UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

Caching strategies for Progressive Web Applications

Name:	**López Bautista Cristian Alexis**
Group:	10-B
Subject:	Progressive Web Applications
Professor:	Dr. Ray Brunet Parra Galaviz
Date:	Friday, March 15th, 2024

1 Introduction

Caching is one of the most exciting features of SWs. It enables us to deliver to our users a responsive, stable, and native-app-like experience. A web app can display some content and perform some function in bad network conditions, and even when the user is completely offline.

There are strategies we can use with service workers to respond to "fetch" events. These determine how the service worker will respond to a fetch event based on the type of event request.

This document will cover five different caching strategies, and describe how and when you should implement each of them:

1. Cache first, Network fallback.
2. The network first, Cache fallback.
3. Stale while revalidate
4. Network only
5. Cache only

2 When to cache resources

A PWA can cache resources at any time, but in practice there are a few times when most PWAs will choose to cache them:

2.1 In the service worker's install event handler (precaching)

When a service worker is installed, the browser fires an event called install in the service worker's global scope. At this point the service worker can precache resources, fetching them from the network and storing them in the cache.

2.2 In the service worker's fetch event handler

When a service worker's fetch event fires, the service worker may forward the request to the network and cache the resulting response, either if the cache did not already contain a response, or to update the cached response with a more recent one.

2.3 In response to a user request

A PWA might explicitly invite the user to download a resource to use later, when the device might be offline. For example, a music player might invite the user to download tracks to play later. In this case the main app thread could fetch the resource and add the response to the cache. Especially if the requested resource is big, the PWA might use the Background Fetch API, and in this case the response will be handled by the service worker, which will add it to the cache.

2.4 Periodically

Using the Periodic Background Sync API, a service worker might fetch resources periodically and cache the responses, to ensure that the PWA can serve reasonably fresh responses even while the device is offline.

3 Caching strategies

3.1 Cache first, Network fallback.

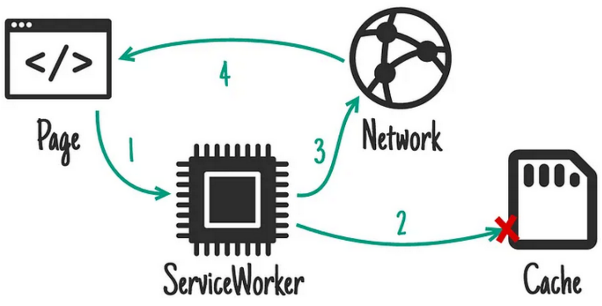

Figure 1: Cache first strategy.

The service worker will loads the local (cached) assets (HTML, CSS, images, fonts, etc.), if possible, bypassing the network. If cached content is not available, the service worker returns a response from the web instead and caches the network response. This strategy can be used when dealing with remote resources that are very unlikely to change, such as static images.

3.2 The network first, Cache fallback.

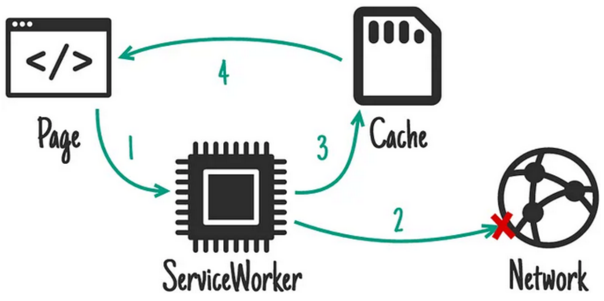

Figure 2: Network first strategy.

In this strategy, the service worker will check the network first for a response and, if successful, returns current data to the page. If the network request fails, then the service worker returns the cached entry instead. Use this when data must be as fresh as possible, such as a real-time API response, but you still want to display something as a fallback when the network is unavailable.

3.3 Stale while revalidate

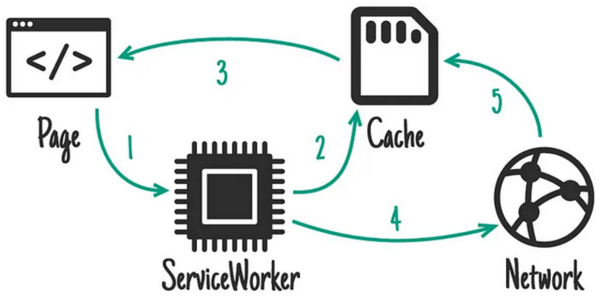

Figure 3: Stale strategy.

The stale-while-revalidate pattern allows you to respond to the request as quickly as possible with a cached response if available, falling back to the network request if it's not cached. The network request is then used to update the cache. This is a fairly common strategy where having the most up-to-date resource is not vital to the application.

3.4 Network only

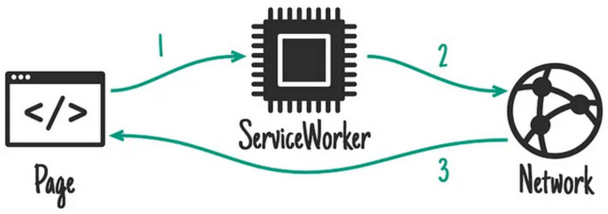

Figure 4: Network only strategy.

In this, the service worker will only check the network. There is no going to the cache for data. If the network fails, then the request fails. This can be used when only fresh data can be displayed on your site.

3.5 Cache only

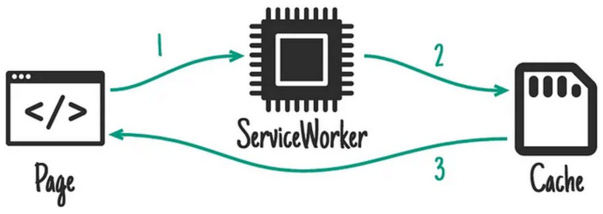

Figure 5: Cache only strategy.

The data is cached during the install event so that you can depend on the information being there. This can be useful if you have your own precaching step.

4 Examples

Pre-cache the home page assets when the user lands on the onboarding page through the install event. With this, once the user completes the onboarding, all the static assets needed to load the home page are already available in the cache.

```
self.addEventListener('install', async function (event) {
  event.waitUntil(caches.open('assets').then(function (cache) {
    return cache.addAll(filesToCache);
 }));
});
```

Network-only strategy to handle all navigations. This ensures that the pages are loaded from the network only and fall back to an offline page if the internet is not available.

```
var networkOnly = new NetworkOnly();
var navigationHandler = async function navigationHandler(params) {
  try {
    return await networkOnly.handle(params);
  } catch (error) {
    return caches.match(FALLBACK_HTML_URL, {
      cacheName: 'offline'
    });
  }
};// Register this strategy to handle all navigations.
var navigationRoute = new workbox.routing.NavigationRoute(navigationHandler);
registerRoute(navigationRoute);
```

Stale While Revalidate strategy for CSS and js assets. All the CSS/js assets are loaded from the cache and returns to the user immediately. Same time, it will check for any updates also from the network as well. If the file is updated in the network, the service worker will update the cache. So next time, when the user loads the same page, he will get the updated version.

```
registerRoute(function(_ref2) {
var request = _ref2.request;
return request.destination === 'style' || request.destination === 'script';
}, new StaleWhileRevalidate({
    cacheName: 'assets',
    plugins: [new CacheableResponsePlugin({
        statuses: [200]
    })]
}));
```

Cache First for the images. The images we use are usually not updated on the server frequently. So I've used the cache first strategy for loading the static images with few options as well. Since the images can bloat your cache disk quickly, we have put a maximum image limit in the cache with maxEntries. Use purgeOnQuotaError along with this to make sure that the cache is cleared when the quota exceeds. Also, I've applied maxAgeSeconds to expire the image cache.

```
registerRoute(function(_ref) {
    var request = _ref.request;
    return request.destination === 'image';
}, new CacheFirst({
    cacheName: 'images',
    plugins: [new CacheableResponsePlugin({
        statuses: [200]
    }), new ExpirationPlugin({
        maxEntries: 50,
        maxAgeSeconds: 60 * 60 * 24 * 30, // 30 Days
        purgeOnQuotaError: true
    })]
}));
```

5 Bibliography

1. Babu, L. V. (2021, december 27th). Best Caching strategies — Progressive Web App (PWA). Medium.

   https://medium.com/animall-engineering/best-caching-strategies-progressive-web-app-pwa-c610d65b2009

2. Caching - Progressive web apps — MDN. (2023, october 25th). MDN Web Docs.

   https://developer.mozilla.org/en-US/docs/Web/Progressive$_web_apps/Guides/Caching$

3. Educative. (s. f.). Educative Answers - Trusted Answers to Developer Questions.

   https://www.educative.io/answers/5-service-worker-caching-strategies-for-your-next-pwa-app