UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TOPIC: Estrategias de manejo de la cache en las PWA's

BY: Toledo Pérez Cristian Alejandro

GROUP: 10B

SUBJECT: PWA

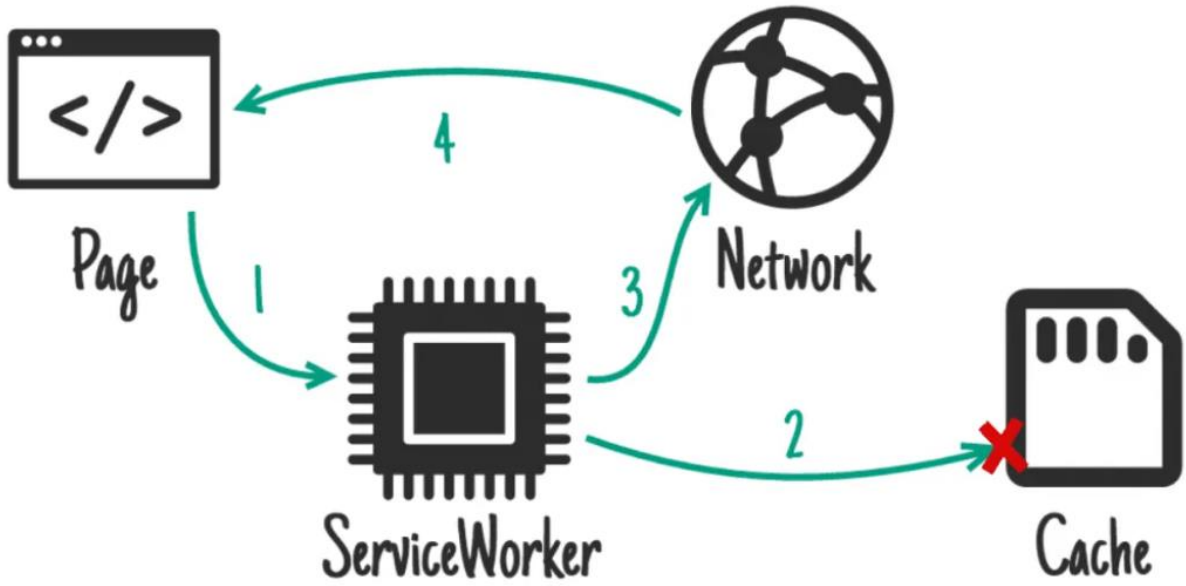
TEACHER: Ray Brunett Parra Galaviz

Tijuana, Baja California, March 15th, 2024

Enhancing User Experience with Cache Management Strategies in Progressive Web Apps (PWAs)

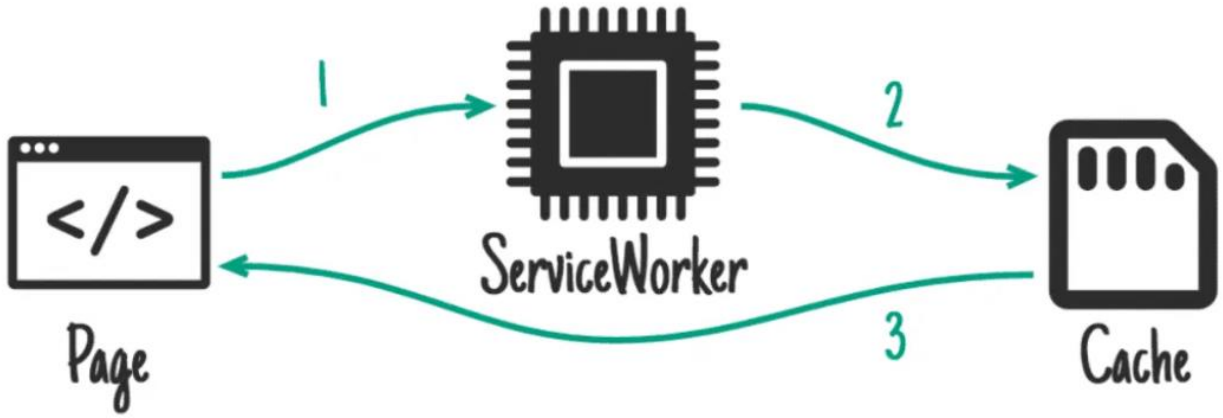
Progressive Web Apps (PWAs) have emerged as an innovative solution to provide a smooth and responsive user experience, even in environments where internet connectivity is intermittent or limited. One of the keys behind their effectiveness lies in the implementation of cache management strategies. These strategies are essential for optimizing content loading by making the most of the local caching capabilities of browsers. By combining the capabilities of the fetch and cache APIs, PWAs can store critical resources such as HTML, CSS, JavaScript, images, and data directly on the user's device. This significantly reduces dependency on the network and improves the loading speed of the application, leading to a faster and smoother experience for the end user. Some of the common strategies include:

- **Cache First:** It responds to requests with the cached version of the resource if available. If it's not available, it retrieves the resource from the network and caches it for future queries. This strategy is ideal for heavy multimedia resources that take time to load.



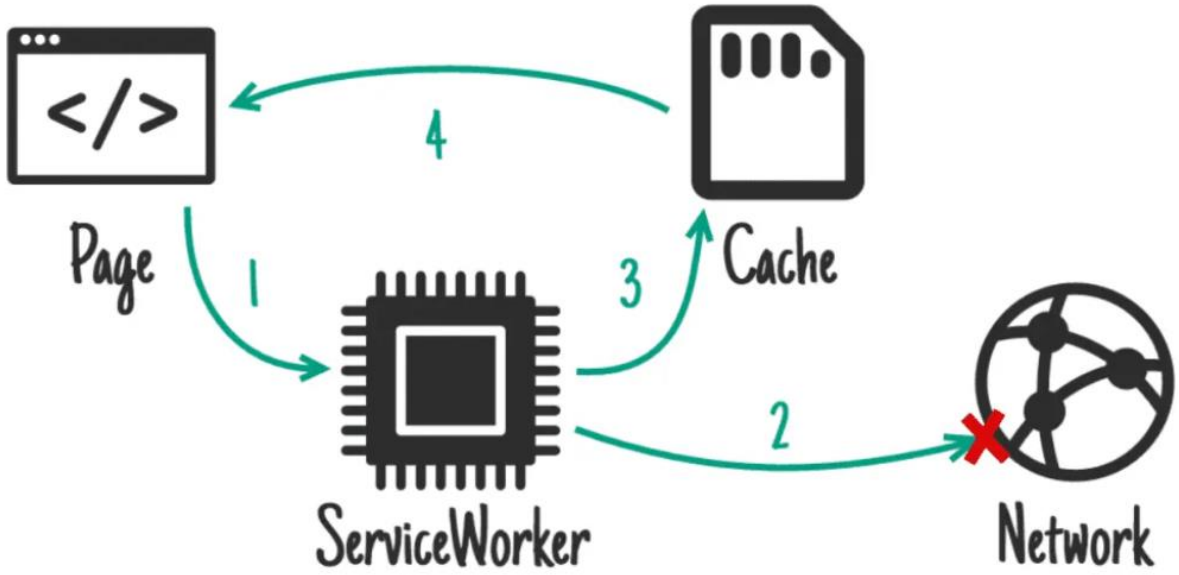

```
const cacheFirst = (event) => {
 event.respondWith(
   caches.match(event.request).then((cacheResponse) => {
     return cacheResponse || fetch(event.request).then((networkResponse) => {
       return caches.open(currentCache).then((cache) => {
         cache.put(event.request, networkResponse.clone());
         return networkResponse;
       })
     })
   })
 )
};
```

- **Cache Only**: It responds to requests directly with the cached version of the resource. If there's no cached version, it returns an error. Although less common, it can be useful in specific situations.



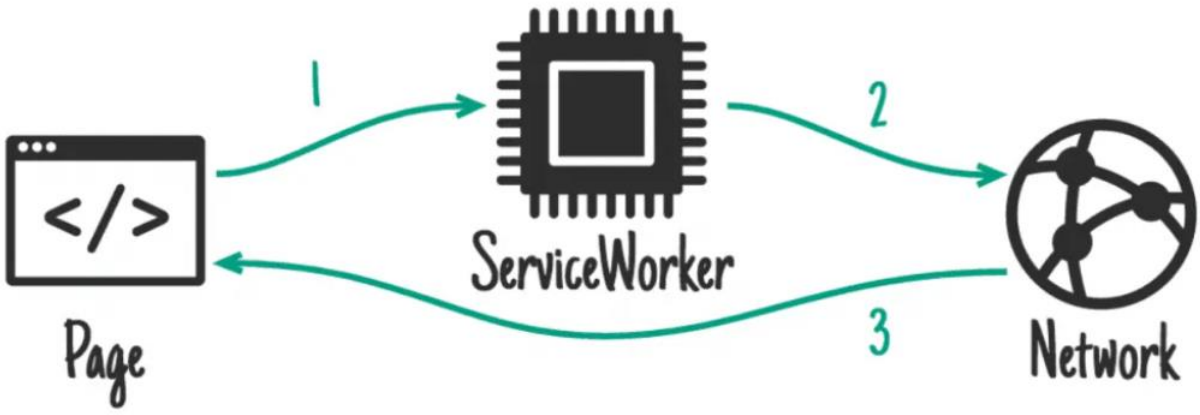

```
const cacheOnly = (event) => {
 event.respondWith(caches.match(event.request));
};
```

- **Network First**: It prioritizes fetching the most updated resource from the network, even if a cached version is available. If the network response is successful, it updates the cache. If there's an error in the network response, it resorts to the cached resource.



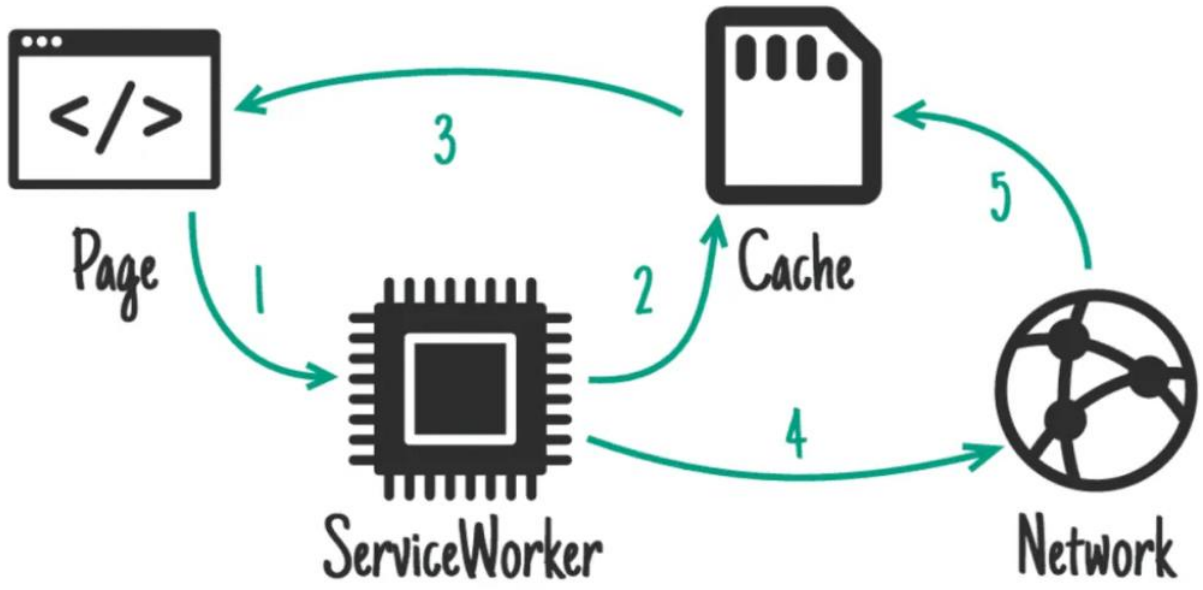

```
const networkFirst = (event) => {
 event.respondWith(
   fetch(event.request)
     .then((networkResponse) => {
       return caches.open(currentCache).then((cache) => {
         cache.put(event.request, networkResponse.clone());
         return networkResponse;
       })
     })
     .catch(() => {
       return caches.match(event.request);
     })
 )
};
```

- **Network Only**: It always prioritizes fetching the most updated resource from the network. If the request fails, it returns an error. It's useful when the latest version of a resource is needed, such as in critical API calls.




```
const networkOnly = (event) => {
 event.respondWith(fetch(event.request));
};
```

- **Stale While Revalidate**: This strategy combines the above by prioritizing cached content to load resources instantly while simultaneously making a network request to update the cache with the latest resource version. If nothing is cached, it makes a request to the network. It's commonly used for static files that change infrequently.




```
const staleWhileRevalidate = (event) => {
 event.respondWith(
   caches.match(event.request).then((cacheResponse) => {
     if (cacheResponse) {
       fetch(event.request).then((networkResponse) => {
         return caches.open(currentCache).then((cache) => {
           cache.put(event.request, networkResponse.clone());
           return networkResponse;
         })
       });
       return cacheResponse;
     } else {
       return fetch(event.request).then((networkResponse) => {
         return caches.open(currentCache).then((cache) => {
           cache.put(event.request, networkResponse.clone());
           return networkResponse;
         })
       });
     }
   })
 );
};
```

Bibliographic Source

Serrano, D. (2021, 9 junio). *Service worker: Estrategias de caché y modo offline* | *Apiumhub*. Apiumhub. Recuperado 15 de marzo de 2024, de https://apiumhub.com/es/tech-blog-barcelona/service-worker-estrategias-de-cache/