Documentação Avançada: Progressive Web App (PWA)

Esta documentação detalha técnicas avançadas para levar seu PWA além do básico, focando em **desempenho**, **reengajamento** e **gerenciamento de dados offline** complexos.

---

1. Estratégias Avançadas de Caching com Service Workers

O Service Worker é o proxy de rede do seu PWA. A forma como ele lida com as requisições de rede define a velocidade e a confiabilidade do seu aplicativo. A biblioteca **Workbox** é a ferramenta recomendada para simplificar a implementação dessas estratégias.

Estratégias de Caching e Seus Usos

Estratégia	Descrição	Melhor Uso
Cache First	Serve o cache imediatamente. Se falhar, busca na rede.	Recursos estáticos (CSS, JS, App Shell) que raramente mudam.
Network First	Tenta a rede primeiro. Se falhar, usa o cache.	Conteúdo dinâmico que precisa ser o mais atualizado (HTML, APIs).
Stale-While-Revalidate	Retorna o cache instantaneamente e, em paralelo, busca a versão mais recente na rede para atualizar o cache.	Recursos que mudam com frequência, mas podem ser "stale" por um breve período (Avatares, feeds).
Cache Only	Apenas busca no cache.	Recursos pré-armazenados na instalação.
Network Only	Apenas busca na rede.	Requisições que nunca devem ser cacheadas (Analytics, POSTs).

Exemplo: Stale-While-Revalidate (Usando Workbox)

Esta estratégia é ideal para imagens, garantindo que o usuário veja algo imediatamente, mesmo que não seja a versão mais recente.

JavaScript

```
// service-worker.js (usando Workbox)
import {registerRoute} from 'workbox-routing';
import {StaleWhileRevalidate} from 'workbox-strategies';
import {ExpirationPlugin} from 'workbox-expiration';

registerRoute(
  // Rota para todas as requisições de imagens
  ({request}) => request.destination === 'image',
  new StaleWhileRevalidate({
    cacheName: 'images-cache',
    plugins: [
      new ExpirationPlugin({
        maxEntries: 60, // Limita o número de imagens cacheadas
        maxAgeSeconds: 30 * 24 * 60 * 60, // Expira após 30 dias
      }),
    ],
  })
);
```

---

2. Reengajamento: Notificações Push e Background Sync

Estes recursos permitem que seu PWA se comunique com o usuário e lide com ações offline de forma assíncrona.

2.1. Notificações Push

Permitem que seu servidor envie mensagens para o dispositivo do usuário, mesmo que o PWA esteja fechado.

Fluxo de Implementação:

1. **Assinatura**: O PWA se inscreve em um serviço de push (Web Push Protocol) e envia o objeto de assinatura para o seu servidor.
2. **Servidor**: Seu servidor envia a mensagem criptografada para o endpoint do serviço de push.
3. **Service Worker**: O Service Worker recebe o evento `push` e exibe a notificação.

Exemplo de Código (Service Worker):

JavaScript

```
// service-worker.js
```

```
self.addEventListener('push', event => {
    const data = event.data.json();
    const title = data.title || 'Nova Notificação PWA';
    const options = {
        body: data.body || 'Você tem uma nova mensagem.',
        icon: '/images/icon-192x192.png',
        data: {
            url: data.url || '/' // URL para abrir ao clicar
        }
    };

    // Exibe a notificação
    event.waitUntil(
        self.registration.showNotification(title, options)
    );
});

self.addEventListener('notificationclick', event => {
    event.notification.close();
    // Abre a janela do PWA para a URL definida
    event.waitUntil(
        clients.openWindow(event.notification.data.url)
    );
});
```

2.2. Sincronização em Segundo Plano (Background Sync)

Garante que ações importantes (como envio de formulários) sejam concluídas quando a conexão for restabelecida, evitando a perda de dados.

Exemplo de Código (PWA Principal - Registro):

JavaScript

```
// main.js (no escopo da página)

async function registerBackgroundSync() {
    if ('serviceWorker' in navigator && 'SyncManager' in window) {
        const registration = await navigator.serviceWorker.ready;
        // Registra uma tag de sincronização única
        await registration.sync.register('enviar-dados-offline');
    }
}
// Chame registerBackgroundSync() após o usuário tentar enviar um formulário
offline.
```

Exemplo de Código (Service Worker - Execução):

JavaScript

```
// service-worker.js

self.addEventListener('sync', event => {
    if (event.tag === 'enviar-dados-offline') {
        // A função enviarDadosPendentes deve buscar dados do IndexedDB e
enviá-los
        event.waitUntil(
            enviarDadosPendentes()
        );
    }
});

async function enviarDadosPendentes() {
    // Lógica para:
    // 1. Abrir IndexedDB e buscar dados.
    // 2. Tentar enviar para o servidor (fetch).
    // 3. Remover do IndexedDB se for bem-sucedido.
}
```

3. Armazenamento Offline Complexo: Cache Storage vs. IndexedDB

Para dados offline, você deve usar as duas APIs em conjunto:

API	Uso	Tipo de Dado
Cache Storage	Armazenamento de **recursos de rede** (HTML, imagens, CSS, JS).	Pares de `Request` e `Response` (objetos HTTP).
IndexedDB	Armazenamento de **dados estruturados** (objetos JSON, dados de usuário, mensagens).	Objetos JavaScript.

Exemplo de Uso do IndexedDB (Usando a Biblioteca `idb`)

O IndexedDB é ideal para dados de aplicativos. A biblioteca `idb` simplifica o uso da API com Promises.

1. Abrir/Criar o Banco de Dados (`initDB`)

JavaScript

```
import { openDB } from 'idb';

const DB_NAME = 'MeuAppDB';
const STORE_NAME = 'mensagens';
const DB_VERSION = 1;

async function initDB() {
  return openDB(DB_NAME, DB_VERSION, {
    upgrade(db) {
      // Cria o Object Store (tabela) se não existir
      if (!db.objectStoreNames.contains(STORE_NAME)) {
        const store = db.createObjectStore(STORE_NAME, {
          keyPath: 'id',
          autoIncrement: true
        });
        store.createIndex('status', 'status'); // Índice para buscar
mensagens pendentes
      }
    }
  });
}
```

2. Adicionar Dados (Mensagem Offline)

JavaScript

```
async function addMessage(message) {
  const db = await initDB();
  const tx = db.transaction(STORE_NAME, 'readwrite');
  const store = tx.objectStore(STORE_NAME);

  // Adiciona a mensagem com status 'pending'
  await store.add({ ...message, status: 'pending', timestamp: Date.now() });

  await tx.done;
  // Após adicionar, registre o Background Sync para tentar enviar depois
  registerBackgroundSync();
}
```

3. Persistência de Dados

Solicite ao navegador que torne o armazenamento persistente para evitar que o cache e o IndexedDB sejam limpos em caso de pouco espaço em disco.

JavaScript

```
async function requestPersistentStorage() {
  if (navigator.storage && navigator.storage.persist) {
    const isPersisted = await navigator.storage.persisted();
    if (!isPersisted) {
      const result = await navigator.storage.persist();
      console.log(result ? "Armazenamento persistente concedido" :
"Armazenamento persistente negado");
    }
  }
}
// Chame esta função no carregamento do seu PWA.
```

Referências

[1] *Dados off-line*. (2022). web.dev. https://web.dev/learn/pwa/offline-data?hl=pt-br

[2] *workbox-strategies*. (2017). Chrome for Developers. https://developer.chrome.com/docs/workbox/modules/workbox-strategies

[3] *Make PWAs re-engageable using Notifications and Push APIs*. (2025). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Re-engageable_Notifications_Push