Tesi - Payment Request API

Daniele Rigon - 857319

17 luglio 2018

Indice

1 Overview

Un service worker è uno script Javascript, che utilizza gli oggetti Promise per poter eseguire operazioni in modalità asincrona (quindi non bloccanti), che il browser avvia in background separato dalla pagina, pertanto non può modificarne gli elementi come i normali script (non ha accesso al DOM) ma può comunicare con essi mediante "messaggi".

Un Service Worker 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 service worker 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 Service Worker dovrebbe essere o meno in esecuzione: questo per 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 Service Worker, il browser la attiverà di nuovo, nel caso in cui non fosse ancora in esecuzione. Quindi vedere il Service Worker bloccato in Dev Tools non significa necessariamente che qualcosa è rotto o non va.

1.1 Impostare i service worker

Molte funzionalità dei Service Worker oggi sono abilitate di default nelle versioni più recenti dei browser. Se il codice demo seguente non funziona bisogna abilitare un pref:

- 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.

2 Architettura di base

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

- L'URL del Service Worker viene recuperato e registrato tramite serviceWorkerContainer.register();
- In caso di esito positivo, il Service Worker viene eseguito in un ServiceWorkerGlobalScope, ovvero un tipo speciale di Service Context che scappa dal thread di esecuzione dello script principale senza accesso DOM.
- Il Service Worker ora è pronto per elaborare gli eventi;
- L'installazione del Service Worker viene tentata quando si accede successivamente alle pagine. Un evento di installazione è sempre il primo inviato a un Service Worker;
- Quando il Service Worker è considerato installato, il passo successivo è l'attivazione; quindi quando il Service Worker è installato riceve un evento di attivazione. L'uso principale di onactivate è per la pulizia delle risorse utilizzate nelle versioni precedenti di uno script di servizio.

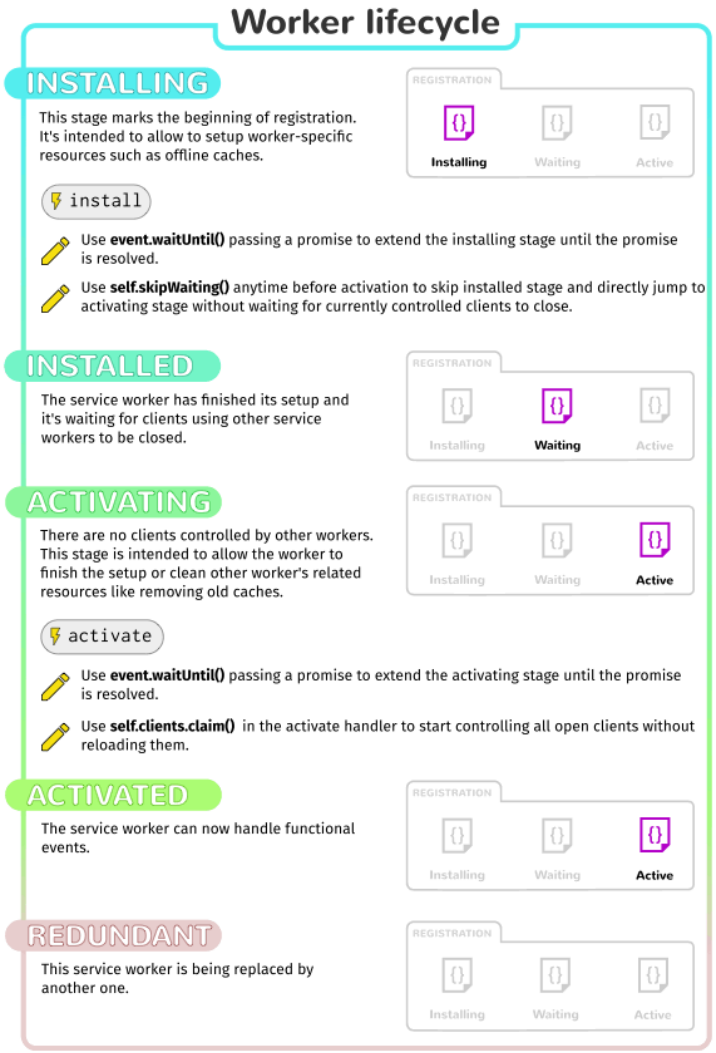

Figura 1: Ciclo di vita del Service Worker

2.1 Casi d'uso

I Service Worker sono anche destinati a essere utilizzati per cose come:

- Sincronizzazione dei dati in background;
- Rispondere alle 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;
- 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.

3 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.

3.1 Registrazione

Come prima cosa bisogna comunicare al browser l'esistenza di un service worker all'interno del sito web. Un Service Worker 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) { // Path che contiene il service worker navigator.
      serviceWorker.register('/service-worker.js').then(function(registration) { console.
      log('Service worker installato correttamente, ecco lo scope:', registration.scope);
      }).catch(function(error) { console.log('Installazione service worker fallita:', error
      ); });
2 }
```

Il codice inizia controllando il supporto da parte del browser verificando la presenza di navigator.serviceWorker. Se supportato il service worker 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 Service Worker il quale al momento non fa nulla di interessante, 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.

3.2 Installazione

Il Service Worker viene scaricato immediatamente quando un utente accede per la prima volta a un sito, o una pagina, controllata dal Service Worker, 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 Service Worker esistente, o risulta essere diverso dal primo Service Worker rilevato per quella pagina/sito. Se è la prima volta che un Service Worker viene reso disponibile viene tentata l'installazione e, dopo un'installazione corretta, viene attivato. Se è disponibile un Service Worker 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 Service Worker. Non appena non ci sono più pagine da caricare, il nuovo Service Worker 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 service worker 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 service worker genererà un errore fatale e l'installazione non andrà a buon fine. Il metodo skipWaiting() consente al service worker di passare allo stato di attivazione ad installazione conclusa e quindi essere subito operativo.

3.3 Attivazione

Una volta installato, il service worker passa nello stato di attivazione. Se la pagina al momento è controllata da un altro service worker, l'attuale passa in uno stato di attesa per poi diventare operativo al prossimo caricamento di pagina, quando il vecchio service worker viene sostituito.

Questo per essere sicuri che solo un service worker (o una sola versione di service worker) per volta possa essere eseguito nello stesso contesto temporale.

A service worker attivato, viene richiamato l'evento activate, l'evento ideale per svuotare la cache obsoleta dell'eventuale precedente versione di service worker. Dopodiché il service worker sarà in grado di effettuare fetching di risorse o restare in attesa di altri eventi.

Di default il nuovo service worker diventa operativo al refresh della pagina o dopo aver richiamato il metodo clients.claim(). Fino a quel momento le eventuali richieste non saranno intercettate.

Di seguito un codice di esempio eseguito al verificarsi dell'evento activate. Il codice effettua il purge della cache di versioni del service worker diverse dall'attuale e successivamente richiama clients.claim() per poter intercettare le richieste fin da subito.

```
1  self.addEventListener('activate', event => { // Questa funzione elimina dalla cache
     tutte le risorse la cui chiave non contiene il nome della versione // impostata sul
     config di questo service worker function clearCacheIfDifferent(event, opts) { return
     caches.keys().then(cacheKeys => { var oldCacheKeys = cacheKeys.filter(key => key.
     indexOf(opts.version) !== 0); var deletePromises = oldCacheKeys.map(oldKey => caches.
     delete(oldKey)); return Promise.all(deletePromises); }); }
2  event.waitUntil( // Se la versione del service worker cambia, svuoto la cache
     clearCacheIfDifferent(event, config) // Con self.clients.claim() consento al service
     worker di poter intercettare le richieste (fetch) fin da subito piuttosto che
     attendere il refresh della pagina .then( () => self.clients.claim() ) ); console.log
     ("Service Worker Avviato");
3  });
```

3.4 Fetch

Grazie all'evento fetch il service worker potrà agire da proxy tra l'applicazione web e la rete.

Il service worker 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 o applicando le più disparate strategie di caching.

Grazie all'evento fetch il service worker diventa un vero e proprio strumento per migliorare le performance di caricamento di un sito web.

3.5 Aggiornare il Service Worker

Se il Service Worker 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 Service Worker si attiverà.

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

```
1  self.addEventListener('install', function(event) {
2    event.waitUntil(
3      caches.open('v2').then(function(cache) {
4        return cache.addAll([
5          '/sw-test/',
6          '/sw-test/index.html',
7          '/sw-test/style.css',
8          '/sw-test/app.js',
9          '/sw-test/image-list.js',
10
11         // include other new resources for the new version...
12       ]);
13     })
14   );
15 });
```

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.

3.6 Disintallare il Service Worker

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

```
1  navigator.serviceWorker.getRegistrations().then(function(registrations) { for(let
     registration of registrations) { registration.unregister() }
2  })
```

Naturalmente è necessario che la pagina contenente il codice di disinstallazione venga visitata dal browser. È possibile procedere anche alla rimozione manuale su Google Chrome semplicemente aprendo la DevTools e cliccando prima su Application e successivamente su Service Workers. Accanto al service worker sarà presente un link dal nome Unregister. Non resta che cliccarci sopra per disinstallare e rimuovere il service worker in via definitiva.

4 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 contesto e del modello di business del sito è possibile adottare una strategia piuttosto che l'altra.

È importante sottolineare che il service worker 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.

4.1 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).

```
1 self.addEventListener('fetch', function(event) { event.respondWith( fetch(event.request).
      catch(function() { return caches.match(event.request); }) );
2 });
```

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.

```
1 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); }) );
2 });
```

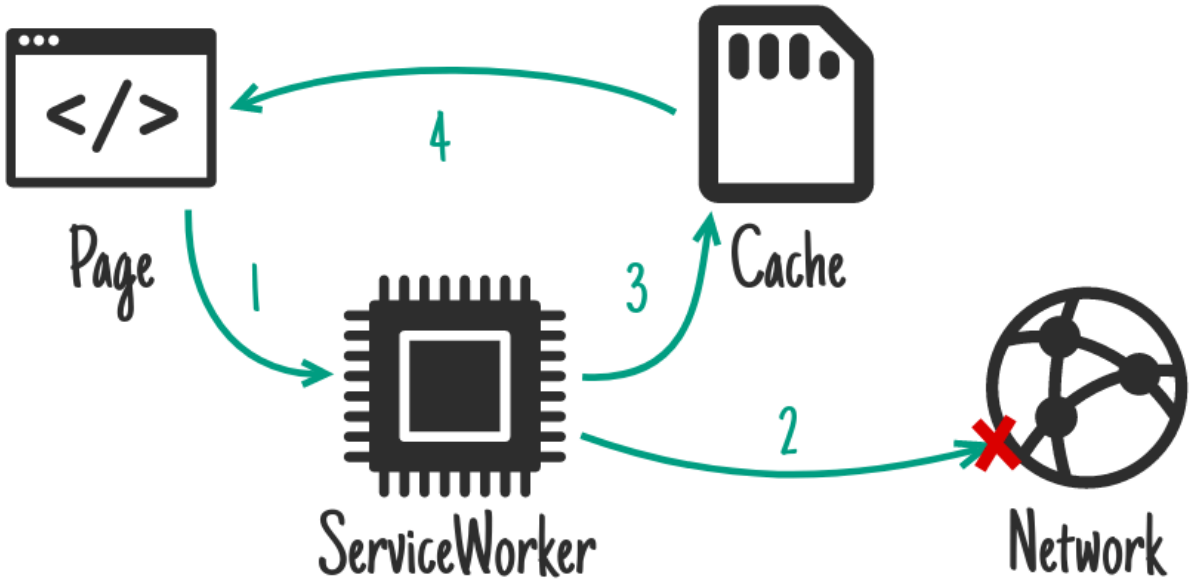

Figura 2: Network first

4.2 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);
  })
  );
});
```

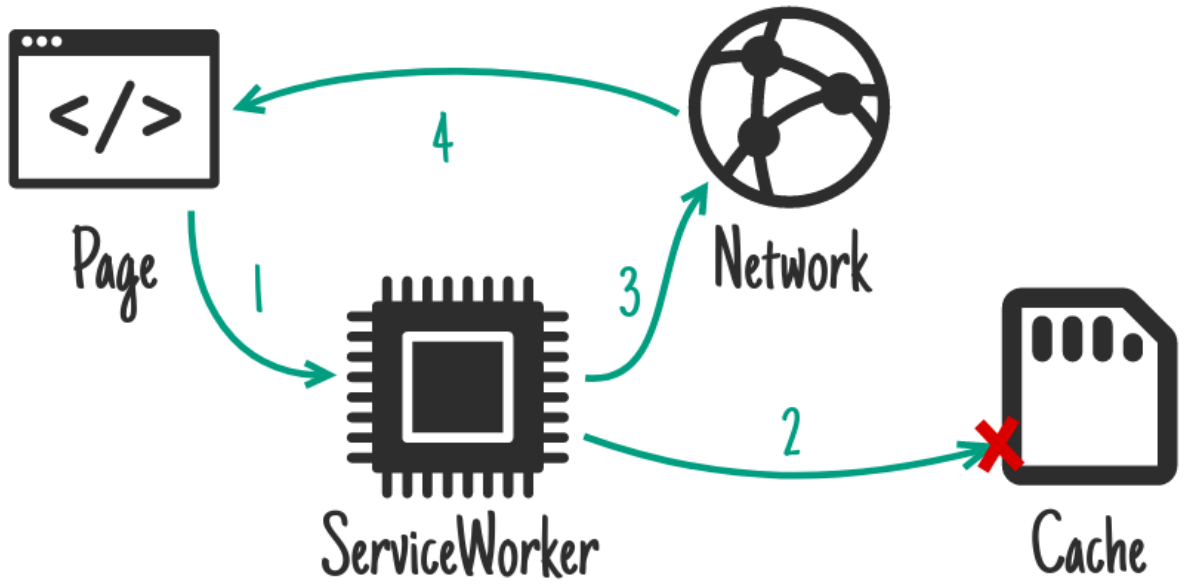

Figura 3: Cache First

4.3 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 });
```

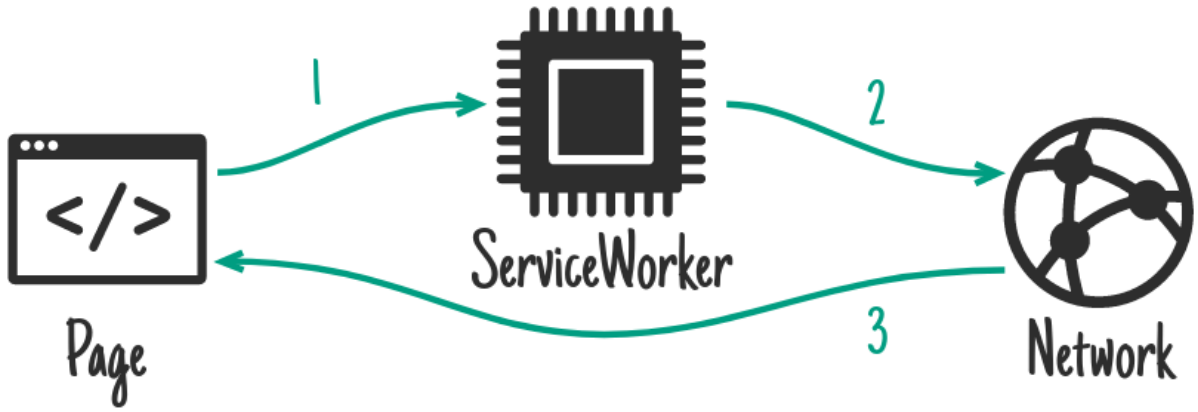

Figura 4: Network only

4.4 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));
});
```

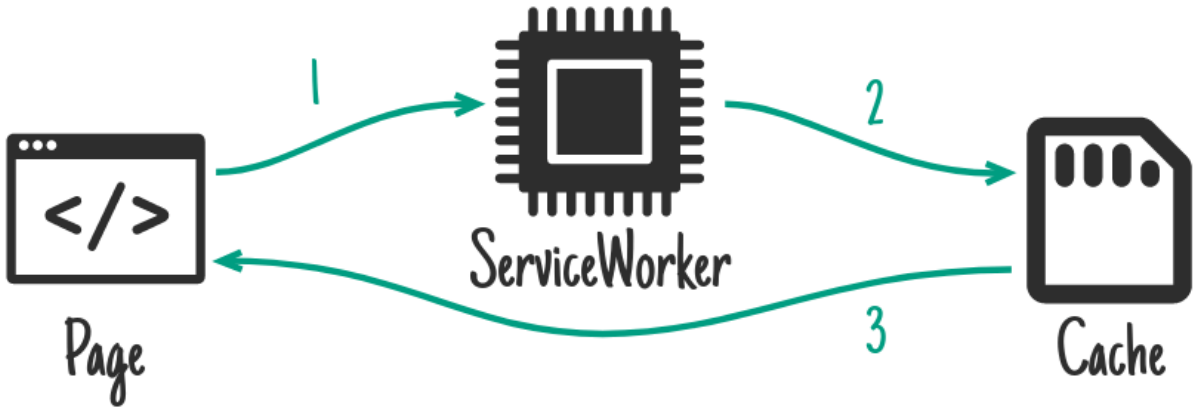

Figura 5: Cache Only

4.5 Fastest

Questa strategia mira a fornire all'utente la risposta più veloce. Il service worker 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 service worker 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.

```
1 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")));
       });
2 };
3 self.addEventListener('fetch', function(event) { event.respondWith( promiseAny([caches.
      match(event.request), fetch(event.request)]) );
4 });
```

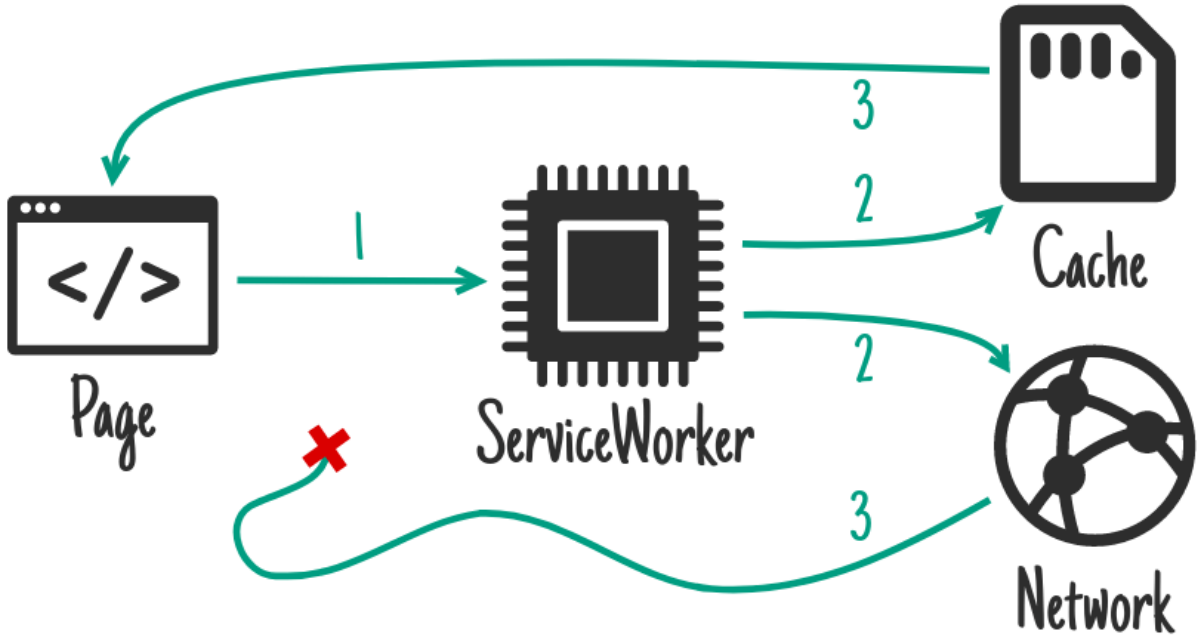

Figura 6: Fastest

4.6 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 service worker. Questo perché il service worker 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 service worker 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.

Di seguito il codice da inserire nella pagina:

```
1  var networkDataReceived = false;
2  var risorsa = '/data.json';
3  startSpinner();
4  // Richiesta di rete (intercettata dal service worker)
5  var networkUpdate = fetch(risorsa).then(function(response) { return response.json();
6  }).then(function(data) { networkDataReceived = true; updatePage();
7  });
8  // Richiesta dalla cache
9  caches.match(risorsa).then(function(response) { if (!response) throw Error("No data");
       return response.json();
10 }).then(function(data) { // Aggiorno la pagina se il contenuto dalla rete non e' stato
       ancora ricevuto if (!networkDataReceived) { updatePage(data); }
11 }).catch(function() { // Contenuto non presente in cache, attendo quello dalla rete
       return networkUpdate;
12 }).catch(showErrorMessage).then(stopSpinner);
```

Di seguito quello del service worker:

```
1  self.addEventListener('fetch', function(event) { event.respondWith( fetch(event.request).
       then(function(response) { cache.put(event.request, response.clone()); return response
       ; }) );
2  });
```

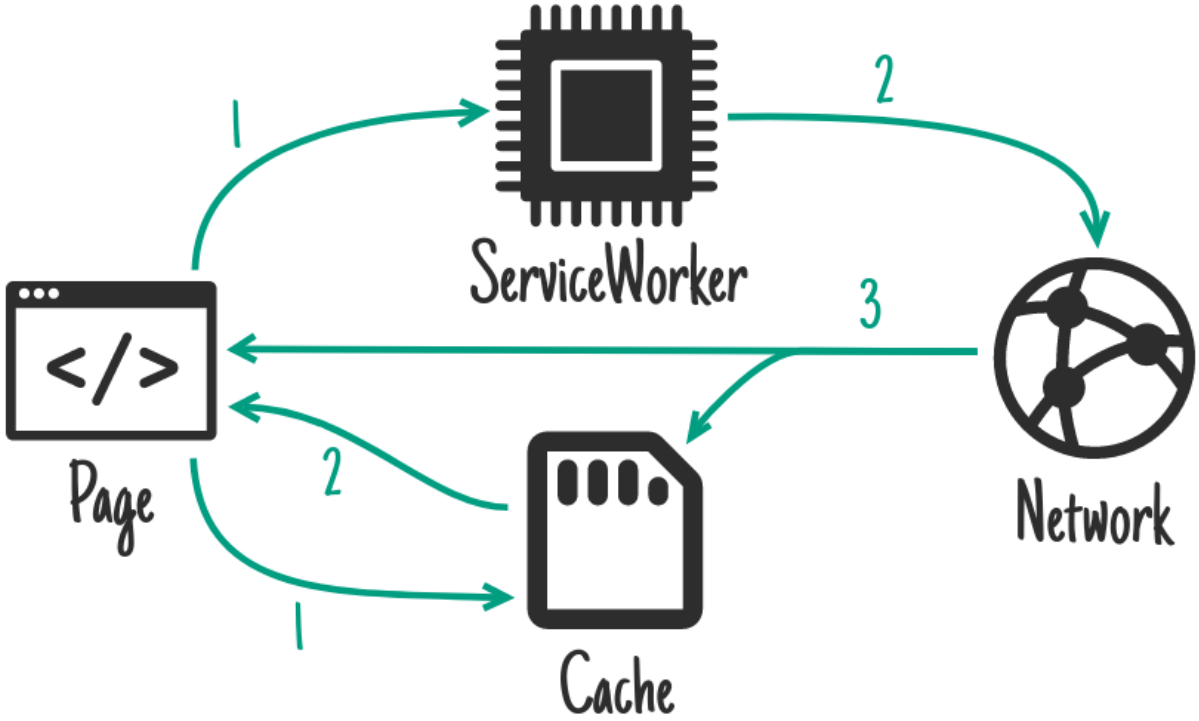

Figura 7: Cache then Network

5 Compatibilità web

5.1 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

5.2 Mobile

6 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.

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