sincronizados os bancos de dados criados pelos usuários da forma mais transparente possível.

2.1 Finalidade da aplicação

A implementação dos conceitos estudados destina-se a uma aplicação para auxílio no estudo de instrumentos musicais. O sistema permite que um usuário registre suas sessões de prática para que, mais tarde, possa visualizar sua evolução. O usuário pode dispor de diversas agendas de treinamento, as quais podem conter diversos exercícios (*licks*). Toda vez que o usuário pratica um exercício, a aplicação registra a duração do exercício, a variação do exercício utilizada - se houver -, a velocidade utilizada em batidas por minuto (BPM) e um *feedback* sobre a execução do exercício para referência futura. No futuro, o usuário deve poder convidar outros usuários para participar de uma mesma agenda de treinamento.

2.2 Service Workers e Web App Manifest

O Nuxt.js facilita a criação de Service Workers e do Web App Manifest, atendendo algumas das exigências das PWAs. Para isso, foi preciso instalar e configurar o módulo do *framework* para PWAs (NUXT-PWA, 2018). O módulo está disponível por gerenciadores de pacotes como o NPM (NPM, 2017) e o Yarn (YARN, 2017), o que tornou o processo bastante rápido e direto.

O módulo é composto por vários outros submódulos destinados a diferentes finalidades. Um destaque interessante é o Workbox (WORKBOX, 2018), que além de gerar os Service Workers para realizar o cacheamento da aplicação e permitir seu uso *offline*, oferece outros serviços, como o Offline Analytics, que captura os dados de acesso à aplicação mesmo enquanto *offline* e os envia para a conta do Google Analytics quando houver conexão com a Internet.

2.3 Segurança e arquitetura

Apesar das facilidades de replicação, o CouchDB oferece alguns desafios quando se trata de segurança e privacidade dos dados. Esses desafios são abordados nesta seção, apresentando possíveis soluções para que os requisitos da aplicação sejam atendidos da maneira mais transparente possível para os usuários, garantindo a integridade e privacidade das suas informações. São explicados os processos de criação de usuários e agendas de treinamento no lado servidor, autenticação e autorização.

2.3.1 Cadastro de usuários e agendas de treinamento no lado servidor

Como o CouchDB não oferece nenhum mecanismo de controle de acesso a nível de documentos nem o conceito de coleções, é necessário que cada usuário possua seu próprio banco de dados. Assim, quando um usuário realiza o cadastro no sistema, um novo banco de dados deve ser criado para seu uso exclusivo. O CouchDB facilita esse processo oferecendo a configuração *couch_peruser*, que realiza a criação automática de um banco de dados com os níveis de acesso corretos toda vez que um novo registro é adicionado ao banco *_users*.

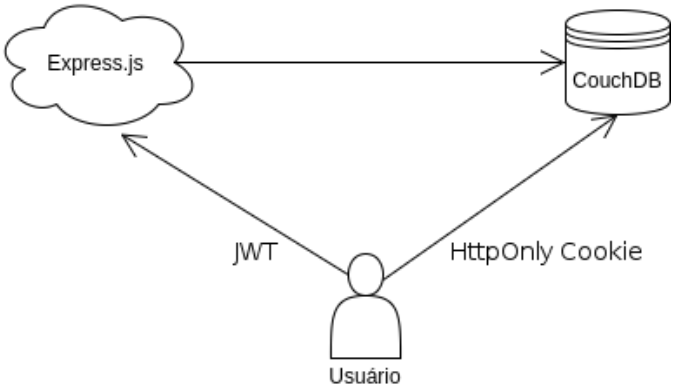
Efetuado o *login*, o usuário passa a poder criar novas agendas de treinamento a serem gerenciadas. No futuro, cada uma dessas agendas poderá ser compartilhada com outros usuários que poderão visualizar e modificar suas informações. Entretanto, assim como acontece com as informações privadas dos usuários, é necessário que cada agenda possua seu próprio banco de dados a fim de controlar quem pode ou não acessá-la. O problema nesse caso é que, diferentemente do que acontece com o banco de dados criado para o usuário com a opção *couch_peruser*, o CouchDB não oferece uma opção que realize esse processo automaticamente. Nem o usuário pode realizá-lo pois, como será visto adiante, a criação de bancos de dados é reservada aos administradores do CouchDB.

2.3.2 Autenticação e autorização

O CouchDB oferece não só a camada de armazenamento de dados, mas também um serviço RESTful para acesso e manipulação das informações, juntamente

com um *framework* de autenticação e autorização *built-in*. O uso desse *framework* é mandatório para que se possa tirar vantagem das funcionalidades de sincronização dessa tecnologia - é preciso que os dispositivos clientes tenham comunicação direta com o servidor do CouchDB para que mantenham os dados em sincronia. Por outro lado, algumas ações só podem ser realizadas por administradores do CouchDB, como, por exemplo, criar novos bancos de dados ou gerenciar o nível de acesso dos usuários. Como algumas dessas ações terão de ser executadas programaticamente, segundo a demanda dos dispositivos clientes, é necessário um servidor intermediário que detenha acesso de nível administrador ao CouchDB. Esse servidor é implementado utilizando Node.js (NODE.JS, 2017) e o *framework* de HTTP Express.js. A arquitetura é representada na Figura 3.

Figura 3 – Arquitetura de autenticação paralela para CouchDB com Express.js



Fonte: Do autor, 2017

Nessa arquitetura, enquanto o *framework* de autenticação do CouchDB fornece autenticação por *tokens* baseada em *cookies*, a autenticação no servidor Express.js pode ser realizada através de JSON Web Tokens (JWT) (JWT, 2017). Assim, ao realizar o *login* na aplicação, o dispositivo realiza a requisição de dois *tokens* em paralelo - ambos com os mesmos critérios de expiração -, uma para cada servidor. Mantém-se assim o contato direto do dispositivo do cliente com o servidor CouchDB e possibilita-se a execução programática de operações restritas a administradores através do servidor intermediário. É interessante que não necessariamente ambas as autenticações precisam ser bem sucedidas para que usuário possa utilizar a aplicação, dado que cada um dos *tokens* dá acesso a serviços distintos. Portanto, no caso de um dos serviços encontrar-se *offline*, o usuário simplesmente terá algumas das funções da aplicação temporariamente indisponíveis.

2.3.3 Sincronização de agendas de treino e compartilhamentos

Apesar do protocolo de replicação do CoucDB e o PouchDB ser projetado para a sincronização de informações contidas em um determinado banco de dados, eles não oferecem nenhuma alternativa que sincronize a criação de novos bancos de dados. Além disso, no caso de uso da aplicação proposta, é necessário que as

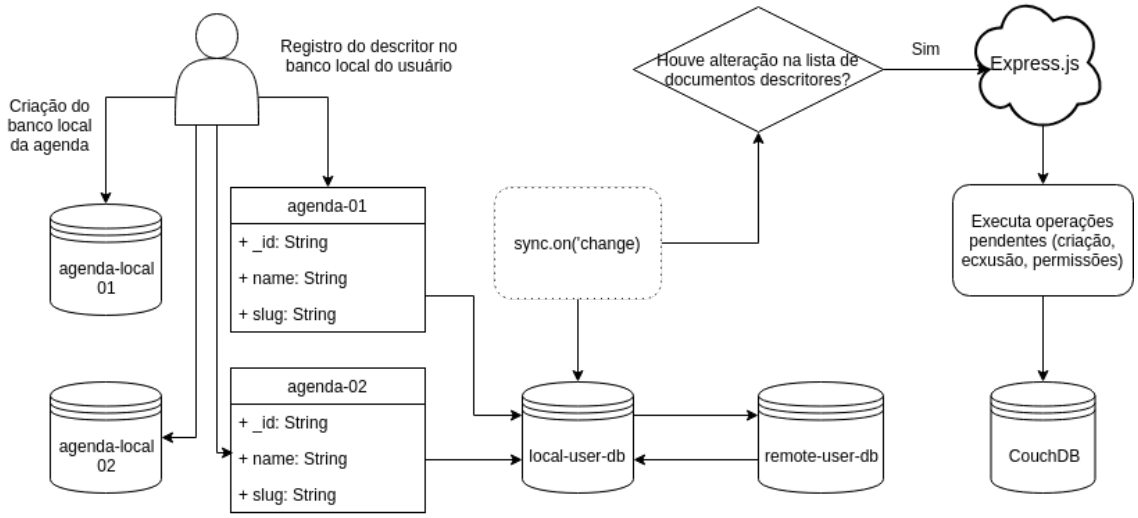
informações de permissões de acesso a determinados bancos de dados também sejam sincronizadas. Como mencionado anteriormente, novos bancos de dados e a modificação das permissões de acesso a esses bancos precisam ser feitas por administradores do CouchDB, portanto, através do servidor Express.js.

A questão torna-se, então: quando realizar essas requisições ao serviço Express.js, visto que, diferentemente do PouchDB, esse serviço não dispõe de um sistema em tempo real? E ainda, como manter uma lista atualizada das agendas de treino de um usuário, dado que esse usuário pode, inclusive, criar novas agendas, deletá-las ou compartilhá-las com outros usuários enquanto estiver *offline*? Ou seja, para que uma nova agenda tenha seu respectivo banco criado no servidor, excluído ou alterado, é necessário ter conhecimento de duas variáveis principais: o *status* da conexão e o *status* da autenticação.

Como a sincronização do banco privado do usuário depende tanto de conexão com a Internet, quanto de autenticação, o melhor momento para a realização dessas operações acontece quando um evento de sincronização bem sucedida é disparado.

Assim, a solução proposta é que cada usuário mantenha uma lista de descrição dessas informações em seu banco privado. Como esse banco pode ser sincronizado normalmente através do PouchDB e apenas o usuário mesmo detém acesso a esse documento, é possível valer-se dos eventos disparados pela *engine* de sincronização e utilizar a lista de descrição remota, sincronizada, para então realizar as operações necessárias. Por exemplo, quando informações alteradas localmente no navegador são sincronizadas com o servidor, a *engine* de sincronização do PouchDB dispara um evento *change* com a propriedade *direction* contendo o valor *push*. A partir disso, pode-se verificar se os documentos da lista de descrição foram modificados e, então, disparar a requisição para o serviço responsável pelas operações de criação de bancos, exclusão de bancos ou modificação de permissões. A Figura 4 apresenta um diagrama desse procedimento.

Figura 4 – Diagrama do mecanismo de sincronização de agendas de treino e suas permissões



Fonte: Do autor, 2017

É preciso considerar que a requisição em questão pode ser falha e que o serviço intermediário não dispõe de sincronização contínua. Também não é possível assumir que um novo evento *change* contendo um mesmo documento descritor irá ocorrer em um momento futuro. Nesse caso, é preciso informar ao usuário e fornecer meios para que realize essa requisição em outro momento. Algo como "Não foi possível realizar o *backup* dos seus dados. Tentar novamente".

2.4 O problema do limite de requisições paralelas em navegadores Web

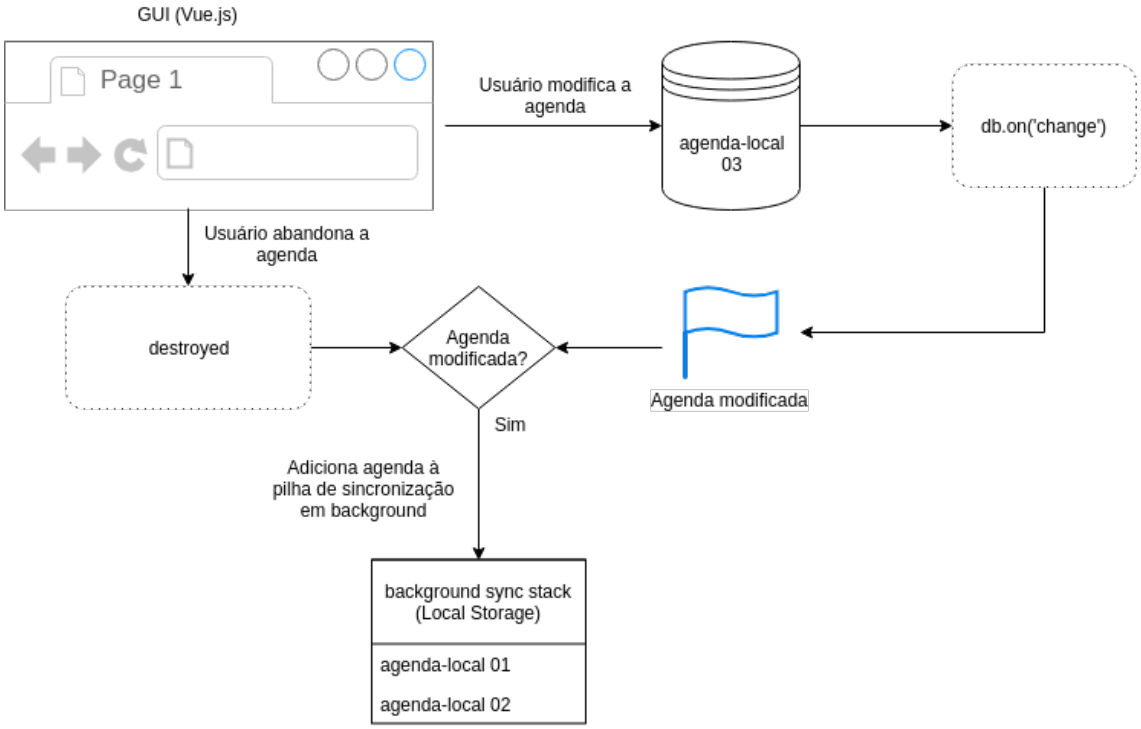
Na aplicação proposta, cada agenda de treinamento representa um banco de dados isolado no PouchDB e no CouchDB. Para que esses bancos permaneçam em sincronia, o PouchDB mantém uma requisição aberta entre o dispositivo do usuário e o servidor CouchDB para cada uma das agendas criadas. Porém, os navegadores Web apresentam limites de requisições paralelas a um mesmo *host* que, em geral, é de seis requisições. Se houver mais de seis bancos tentando sincronizar dados com o servidor, qualquer requisição adicional após se atingir o limite será completamente ignorada.

Uma alternativa para a solução desse problema seria o uso de Web Sockets, porém, a manutenção de *sockets* abertos o tempo todo resultaria em consumo desnecessário de bateria, além de carecer de suporte por alguns navegadores. Assim, a solução proposta aqui prevê a sincronização de uma agenda apenas durante o tempo em que ela estiver em uso. Entretanto, essa abordagem traz algumas complicações. Por exemplo, se uma agenda for modificada quando *offline* e não for utilizada quando houver conexão, os seus dados não serão enviados ao servidor CouchDB e, portanto, não serão sincronizados com outros dispositivos.

Para minimizar o problema, foi implementado um mecanismo de *flags* que

sinaliza uma agenda como modificada sempre que ela deixar de ser utilizada e tenha sofrido alterações. Essa sinalização é realizada caso o dispositivo esteja *offline* ou caso o usuário não esteja autenticado no momento do término da utilização e a pilha é armazenada na Local Storage. Isso é feito através do *lifecycle hook* ***destroyed***, do Vue.js, que é executado quando o usuário abandona a agenda na GUI (Graphic User Interface) da aplicação. O esquema é ilustrado na Figura 5.

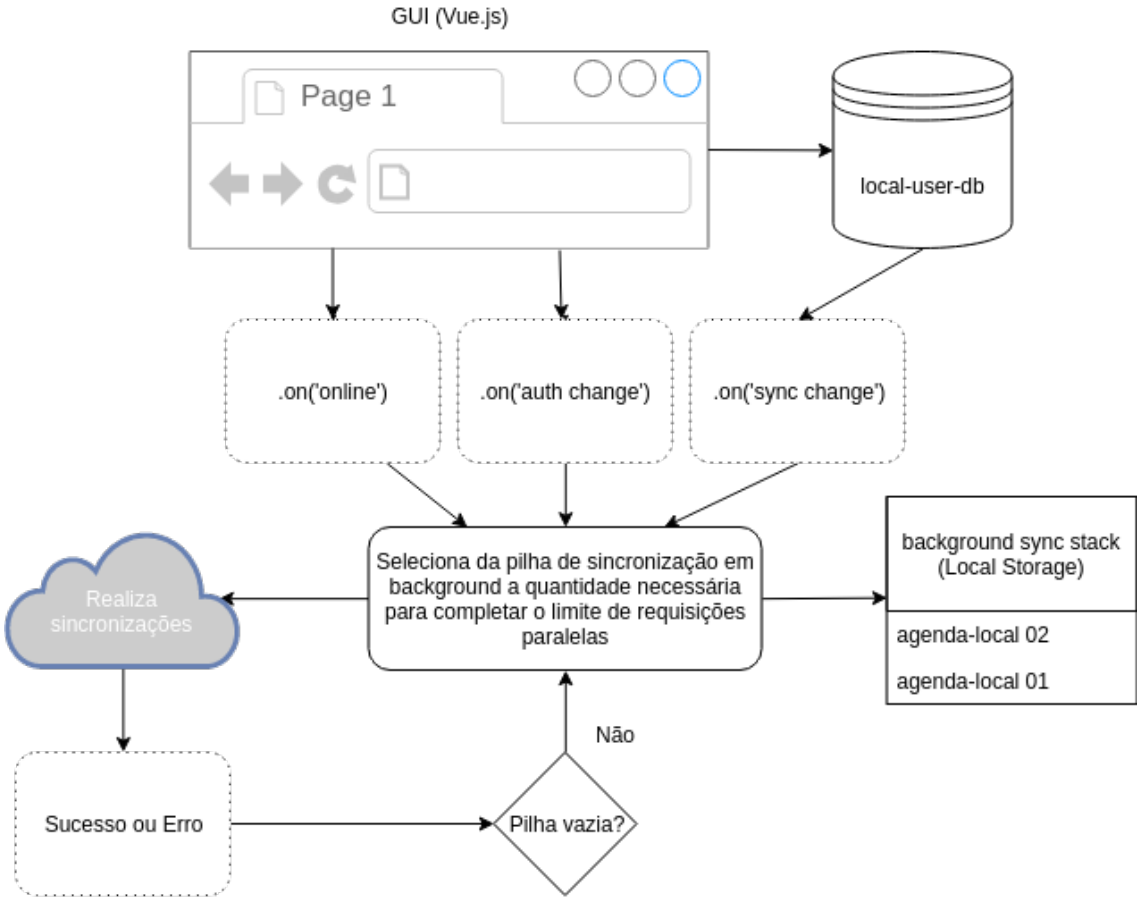
Figura 5 – Diagrama do mecanismo de sinalização de modificação *offline* de agendas



Fonte: Do autor, 2017

Após, observam-se alguns eventos que, quando disparados, realizam a sincronização das agendas da pilha em *background*, respeitando o limite de requisições paralelas definido pelo desenvolvedor. A Figura 6 ilustra a ideia.

Figura 6 – Diagrama do mecanismo de sincronização em *background*



Fonte: Do autor, 2017

3 Resultados

O desenvolmineto com Nuxt.js permitiu que algumas características almejadas para a aplicação fossem alcançadas com relativa facilidade. O cacheamento do *app shell* e a integração da interface da aplicação com a interface nativa do dispositivo não necessitaram de mais nada além da instalação do módulo para PWAs do Nuxt.js e sua configuração, já que esse módulo cria o Web App Manifest e os Service Workers necessários aos objetivos mencionados sem que seja preciso escrever esses arquivos desde as suas bases. O Vuetify, por sua vez, tornou possível o desenvolvimento de uma interface gráfica completamente responsiva e extremamente semelhante à interface nativa dos aplicativos para Android. Com isso, a interface da aplicação atendeu à diversas exigências das PWAs, sendo responsiva e *app-like* (através do Vuetify), independente de conectividade, instalável e transparentemente atualizada (através dos Service Workers) e descobrível e linkável (por tirar proveito da plataforma Web).

Com o par PouchDB/CouchDB, a característica *offline-first* foi muito bem incorporada à aplicação, resultando em uma GUI extremamente rápida em que, após o carregamento inicial da aplicação, todas as interações do usuário ocorrem de

maneira instantânea. Isso porque nenhuma das operações realizadas pelo usuário depende da realização de requisições HTTP. Por outro lado, não foi possível verificar se houve economia real de bateria dos dispositivos, dado que a aplicação foi executada apenas em ambiente de desenvolvimento por meio de dispositivos conectados via USB e, até o momento, não foi disponibilizada em ambiente de produção.

O mecanismo de sincronização de tarefas restritas a administradores do CouchDB através de um servidor intermediário, descrito na subseção 2.2.3, mostrou-se sólido. Porém, como mencionado, a dependência de uma requisição HTTP convencional para a execução dessas operações, sua suscetibilidade à falhas e a ausência de um mecanismo que tente realizar essas requisições repetidamente até que sejam bem sucedidas, obriga o desenvolvedor a apresentar ao usuário opções para executar essas operações manualmente em caso de falha. Esse fator implica em uma pequena perda de usabilidade, já que operações presumidamente transparentes precisam ser explicitadas aos usuários.

Enfim, apesar de minimizar o problema, a abordagem proposta na subseção 2.3 para sincronização de agendas modificadas quando *offline* serve apenas para o envio de informações ao servidor. Ou seja, um dispositivo ocioso apenas saberá das modificações em uma agenda específica quando esta for acessada na sua própria interface. Porém, esse mecanismo impede que dados deixem de ser enviados ao servidor quando um dispositivo estiver *online* e, tão logo a agenda for utilizada *online* em um dispositivo antes ocioso, os dados estarão disponíveis para sincronização no servidor.

Considerações finais

Atualmente, a abordagem *multidabase* é a única capaz de manter verdadeiramente a privacidade dos dados de um usuário em um sistema CouchDB. Entretanto, essa abordagem acarreta enormes limitações que precisam ser vencidas através de mecanismos de menor estabilidade e coesão em relação ao mecanismo de replicação de documentos do qual usufruem os bancos de dados contidos no CouchDB. Todos esses problemas seriam sanados caso o CouchDB provesse alguma espécie de controle de acesso a nível de documento, o que, sem nenhuma dúvida, diminuiria drasticamente a quantidade de bancos a serem criados e eliminaria a necessidade dos mecanismos descritos nas seções 2.2.3 e 2.3 do presente estudo.

Essa abordagem *multidatabase* também apresenta desafios de arquitetura para a aplicação que devem ser cuidadosamente ponderados. Como cada usuário e cada agenda de treino detém seus bancos de dados isolados, a manutenção de Design Documents precisa atentar ao suporte retroativo e deve ser realizada através da atualização da aplicação nos dispositivos de usuários, já que realizá-la diretamente no servidor CouchDB implicaria na atualização de todos os bancos de dados manualmente ou através de um *script* que varresse todo o SGBD realizando as alterações.

Quando tratamos de performance, a ideia de uma enorme quantidade de bancos de dados pode representar problemas que dependerão do tipo de *hardware* e do

sistema operacional utilizados para o servidor CouchDB. Isso se deve ao fato de que requisições ao CouchDB dependem do quão rápido o sistema operacional consegue encontrar, abrir, acessar e fechar os arquivos do banco de dados. Também podem ocorrer problemas em relação à nomenclatura dos bancos já que caracteres especiais pode influenciar a estrutura de pastas do sistema operacional. Por essas razões, torna-se interessante o uso de serviços focados no escalonamento *multidatabase*, como o Cloudant, da IBM, em detrimento de uma instalação nativa do CouchDB.

Por fim, independentemente do uso do par PouchDB/CouchDB ou de qualquer outra ferramenta alternativa dedicada aos objetivos deste estudo, o conceito das PWAs coloca-se como uma alternativa viável e extremamente atrativa aos aplicativos nativos, sendo eles de uso pontual ou não. A distribuição fácil proporcionada pela Web, a ausência de processos de empacotamento em executáveis de instalação dependentes de versões de sistemas operacionais e a proximidade da experiência nativa obtida com as novas tecnologias Web parecem dar início a um movimento em direção à obsolescência dos aplicativos nativos.

Offline-first Multidatabase Progressive Web Apps with PouchDB/CouchDB

Matheus Dal'Pizzol* Anubis Graciela de Moraes Rossetto †

2018

Abstract

This article discusses the development of Progressive Web Apps (PWAs) using the offline-first approach. The problems presented by native mobile applications and the solutions offered to them by the PWAs are discussed, as well as the mechanisms that enable the execution of Web applications in an offline scenario and its integration with the native interface of mobile devices, such as the Service Workers and the Web App Manifest. The choice of PouchDB for the local storage of information is justified by the analysis of alternative tools to the task. Strategies are presented for the synchronization and security of multiple databases for each user in a CouchDB DBMS (Data Base Management System). These strategies are implemented in an app for the study of musical instruments that keeps track of the students practice sections for analysis, future reference and visualization of their learning progress on the instrument. The obtained results are discussed as well as the future of the PWAs.

Keywords: Progressive Web Apps. Offline-firts. PouchDB. CouchDB. Mobile Apps.

Referências

9TO5MAC. *Job's original vision for the iPhone No third-party native apps*. 2011. Disponível em: <https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps>. Acesso em: 12 out 2017. Citado 2 vezes nas páginas 2 e 4.

*<matheusdalpizzol@gmail.com>
†<anubisrossetto@gmail.com>

ARCHIBALD, J. *The Offline Cookbook*. 2017. Disponível em: <https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/>. Acesso em: 15 nov 2017. Citado na página 6.

BROWN, T. *Building an Offline First App with PouchDB*. 2014. Disponível em: <https://www.sitepoint.com/building-offline-first-app-pouchdb>. Acesso em: 15 out 2017. Citado na página 8.

COUCHDB. 2017. Disponível em: <http://couchdb.apache.org/>. Acesso em: 11 dez 2017. Citado na página 3.

CSELLE, G. *Every Step Costs You 20% of Users*. 2012. Disponível em: <http://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html>. Acesso em: 2 out 2017. Citado na página 2.

DA-14. *Eight Reasons to Consider Progressive Web App Development*. 2017. Disponível em: <https://da-14.com/blog/eight-reasons-consider-progressive-web-app-development>. Acesso em: 2 out 2017. Citado na página 2.

EXPRESS. *Express.js*. 2017. Disponível em: <http://expressjs.com>. Acesso em: 14 nov 2017. Citado na página 3.

FEYERKE, A. *Say Hello to Offline First*. 2013. Disponível em: <http://hood.ie/blog/say-hello-to-offline-first.html>. Acesso em: 2 out 2017. Citado 2 vezes nas páginas 2 e 3.

FEYERKE, A. *Alex Feyerk - Offline First*. 2014. Disponível em: <https://www.youtube.com/watch?v=dPz_5-MEvcg>. Acesso em: 2 out 2017. Citado na página 3.

GAUNT, M. *Service Workers uma Introdução*. 2017. Disponível em: <https://developers.google.com/web/fundamentals/primers/service-workers>. Acesso em: 12 out 2017. Citado na página 5.

GAUNT, M.; KINLAN, P. *O manifesto do aplicativo web*. 2017. Disponível em: <https://developers.google.com/web/fundamentals/web-app-manifest/?hl=pt-br>. Acesso em: 12 out 2017. Citado na página 7.

HOLT, B. *Offline-First Apps with PouchDB*. 2015. Disponível em: <https://www.youtube.com/watch?v=7L7esHWAjSU>. Acesso em: 31 out 2017. Citado na página 3.

JWT. *JWT*. 2017. Disponível em: <https://jwt.io/>. Acesso em: 10 dez 2017. Citado na página 11.

KUMAR, M. *Progressive Web Apps Better and Low Cost Mobile Presence*. 2016. Disponível em: <https://dzone.com/articles/progressive-web-app-better-low-cost-mobile-presenc>. Acesso em: 2 out 2017. Citado na página 2.

KUMAR, N. *Intro PouchDB, unroll CouchBase crust and offline data Sync in Ionic with 4-way Data Binding*. 2016. Disponível em: <https://blog.knoldus.com/2016/01/08/intro-pouchdb-unroll-couchbase-crust-and-offline-data-sync-in-ionic-with-4-way-data-binding/>. Acesso em: 10 dez 2017. Citado na página 9.

LAWSON, N. *PouchDB and CouchDB an interview with Nolan Lawson*. 2017. Disponível em: <https://blog.couchdb.org/2017/04/04/pouchdb-couchdb-an-interview-with-nolan-lawson>. Acesso em: 15 out 2017. Citado na página 8.

LOPES, S. *Progressive Web Apps por Sergio Lopes - Devfest São Paulo 2016*. 2016. Disponível em: <https://www.youtube.com/watch?v=tECW-YJXV1o>. Acesso em: 2 out 2017. Citado 2 vezes nas páginas 2 e 7.

MARTYNUS, G. *Building Offline First apps with Hoodie | js.la March 2017*. 2017. Disponível em: <https://www.youtube.com/watch?v=TSDyxtVbbME>. Acesso em: 31 out 2017. Citado na página 8.

MATERIAL-DESIGN. *Material Design*. 2018. Disponível em: <https://material.io/>. Acesso em: 12 jun 2018. Citado na página 7.

NODE.JS. *Node.js*. 2017. Disponível em: <https://nodejs.org>. Acesso em: 10 dez 2017. Citado na página 11.

NPM. *NPM*. 2017. Disponível em: <https://www.npmjs.com/>. Acesso em: 10 dez 2017. Citado na página 10.

NUXT. *Nuxt*. 2017. Disponível em: <http://nuxtjs.org>. Acesso em: 14 nov 2017. Citado na página 3.

NUXT-PWA. *Nuxt-PWA*. 2018. Disponível em: <https://github.com/nuxt-community/pwa-module>. Acesso em: 12 jun 2018. Citado na página 10.

OSMANI, A. *Offline Storage for Progressive Web Apps*. 2016. Disponível em: <https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c>. Acesso em: 15 out 2017. Citado na página 8.

OSMANI, A.; COHEN, M. *Armazenamento off-line para Progressive Web Apps*. 2017. Disponível em: <https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa?hl=pt-br>. Acesso em: 15 out 2017. Citado na página 8.

POUCHDB. *Introduction to PouchDB*. 2017. Disponível em: <https://pouchdb.com/guides>. Acesso em: 15 out 2017. Citado 2 vezes nas páginas 3 e 8.

PRASAD, A. *Offline Support is Valuable, and You Can't Add it Later*. 2011. Disponível em: <http://aanandprasad.com/articles/offline>. Acesso em: 2 out 2017. Citado 2 vezes nas páginas 2 e 3.

RUSSEL, A. *Progressive Web Apps Escaping Tabs Without Losing Our Soul*. 2015. Disponível em: <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul>. Acesso em: 12 out 2017. Citado 2 vezes nas páginas 1 e 4.

SMITH, G. *Dexie.js - Because IndexedDB should be Awesome*. 2016. Disponível em: <http://blogs.bytecode.com.au/glen/2016/01/11/dexie.html>. Acesso em: 15 out 2017. Citado na página 8.

TABALIN, A. *Create Offline Web Apps Using Service Workers and PouchDB*. 2017. Disponível em: <https://www.sitepoint.com/offline-web-apps-service-workers-pouchdb>. Acesso em: 15 out 2017. Citado na página 8.

THYGESEN, T. *60% of Apps have never been used. Stats for app producers*. 2013. Disponível em: <http://www.tinethygesen.com/post/42502739988/60-of-apps-have-never-been-used-stats-for-app>. Acesso em: 2 out 2017. Citado na página 2.

VUE-ROUTER. *Getting Started*. 2017. Disponível em: <https://router.vuejs.org/en/essentials/getting-started.html>. Acesso em: 31 out 2017. Citado na página 7.

VUE.JS. *Introduction*. 2017. Disponível em: <https://vuejs.org/v2/guide>. Acesso em: 31 out 2017. Citado na página 3.

VUETIFY. *Vuetify for Vue.js*. 2017. Disponível em: <https://github.com/vuetifyjs/vuetify>. Acesso em: 31 out 2017. Citado na página 3.

WEBPACK. *Code Splitting*. 2017. Disponível em: <https://webpack.js.org/guides/code-splitting>. Acesso em: 31 out 2017. Citado na página 7.

WORKBOX. *Workbox*. 2018. Disponível em: <https://developers.google.com/web/tools/workbox/>. Acesso em: 12 jun 2018. Citado na página 10.

YARN. *Yarn*. 2017. Disponível em: <https://yarnpkg.com>. Acesso em: 10 dez 2017. Citado na página 10.