Tesi - Payment Request API

Daniele Rigon - 857319

24 luglio 2018

Indice

1 Overview

Un ServiceWorker è uno script Javascript che utilizza le Promises per poter eseguire operazioni in modalità asincrona nel browser, avviate in background separato dalla pagina; pertanto non possono modificarne gli elementi del DOM come i normali script ma può comunicare con essi mediante "messaggi". Un ServiceWorker si trova tra la nostra applicazione Web e la rete e, come un server proxy, può intercettare tutte le richieste a pagine web e file statici e rispondere secondo politiche che siamo noi stessi a decidere. I ServiceWorker sono pensati per consentire la creazione di esperienze offline efficaci, intercettare le richieste di rete e intraprendere azioni appropriate in base al fatto che la rete sia disponibile o meno e aggiornare le risorse che risiedono sul server, oltre a consentire l'accesso alle notifiche push e alle API di sincronizzazione in background. È il browser che in qualsiasi momento deciderà se il ServiceWorker dovrebbe essere o meno in esecuzione così da risparmiare risorse, specialmente sui dispositivi mobili. Per questo può essere che se non facciamo alcuna richiesta HTTP per un certo periodo di tempo o non riceviamo alcuna notifica per un po' è possibile che il browser spenga il Service Worker. Se attiviamo una richiesta HTTP che deve essere gestita dal ServiceWorker il browser la attiverà di nuovo, nel caso in cui non fosse ancora in esecuzione.

1.1 Impostare i service worker

Molte funzionalità dei Service Worker oggi sono abilitate di default, ma nel caso non lo fossero bisogna abilitarle nel browser:

- Firefox: su `about:config` impostare dom.serviceWorkers.enabled su true, riavvia il browser;
- Chrome : su `chrome://flags` accendere experimental-web-platform-features, riavvia browser;
- Opera : su `opera://flags` attivare Support for ServiceWorker, riavvia il browser;
- Microsoft Edge : su `about:flags` spuntare Enable service workers, riavvia il browser.

1.2 Architettura di base

Per quanto riguarda i ServiceWorker generalmente vengono eseguiti questi passaggi per l'impostazione di base:

- L'URL del ServiceWorker viene recuperato e registrato tramite serviceWorkerContainer.register();
- In caso di esito positivo il ServiceWorker viene eseguito in un ServiceWorkerGlobalScope, ovvero un tipo speciale di ServiceContext che scappa dal thread di esecuzione dello script principale senza accesso DOM. Il ServiceWorker ora è pronto per elaborare gli eventi;
- L'installazione del ServiceWorker viene tentata quando si accede successivamente alle pagine: un evento di installazione è sempre il primo inviato a un ServiceWorker;
- Quando il ServiceWorker è considerato installato il passo successivo è l'attivazione, quindi quando il ServiceWorker è installato riceve un evento di attivazione.

1.3 Casi d'uso

I Service Worker sono destinati anche ad altri usi:

- Sincronizzazione dei dati in background;
- Risposta a richieste di risorse da altre origini;
- Ricezione di aggiornamenti centralizzati a dati costosi da calcolare in modo che più pagine possano utilizzare un set di dati;
- Modelli personalizzati basati su determinati pattern URL;

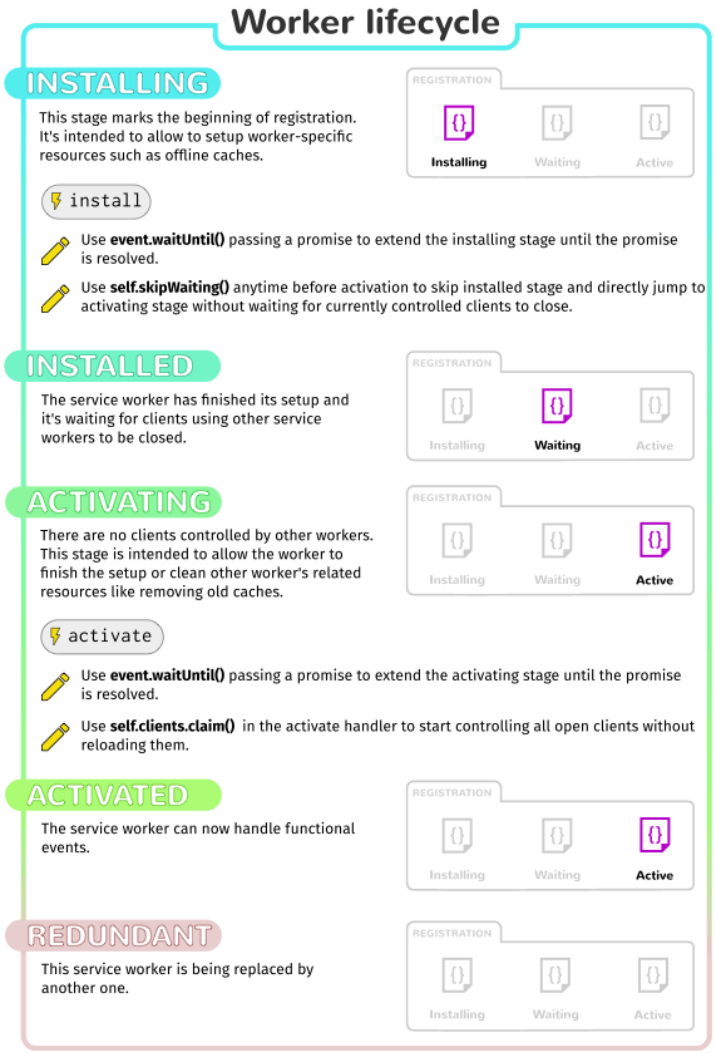

Figura 1: Ciclo di vita del Service Worker

- Miglioramenti delle prestazioni, ad esempio prelettura delle risorse che l'utente probabilmente avrà bisogno nel prossimo futuro.

Altre specifiche sono utilizzate dal Service Context, ad esempio:

- Sincronizzazione in background : avvia un operatore di servizio anche quando nessun utente si trova sul sito, quindi le cache possono essere aggiornate, ecc;
- Reagire per inviare messaggi : si può avviare un Service Worker per inviare agli utenti un messaggio per comunicare loro che sono disponibili nuovi contenuti;
- Reagendo ad orari e date particolari.

2 Ciclo di vita di un Service Worker

Il ciclo di vita di un service worker è composto da quattro fasi:

- Registrazione: il service worker viene scaricato dal browser, analizzato ed eseguito;
- Installazione: il service worker viene installato;

- Attivazione: il service worker è pronto ed è in grado di poter controllare gli eventi generati dal client;
- Fetch: evento generato dal client. Il service worker è in grado di intercettare le richieste e rispondere secondo le opportune strategie di caching.

2.1 Registrazione

Come prima cosa bisogna comunicare al browser l'esistenza di un ServiceWorker all'interno del sito web. Un ServiceWorker viene prima registrato utilizzando il metodo ServiceWorker.register() e per farlo basta inserire su tutte le pagine del sito uno script come il seguente:

```
1 if ('serviceWorker' in navigator) {
2   // Path che contiene il service worker navigator.serviceWorker.register('/service-
      worker.js').then(function(registration) {
3   console.log('Service worker installato correttamente, ecco lo scope:', registration.
      scope); }).catch(function(error) {
4       console.log('Installazione service worker fallita:', error);
5     });
6 }
```

Il codice inizia controllando il supporto da parte del browser verificando la presenza di navigator.serviceWorker. Se supportato, il ServiceWorker viene registrato per mezzo di navigator.serviceWorker.register che restituisce un oggetto Promise il quale si risolve con successo a registrazione avvenuta correttamente. service-worker.js è il file Javascript residente nella root del sito web e che contiene il codice del service worker, il cui codice è:

```
1 // Evento install
2 self.addEventListener('install', event => { // Codice da eseguire su installazione
      console.log("Service Worker Installato");
3 });
4 // Evento activate
5 self.addEventListener('activate', event => { // Codice da eseguire su attivazione console
      .log("Service Worker Attivo");
6 });
7 // Evento fetch
8 self.addEventListener('fetch', event => { // Codice da eseguire su fetch di risorse
      console.log("Richiesta URL: "+event.request.url);
9 });
```

In questo modo viene registrato un ServiceWorker il quale viene semplicemente installato e ad ogni richiesta stampa in console un messaggio con la URL che il browser tenta di scaricare dal server web. Per controllare il caricamento di un Service Worker il codice di questo deve essere eseguito al di fuori delle normali pagine.

Possono esserci diversi motivi per cui il Service Worker non si registra:

- Non si sta eseguendo l'applicazione tramite HTTPS;
- Il path del Service Worker non è scritto correttamente: deve essere scritto in relazione all'origine, non alla directory radice dell'app;
- Il Service Worker a cui ci si riferisce ha un'origine diversa da quella della tua app.

2.2 Installazione

Il ServiceWorker viene scaricato immediatamente quando un utente accede per la prima volta a un sito, o una pagina, controllata dal ServiceWorker, e sarà poi scaricato periodicamente ogni tot periodo di tempo.

L'installazione viene tentata quando il file nuovo che è stato scaricato risulta diverso da un ServiceWorker esistente, o risulta essere diverso dal primo ServiceWorker rilevato per quella pagina/sito. Se è la prima volta che un ServiceWorker viene reso disponibile viene tentata l'installazione e, dopo un'installazione corretta, viene attivato. Se è disponibile un ServiceWorker esistente la nuova versione viene

installata in background, ma non ancora attivata; si attiva solo quando non ci sono più pagine caricate che stanno ancora utilizzando il vecchio ServiceWorker. Non appena non ci sono più pagine da caricare il nuovo ServiceWorker si attiva.

Conseguentemente all'installazione viene richiamato l'evento install: tale evento consente di effettuare il precaching, ovvero inserire in cache pagine e file statici del sito web prima di intercettarne le richieste. Per farlo occorre utilizzare gli oggetti Promise event e cache come segue:

```
1  'use strict';
2  // Array di configurazione del service worker
3  var config = {
4  version: 'versionesw1::',
5  // Risorse da inserire in cache immediatamente - Precaching
6  staticCacheItems: [ '/wp-includes/js/jquery/jquery.js', '/wp-content/themes/miotema/
     logo.png', '/wp-content/themes/miotema/fonts/opensans.woff', '/wp-content/themes/
     miotema/fonts/fontawesome-webfont.woff2',
7  ],
8  };
9  // Funzione che restituisce una stringa da utilizzare come chiave per la cache
10 function cacheName (key, opts) { return `${opts.version}${key}`;
11 }
12 // Evento install
13 self.addEventListener('install', event => { event.waitUntil( // Inserisco in cache le
     URL configurate in config.staticCacheItems caches.open( cacheName('static', config) )
     .then(cache => cache.addAll(config.staticCacheItems)) // self.skipWaiting() evita l'
     attesa, il che significa che il service worker si attivera immediatamente non appena
     conclusa l'installazione .then( () => self.skipWaiting() ) ); console.log("Service
     Worker Installato");
14 });
```

Se si decidesse di aggiungere/eliminare nuove risorse da inserire in cache bisognerà avere l'accortezza di cambiare il nome della versione del ServiceWorker ed eliminare dalla cache le risorse già presenti. Una cosa molto importante da sapere è che le risorse da inserire in cache in fase di precaching devono esistere realmente sul server web altrimenti il ServiceWorker genererà un errore fatale e l'installazione non andrà a buon fine. Il metodo skipWaiting() consente al ServiceWorker di passare allo stato di attivazione ad installazione conclusa e quindi essere subito operativo.

2.3 Attivazione

Una volta installato il ServiceWorker passa nello stato di attivazione. Se la pagina al momento è controllata da un altro ServiceWorker quello attuale passa in uno stato di attesa per poi diventare operativo al prossimo caricamento di pagina quando il vecchio ServiceWorker viene sostituito. Questo per essere sicuri che solo un ServiceWorker (o una sola versione di ServiceWorker) per volta possa essere eseguito nello stesso contesto. A ServiceWorker attivato viene richiamato l'evento activate, ovvero l'evento per svuotare la cache obsoleta dell'eventuale precedente versione di ServiceWorker. Dopodiché il ServiceWorker sarà in grado di effettuare il fetching di risorse o di restare in attesa di altri eventi. Di default il nuovo ServiceWorker diventa operativo al refresh della pagina o dopo aver richiamato il metodo clients.claim(); fino a quel momento le eventuali richieste non saranno intercettate.

2.4 Fetch

Grazie all'evento fetch il ServiceWorker potrà agire da proxy tra l'applicazione web e la rete. Il ServiceWorker intercetterà ogni richiesta HTTP del browser e sarà in grado di rispondere a quest'ultimo prendendo la risorsa dalla cache piuttosto che scaricarla dalla rete. Grazie all'evento fetch il ServiceWorker diventa un vero e proprio strumento per migliorare le performance di caricamento di un sito web.

2.5 Aggiornare il Service Worker

Se il ServiceWorker è già stato installato ma una nuova versione è disponibile per l'aggiornamento o il caricamento della pagina, la nuova versione viene installata in background ma non sarà ancora attivata.

Si attiva solo quando non ci sono più pagine caricate che stanno ancora utilizzando il vecchio servizio. Non appena non ci sono più pagine di questo tipo ancora caricate, il nuovo ServiceWorker si attiverà.

Si dovrà aggiornare il listener install di eventi nel nuovo Service Worker, similmente a questo:

```
self.addEventListener('install', function(event) {
  event.waitUntil(
    caches.open('v2').then(function(cache) {
      return cache.addAll([
        '/sw-test/',
        '/sw-test/index.html',
        '/sw-test/style.css',
        '/sw-test/app.js',
        '/sw-test/image-list.js',

        // include other new resources for the new version...
      ]);
    })
  );
});
```

Mentre accade questo è ancora la versione precedente (v1) quella responsabile per i recuperi, mentre la nuova versione (v2) si sta installando in background. Quando nessuna pagina sta utilizzando la versione corrente, il nuovo operatore si attiva e diventa responsabile dei recuperi.

2.6 Disintallare il Service Worker

Rimuovere/disinstallare un ServiceWorker è un'operazione semplice. È possibile eseguirla manualmente dal proprio browser oppure inserendo un semplice script al posto di quello di registrazione del service worker:

```
  navigator.serviceWorker.getRegistrations().then(function(registrations) {
    for(let registration of registrations) {
      registration.unregister()
    }
  })
```

Naturalmente è necessario che la pagina contenente il codice di disinstallazione venga visitata dal browser, oppure è possibile rimuovere il ServiceWorker manualmente tramite DevTools.

3 Strategie di caching

Diverse sono le strategie che possono essere adottate per migliorare le performance di un sito web mediante i service worker. A seconda del sito e del contesto è possibile adottare una strategia piuttosto che l'altra. È importante sottolineare che il ServiceWorker non utilizza cache a meno che non siamo noi a dirlo, quindi di default il comportamento nella fase di fetch delle risorse sarà quello nativo del browser. Di seguito l'elenco completo delle strategie con esempi di codice di implementazione.

3.1 Network first

Questa strategia mira ad avere un contenuto sempre fresco scaricandolo dalla rete, fornendo la copia in cache solo in caso di problemi di connettività (ad esempio in caso di connessione offline).

```
self.addEventListener('fetch', function(event) {
    event.respondWith(fetch(event.request).catch(function() {
      return caches.match(event.request);
    })
  );
});
```

Una modifica interessante in questo caso potrebbe essere quella di aggiornare la copia in cache quando la risorsa viene scaricata dalla rete, cosicché in caso di errori di connessione viene restituita la copia più giovane.

```
self.addEventListener('fetch', function(event){
    event.respondWith(fetch(event.request).then(function(response) {
      cache.put(event.request, response.clone());
      return response;
    }).catch(function() {
      return caches.match(event.request);
    })
  );
});
```

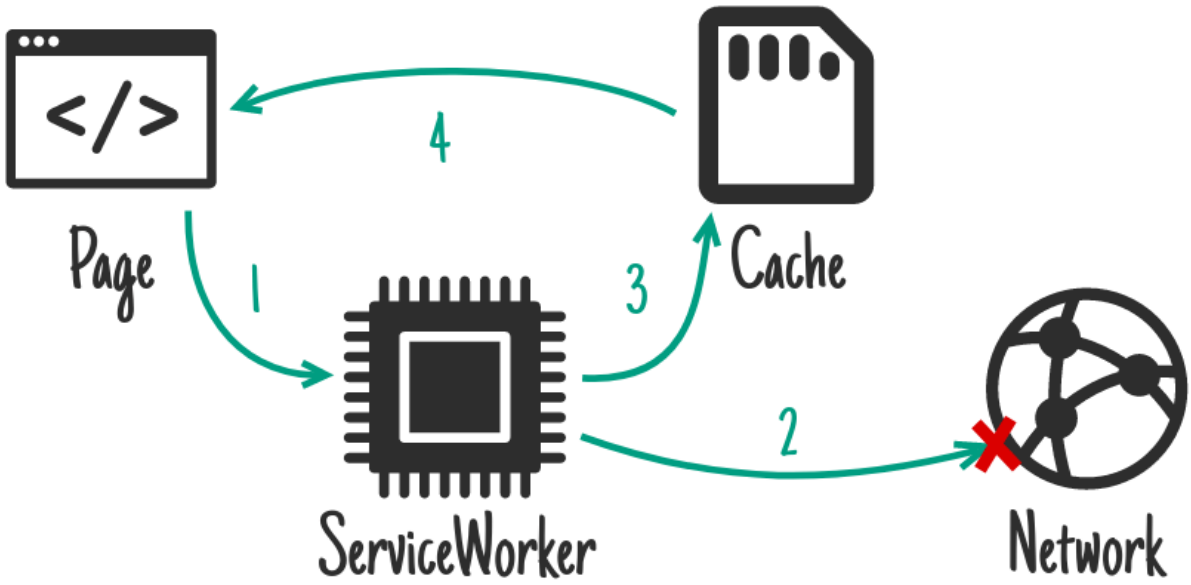

Figura 2: Network first

3.2 Cache first

Chiamata anche cache, falling back to network, questa strategia verifica se la risorsa è disponibile in cache. Se così fosse viene restituita la copia in cache. In caso contrario la risorsa viene scaricata dalla rete.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(
    caches.match(event.request).then(function(response) {
      return response || fetch(event.request);
    })
  );
});
```

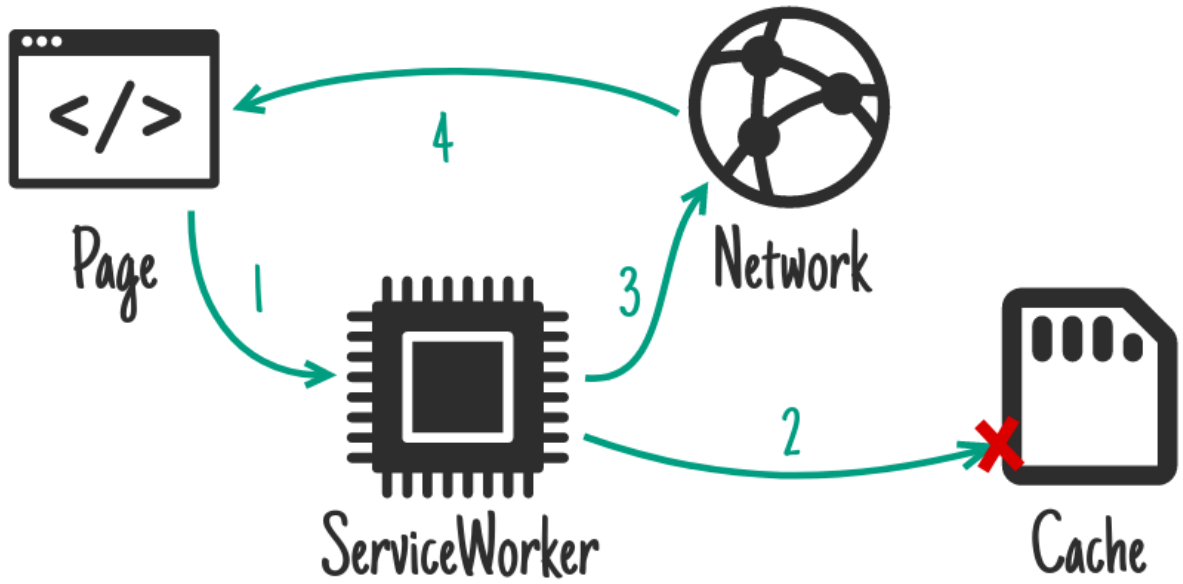

Figura 3: Cache First

3.3 Network only

È la strategia più banale in quanto viene simulato il normale comportamento del browser, ovvero scaricare le risorse direttamente dalla rete.

Per applicare questa strategia basta non inserire alcuna riga di codice all'interno dell'evento fetch:

```
1 self.addEventListener('fetch', function(event) {});
```

o al limite inserire semplicemente la seguente riga:

```
1 self.addEventListener('fetch', function(event) {
2   event.respondWith(fetch(event.request));
3 });
```

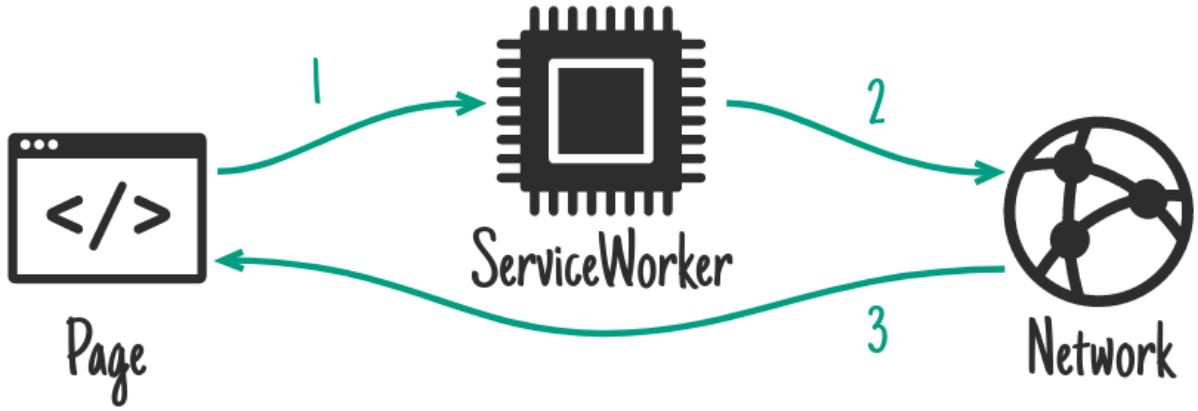

Figura 4: Network only

3.4 Cache only

Esattamente opposta alla strategia network only, in questo caso il service worker risponde solo con elementi conservati in cache. In caso di miss la risposta restituita al browser simulerà l'errore di connessione.

```
self.addEventListener('fetch', function(event) {
  event.respondWith(caches.match(event.request));
});
```

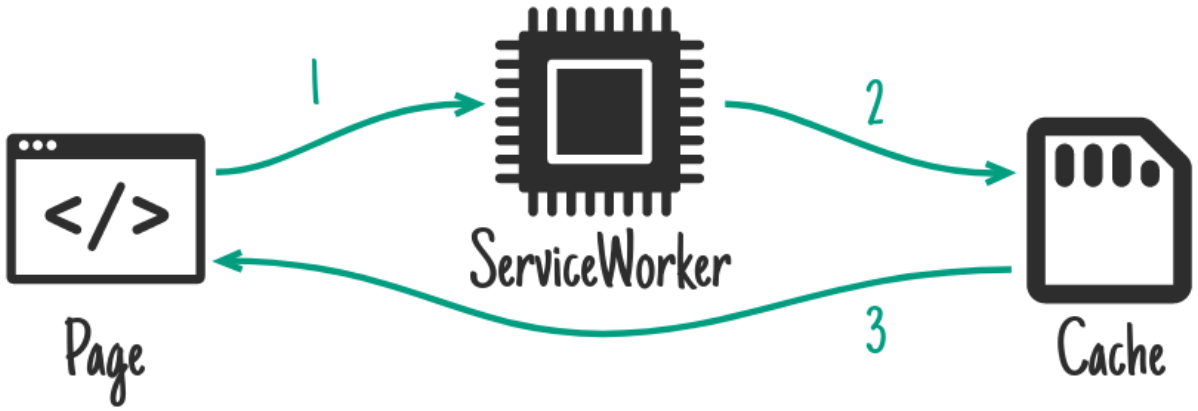

Figura 5: Cache Only

3.5 Fastest

Questa strategia mira a fornire all'utente la risposta più veloce. Il ServiceWorker avvia contemporaneamente una richiesta in cache ed una in rete. La prima che risponde verrà restituita all'utente. Questa soluzione può essere l'ideale per quei dispositivi con vecchi hard drive dove la lettura da disco può addirittura rivelarsi più lenta del fetch dalla rete. Per i dispositivi moderni è meglio utilizzare la strategia cache then network. Siccome il ServiceWorker può ritornare un solo Promise, occorre realizzare una funzione a cui passare un array di oggetti Promise, in questo caso cache e fetch, e risolverli quasi contemporaneamente ritornando quello che si risolve per primo.

```
function promiseAny(promises) {
  return new Promise((resolve, reject) => {
    promises = promises.map(p => Promise.resolve(p));
    promises.forEach(p => p.then(resolve));
    promises.reduce((a, b) => a.catch(() => b)).catch(() => reject(Error("All failed")));
  });
};
self.addEventListener('fetch', function(event) {
  event.respondWith( promiseAny([caches.match(event.request), fetch(event.request)]) );
});
```

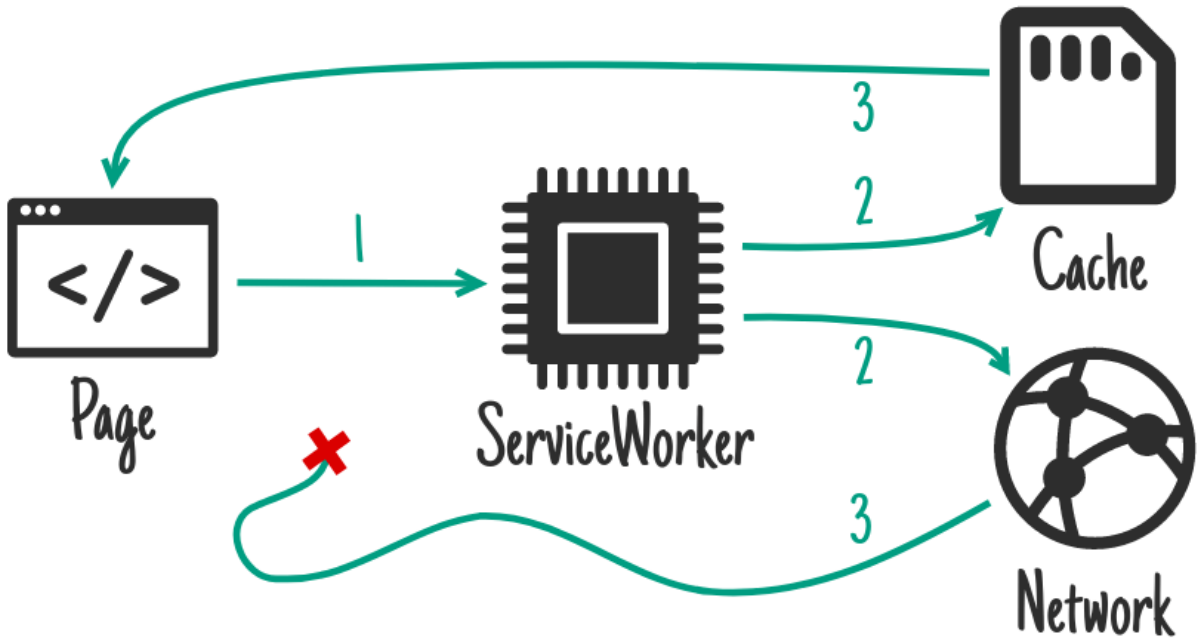

Figura 6: Fastest

3.6 Cache then network

Questa strategia mira a fornire il contenuto dalla cache per una risposta molto rapida. Dopodiché in parallelo si avvia una richiesta in rete per scaricare una copia aggiornata della risorsa e sostituirla con quella in cache. La risorsa ricevuta dalla rete viene poi sostituita con quella presente sulla pagina. Per ottenere questo obiettivo occorre avere sia codice lato pagina che lato ServiceWorker. Questo perché il ServiceWorker deve rispondere subito e non può attendere il completamento di un secondo task senza rallentare l'intera operazione. Per ottenere qualcosa di analogo usando il solo ServiceWorker occorre utilizzare postMessage affinché la pagina comunichi al service worker la risorsa da interpellare con un secondo fetch, sia esso dalla cache o dalla rete. La complessità rimane uguale ma molto utile in caso si utilizzi il service worker per fare page caching.

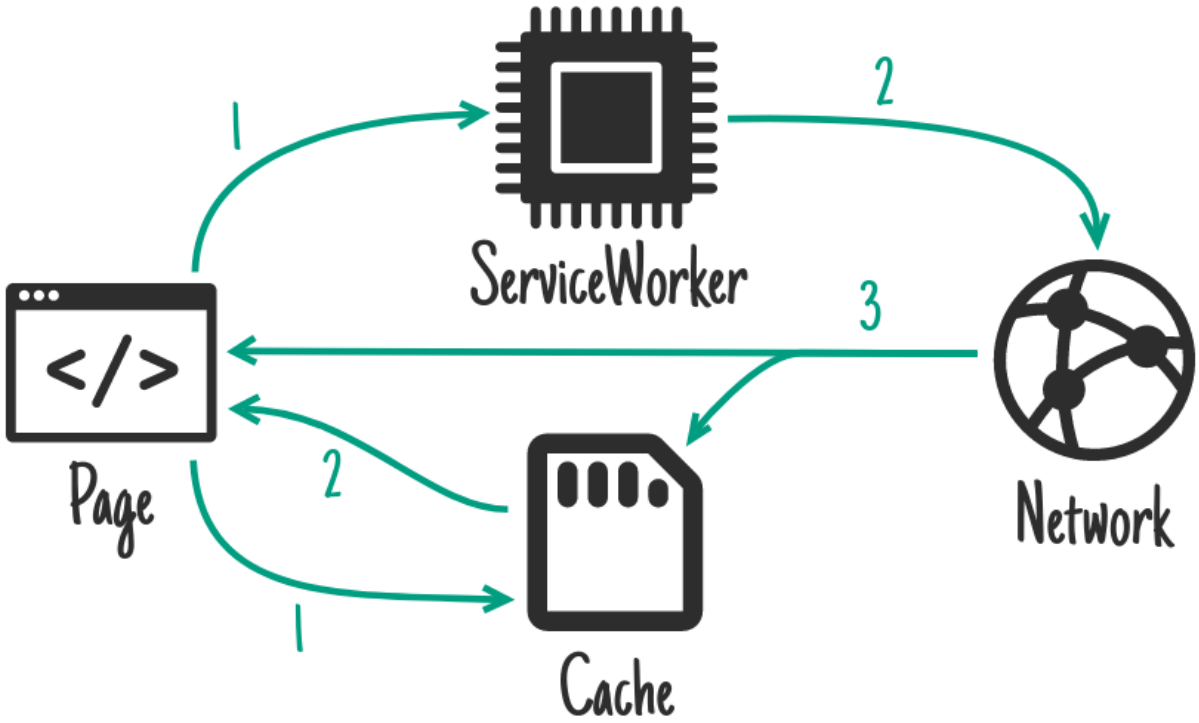

Figura 7: Cache then Network

4 Compatibilità web

4.1 Desktop

Desktop | Mobile

Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support	40.0	16[2]	33.0 (33.0)[1]	No support	24	No support

Figura 8: Compatibilità web

4.2 Mobile

Desktop | **Mobile**

Feature	Android Webview	Chrome for Android	Firefox Mobile (Gecko)	Firefox OS	IE Phone	Opera Mobile	Safari Mobile
Basic support	No support	40.0	(Yes)	(Yes)	No support	(Yes)	No support

Figura 9: Compatibilità mobile

5 Conclusioni

Se realizzati da un esperto i service worker possono rendere la navigazione del sito web molto ma molto veloce. Grazie alla loro implementazione esterna, non richiedono modifica alcuna al sito web, motivo per cui sono molto apprezzati nel ramo delle web performance.