UNIVERSITÀ CA' FOSCARI DI VENEZIA

CORSO DI LAUREA TRIENNALE IN INFORMATICA

TESI DI LAUREA

REALIZZARE UNA PROGRESSIVE WEB APP A PARTIRE DA UNA SINGLE-PAGE WEB APP SVILUPPATA IN ANGULAR 8

Relatore:
Prof. Albarelli Andrea
Laureando:
Davide Lovat
Matricola: 838654

ANNO ACCADEMICO 2019/2020

INDICE

1 INTRODUZIONE

Le applicazioni web (web app) costituiscono un'alternativa alle applicazioni tradizionali, specialmente per i dispositivi mobili. Nel mondo della programmazione web molti framework permettono di fornire strumenti di sviluppo delle applicazioni web rapidi e facili da apprendere. In particolare, il framework open source Angular semplifica lo sviluppo di applicazioni web a singola pagina, garantendone la compatibilità con qualsiasi piattaforma.

Con questo elaborato ci si prefigge come obiettivo di indagare la possibilità di realizzare una Progressive Web-App (PWA) a partire da una Single-Page Web-App (SPWA) realizzata con il framework Angular 8.
Impiegheremo un service worker personalizzato integrandolo con Angular 8 per rendere un'applicazione web disponibile offline. Vedremo come è stato realizzato il servizio offline e come abbiamo impostato la logica di gestione delle risorse in grado di servire le esigenze di chi usa l'applicazione, determinando quali risorse devono essere messe in memoria. Inoltre vedremo come è stata realizzata la gestione dello spazio di memoria allocato alla nostra web app.

2 DALLE WEB APP ALLE PROGRESSIVE WEB APP

Negli anni dal 2007 a oggi gli sviluppatori web hanno cambiato il loro modo di operare, nuovi termini sono entrati a far parte del vocabolario IT comune. Nello specifico ci riferiamo all'introduzione ed evoluzione nel panorama moderno delle web app , termine coniato per riferirsi ad applicazioni web-based, cioè accessibili e fruibili via web per mezzo di un network. Oggi le web app hanno trovato un largo impiego specialmente all'interno delle business software.
Secondo uno studio del 2018 in cui si è effettuato un confronto tra il numero di utenti che impiegano una web app rispetto a quelli che usano la corrispettiva applicazione nativa (native app), è stato notato che: "mobile web reach is way higher than native app reach, It was 11.4 million unique visitors per month compared to 4 million visitors".
[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9441)
Oggi quindi si può notare un cambiamento di paradigma che va dalle app native alle applicazioni fruibili attraverso il web. Questo cambiamento lo si deve sicuramente al consolidamento delle web app ma soprattutto all'introduzione nel 2014 di un nuovo modello di applicazioni web, cioè quello che oggi viene chiamato "Progressive Web App".
Negli anni successivi all'introduzione in commercio dei primi smartphone ci fu un tentativo di spingere le applicazioni web-based in primo piano, cercando di competere con le apps native, specialmente con la crescita rilevante dei dispositivi mobili che tra il 2007 e il 2013 si era stimato aver raggiunto 1 miliardo di utenti per dispositivi smartphone con una previsione futura della crescita del 47% annua.**[2](Native Apps vs. Mobile Web Apps, iJIM – Volume 7, Issue 4, October 2013, pg.27)**
Oggi le statistiche mostrano che nel 2019 il numero di dispositivi mobili venduti globalmente ha superato il miliardo e mezzo **[58]** e il numero di utenti nel 2019 supera i tre miliardi e si stima una crescita di diverse centinaia di milioni nei prossimi anni **[59]**.
Tuttavia il tentativo di competere con le applicazioni native si dimostrò fallimentare se confrontato con il

successo che riscontrarono quest'ultime.**[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 789)**

Infatti le app native non solo garantiscono il diretto accesso all'hardware ma in generale forniscono una migliore esperienza utente e un avvio diretto che non richiede ad un'applicazione di essere caricata su un browser a runtime.**[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 789)**

In sei anni le app native si erano già da tempo consolidate all'interno dell'ambiente mobile e oggi l'uso delle app native con i dispositivi mobili è ormai da considerarsi universale.

[2](Native Apps vs. Mobile Web Apps, iJIM – Volume 7, Issue 4, October 2013, pg.27)

Nell'Ottobre del 2013 secondo uno studio condotto all'Università di Stoccolma, dove si è indagato sulla fattibilità di sostituire delle app native con le applicazioni web, si è arrivato a sostenere che ancora una volta le app native sono la migliore scelta per quelle applicazioni che necessitano di un uso intenso dell'hardware del dispositivo, delineando quindi i confini delle web app e concludendo che: "mobile applications that only require a native interface and content consumption are suitable substitutes for native applications". **[2](Native Apps vs. Mobile Web Apps, iJIM – Volume 7, Issue 4, October 2013, pg.27)**

Infatti durante lo studio è stato fatto notare come le apps web per mobile, che fanno uso di componenti hardware, sono meno performanti di una applicazione nativa autorizzata ad accedere all'hardware del dispositivo.

In questi ultimi anni, con lo sviluppo di API Web che permettono l'accesso diretto all'hardware anche per le applicazioni web, lo spartiacque tra web app e apps native è stato ridotto, permettendo di rendere le funzionalità di una web app più vicine a quelle di una app nativa e aumentare la competitività delle applicazioni web nei confronti delle applicazioni native. Stiamo assistendo nuovamente ad un cambio di paradigma che va dalle app native ai browser.

Questo passaggio, lo si deve in larga misura alla rivoluzione delle progressive web apps che ha coinvolto le applicazioni web. **[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 789)**

3 SINGLE PAGE WEB APPLICATION E METODOLOGIA AJAX

Le prime applicazioni web classiche impiegavano delle tecnologie web che consentivano un'implementazione multipagine (multi-page application MPA). Il modello classico di una web app richiedeva infatti che, attraverso l'interfaccia dell'applicazione web, l'utente facesse una richiesta HTTP verso un web server. Il server si occupava di processare la richiesta, eseguendo una serie di operazioni necessarie a soddisfare la richiesta, per poi tornare al client una pagina HTML di risposta. J.J. Garrett coniatore del termine Ajax e fondatore di Adaptive Path, afferma in una sua pubblicazione che sebbene il modello appena presentato funzioni per il web ipertestuale non è necessariamente il migliore per fornire applicazioni software attraverso il web.**[4](Ajax: A new approach to web applications, pg. 2)** Infatti sotto l'aspetto dell'esperienza utente un'applicazione web che usa l'approccio appena descritto provocherebbe, per ogni interazione utente che abbia bisogno di interpellare il server, il blocco dell'applicazione e conseguentemente l'impossibilità ingiustificata da parte dell'utente di poter utilizzare altre funzionalità dell'applicazione durante il tempo di risposta del server.

La tecnologia Ajax conferisce all'applicazione un metodo per svincolarsi dalle richieste dirette al server.
Ajax ha permesso un nuovo approccio nella realizzazione delle applicazioni web che consente di introdurre le applicazioni a singola pagina (single-page application SPA), che costituiranno un'alternativa alle applicazioni multipagine fino ad allora impiegate per le applicazioni web classiche.
Grazie al modello AJAX e alla combinazione delle tecnologie esistenti (HTML, javascript, XML, ...), le applicazioni web possono eseguire aggiornamenti rapidi e incrementali dell'interfaccia utente senza ricaricare nel browser l'intera pagina. Questo rende l'applicazione più performante e più reattiva alle azioni dell'utente. Qualsiasi azione utente, che normalmente generava una richiesta HTTP, invece di essere gestita direttamente dall'app, prende la forma di una chiamata javascript ad Ajax che gestisce la richiesta in modo asincrono.
[4](Ajax: A new approach to web applications, pg. 2) [5]
Le SPA cercano di imitare le applicazioni native, ma attraverso il browser, rimuovendo i tempi di attesa nel caricamento delle pagine. Infatti si tratta di un'unica pagina web che carica attraverso uno script Javascript tutti i contenuti.
Questo approccio allo sviluppo di applicazioni web a singola pagina permette di contenere l'informazione in un unica pagina e presentarla all'utente in un unico contenitore. L'implementazione viene semplificata grazie ad una serie di framework messi a disposizione degli sviluppatori web, come ad esempio Angular e AngularJs.
Le SPA durane questi anni sono state adottate largamente per fornire servizi internet. Angular è un framework che può offrire molti benefici se impiegato per lo sviluppo di applicazioni con un'architettura SPA.
Nello specifico per il nostro caso di studio si tratterà di realizzare una progressive web app a partire da una SPA "Twipping App", sviluppata su framework Angular 8.

4 CHE COS'È UNA PWA

Le Progressive Web Apps sono una nuova classe di applicazioni web che permettono di usufruire di vantaggi che fino a prima si potevano trovare solo nelle apps native e le cui features erano semplicemente impensabili per le applicazioni web dell'epoca.
Questo cambiamento è stato reso possibile con la maturazione delle piattaforme web e l'implementazione di nuove API browser per l'accesso all'hardware del dispositivo, e in ultima dall'introduzione nel 2014 dei Service Worker da parte del Chromium browser che hanno sbloccato questa nuova classe di applicazioni web.
[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9441)
Il termine PWA fu coniato per fare riferimento a delle web app capaci di adattarsi ad un ambiente mobile.
Essenzialmente le PWA sono da considerare come regolari applicazioni web per desktop e mobile accessibili in qualsiasi browser in grado di supportare i nuovi standard web. Ciò che però le rende differenti dalle semplici web app è l'uso dei service worker che gli permettono di utilizzare molte più funzionalità di ciò che è tradizionalmente disponibile attraverso un browsers.
[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9441)
Il confine che permette di stabilire se una web app è una PWA non è delineato con estrema rigidità, esistono diverse definizioni che convengono sulle caratteristiche di base che permettono di identificare una PWA, ma bisogna considerare che anche se una o più di una di queste caratteristiche non sussistono all'interno di una progressive web app questa non va considerata di meno di una PWA che le possiede tutte.

In linea generale le caratteristiche che identificano una PWA si possono riassumere come segue: Una web app deve fornire l'esperienza via web direttamente, quasi come quella di un'applicazione nativa eseguita su desktop o da mobile. Questo significa che deve essere **veloce, affidabile e installabile**.
Veloce significa che il tempo per rendere disponibile il contenuto e fornire un'esperienza interattiva deve essere relativamente breve
Affidabile significa che una PWA deve mantenere una performance affidabile, rendendo i costi di avvio della web app pari a quelli che un utente si aspetta da un'applicazione installata su dispositivo, il che significa annullare i tempi di ritardo nell'avvio dell'app.
Installabile, le PWAs oltre a poter essere eseguibili da un browser, devono apparire e comportarsi sul dispositivo dell'utente come tutte le altre apps installate su un dispositivo, cioè deve essere possibile lanciare la web apps dalla schermata del dispositivo tramite un icona come avviene con le app native, consentendo di poter essere eseguita in una sua finestra e non in quella di un browser e deve figurare come tutte le altre applicazioni installate nel task switcher.
Un utente che installa una PWA si deve aspettare che funzioni come tutte le altre apps indipendentemente dal tipo di connessione di rete. **[6]**

A livello più tecnico tutte le PWA si riconoscono da un elemento specifico, il service worker, che è sempre presente e svolge l'arduo compito di implementare le funzionalità caratteristiche definite da una progressive web app, che riportiamo parzialmente qui sotto, con una breve descrizione:

Funzione	**Descrizione**
Servizio Offline	Capacità di lavorare anche se il dispositivo è offline.
Notifiche Push	Capacità di visualizzare notifiche quando la web app non è aperta su un browser.
Background Sync	Sincronizzazione in background dei dati.
Add to Home Screen	Installare l'applicazione web su un dispositivo, permettendo l'avvio dell'app web tramite il semplice click o touch su un'icona.
Precaching	Capacità di copiare gli assets o risorse nella memoria dedicata alla PWA, al primo avvio del service worker.
Storage Estimation	Capacità di stimare la quantità di memoria disponibile per la PWA.
Persistent Storage	Capacità di garantire dati permanenti nella memoria, rimuovibili solo manualmente.

Tabella 1 *Caratteristiche PWA*
[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 791,792)

Nel caso di studio da noi condotto realizzeremo una PWA come un'applicazione web che fa uso di un service worker limitato al servizio offline e alla gestione della memoria.

5 PERCHÈ PASSARE ALLE PWA

Sia le applicazioni native che le applicazioni web presentano delle limitazioni.
[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9439)
[2](Native Apps vs. Mobile Web Apps, iJIM – Volume 7, Issue 4, October 2013, pg.27)
Riportiamo in tabella 2 le caratteristiche distintive delle app native e delle web app, confrontando i diversi vantaggi e svantaggi dei due tipi di implementazione.

	Description	Native app	Mobile web app
No Installation	L'app non richiede di essere installata.	NO	SÌ
Fast Updates	Update istantaneo.	NO	SÌ
Size	Non occupa molto spazio in memoria.	NO	SÌ
Hardware Acess	Ha accesso all'hardware del dispositivo.	SÌ	NO
Development	Non richiede uno specifico sviluppo per ogni sistema operativo.	NO	SÌ
OS Functionality	Tutte le funzionalità del sistema operativa sono disponibili.	SÌ	NO
Content Creation	Sono adatte alla creazione di contenuti.	SÌ	NO
Content Consumption	Sono adatte per consumare contenuti.	SÌ	SÌ
Offline	Permette di essere utilizzata anche in remoto.	SÌ (esclusi casi specifici)	NO

Tabella 2 *Native app e Mobile web app vantaggi e svantaggi messi a confronto*

[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9441)
[2](Native Apps vs. Mobile Web Apps, iJIM – Volume 7, Issue 4, October 2013, pg.28)
Le PWA combinano il meglio delle web app con le caratteristiche delle app native, da questa combinazione possiamo superare molte delle limitazioni indicate nella tabella 2.
[1](Impact of Progressive Web Apps on Web App Development, International Journal of Innovative Research in Science, Engineering and Technology, vol.7, issue 9, September 2018 pg.9439)
Unendo gli aspetti positivi delle applicazioni native con quelli delle applicazioni web otteniamo che le PWA

presentano i vantaggi esposti in tabella 3.

PWA Benefits	Description
No Installation	L'app non richiede di essere installata.
Fast Updates	Update istantaneo.
Size	Non occupa molto spazio in memoria.
Hardware Acess	Ha accesso all'hardware del dispositivo.
Development	Non richiede uno specifico sviluppo per ogni sistema operativo.
OS Functionality	Tutte le funzionalità del sistema operativa sono disponibili.
Content Creation	Sono adatte alla creazione di contenuti.
Content Consumption	Sono adatte per consumare contenuti.
Offline	Permette di essere utilizzata anche in remoto.

Tabella 3 *PWA Benefici*

6 SERVICE WORKER

Nel concreto le PWAs si appoggiano su un componente fondamentale, il service worker (SW). Quest'ultimo permette di realizzare la maggior parte delle features di una PWA. Le più comuni implementazioni di un service worker garantiscono alle apps di lavorare offline, di ricevere notifiche push e sincronizzare i dati in background anche quando l'app non è in esecuzione, inoltre permettono all'utente di installare le PWA sull' home screen del proprio dispositivo.

[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 791,792)

I service workers vengono introdotti nel 2014 con il progetto Chromium, un browser open-source che mira a costruire un esperienza web per l'utente più sicura, stabile e veloce. Con il rilascio di Chrome 40 Beta viene rilasciata anche la nuova API service worker per garantire ai siti di funzionare anche offline e velocizzare i tempi di caricamento delle pagine. In concomitanza con l'API ServiceWorker, altre due API altrettanto importanti e che tratteremo più dettagliatamente in seguito, sono state sviluppate per essere usate unicamente all'interno

dei service worker, che sono l'API Fetch e l'API Cache. L'una usata per gestire le richieste al network, mentre l'altra permette di gestire lo spazio di memoria di cache dedicato al sito. **[7]**

[3] (What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser, WWW 2018, April 23-27, 2018, Lyon, France, pg. 791,792)

6.1 COSA SONO I SERVICE WORKER

Il service worker è a tutti gli effetti un web worker(WW) guidato da eventi, cioè uno script Javascript(JS) eseguito in background, separatamente dal thread principale del browser e indipendentemente da qualsiasi altro script dell'applicazione a cui sono associati, che risponde agli eventi dei documenti HTML o di altre sorgenti.

[8](W3C Candidate Recommendation, Service Worker 1,
https://www.w3.org/TR/service-workers-1/#service-worker-concept)

Nello specifico, date alcune delle caratteristiche che condivide con un particolare tipo di worker, il service worker può essere associato ad uno shared web worker, infatti entrambi:

- vengono eseguiti in un proprio script con contesto globale, tipicamente in un proprio thread.
- non sono legati ad una pagina in particolare
- non possono accedere direttamente al DOM dei documenti

Diversamente da uno shared worker il service worker può inoltre:

- essere eseguito senza che vi sia alcuna pagina web aperta
- è event-driven, cioè termina quando non è in uso e si riavvia quando viene richiesto
- ha un modello di update definito
- funziona solo con il protocollo HTTPS

[8](W3C Candidate Recommendation, Service Worker 1,
https://www.w3.org/TR/service-workers-1/#motivations)
[9] (Github.com, The World Wide Web Consortium (W3C),
https://github.com/w3c/ServiceWorker/blob/master/explainer.md#whats-all-this-then)
[10]
[11] (Web Hypertext Application Technology Working Group. HTML Living Standard. Web Workers,
https://html.spec.whatwg.org/multipage/workers.html#scope-2)

Essendo il SW un WW, sarà in grado di essere in esecuzione per lunghi periodi di tempo, con un lungo tempo di vita. Generalmente avrà un alto costo di start-up in fatto di performance e un alto costo di memoria per istanza. Siccome i web workers sono componenti relativamente heavy-weight è sconsigliato utilizzarli in largo numero, tipicamente una progressive web app si limita all'uso di un singolo service worker.

[11] (Web Hypertext Application Technology Working Group. HTML Living Standard. Web Workers,
https://html.spec.whatwg.org/multipage/workers.html#scope-2)

Un service worker una volta attivato dal browser sarà in grado di lavorare in autonomia anche in assenza di pagine web o di una finestra del browser per un periodo di vita indefinito, in quanto deve essere rimosso manualmente dall'utente o dalla stesso sito web, poiché il browser non si occupa di ripulire la memoria dai service worker installati dai siti web.

Qui sotto riportiamo due esempi generici di come un sito web funziona senza e con un service worker (Figura 1, Figura 2). Possiamo notare dalla Figura 2 come il service worker funga da intermediario tra il nostro sito web o applicazione web, che impiega il service worker e il network. Ogni richiesta che parte da una pagina sotto il controllo del service worker o risposta ritornata dal web server, viene intercettata dal service worker e gestita internamente (Figura 2).

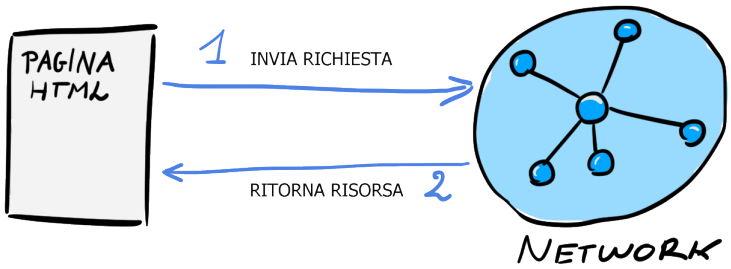

Figura 1 *Comunicazione tra la pagina web e il network in un sito che non impiega un service worker*

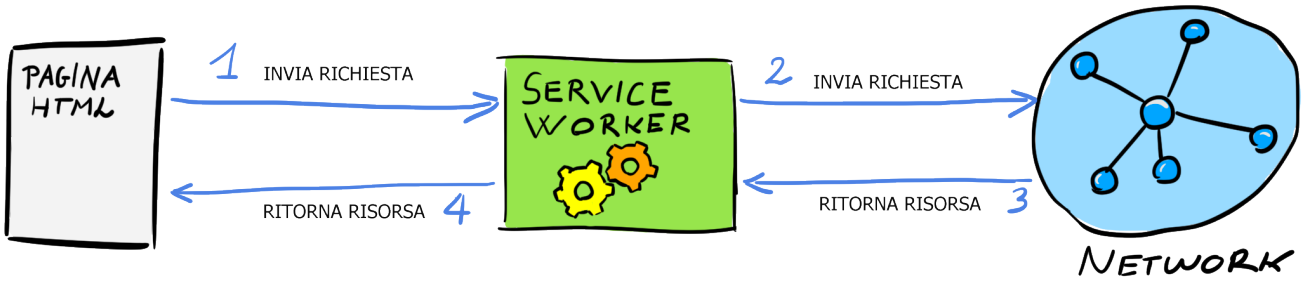

Figura 2 *Comunicazione tra pagina web e network che impiega un service worker*

6.2 SUPPORTO DEI BROWSER AI SERVICE WORKER

Essendo le PWA il futuro delle applicazioni web, la grande maggioranza dei browser hanno iniziato a fornire un supporto per le API dei service worker. Attualmente quasi tutte le nuove versioni dei browser più diffusi supportano i service worker. Le tabelle 5 e 6 mostrano la compatibilità di alcuni browser sviluppati per dispositivi desktop e mobile **[12]**.

DESKTOP BROWSER	SERVICE WORKER SUPPORT	LATEST VERSION	RELASE DATE
Edge	Full support	18	2018
Chrome	Full support	78	2019
Firefox	Full support	71	2019
Safari	Full support	13	2019
Opera	Full support	64	2019
Internet Explorer	No support	/	2013

Tabella 5

Come si può notare l'unico browser desktop che non supporta le API service worker è Internet Explorer,

mentre tutte le versioni più recenti dei browser Edge, Safari, Chrome, Firefox e Opera supportano questa tecnologia **[12]**.

MOBILE BROWSER	SERVICE WORKER SUPPORT	LATEST VERSION	RELASE DATE
iOS Safari	Full support	13.2	2019
Chrome per Android	Full support	78	2019
Firefox per Android	Full support	68	2019
Samsung Internet	Full support	10.1	2019
UC Browser per Anrdoid	Full support	12.12	2016
Android Browser	Partial support	76	2019
Opera Mobile	Partial support	46	2016
Opera Mini	No support	/	2015

Tabella 6

Dal lato dei browser per dispositivi mobili invece riscontriamo che alcuni non forniscono un supporto completo dei service worker. Come prima abbiamo che i browser iOS Safari, Chrome e Firefox per Android con l'aggiunta di Samsung Internet aggiornati alla versione del 2019 e UC Browser per Anrdoid con ultima data di rilascio risalente al 2016, offrono un completo supporto ai service worker. Invece offrono un supporto parziale Android Browser del 2019 e Opera Mobile 2016. Non offrono invece alcun supporto le versioni di Opera Mini con l'ultima versione aggiornata al 2015 **[12]**.

6.3 RESTIZIONI NEL SERVICE WORKER

Abbiamo visto che il service worker è uno script che viene eseguito in modo asincrono su un altro thread rispetto al thread principale del browser. Questo comporta anche delle limitazioni nell'uso del service worker.

Origine del service worker

Quando un service worker viene chiamato da una pagina per essere registrato, per poter essere installato, il service worker deve essere sulla stessa origine della pagina che effettua la chiamata.
[9] (Github.com, The World Wide Web Consortium (W3C), https://github.com/w3c/ServiceWorker/blob/master/explainer.md#whats-all-this-then)

Scope del service worker

Lo scope determina il percorso da cui il service worker può ricevere le richieste dalle pagine del sito, cioè limita il controllo del service worker alle sole pagine che cadono sotto il percorso specificato dallo scope.

[16] (Web Fundamentals, Introduction to Service Worker, https://developers.google.com/web/ilt/pwa/introduction-to-service-worker#registration_and_scope)

Stato globale di un servie worker

Un service worker viene terminato quando non è più in uso e riavviato quando è richiesta una sua azione. Lo

stato di un service worker nel ciclo di terminazione e riavvio non è persistente, tutti i dati impiegati all'interno dei gestori di eventi fetch e message non sono recuperabili. Si deve assumere che per ogni invocazione ad un gestore di eventi interno al service worker, questo viene invocato con uno stato globale di default. In conclusione non si può utilizzare lo stato globale interno ad un service worker per memorizzare le informazioni persistenti da poter essere riutilizzate al riavvio del service worker. Invece esistono altri meccanismi come la memorizzazione di dati in Cache o IndexDB.

[14] Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#what_is_a_service_worker

L'interfaccia "ServiceWorkerGlobalScope" della Service Worker API rappresenta il contesto di esecuzione globale di un service worker **[13]**.

Comportamento asincrono

Essendo il SW non bloccante, cioè realizzato in modo completamente asincrono, non sarà possibile utilizzare le API sincrone come XMLHttpRequest (XHR API) e localStorage all'interno del service worker.

La **XHR API** permette di recuperare file XML o altri tipi di dati da e verso un web server tramite una richiesta HTTP effettuata in modo sincrono e viene usata pesantemente nella programmazione AJAX per consentire ad una parte di una pagina web di recuperare dati da un URL senza dover fare un refresh dell'intera pagina. Nel caso dei service worker non sarà disponibile per fare richieste al web server.

La **Web Storage API** fornisce dei meccanismi di memorizzazione per il browser, in particolare il meccanismo **localStorage** che permette di mantenere un'area di memoria separata per ogni origine, persistente anche dopo che il browser viene chiuso. Nel caso del service worker non sarà possibile impiegare l'API localStorage per accedere al contenuto della memoria di un particolare dominio. Per questo è stata definita l'API Cache, per essere usata come meccanismo di memorizzazione permanente nei service worker. **[15, 17]**

[16] (Web Fundamentals, Introduction to Service Worker,
https://developers.google.com/web/ilt/pwa/introduction-to-service-worker#what_is_a_service_worker)

HTTPS only

I service worker possono essere eseguiti solo se si impiega il protocollo HTTPS per le comunicazioni sicure tra computer connessi in rete. Durante lo sviluppo di un service worker è consentito l'uso di un service worker su localhost, ma nella fase di attivazione del service worker in produzione sarà richiesto che il server comunichi attraverso il protocollo HTTPS. Nell'esempio da noi implementato per realizzare una progressive web app con i service worker, il web server è regolarmente configurato per consentire una comunicazione sicura.

Le ragioni che ci spingono ad utilizzare una connessione criptata, risiede nella natura del service worker che permette di intercettare, modificare risposte e fabbricarne di nostre.

Servire le pagine tramite HTTPS ci permette di proteggere la comunicazione dagli attacchi di un possibile "man in the middle", evitando quindi che il browser riceva, invece del nostro service worker, un service worker alterato da un attacco informatico durante il suo viaggio attraverso il network.

[14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#you_need_https)
[16] (Web Fundamentals, Introduction to Service Worker,
https://developers.google.com/web/ilt/pwa/introduction-to-service-worker#what_is_a_service_worker)

Come fatto notare in **[18]** le PWA non sono comunque immuni da possibili attacchi di phishing, infatti sebbene l'uso di HTTPS nella comunicazione possa prevenire questo tipo di attacco informatico ci sono altre

caratteristiche del service worker che possono mettere a rischio la sicurezza e privacy di chi ne fa uso.
[18] (CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications, pg.1731)

Nessun accesso al DOM

Siccome un service worker viene eseguito nel contesto di un web worker, come da specifiche ci sono alcune eccezioni al codice che il worker thread o javascript worker può eseguire al suo interno. Un service worker è in grado di controllare molte pagine, ma esattamente come tutti i web worker non sarà in grado di manipolare direttamente il DOM di una pagina. Sarà invece possibile comunicare con le pagine attive sotto il suo scope,impiegando l'interfaccia "postMessage" e l'event listener "message", che gli permettono rispettivamente di inviare dati alle pagine attraverso un messaggio e ricevere dati dalle pagine. Attraverso questo meccanismo di messaggistica sarà possibile manipolare indirettamente il DOM delle pagine con il service worker.
[10, 20]
[14] (Web Fundamentals, Service Worekers: an Introducution, https://developers.google.com/web/fundamentals/primers/service-workers#what_is_a_service_worker)
[16] (Web Fundamentals, Introduction to Service Worker, https://developers.google.com/web/ilt/pwa/introduction-to-service-worker#what_is_a_service_worker)
[9] (Github.com, The World Wide Web Consortium (W3C), https://github.com/w3c/ServiceWorker/blob/master/explainer.md#whats-all-this-then)
[21]

6.4 SICUREZZA NELLE PWA

Abbiamo visto come il service worker sia implementato per essere impiegato unicamente per una comunicazione HTTPS, ma questo può bastare a proteggere la nostra PWA da attacchi di malintenzionati?**[19]**
Si parlava in precedenza di una possibile minaccia nell'uso dei service worker, quindi prima di addentrarci nello sviluppo di una PWA, soprattutto in un ambiente business software, è importante conoscere tutti i possibili rischi che possono sorgere. Abbiamo visto che un service worker è relativamente sicuro, gli stessi autori del progetto Chromium, sostengono che i service worker sono sicuri, ma come precedentemente accennato, uno studio condotto dalla School of Computing del KAIST ha dimostrando che ci possono essere altri sistemi che rendono una PWA insicura per chi ha installato il service worker di una PWA nel proprio dispositivo. Secondo quanto riportano da questo studio non ci sono abbastanza ricerche sulla sicurezza e i rischi di privacy unici legati alle PWA.
[18] (CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications, pg. 1732)
Ciò che è di alto interesse per implementazioni future nella nostra PWA è il rischio di phishing a cui può essere sottoposta la PWA se si utilizzano delle notifiche push nel nostro sito.
Sebbene non siano state implementate nella nostra versione del service worker, in futuro potremmo sicuramente utilizzarle per mandare notifiche agli utenti, questo potrebbe costituire un problema per chi usa la nostra PWA se qualcuno decidesse di avvantaggiarsi di questa feature implementata dal service worker per

raggirare l'utente e indurlo in errore. Infatti come riportato da **[18]** la modalità con cui un utente riconosce una notifica push da un sito è prevalentemente dettata dal logo. Un utente potrebbe essere vittima di un attacco phishing da parte di un sito all'apparenza innocuo che è stato autorizzato dall'utente stesso a invia notifiche push, ma che però nasconde un intento maligno, in quanto nell'inviare le notifiche push potrebbe imitare il brand o logo di un'altra compagnia, che può indurre l'utente a confondere il mittente del messaggio.
[18] (CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications, pg. 1732, 1735 - 1737)
Secondo quanto riportato da questa ricerca , l'unico punto di riferimento su cui un utente può fare affidamento per riconoscere l'origine del mittente della notifica è il nome del dominio mostrato nella notifica, ma è stato notato che alcuni browser popolari come Samsung Internet, Firefox per Linux desktop e Firefox per Anrdoid, quando il pannello delle notifiche è pieno, non mostrano il dominio nel corpo della notifica push.
[22] (CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications, pg. 1732)

7 SERVICE WORKER LIFECYCLE

Al fine di comprendere come un service worker agisce in background, tra il browser e il web server, è importante per lo sviluppatore conoscere le fasi del ciclo di vita di un service worker.

Si identificano sei stati logici:

- Installazione
- Attivazione
- Idle
- Terminato
- Fetch
- Errore

A livello effettivo il service worker per essere attivo e funzionante deve passare attraverso tre passaggi sequenziali chiave: **registrazione --> installazione --> attivazione**

Se in qualsiasi punto durante uno di questi step si dovesse verificare un errore il service worker viene scartato. Il service worker non sarà attivo finché le fasi di registrazione, installazione e attivazione non saranno state completate con successo, cioè senza errori. Se la fase di registrazione, installazione, o attivazione fallisce, il browser tenterà di registrare nuovamente il service worker ad ogni caricamento di pagina. È quindi importante evitare possibili errori di sviluppo, vedremo più avanti man mano che spieghiamo gli altri passaggi, quali sono gli errori che uno sviluppatore di PWA deve evitare.
[22] (Progressive Web Apps, chp. 1, pg.9-10)
Va tenuto presente che se l'abilitazione di un service worker fallisce, per ragioni non legate a errori commessi dallo sviluppatore nell'implementazione, come potrebbe essere ad esempio l'interruzione della connessione nella comunicazione tra il browser e il server, allora in questo caso al prossimo tentativo o tentativi (con il

refresh della pagine o il caricamento di una pagina), in presenza di connettività, il service worker eseguirà la sua installazione correttamente. Se Il service worker deve essere aggiornato, la nuova versione del SW viene scaricata, registrata e installata, ma non andrà in esecuzione finché il vecchio service worker non ha concluso tutte le operazione e sarà portato in uno stato inattivo. Presentiamo di seguito le definizioni generali per ogni stato del ciclo di vita di un service worker.

[14] (Web Fundamentals, Service Worekers: an Introducution, https://developers.google.com/web/fundamentals/primers/service-workers#the_service_worker_life_cycle)

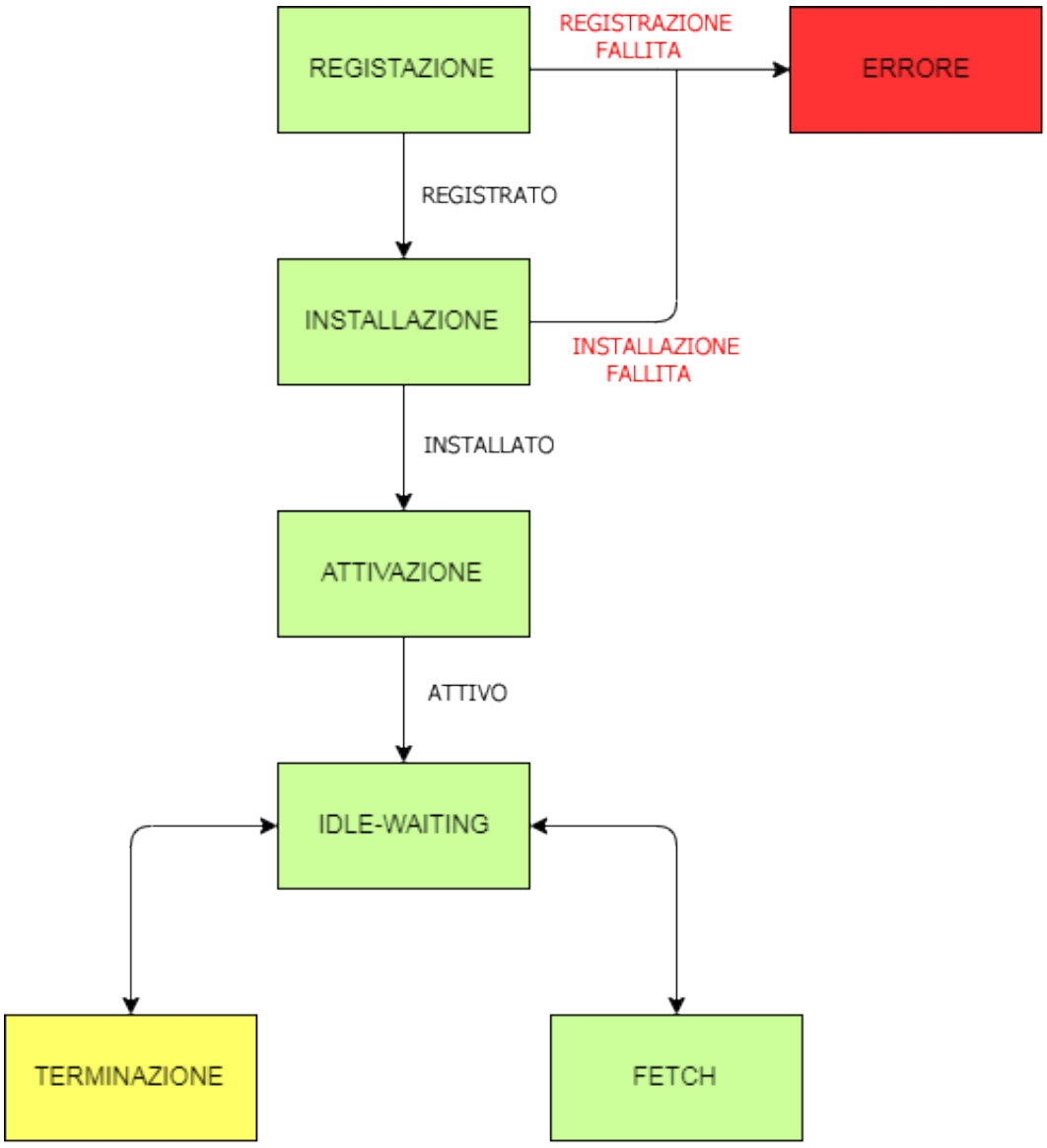

Figura 3 *Ciclo di vita di un service worker*

REGISTRAZIONE

In questa fase il service worker non esiste ancora. Lo script del service worker deve essere scaricato dal server e registrato nell'origine da cui viene recuperato, l'origine è determinante poiché stabilisce che le pagine

controllate dal service worker sono solo quelle con stessa origine e le pagine localizzate in una livello gerarchico inferiore rispetto all'origine del service worker.

INSTALLAZIONE
La fase di installazione può essere impiegata per inserire gli asset statici nella cache.

ATTIVAZIONE
La fase di attivazione è la fase in cui il service worker viene preparato per essere attivato, cioè eseguibile.

IDLE
La fase di attesa, si verifica quando il service worker non sta svolgendo alcuna operazione e non è in esecuzione ma è stato installato e attivato.

FETCH
La fase di fetch si verifica quando il service worker intercetta una chiamata al server dal DOM di una pagina. In questa fase il service worker gestisce la chiamata, la risposta dal server e le risorse nella cache.

TERMINAZIONE
Il service worker può essere terminato dal browser in qualsiasi momento. La terminazione indica semplicemente che l'esecuzione del service worker viene interrotta. Vedremo alcuni metodi per evitare che il browser sopprima l'esecuzione del service worker, invalidando le operazioni ancora in fase di conclusione.

ERRORE
Il service worker va nello stato ERROR. In questo caso si è verificato un errore che non ha permesso l'installazione del service worker. Un tipico errore nella fase di installazione può avvenire quando si verifica un problema di connessione o si cerca di mettere in cache una risorsa che non è raggiungibile, perché non esiste o la sua posizione è stata modificata. Un errore invalida l'installazione del service worker.

7.1 REGISTRAZIONE DI UN SERVICE WORKER

Iniziamo subito con il dire che il service worker di un particolare sito web, per essere installato deve prima essere registrato dal browser.
Per assicurarsi che un service worker venga registrato da un browser lo sviluppatore web deve inserire il codice javascript di registrazione all'interno del corpo o header della pagina web da cui si desidera far partire la registrazione.
Automaticamente al caricamento della pagina verrà eseguito il codice javascript di registrazione del service worker. Molto banalmente lo script effettua una chiamata alla funzione "register" che prende come argomento la posizione del file javascript del service worker che si vuole registrare. La chiamata alla funzione fa sì che il browser avvii il processo di registrazione, che consiste nel download, parsing ed esecuzione del file del service worker.

Se il percorso dello script del service worker è valido, la registrazione ha successo e non appena il service worker viene eseguito, viene attivato in background l'evento di installazione del service worker. Nel caso venga specificato un percorso sbagliato si entrerà immediatamente in uno stato di errore, che provocherà conseguentemente il fallimento della registrazione.
La chiamata alla funzione "register" nello script di registrazione può essere effettuata in qualsiasi momento e ogni volta che viene caricata la pagina senza che il programmatore si debba preoccupare se il service worker sia già stato registrato o meno, infatti sarà il browser a gestire automaticamente questa situazione determinando se il service worker è già stato registrato o meno, confrontando byte per byte il service worker già installato con quello recuperato dal server. Questa capacità del browser di distinguere se un service worker è stato registrato è molto importante quando un service worker deve essere aggiornato ad una nuova versione.
[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#the_service_worker_life_cycle)
[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#install_a_service_worker)
[22] (Progressive Web Apps, chp. 1, pg.9-10)

Scope di registrazione

Lo scope è il dominio entro il quale il service worker entra in funzione ed è dipendente dalla posizione del file del service worker che si deve registrare. Un service worker entra in funzione solo per quegli eventi ricevuti da pagine che si trovano allo stesso livello o al di sotto del livello di gerarchia specificato nello scope del service worker.
Per tutte le altre pagine del sito che non rientrano nello scope del service worker non sarà possibile usufruire dei vantaggi tipici di una PWA. Quindi lo scope dal service worker è influenzato dalla struttura gerarchica del nostro sito, non di meno il modo per registrare un service worker su una web app cambia a seconda che la web app sia una SPWA o una MPWA.
In una SPWA risulta particolarmente semplice registrare un service worker, in quanto si impiega una singola pagina per caricare dinamicamente i contenuti del sito. Quindi risulta possibile registrare il service worker unicamente sulla pagina principale o root del dominio, per includere anche tutte le "altre pagine". Vedremo nel nostro caso di studio come è stato possibile registrare un service worker su una SPWA in Angular.
[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#the_service_worker_life_cycle)
[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#register_a_service_worker)
Invece risulta particolarmente più oneroso registrare un service worker su una MPWA o un sito web multipagina, in quanto il codice di registrazione del service worker deve essere inserito in ogni pagina web se ci si vuole assicurare che l'utente entrando a visitare una pagina qualsiasi del nostro sito possa usufruire sempre dei vantaggi offerti da una PWA e non solo quando visita il sito passando attraverso la pagina principale o home page.
La registrazione è solo il primo passo verso l'abilitazione di un service worker, in quanto abbiamo ancora due

passi da seguire prima che il service worker entri realmente in funzione, che sono l'installazione e l'attivazione.

7.2 INSTALLAZIONE DI UN SERVICE WORKER

I service worker sono web worker guidati dagli eventi che vengono registrati con un origine e un percorso.
Questo ci permette di implementare all'interno dello stesso, l'insieme delle azioni che vogliamo che il nostro service worker esegua non appena viene intercettato un evento che l'API ServiceWorker è autorizzata a gestire.
Tra i vari eventi che possono essere gestiti, abbiamo l'evento 'install'.
L'evento 'install' è il primo evento che un service worker riceve e si verifica una sola volta. Ci permette di specificare l'insieme di istruzioni da eseguire durante la fase di installazione di un service worker. Direttamente all'interno del service worker, definiamo l'insieme delle istruzioni che l'event handler dovrà richiamare al verificarsi dell'evento 'install'.

[23] (Web Fundamentals, The Service Worker Lifecycle,
https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle#the_first_service_worker)

Durante la fase di installazione il service worker non potrà di default gestire altri eventi come ad esempio gli eventi 'fetch', 'message' o 'push'. Solo quando ha finito l'installazione ed è nello stato 'active' sarà ricettivo anche agli altri eventi lanciati dalla pagina web.
Questo comportamento di default del service worker causa un ritardo nella disponibilità delle risorse che debbono essere memorizzate in cache , che si protrae fino alla seconda iterazione, infatti solo con un secondo refresh o caricamento della pagina, quando il service worker è ormai installato, possiamo controllare anche le risorse richieste tramite una 'fetch' dalla pagina. È comunque possibile sovrascrivere questo comportamento di default del service worker per intercettare le richieste anche delle pagine che di default non intercetta durante l'installazione.

[23] (Web Fundamentals, The Service Worker Lifecycle,
https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle#the_first_service_worker)

7.3 ATTIVAZIONE DI UN SERVICE WORKER

Se l'ultimo step si è concluso correttamente allora adesso il service worker sarà entrato nello stato 'activated'. In questo stato il service worker di default è in grado di controllare risorse in memoria. Tipicamente in questa fase si gestiscono le vecchie cache appartenenti a service worker segnati con versioni più vecchie. La gestione della memoria nella fase di attivazione è utilizzata esclusivamente quando si effettua l'update di un service worker. La ragione per cui non si effettua la pulizia della memoria cache controllata da un service worker con una versione più vecchia, è perché nella fase di installazione del nuovo service worker, il service worker che deve essere sostituito è ancora in grado di servire i file dalla cache che noi intendiamo ripulire. Quindi conviene liberare la cache solo quando siamo nella fase di attivazione, cioè quando il vecchio service worker viene effettivamente rimosso e sostituito.

[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers: an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#the_service_worker_life_cycle)
[14] (Web Fundamentals, Service Worekers: an Introducution, [14] (Web Fundamentals, Service Worekers:

an Introducution,
https://developers.google.com/web/fundamentals/primers/service-workers#update-a-service-worker)

7.4 FETCHING DELLE RICHIESTE IN UN SERVICE WORKER

Abbiamo già parlato di come, prima che fosse introdotta la tecnica AJAX, le applicazioni web classiche ma anche più in generali i siti web, utilizzassero tutti un sistema di caricamento delle pagine o update molto semplice, che consiste nel ricaricare l'intera pagina ogni volta che il client invia una request al server del sito indipendentemente che si voglia aggiornare una sola porzione della pagina o fare il refresh dell'intera pagina.
Quando il client invia una request di uno specifico sito web ad un server, riceve come risposta l'insieme degli assets che costituiscono l'intera pagina web, che vengono scaricati e visualizzata dal client che ricarica l'intera pagina web.
Questo metodo di caricare le pagine risulta un grande spreco di tempo, specialmente per una web app e consegna all'utente un'esperienza relativamente povera.
A questo proposito la tecnica AJAX ha reso possibile agli sviluppatori di applicazioni web di realizzare pagine web in grado di richiedere solo una piccola porzione di dati, evitando di scaricare nuovamente l'intera pagina web. Questo è reso possibile oggi da due API web.
Le due API in questione sono XMLHTTPRequest (XHR) e Fetch API. La prima è una vecchia tecnologia, alla base della tecnica AJAX, che permette al client di trasferire dati in modo sincrono (non supportato dai service worker) o asincrono senza aver bisogno di fare un intero refresh della pagina ma di fare l'update solo di una parte della pagina senza interrompere le azioni dell'utente. La Fetch API invece è recente e sarà fondamentale per poter gestire le richieste della web app dal nostro service worker. Si tratta di una API nata per rimpiazzare l'API XMLHTTPRequest che permette di effettuare richieste HTTP asincrone in JavaScript in modo più semplice anche se non supportata da tutti i browser come la tecnologia XHR. In pratica esattamente come per l'API XHR il concetto di base è sempre lo stesso, fornire un'interfaccia che permetta di accedere e manipolare le richieste fatte dal client e le risposte ricevute dal web server.
Come si vede in figura 2 il service worker fa da intermediario tra il client e il server quando un'evento fetch request viene lanciato dal client verso il server.

Nell'implementazione del nostro service worker utilizzeremo per gestire le request lato client l'API fetch in quanto nelle specifiche dell'API service worker viene implementata la gestione degli eventi di tipo Fetch che consentono al service worker di rispondere ad una request fatta dal client. Questo permette al service worker di modificare le risposte alle richieste fatte dal client da pagine con lo stesso scope del sw. Ribadiamo infatti che il service worker sarà in grado di controllare solo le richieste fatte dalle pagine che figurano sotto lo stesso scope del service worker.
[24, 25]
[8](W3C Candidate Recommendation, Service Worker 1,
https://www.w3.org/TR/service-workers-1/#on-fetch-request-algorithm)

8 CARATTERISTICHE DEL NOSTRO SERVICE WORKER

Abbiamo già presentato un elenco di alcune delle caratteristiche che si possono implementare in una progressive web app, tramite un service worker. In questo paragrafo definiamo le più rilevanti per il nostro caso di studio, presentando due delle implementazioni più comuni in molte PWA. Nello specifico per il nostro caso di studio siamo interessati principalmente a due caratteristiche delle PWA:

1. La capacità di poter usufruire della nostra web app anche offline.
2. La possibilità di gestire le risorse statiche memorizzate nella cache della nostra web app e lo spazio di memoria a nostra disposizione.

Servizio Offline:

Questa feature è la principale ragione per cui si vuole fare un upgrade di una semplice web app in una PWA e consiste nella capacità di garantire in una certa misura il funzionamento di una web app anche se il dispositivo è offline. L'implementazione di questa caratteristica si appoggia ad un'altra caratteristica dei service worker, che è la gestione degli asset statici o risorse statiche nella memoria riservata dal browser alla web app stessa.

Gestione della Memoria:

Una web app che fa uso di un service worker ha riservato uno spazio di memoria che gli viene assegnato dal browser e le cui dimensioni variano in base alle specifiche indicate da quest'ultimo.
Il service worker ha la capacità di memorizzare i dati in modo persistente all'interno di questo spazio di memoria con l'assicurazione che il browser non potrà ripulirne il contenuto senza l'intervento diretto dell'utente, anche nel caso in cui la memoria dedicata sia quasi esaurita.
Quest'ultima caratteristica ci permette di rendere disponibili le risorse alla web app non solo quando è offline, ma anche nel caso in cui il client sia online, permettendoci di velocizzare i tempi di risposta ad una richiesta fatta dal client e caricando le risorse dalla cache invece che attendere il recupero di un risorsa direttamente dal server, che incide sulle prestazioni della nostra web app allungandone i tempi di risposta.
Inoltre per gestire meglio lo spazio di memoria è possibile stimare la quantità di spazio riservata dal browser alla web app e fare una stima della quantità di spazio usato. Conoscere lo spazio a disposizione e se è in esaurimento, ci permette di implementare, se vogliamo, un sistema di gestione delle risorse per determinare quando e quali risorse vanno rimosse per fare spazio a risorse nuove con una priorità ben più alta.

9 ANGULAR 8

Angular è un popolare framework sviluppato da Google per la realizzazione di applicazioni web compatibili con i dispositivi mobili e desktop ed è anche alla base della nostra applicazione web classica.
Angular è incentrato sullo sviluppo di applicazioni web dinamiche principalmente a singola pagina.
L'implementare della nostra PWA, si basa sul framework di Angular 8 attualmente la versione più recente che sia stata rilasciata.

9.1 FILE INTERNI E FILE DI SISTEMA DI UN'APPLICAZIONE ANGULAR 8

Un'applicazione sviluppata con il framework Angular viene sviluppata nel contesto di uno spazio di lavoro di

Angular chiamato workspace.
Un workspace è una collezione di applicazioni e librerie o più semplicemente di progetti, gestibili dalla Command Line Interface di Angular (Angular CLI), l'Angular CLI è uno strumento per gestire il ciclo di sviluppo di Angular, ed è anche impiegata per creare i file bundle iniziali per il nostro spazio di lavoro. Un workspace contiene i file di uno o più progetti. Un progetto è il set di file che costituiscono una applicazione standalone.
Nella creazione di uno spazio di lavoro impieghiamo i comandi della CLI di Angular, per creare il root del workspace "worspace root", anche identificabile come la directory del file bundle. La creazione del workspace root, crea inoltre i file di configurazione "angular.json" del workspace e un progetto iniziale dell'applicazione.
[26] (Angular, Workspace and project file structure,
https://angular.io/guide/file-structure#workspace-and-project-file-structure)
[27] (Angular, Glossary, https://angular.io/guide/glossary#command-line-interface-cli)
[27] (Angular, Glossary, https://angular.io/guide/glossary#workspace)
Il top-level del workspace contiene diversi files:

- workspace-wide configuration files
- configuration files per il root-level dell'applicazione
- sottocartelle per il root-level dell'applicazione sorgente

[26] (Angular, Workspace and project file structure,
https://angular.io/guide/file-structure#workspace-configuration-files)

9.1.1 WORKSPACE CONFIGURATION FILES

Presentiamo alcuni file importanti di cui tenere nota, appartenenti al workspace di un'applicazione.

File e Directory	Descrizione
angular.json	file di configurazione di default per tutti i progetti nello spazio "workspace", che include le opzioni di configurazione della CLI di Angular.
node_modules/	Cartella che fornisce i pacchetti Node package manager (npm) per l'intero spazio di lavoro (workspace). I pacchetti npm vengono impiegati per distribuire e caricare i moduli e le librerie di Angular.
src/	Cartella che contiene i file sorgente dell'applicazione al livello radice del progetto.

Tabella 7
[26] (Angular, Workspace and project file structure,
https://angular.io/guide/file-structure#workspace-configuration-files)

9.1.2 APPLICATION SOURCE FILES E APPLICATION PROJECT FILES

Per il workspace di una singola applicazione, la cartella src/ del workspace contiene i file sorgente della radice o

root dell'applicazione. Sempre tramite i comandi della CLI, quando si crea la cartella del workspace viene generato anche lo scheletro di una nuova applicazione nella cartella src/ nel livello più alto del workspace.
I file a livello più alto dell'applicazione nella cartella src/ permettono l'esecuzione dell'applicazione, le sottocartelle invece contengono i file sorgente dell'applicazione e i file configurazione specifici dell'applicazione.
[26] (Angular, Workspace and project file structure, https://angular.io/guide/file-structure #application-project-files)
[26] (Angular, Workspace and project file structure, https://angular.io/guide/file-structure #application-source-files)
All'interno della cartella src/ e la cartella app/ è presente la logica e i dati del progetto, i componenti di Angular, i template e gli stili.
[26] (Angular, Workspace and project file structure, https://angular.io/guide/file-structure #application-source-files)
File sorgente generati dalla CLI:

APPLICATION SOURCE FILES	
app/	Contiene i componenti in cui sono definite la logica e i dati della nostra applicazione.
assets/	Contiene file immagini e altri file asset da copiare come sono quando l'applicazione viene assemblata 'build'.
index.html	è la pagina HTML principale che viene servita quando qualcuno visita il sito. Nella pagina principale vengono aggiunti automaticamente tutti i file Javascript, Javascript bundle e file CSS dalla linea di comando CLI quando viene assemblata l'applicazione.
main.ts	è il punto di entrata per la nostra applicazione.
polyfills.ts	Provides polyfill scripts per il supporto ai browser.
styles.sass	Lista di file CSS che forniscono gli stile per il sito.

Tabella 8

index.html
Nella pagina principale HTML "index.html" vengono aggiunti automaticamente dalla CLI nella fase di building alcuni file Javascript che costituiscono il motore della nostra applicazione. I file main.js, vendor.js, polyfills.js e styles.js.

main.js
Come abbiamo visto in tabella 8 questo è il punto di entrata della nostra applicazione. Questo file contiene tutto il codice dell'applicazione scritta in Angular.

vendor.js
Questo file contiene tutte le librerie importate nella nostra applicazione app.module, incluse le librerie Angular e le librerie di terze parti.
Questo file può raggiungere dimensioni veramente grandi dopo la compilazione del progetto perché contiene

ogni cosa che richiede di compilare Angular in un browser

polyfills.js
Questo file contiene tutti i polyfills dichiarati nel file polyfills.ts presente nel workspace dell'applicazione
This one should be self explanatory. It contains all the polyfills you declare in your polyfills.ts file. Permette ad Angular di funzionare su vecchi browser.

styles.js
In questo file vengono importati tutti gli stili dichiarati nel file styles.ts

[26] (Angular, Workspace and project file structure, https://angular.io/guide/file-structure #application-project-files)

In figura 4 viene riportato il codice della pagina principale 'index.html' in un applicazione sviluppata in Angular 8.
Nella prima sezione viene riportato il codice come si presenta nella cartella che contiene i file sorgente dell'applicazione 'src/' e che costituisce il nostro ambiente di sviluppo. Nel codice è presente unicamente il tag <app-root> che è il primo componente ad essere caricato, ed è genitore di tutti gli altri componenti implementati dallo sviluppatore per la sua web app.

Nella seconda sezione invece abbiamo il codice del file 'index.html' generato all'interno della cartella 'dist/' dopo che l'applicazione è stata assemblata in produzione tramite il comando della CLI:
`ng build -prod`
Possiamo vedere che la pagina 'index.html' oltre a contenere il componente di Angualr <app-root> contiene i bundle Javascript. Tutto il contenuto della pagina viene aggiunto tramite i file javascript, tutto il contenuto sarà caricato dinamicamente in <app-root>.

src/index.html in progetto Angular

```
1.  <!doctype html>
2.  <html lang="en">
3.  <head>
4.    <meta charset="utf-8">
5.    <title>MyApp</title>
6.    <base href="/">
7.    <meta name="viewport" content="width=device-width, initial-scale=1">
8.    <link rel="icon" type="image/x-icon" href="favicon.ico">
9.  </head>
10. <body>
11.   <app-root>
12.     <!-- contiene i componenti realizzati per la nostra web app -->
13.   </app-root>
14. </body>
15. </html>
```

```
dist/index.html in scheda del Browser dopo il build del progetto

--------------------------------------------------------------------------------

<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>MyApp</title>
    <base href="/">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link href="styles.jkl8cd98f99b145e098.bundle.css" rel="stylesheet"/>
</head>
<body>
    <app-root>
    <!-- contiene i componenti realizzati per la nostra web app -->
    </app-root>
    <script type="text/javascript" src="inline.2br89a223b689m1x55az.bundle.js"></script>
    <script type="text/javascript" src="polyfills.1bw89a223b689m1x55az.bundle.js"></script>
    <script type="text/javascript" src="scripts.769m89a223b689m1x55az.bundle.js"></script>
    <script type="text/javascript" src="main.7mn10a223b689m1x55az.bundle.js"></script>
</body>
</html>
```

Figura 4 *File "index.html" prima e dopo il build in produzione*

9.2 SERVICE WORKER IN ANGULAR

L'implementazione di un service worker in un'applicazione web realizzata con Angular può rivelarsi vantaggioso, in quanto le applicazioni sviluppate in Angular sono principalmente applicazioni a singola pagina che ne beneficerebbero.

[28] (Angular, Service workers in Angular, https://angular.io/guide/service-worker-intro#service-workers-in-angular)

Come vedremo anche in seguito nel nostro caso di studio, una feature delle nuove versioni di Angular è l'inclusione del supporto ai service worker nei progetti.

[28] (Angular, Service workers in Angular, https://angular.io/guide/service-worker-intro #prerequisites)

[29] Angular, Getting started with service workers, https://angular.io/guide/service-worker-getting-started#getting-started-with-service-workers)

Tramite la CLI di Angular 8, con una semplice linea di comando, sarà possibile configurare l'applicazione per fare uso dei service worker aggiungendo il pacchetto di Angular "service worker" che inoltre si occupa di generare e impostare i file addizionali di supporto.

[29] Angular, Getting started with service workers,
https://angular.io/guide/service-worker-getting-started#adding-a-service-worker-to-your-project).
L'inclusione del pacchetto "service worker", rende la gestione della registrazione e installazione dello script del service worker e lo storage dei file di Angular più accomodante come vedremo in seguito in quanto sono gestiti da Angular e possono essere configurati manualmente dall'utente tramite i file di conifgurazione interni al progetto.
Un service worker creato internamente al motore di Angular è supportato da tutti i browser che supportano i service worker in generale.
Se il browser supporta i service worker, quando viene aperta la pagina 'index.html' dell'app web con un Angular Service Worker implementato, il browser scarica il service worker script 'ngsw-worker.js' di cui parleremo nel nostro caso di studio.
[28] (Angular, Service workers in Angular, https://angular.io/guide/service-worker-intro #browser-support)
[28] (Angular, Service workers in Angular,
https://angular.io/guide/service-worker-intro#service-workers-in-angular)

10 LA MEMORIA CACHE DI UNA PWA

10.1 TIPI DI CACHE : HTTP Cache & Cache API

Attualmente siamo interessati alla Cache del service worker manipolabile tramite la Cache API, ma bisogna anche considerare un altro tipo di memoria per non incorrere nella memorizzazione di informazioni non desiderate e stiamo parlando della memoria HTTP Cache o Cache del Browser.
Bisogna distinguere tra la memoria cache dedicata alla nostra progressive web app dalla memoria cache HTTP integrata del browser. Sono entrambe allocate in spazi di memoria separati e controllate differentemente.
Infatti è importante sapere che utilizzando la Cache del Service Worker non stiamo disabilitando la Cache del Browser che continuerà a effettuare anch'essa lo storage dei dati.
Ad esempio, prendiamo una web app con un service worker con un funzionamento di base, che semplicemente ad una richiesta del client restituisce la corrispettiva risorsa in cache se presente, altrimenti la richiesta viene spedita al server. Tale richiesta verrà prima intercettata dal Service Worker che controllerà nello spazio di memoria della cache sotto il suo controllo, dopodiché se non è stata trovata alcuna risposta allora verrà interpellata la Cache HTTP del Browser, in ultima se anche la Cache del Browser non ha dato responso positivo allora la richiesta verrà inviata al server web che recupera la risposta e la invia alla nostra web app. (Figura 5)

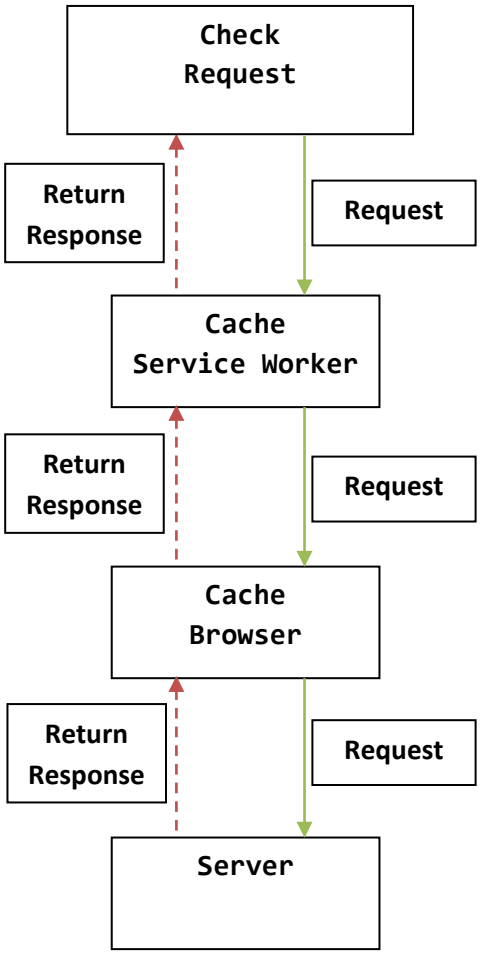

Figura 5 *Ordine con cui le memorie cache vengono controllate quando viene effettuata una richiesta.*

Vediamo quindi che la Cache del service worker ha una priorità maggiore rispetto a quella del browser, questo significa che il service worker può dirottare qualsiasi richiesta. Si può anche vedere che però un service worker non ha alcun controllo sulle fetch effettuate dal browser, per questo è importante riconoscere se la nostra web app fa anche uso della cache del browser.

Vediamo come funzionano le due memorie Cache e quali sono le differenze tra di loro:

La **cache HTTP di un browser** è la memoria di cache impiegata dal browser per copiare i dati lato client. La cache HTTP viene controllata tramite headers HTTP generato dal server da cui il browser prende istruzioni per determinare quali operazioni devono essere compiute su una risorsa. Non esiste alcuna API che ci permetta di controllare questo spazio di memoria direttamente dal codice Javscript la gestione della cache è affidata in modo automatico al browser. L'headers HTTP rappresenta l'header di configurazione delle opzioni per le richieste HTTP, ed è un elemento lato server controllato dall'amministratore di sistema. In Angular 8 possiamo modificare il comportamento della cache del browser, attaccando certi headers HTTP alla richiesta, un esempio è l'header Cache-Control che permette di prevenire lo store di una risorsa web se settato a 'no-cache'. Per evitare di dover aggiungere l'header Cache-Control su ogni richiesta HTTP, possiamo utilizzare una caratteristica di Angular chiamata 'Interceptor', che permette di dichiarare degli intercettatori che ispezionano

e trasformano ogni HTTP requests dall'applicazione al server, inoltre l'interceptor può anche trasformare la risposta del server che deve essere ritornata all'applicazione.
[38] (Angular, Http Client, https://angular.io/guide/http#http-interceptors)
La **cache del Service Worker** è la memoria dedicata alla nostra web app nella così detta 'cache' gestibile attraverso i metodi dell'API Cache, è riservata ad essere impiegata con i service worker e non sarà accessibile per modifiche autonome da parte del browser, ma solo dal service worker, infatti la cache è completamente separata dalla cache standard del browser. Quindi lo sviluppatore, può decidere come gestire questo spazio di memoria rimuovendo, aggiungendo o aggiornando i dati in base a come è stata implementata la logica di gestione della memoria senza doversi preoccupare che il suo spazio di memoria venga intaccato da eventuali modifiche perpetrate dal browser in background, poiché gli HTTP headers non possono essere impiegati sulla Cache Service Worker. **[30, 37, 39]**
Per il nostro caso di studio non siamo interessati a gestire le risorse tramite la cache HTTP del browser ma solo la cache del service worker.

10.2 CACHE API

Con l'introduzione della service worker API, in contemporanea è stata resa disponibile anche la Cache API, che mette a punto un'insieme di meccanismi per fornire agli sviluppatori la capacità di gestire manualmente la memoria dedicata alla loro web app e quindi anche le risorse che devono essere inserite o rimosse.
Le cache API sono definite nelle specifiche del service worker e vengono utilizzate congiuntamente con questi ultimi.
Le librerie in grado di gestire la memoria esistevano anche prima dell'API Cache, come ad esempio Storage API. Il problema come abbiamo visto sta nel fatto che queste API sono sincrone e quindi non è possibile adottarle per gestire la memoria all'interno di un service worker, che deve appoggiarsi a metodi e procedure asincrone dell'interfaccia Cache.
Nel caso fosse richiesto di separare le risorse in più cache con nomi diversi una web app può dichiarare più di un oggetto Cache. Questo permette una gestione più semplificata delle risorse in cache in quanto è possibile separare se si vuole le risorse in base a una classificazione di priorità che comporta una più facile eliminazione delle risorse in cache nel caso sia necessario liberare spazio nella memoria della web app.
La cache API ci permette di gestire manualmente tramite script il modo con cui fare gli update della cache. Gli elementi in una Cache non vengono aggiornati finché non viene esplicitamente richiesto e non scadono ammeno che non vengano eliminate. **[30]**
Gli oggetti memorizzati attraverso l'uso dell'interfaccia Cache, sono costituiti da una coppia di elementi "key/value". Nel campo "key" l'oggetto memorizzato sarà un oggetto request, o l'attributo stringa url della request. Nel secondo campo "value" viene registrato l'oggetto response.
Un elemento copiato nella cache ha quindi la seguente struttura: **(request, response) [31, 32]**

10.3 LIMITAZIONI DELLA CACHE API

È molto comune che in una pagina HTML un utente faccia richieste al web server con metodi GET, POST, PUT, etc., a questi metodi se ne possono trovare di diversi altri tipi. Nel nostro caso di studio ci siamo limitati a

servire richieste con metodi di due tipi, cioè GET e POST. Una limitazione della Cache API a quanto abbiamo scoperto è l'impossibilità di memorizzare oggetti request effettuati con metodo POST.
Questa restrizione ci impedisce dunque di salvare direttamente un oggetto request con metodo POST ma solo gli oggetti request con metodo GET.

10.4 COS'È LA FASE DI CACHING

La comunicazione all'interno di una rete è spesso un'attività lenta e costosa, specialmente se
si prendono in considerazione quelle comunicazioni in cui le risposte dal server sono ottenibili solo attraverso un numero elevato di scambi tra client e server.
Di conseguenza, la possibilità di mettere in cache e riutilizzare risorse recuperate in precedenza costituisce un aspetto fondamentale nell'ottimizzazione delle prestazioni **[37]**.
La fase di caching è la fase più importante di una PWA in quanto ci permette di rendere la nostra web app attiva e funzionante anche senza connessione Internet, inoltre fornisce una velocità di risposta migliore. Le risorse richieste più frequentemente possono essere memorizzate, così per tutte le richieste future, se la cache o le caches hanno un riferimento alla richiesta aggiornato, il client utilizzerà la risorsa copiata in cache per soddisfare la richiesta, altrimenti sarà necessario inviare la richiesta all'origine del server.
Ovviamente il processo di caching deve essere tenuto sotto controllo, infatti un uso incontrollato della cache può portare ad un'eccessiva quantità di dati memorizzati in cache, che rischia di saturare la memoria della web app, a questo si aggiunge inoltre il rischio di stagnazione dei dati in cache se non viene impiegata una politica di aggiornamento delle risorse salvate in cache per mantenerle in sincrono con il server. Viceversa se in cache non vengono o non possono (per scarsezza di spazio libero) essere inserite tutte le risorse necessarie a rendere la web app indipendente dal network, allora non saremo in grado di offrire una web app in grado di lavorare anche off-line.

In questo capitolo introduciamo alcuni dei metodi impiegati per controllare lo stato della memoria di una PWA.
Ma prima di vedere come è stato implementato il codice di gestione dello spazio di memoria della PWA è necessario introdurre i tipi di cache, di nostro interesse per questo particolare frangente, che si possono incontrare e come interagiscono tra loro.

10.5 STRATEGIE DI CACHING CON IL SERVICE WORKER

Con strategie di caching nel service worker si intende un'insieme di tecniche di base messe a nostra disposizione, come linee guida e che si distinguono l'una dall'altra in base alla priorità che si decide di assegnare all'accesso delle risorse in cache rispetto al recupero delle risorse fornite dal web server. Tutte queste strategie vengono implementate dagli sviluppatori web in base alle esigenze del sito web stesso e del contesto in cui esso lavora.
Presentiamo una serie di strategie di pratica comune. Le strategie di caching esposte possono essere combinate tra loro per cambiare il comportamento del service worker in situazioni particolari. **[33]**

NETWORK FIRST (NETWORK, FALLING BACK TO CACHE)

Questa strategia scarica direttamente la risorsa dal network, servendo il client dalla cache solo in caso di problemi di connettività.

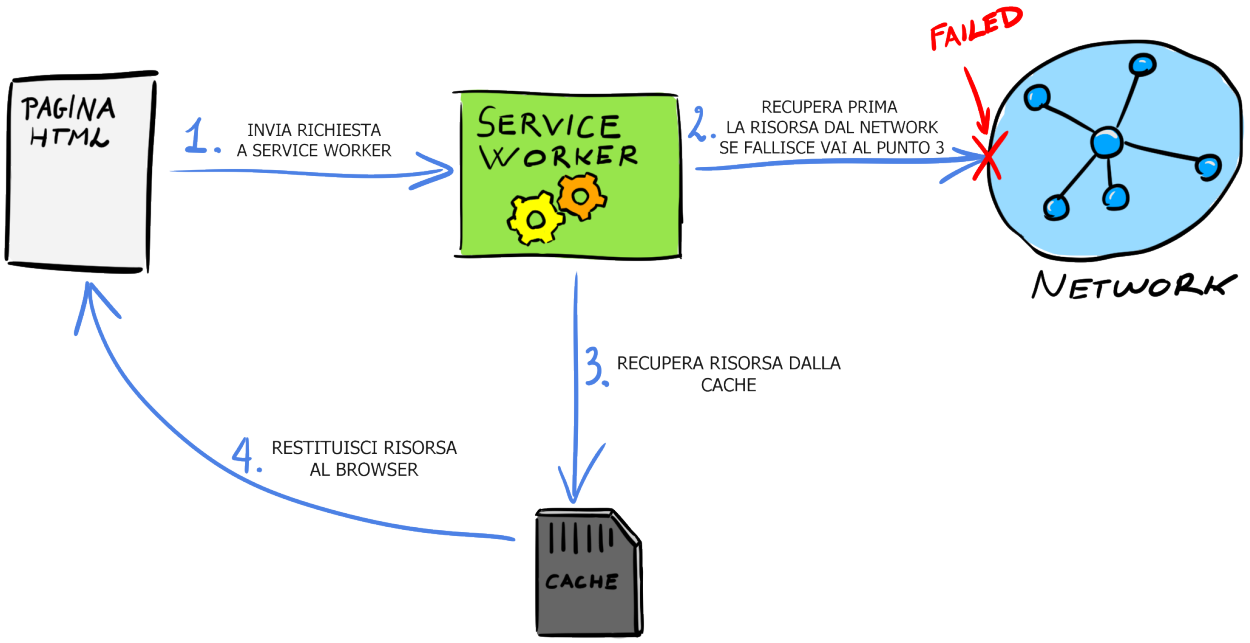

Figura 6 *Network First*

Come si vede dalla Figura # la richiesta viene inviata al service worker (1), che poi la invia prima al web server (2). In caso di successo la risposta viene recuperata dal server e spedita al client. Nel caso l'operazione di recupero della risorsa dal server fallisca il service worker cerca di recuperare la risorsa in cache (3). Una risposta viene ritornata al client (4) in base all'esito della ricerca nella cache.
Questa strategia viene impiegata quando i dati cambiano frequentemente e l'utente ha bisogno di risorse costantemente aggiornate, ma si vuole dare la possibilità all'utente di vedere i dati anche quando è offline.

NETWORK FIRST AND CACHE UPDATE

Una variazione alla strategia Network First appena esposta è la Network First then Cache Update che aggiorna la risorsa in cache dopo che la risorsa è stata scaricata dal web server, con questo sistema siamo quasi certi che le risorse restituite all'utente in caso di problemi di connessione sono aggiornate.

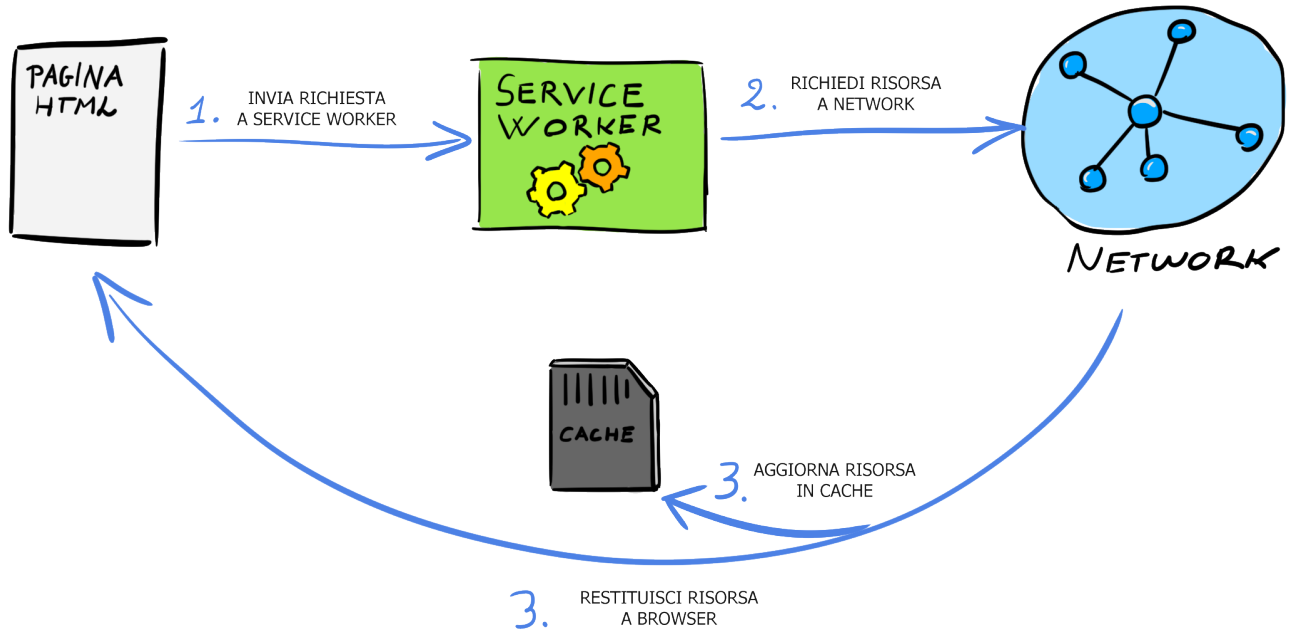

Figura 7 *Network first and cache update caso 1*

In figura # viene presentato il caso in cui il server risponde alla richiesta del service worker con una risorsa. Come si vede in modo generico nel punto 3, quando il service worker recupera la risorsa questa viene ritornata alla pagina web, ma prima viene copiata in cache aggiornando l'istanza della risorsa se esiste già in memoria.

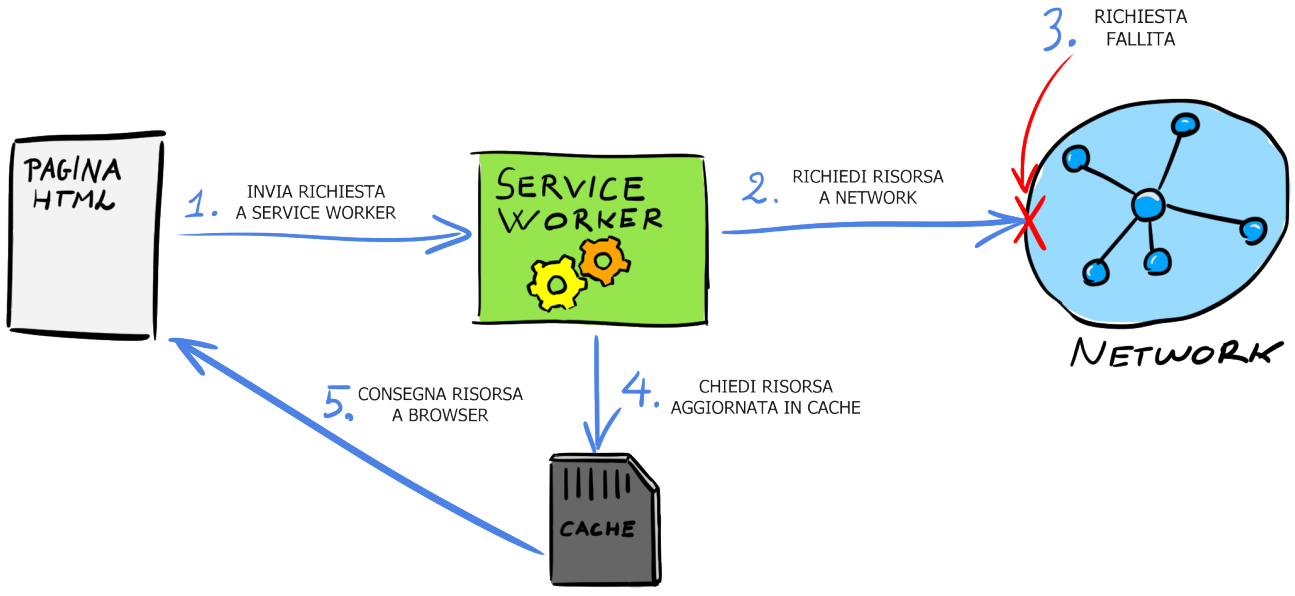

Figura 8 *Network first and cache update caso 2*

In Figura # invece abbiamo una rappresentazione in cui il server non restituisce una risorsa in questo ultimo caso dopo aver fatto la chiamata, il service worker decide di controllare la cache e restituire la risorsa dalla memoria. Grazie all'aggiornamento costante delle risorse siamo certi in questo caso che la risorsa, recuperata

dalla cache sarà anche la più recente dall'ultima chiamata.

CACHE FIRST (CACHE, FALLING BACK TO NETWORK)

Questa strategia restituisce la risorsa direttamente dalla cache. Se però la risorsa non è disponibile in cache allora la richiesta dell'utente viene servita dal network.

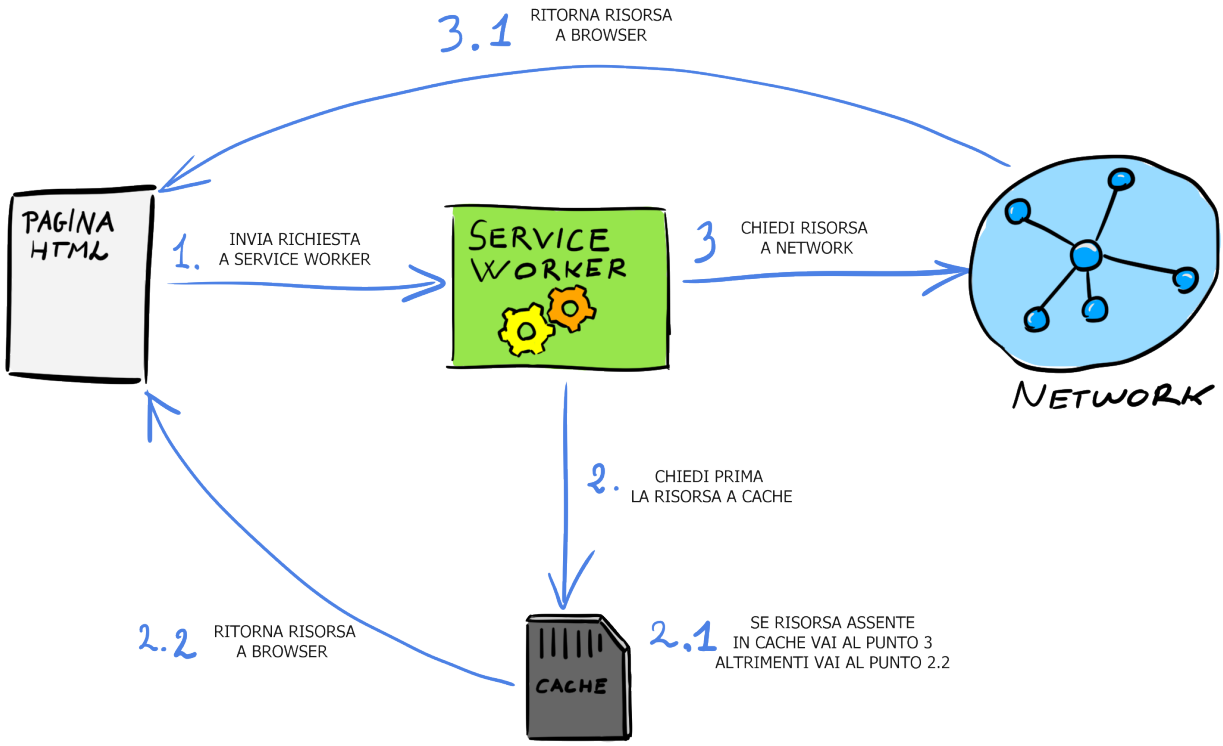

Figura 9 *Cache first*

CACHE ONLY

In netto contrasto con la strategia appena presentata, il service worker quando cattura un evento fetch lanciato dal client, si limita molto banalmente a servire il client dalla cache indipendentemente che il network sia disponibile oppure no.

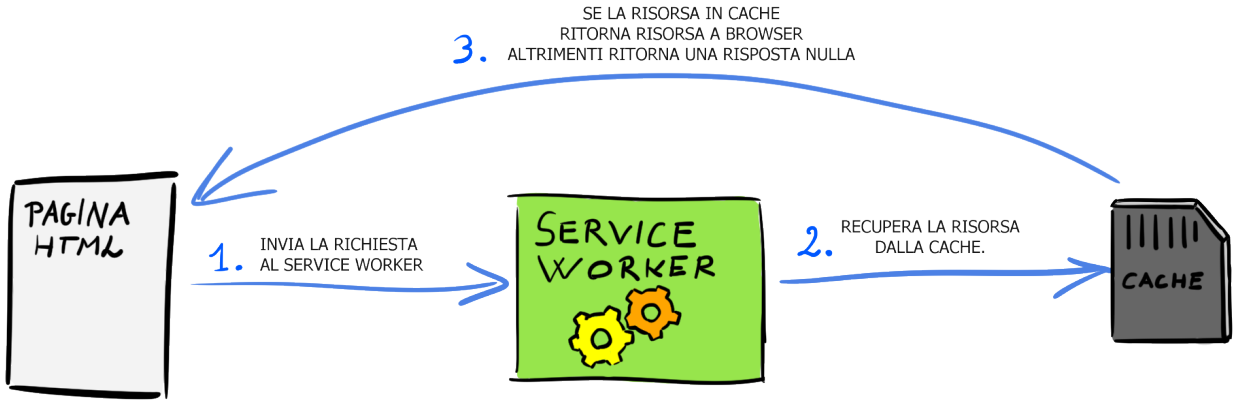

Figura 10 *Chace only*

È utile quando si deve offrire del contenuto statico che sappiamo non cambia mai e siamo sicuri che sarà disponibile dalla cache del nostro sito. Per implementare questa strategia dobbiamo assicurarci che il client ottenga sempre una risorsa dalla cache quando il service worker diventa attivo. Dobbiamo quindi, durante l'installazione del service worker, effettuare un precaching delle risorse statiche. **[34]**

NETWORK O CACHE

In questa strategia il service worker da una maggiore priorità al network senza però rifiutare l'intervento della cache nel caso i tempi di risposta del server si prolunghino oltre modo, infatti il service worker cerca di recuperare le risorse più aggiornate dal server web, e interpella la cache solo se il network impiega un'eccessiva quantità di tempo a rispondere.

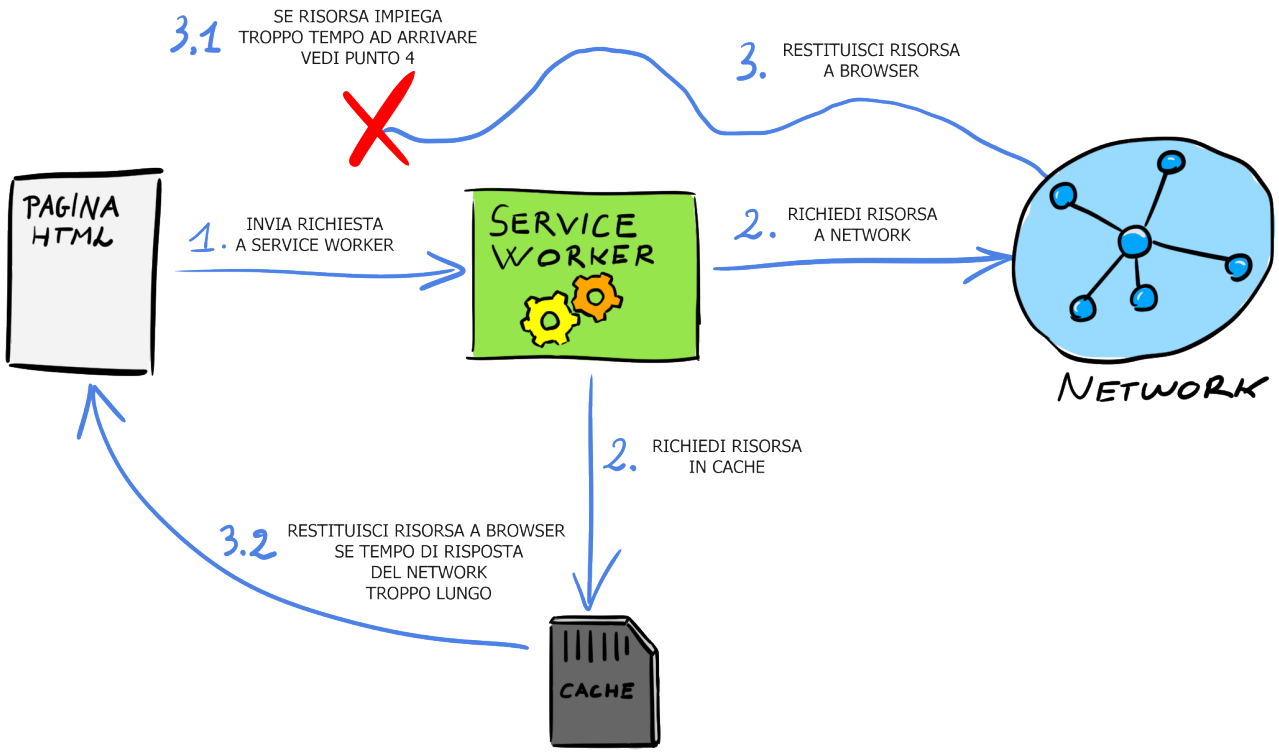

Figura 11 *Network o Cache*

Questo metodo può essere utile se si vuole fornire all'utente contenuti sempre aggiornati ma che sia anche veloce a caricarsi, nel caso la connessione al server non sia disponibile o si riscontrano possibili tempi di risposta lenti a causa di un eccessivo traffico sulla rete il service worker deve intervenire sulla cache per non degradare l'esperienza utente. **[35]**

CACHE AND UPDATE (CACHE THEN NETWORK) E CACHE, UPDATE AND REFRESH

Questa strategia consente di ottenere risposte rapide ma anche di aggiornare le risorse nella cache dal network. Il service worker risponde ad una fetch prelevando la risorsa direttamente dalla cache per fornire il contenuto in modo rapido e solo in seguito dopo aver servito subito l'utente effettua anche un update della risorsa nella cache attraverso una chiamata al network.

Si impiega nel caso si voglia consegnare istantaneamente il contenuto richiesto e non sia prioritario che sia in sincrono con il web server.
Viene implementata servendo la risorsa dalla cache, ed effettuando in contemporanea una chiamata fetch al server. La prossima volta che la pagina sarà visitata dall'utente la risorsa servita dalla cache sarà in sincrono con il web server.

Una variazione a questa strategia è la strategia 'Cache, Update and Refresh'. Il funzionamento è molto simile alla precedente strategia: il contenuto viene servito immediatamente all'utente e la risorsa viene aggiornata in cache, ma non appena la nuova risorsa aggiornata è disponibile le informazioni visualizzate dall'utente vengono

aggiornate automaticamente senza aspettare il refresh manuale della pagina. **[36]**

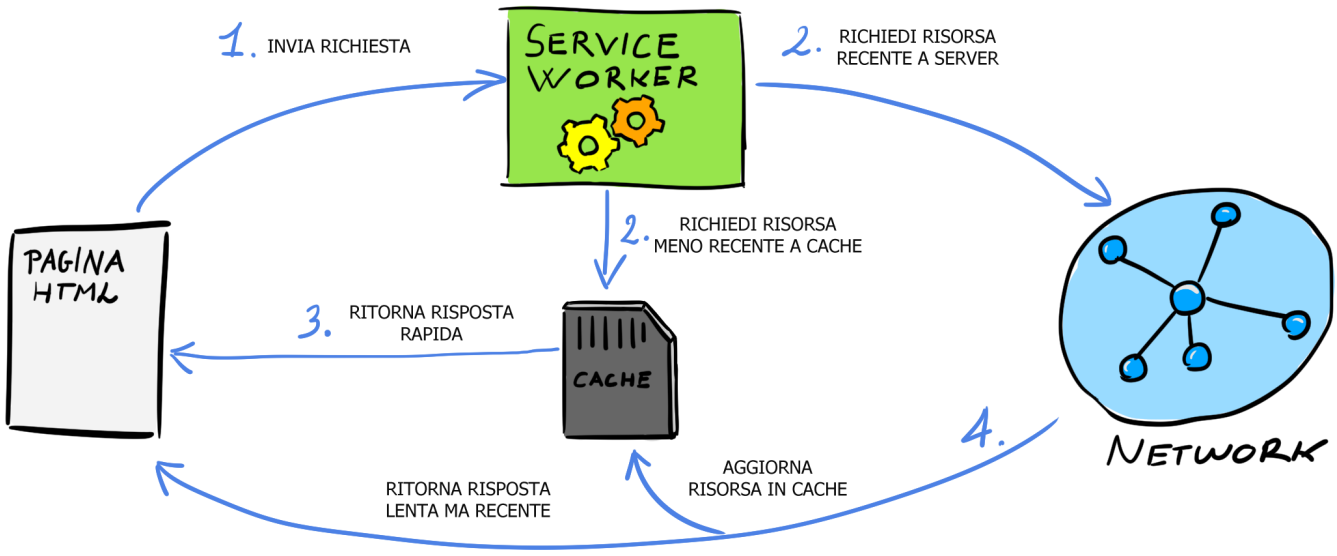

Figura 12 *Cache, Update and Refresh*

11 GESTIONE DELLA MEMORIA CACHE DEL SERVICE WORKER

Per poter gestire la memoria della nostra web app è stato realizzato un modulo da precaricare nel service worker, costituito da funzioni e procedure che permettono di ricavare informazioni sullo stato della memoria e di conseguenza ricavare informazioni utili al service worker per permettergli di decidere se deve liberare spazio in cache, per creare spazio per nuove risorse, rimuovendo magari quelle risorse a priorità più bassa.
Sebbene questa features sia presente solo come modulo non integrato ancora nella nostra implementazione di un service worker, le procedure e funzioni che lo costituiscono sono da considerarsi valide per poter gestire al meglio lo spazio di memoria destinato alla nostra PWA.
Presentiamo di seguito le informazioni che il nostro modulo di gestione della memoria allocata, deve essere in grado di fornire al service worker sulla mamoria cache della nostra PWA.

1 - Spazio di memoria della PWA: Questa informazione ci deve fornire le dimensioni totali riservate dal browser web alla nostra applicazione in cache.
2- Spazio libero: è lo spazio che la nostra PWA ha ancora a disposizione per memorizzare nuove risorse in cache. Può essere espresso in bytes o in percentuale.
3- Spazio occupato: è lo spazio che la nostra PWA ha già occupato con i dati copiati in cache. Può essere espresso in Bytes o percentuale.

Per poter effettuare i nostri calcoli sulla memoria cache esistono già delle API che corrono in nostro soccorso e che ci permettono di ricevere informazioni riguardo le dimensioni dello spazio di memoria utilizzato o ancora da utilizzare. Le più recenti API sono lo Storage API e la Quota Management API. **[40, 41]**
Come si può dedurre già dal nome, le informazioni della cache ritornate da queste API, sono da considerarsi stime approssimative dello spazio utilizzato e spazio totale della memoria della web app.

11.1 SPAZIO DI MEMORIA ALLOCATO IN CACHE DA UN BROWSER

Secondo una ricerca condotta nel 2014 da Eiji Kitamura sui limiti di allocazione di spazio di memoria su dispositivi mobili concesso da diversi browser ai siti web, che usano differenti tecnologie per memorizzare dati, quali WebSQL Database, Indexed Database, FileSystem API, Application Cache,
[42] (HTML5 Rocks, Working with quota on mobile browsers - A research report on browser storage, https://www.html5rocks.com/en/tutorials/offline/quota-research/#toc-introduction) si è concluso che ogni browser ha caratteristiche uniche quando si lavora con storage, quota, e limits specialmente per i dispositivi mobili. Anche lo spazio di memoria della cache allocata per la web app dall'API Cache non è indipendente dal tipo di browser. **[42] (HTML5 Rocks, Working with quota on mobile browsers - A research report on browser storage, https://www.html5rocks.com/en/tutorials/offline/quota-research/#toc-conclusion)**
Per i browser più utilizzati da un'ampia fetta di pubblico, quali Chrome, Firefox, Safari, Internet Explorer 10 sono state condotte delle ricerche da parte del gruppo web developer di Google. Da una relazione redatta da Addy Osmari e Marc Cohen sono state riportate in una tabella il limite massimo di spazio libero permesso dai browser più popolari. Qui prendiamo in considerazione solo la memoria allocata per la service worker cache.
[43] (Web Fundamentals, Offline Storage for Progressive Web Apps, https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa#how_much_can_i_store)

Chrome Storage Limitation for Desktop and Mobile
In un ambiente Chrome la memoria allocata per una PWA è per origine. *es. di origine*

`Origin: <scheme> "://" <hostname> [ ":" <port> ]`
 	
`Origin: https :// myHostName:400/`

La memoria allocata per origine è <6% dello spazio libero. Chrome in Incognito impone una restrizione sulla quota massima di spazio allocabile di 100 megabytes, indipendentemente dalla quantità di spazio disponibile
Lo spazio di memoria è condiviso tra Local storage, session storage, service worker cache, IndexedDB

Sarà possibile memorizzare dati finche il browser non raggiunge la quota massima di spazio allocato.

Opera Storage Limitation
In un ambiente Opera la memoria allocata per una PWA è per origine.

Mobile Safari Storage Limitation
In un ambiente iOS Safari il limite per il service worker cache è al massimo di 50Megabytes, quindi è molto restrittivo.
Inoltre la memoria cache di un service worker che non viene impiegato per molto tempo viene eliminata dal dispositivo iOS.
Desktop Safari Storage Limitation
In un ambiente Safari Desktop non c'è limite alla quantità di spazio utilizzato dalla cache del service worker.

Mobile Firefox Storage Limitation
In un ambiente Firefox per dispositivi mobili lo spazio di allocazione massimo consentito in cache è al massimo del 10% dello spazio libero
L'applicazione può oltre fino oltre il 10% dello spazio libero.
Desktop Firefox Storage Limitation
Non c'è limite alla memoria massima consentita per la service worker cache.

Internet Explorer 10 Storage Limitation

L'applicazione web è autorizzata a memorizzare dati fino a un massimo di 250MegaBytes
Ricordiamo che Internet Explorer non supporta i service worker.

[43] (Web Fundamentals, Offline Storage for Progressive Web Apps, https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa#how_much_can_i_store)
[44] (Web Fundamentals. Progressive Web Apps Training. Live Data in the Service Worker. https://developers.google.com/web/ilt/pwa/live-data-in-the-service-worker#how_much_can_you_store)

12 PROGETTAZIONE SERVICE WORKER

12.1 TWIPPING APP

Iniziamo con il descrivere quella che è la nostra web app, che dovrà subire una trasformazione da semplice web app a PWA.
Twipping è una piattaforma web che offre alle imprese e aziende la possibilità di richiedere e costruire il proprio spazio di lavoro. Il modo con cui twipping fa questo è abbastanza innovativo in quanto si basa su un sistema di incapsulamento, che sfrutta il concetto di riusabilità e dinamicità offerto dal sistema a componenti di Angular, permettendo di caricare su richiesta dalla piattaforma il template della nostra web app e il suo contenuto, impiegando un unica pagina. Molto più semplicemente abbiamo una SPWA che è mutevole e che cambia la propria struttura in funzione dell'utente, in quanto, per ogni utente registrato, gestisce uno spazio di lavoro, detta capsula, che contiene il template e il contenuto della nostra SPWA. I componenti Angular di cui abbiamo bisogno vengono caricati sul momento, da twipping sempre attraverso la stessa singola pagina principale della nostra web app.
l' 'API 'getTemplateData' contiene i componenti e la logica della capsula che è stata caricata e viene scaricata dal browser insieme alle altre risorse richieste dalla pagina.
Il nostro obbiettivo primario è stato quello di comprendere come si possa integrare un service worker customizato in una SPWA realizzata tramite il framework Angular 8 per garantire un servizio offline stabile della web app.
Si sono scelti due approcci al problema, nel primo approccio decidiamo di gestire tutti i file di Angular manualmente. Diversamente nel secondo approccio impieghiamo gli strumenti messi a disposizione da Angular per gestire un service worker, questo secondo approccio permette di svincolare lo sviluppo del service worker dalla gestione dei file interni ad Angular.
Ci siamo poi posti come secondo obbiettivo l'implementare di una logica di gestione delle risorse tramite il service worker che sia in grado di controllare in cache solo un certo numero di risorse specifico per ogni singolo spazio o capsula gestito dalla nostra 'Twipping app'.

12.2 CASI DI STUDIO

Abbiamo due casi da studiare separatamente, in entrambi i casi ci prefiggiamo due obbiettivi:

1. Fornire un servizi offline per la nostra web app.
2. Gestire la memoria della nostra web app

Nel primo caso di studio (Figura 13) il service worker non si avvale dell'aiuto dei paccheti Angular Service Worker. La registrazione e il caricamento in memoria dei file di Angular dovranno essere gestiti dallo sviluppatore insieme agli altri assets da copiare in cache.
La registrazione viene fatta manualmente attraverso uno script javascript inserito nella pagina principale che si occupa anche di aggiornare il service worker, mentre la copia dei file bundle di Angular non potrà essere fatta durante la fase di precaching ma sarà necessario individuare le chiamate ai file javascript di Angular come main, runtime, polyfills, styles e vendor nella fase di fetch all'interno del service worker.

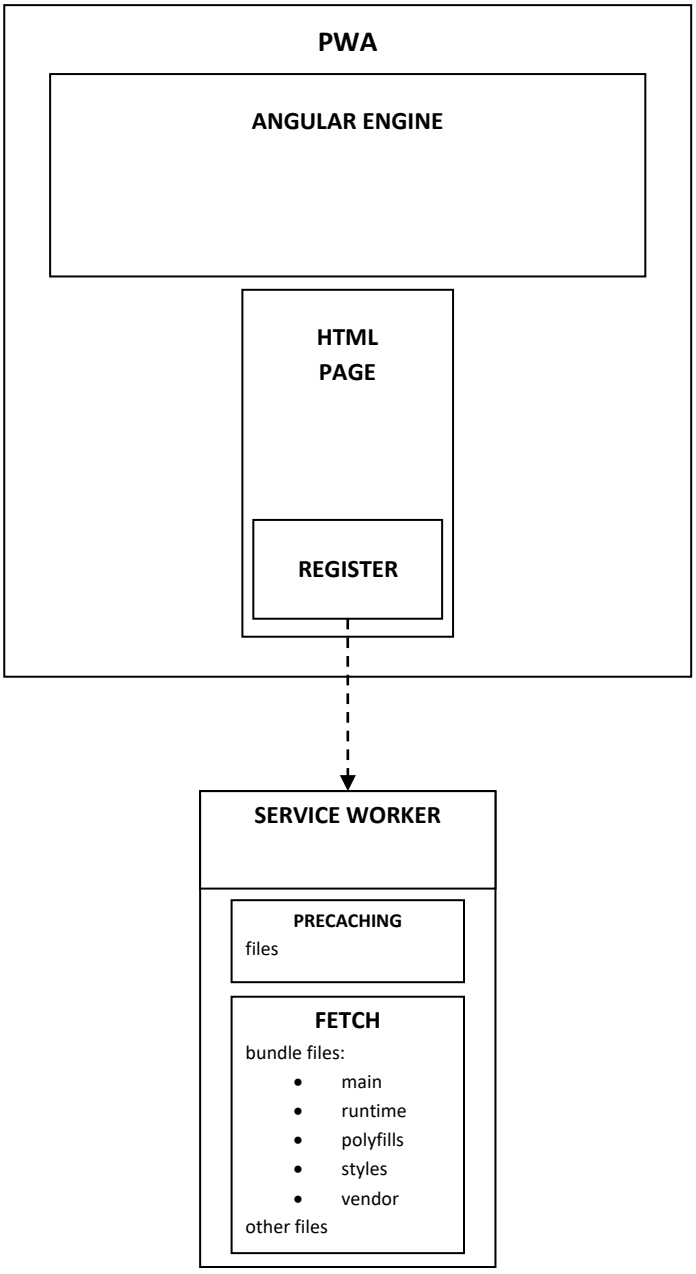

Figura 13 *CASO 1: Gestione manuale dei file Angular 8 all'interno dello script service worker*

Nel secondo caso invece installiamo il pacchetto di Angular Service Worker che crea un service worker di base che ci permette di registrare e gestire i file di Angular direttamente tramite l'impiego dei file di configurazione del service worker (Figura 14). Con questa implementazione non abbiamo bisogno di un realizzare uno script di registrare per registrare e aggiornare il service worker e inoltre potremmo copiare nella fase di precaching i file di sistema di Angular, semplicemnente agendo sui file di configurazione del service worker di Angular. Sarà inoltre possibile estendere lo script del service worker generato da Angular con il nostro service worker personalizzato per gestire le altre risorse da inserire in cache e la memoria della nostra web app.

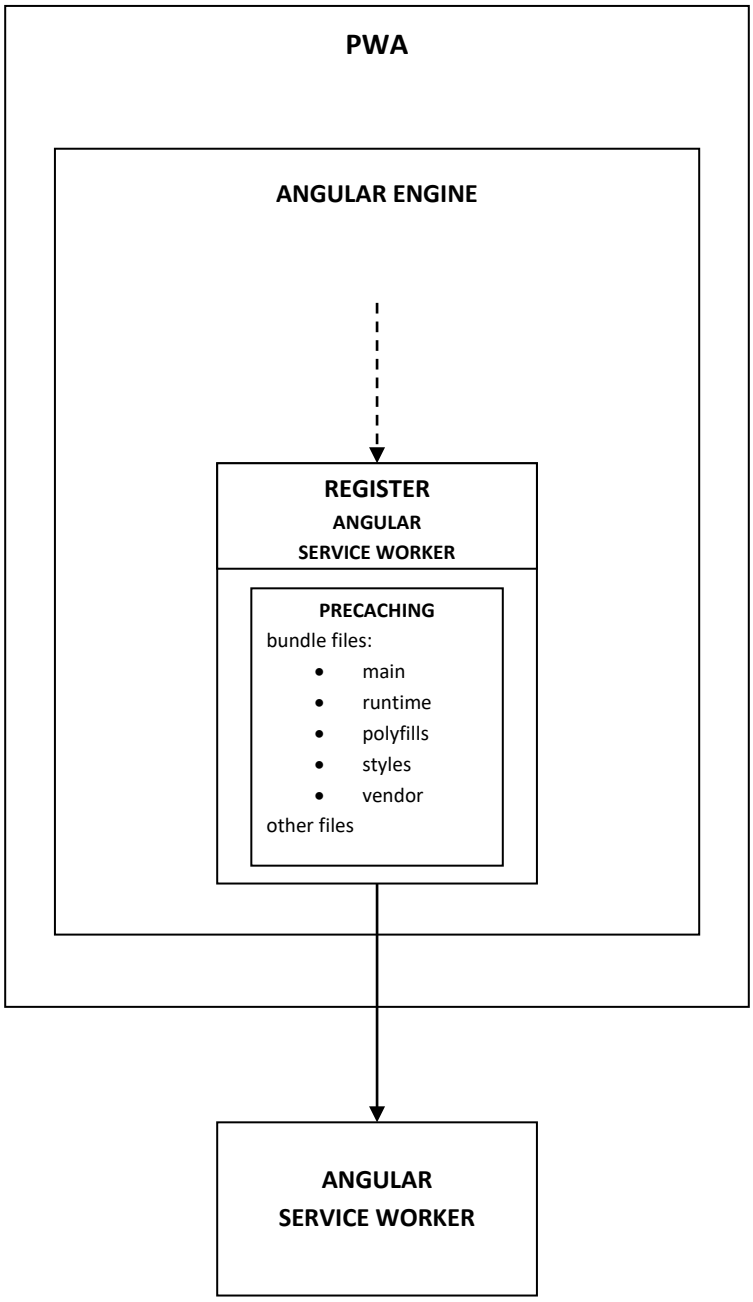

Figura 14 *CASO 2: Gestione interna file Angular con Angular Service Worker*

Molti degli elementi che verranno presentati nel primo caso di studio sono validi anche per il secondo caso di studio.

12.3 PRIMO CASO DI STUDIO - GESTIONE MANUALE DEI FILE BUNDLE

In questo caso di studio per realizzare la nostra PWA, dobbiamo innanzitutto registrare il service worker manualmente. Quindi sono due gli script principali che devono essere integrati con il progetto della nostra web application. Il primo file javascript è il 'service-worker-register.js' che si occupa di registrare il nostro service

worker con l'applicazione web 'Twipping'. Il secondo script è il file 'service-worker.js', che verrà implementato per intercettare gli eventi lanciati dal client, fetchando le richieste verso il web server e decidendo queli memorizzare in cache.

12.3.1 STRUTTURA DEL SERVICE WORKER

Il file nominato "service-worker.js" costituisce il cuore del nostro progetto, quando il browser deve effettuare la registrazione del service worker richiederà al web server questo file.
Trattandosi di un file di dimensioni estese gran parte della logica del nostro service worker è stata separata in moduli.
Una struttura a moduli ci permette di organizzare la logica di funzionamento del service worker in blocchi che saranno più facili da sostituire. L'inserimento dei moduli all'interno di un service worker può essere effettuato unicamente impiegando il metodo "importScripts", che permette di importare i moduli in modo sincrono all'interno dello spazio di lavoro del service worker **[45]**. Tutti i moduli vengono richiamati dal browser in una prima chimata e memorizzati nel browser. I moduli possono essere importati solo in due modi, specificando le chiamate all'interno dell'install handler, in questo caso vengono caricate in modo asincrono, o effettuando la chiamata ad importScripts() durante l'esecuzione sincrona dello script di altro livello (top-level) del service worker.
Quando viene fatta una chiamata al metodo "importScripts", l'esecuzione del service worker viene messa in pausa e il codice addizionale viene scaricato dall'URL specificato come argomento del metodo "importScripts". Quando l'operazione è conclusa, il service worker riprende l'esecuzione.

12.3.2 COMPATIBILITÀ DI IMPORT SCRIPTS

"importScripts" risulta essere compatibile praticamente con tutti i browser più comuni, aggiornati alla versione più recente. La compatibilità è garantita sia per i browser per dispositivi desktop che per i dispositivi mobile. Chrome, Edge, Safari, Opera, Firefox, Internet Explorer, tutti supportano l'importazione di moduli.
Va fatto notare che però il comportamento di "importScripts" può variare da browser a browser.
Ad esempio Firefox 56, Chrome 78 e Safari ogni volta che il service worker file deve essere aggiornato, vengono effettuati dei controlli anche sui file che devono essere importati. L'aggiornamento viene effettuato anche solo nel caso in cui da un controllo byte per byte sui file importati risulta che uno dei moduli è diverso, in tal caso l'aggiornamento viene esteso anche al file del service worker, anche se si tratta dello stesso identico file che non ha subito modifiche.
Si notano invece comportamenti diversi per le versioni precedenti a Chrome 78, infatti dopo la prima chiamata a "importScripts", le successive richieste fetch allo stesso file non vengono eseguite.

[46] (Web Fundamentals, Tweaks to cache.addAll() and importScripts() coming in Chrome 71, https://developers.google.com/web/updates/2018/10/tweaks-to-addAll-importScripts#disallowing_asynchronous_importscripts)

[47] (Web Fundamentals, Fresher service workers, by default, https://developers.google.com/web/updates/2019/09/fresher-sw#checks_for_updates_to_imported_scripts)

12.3.3 ORGANIZZAZIONE DEI MODULI

Abbiamo realizzato complessivamente sei moduli, tre implementano le funzioni di gestore degli eventi di installazione, attivazione e fetching. Degli altri tre moduli due vengono richiamati dal modulo di fetching, come sotto-moduli e contengono algoritmi utili ad elaborare le richieste intercettate dal service worker. In ultima abbiamo il modulo di gestione dello spazio di memoria della nostra Applicazione Web, che contiene gli algoritmi per controllare lo spazio allocato alla nostra PWA dal browser.

DIAGRAMMA DEI MODULI

- Files Service Worker
- Main:
  - service-worker.js
  - service-worker-registration.js
- Moduli:
  1. swInstall.js
  2. swActivate.js
  3. swFetch.js
  4. swStorageManagement.js
- Sotto-Moduli del modulo swFetch.js:
  1. swSpecialRequest.js
  2. swCacheLogic.js

Codice Principale Service Worker in 'service-worker.js'

```
service-worker.js
-----------------------------------------------------------
    // moduli da caricare nello script principale del service worker.
    const _modules = {
        install: '/assets/js/service-worker-modules/md_swInstall.js',
        activate: '/assets/js/service-worker-modules/md_swActivate.js',
        fetch: '/assets/js/service-worker-modules/md_swFetch.js'
    };

        // * Evento Install * //
        importScripts(_modules.install);
        self.addEventListener('install', function(event){
            ...
        });

        // * Evento Activate * //
```

```
17.         importScripts(_modules.activate);
18.         self.addEventListener('activate', function(event)
19.         {
20.             ...
21.         });
22.
23.         // * Evento Fetch * //
24.         importScripts(_modules.fetch);
25.         self.addEventListener('fetch', function (event) {
26.             ...
27.         });
```

Figura 15 *Struttura interna del file javascript 'service-worker.js'*

12.3.4 REGISTRAZIONE MANUALE DEL SERVICE WORKER

Per registrare manualmente il service worker abbiamo realizzato lo script "service-worker-registration.js".
La sua implementazione è molto semplice poiché richiede semplicemente di controllare se il browser supporta il service worker, controllando se l'API Service Worker è presente nel browser e richiede di fare una chiamata al metodo "register" dell'API Service Worker indicando il percoso del nostro script "service-worker.js", che ci permette di creare o aggiornare la registrazione del service worker. Facendo la chiamata all'url del service worker, lo script del service worker viene scaricato dal browser.
[8] W3C Candidate Recommendation, 19 Novembre 2019, Service Workers 1, https://www.w3.org/TR/service-workers-1/#navigator-service-worker-register)
Nello specifico il metodo di registrazione, crea una tupla chiamata appunto service worker registration, che contiene il service worker stesso. La tupla contenente l'url scope, che identifica univocamente il service worker e che deve essere sulla stessa origine del documento che registra il service worker e un set di service workers settati a null. Questo set di service workers è composto da un installing worker, un waiting worker e un active worker. Ognuno di questi web worker è attivo in uno stato preciso del service worker. Un service worker a livello tecnico ha sei stati "parsed", "installing", "installed", "activating", "activated" e "redundant". Inizialmente il service worker ha stato settato a "parsed". L'installing worker è settato sullo stato "installing", mentre il waiting worker su stato "installed". "acive worker" può avere lo stato "activating" o lo stato "activate".
[8] W3C Candidate Recommendation, 19 Novembre 2019, Service Workers 1, https://www.w3.org/TR/service-workers-1/#service-worker-concept)
[8] W3C Candidate Recommendation, 19 Novembre 2019, Service Workers 1, https://www.w3.org/TR/service-workers-1/#service-worker-registration-concept)
Abbiamo già introdotto il concetto di scope nei paragrafi precedenti e di come questo sia importante per un service worker worker poiché determina la visibilità di un service worker sulle pagine della nostra web app. Per permettere che il service worker sia in grado di controllare tutti i documenti della nostra web app, dobbiamo piazzare il service worker script nel root-level di progetto, che detto più semplicemente significa inserire il "service-worker.js" file nella directory "src/" del nostro progetto. Alternativamente per ragioni di ordine e

consistenza è possibile, inserire il file del service worker e dei moduli associati al service worker in una cartella unica, in questo caso si è deciso di usare la cartella predefinita di Angular per gli asset di progetto, "src/assets", e creare una copia del file "service-worker.js" nel root del progetto, agendo sul file di configurazione del workspace, "angular.json". Dentro il file "angular.json" scriviamo:

```
/*
 The architect section of angular.json contains a set of Architect targets.
 Many of the targets correspond to the CLI commands that run them.
 */
 ....
 ....
 "architect": {
        "build": {
          ....
          ....
            "assets": [
              "src/assets",
              {
                "glob": "service-worker.js",
                "input": "src/assets/service worker",
                "output": "/"
              }
            ],
          ....
          ....
        },
        "serve": {
          ....
        },
        "test": {
          ....
        },
        "lint": {
          ....
        }
        ....
    }
....
....
```

Figura 16 *Configurazione del file "angular.json", per copiare il file "service-worker.js" dalla cartella asset nel root di progetto*

```
// Servie Worker Registrazione

// service worker script path
const _swPath = './service-worker.js';

function serviceWorkerRegistration(path){
    if('serviceWorker' in navigator){
        navigator.serviceWorker.register(path, {scope: './'}).then(function(registra
tion){
        // Esegui se registrazione ha successo.
        console.log('Service worker registrato');
        console.log ('Service worker scope è:', registration.scope);
        }).catch(function(error){
```

```
13.         // Esegui se registrazione fallisce.
14.         console.log('Registrazione service worker fallita:', error);
15.         });
16.     }else{
17.         throw new Error("No Service Worker support!");
18.     }
19. }
```

Figura 17 *Registrazione del service worker in 'service-worker-registration.js'*

L'ultimo passo è inserire lo script "service-worker-registration.js" nella pagina HTML principale di Angular, "index.html", da dove viene avviata e gestita tutta la nostra applicazione web a singola pagina. Sarà necessario inserire il codice javascript solo in una pagina poiché come abbiamo già visto, il vantaggio di usare un service worker in una applicazione a singola pagina, rispetto alle applicazioni web a pagine multiple è che non si deve richiamare il codice di registrazione da tutti i documenti.
Quando l'utente apre la nostra web app sulla pagina "index.html" il codice di registrazione viene eseguito dal browser.
Ad ogni refresh della pagina "index.html" il file di registrazione viene eseguito, questo permette anche al browser di controllare se il service worker deve essere aggiornato con una nuova versione presente sul server che differisce da quella attuale. Nel caso in cui si pensa che l'utente utilizzi il sito per un lungo tempo senza caricare la pagina, possiamo richiamare la funzione "registration.update()" periodicamente all'interno della pagina dopo un certo intervallo di tempo.

Codice di registrazione del service worker nel file index.html (pagina principale)

```
index.html

--------------------------------------------------------------------------------

1.  <!doctype html>
2.  <html lang="en">
3.  <head>
4.    <meta charset="utf-8">
5.    <title>Twipping App</title>
6.    <base href="/">
7.    <meta name="viewport" content="width=device-width, initial-scale=1">
8.    <link rel="icon" type="image/x-icon" href="favicon.ico">
9.     ...
10. </head>
11. <body>
12.   <app-root>
13.       <!-- qui inserisco tutti i componenti Angular della mia web app -->
14.   </app-root>
15.   ...
16.   <!-- register the Service Worker -->
17.   <script src= "./service-worker-registration.js"></script>
18.   ...
19. </body>
20. </html>
```

Figura 18 *Pagina principale della PWA 'index.html' con link a script di registrazione del service worker*

12.3.5 GESTIONE DEGLI EVENTI NEL SERVICE WORKER

Un service worker è un event-driven web worker, questo significa che il service worker reagisce esclusivamente agli eventi, rispondendo agli eventi inviati dai documenti e altre risorse.
All'interno dello script del service worker gestiremo solo tre tipi di eventi per il nostro caso di studio che sono gli eventi install, activate e fetch, mettendo il service worker in ascolto di tali eventi. Per ogni evento specificato, che viene consegnato al service worker, il service worker richiama una funzione, da noi creata, in risposta all'evento.
A permettere la gestione degli eventi nell'API Service Worker è sempre l'interfaccia 'ServiceWorkerGlobalScope' che implementa dei gestori di eventi anche detti proprietà. Le più rilevanti per il nostro caso di studio sono le proprietà 'oninstall', 'onactivate', 'onfetch' che rispondono agli eventi di installazione, attivazione e fetch.
Abbiamo già visto quali sono nel ciclo di vita del service worker le fasi che lo interessano e i suoi possibili stati, quando un service worker deve essere installato e poi passa in fase di attivazione vengono lanciati due eventi, rispettivamente un'evento 'install',che viene lanciato subito dopo che la fase di registrazione è stata completata e un evento 'activate' che viene lanciato dopo che il service worker è stato installato. Entrambi i due eventi possono essere gestiti all'interno del service worker grazie agli event handler 'oninstall' e 'onactivate' presentati sopra e che si attivano non appena uno dei due eventi viene intercettato dal service worker. Lo stesso vale anche per l'evento fetch, gestito tramite l'event handler 'onfetch'.
Il service worker è in grado di mettersi in ascolto di altri eventi, come ad esempio gli eventi 'message' che permettono al service worker di ricevere messaggi dalle pagine da lui controllate e gli eventi 'push' che permettono al server di inviare notifiche push al client, in questo testo tratteremo unicamente gli eventi install, activate, fetch in quanto sono gli unici eventi che sono stati gestiti nel nostro caso di studio di un service worker **[13]**.

12.3.5.1 GESTIONE DEGLI EVENTI INSTALL E ACTIVATE

L'evento Install e l'evento Activate diversamente dall'evento fetch ma anche tutti gli altri eventi gestibili da un service worker sono eventi del ciclo di vita del service worker, ciò significa che non sono eventi spediti dal DOM al service worker come lo sono ad esempio gli eventi funzionali fetch, message o push, ed altri, che non fanno parte del ciclo di vita del service worker.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
[8] W3C Candidate Recommendation, 19 Novembre 2019, Service Workers 1, https://www.w3.org/TR/service-workers-1/#service-worker-events)
La fase di Installazione di un service worker avviene subito dopo la fase di registrazione se la fasi di fetching e parsing dello script del service worker e degli script importati al suo interno sono avvenute correttamente in caso contrario la fase di installazione viene interrotta.

Tutte gli script del service worker o importati nel service worker scaricati nella fase di start up vengono messi in cache.
Nel nostro caso il browser deve scaricare il file 'service-worker.js' tramite l'URL fornito nella chiamata di registrazione: *navigator.serviceWorker.register('./service-worker.js')*
Il file del service worker, viene scaricato dal nodo root della nostra web app, con una fetch del browser e in seguito parsarto ed eseguito. All'interno dello script 'service-worker.js' abbiamo i nostri moduli importati con 'importScripts' che verranno chiamati tramite una serie di fetch. Anche i moduli che svolgono le operazioni in risposta agli eventi del service worker, vengono scaricati, parasati e mandati in esecuzione a turno.
Dopo la fase di fetching del service worker, questi sarà nello stato "parsed" e sarà pronto a processare gli eventi inviatogli.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
L'evento install è il primo evento ad essere inviato al service worker. Durante la fase di installazione il service worker cambia il suo stato da 'parsed' a 'installing', finché la fase di installazione non viene completata. L'evento install è anche il primo evento che dobbiamo gestire tramite l'event handler 'oninstall' del service worker, il quale riceve come argomento l'interfaccia 'InstallEvent', che rappresenta un'azione di installazione. Durante la fase di installazione dobbiamo assicurarci che non saranno spediti eventi funzionali, lo stesso vale anche per la fase di attivazione. Durante la fase di installazione il service worker non deve essere in grado di ricevere eventi funzionali dalle pagine della nostra web app. Questo deve essere fatto per assicurarci che tutte le risorse nella fase di pre-caching siano state inserite nella cache, prima di rendere attivo il service worker passando alla fase successiva 'activate'.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/# wait-until-method)
Siccome per popolare la cache si fa uso di chiamate a funzioni asincrone, per comunicare allo user agent che l'event handler oninstall, non ha ancora terminato tutte le operazioni e che il service worker non è ancora pronto a ricevere e gestire futuri eventi, si fa uso di un meccanismo, il metodo **waitUntil()**, che previene al service worker di entrare nella fase di attivazione se non ha ancora finito di popolare la cache. Il metodo **waitUntil()** permette di estendere il tempo di vita di un evento, comunicando al mittente dell'evento, cioè il browser in questo caso, che c'è ancora del lavoro da fare. Finché la promise, della nostra operazione asincrona passata a waitUntil() non si risolve con successo, il service worker non viene terminato e lo user agent attende dal service worker il via libera per inviare alti eventi.
Al termine dell'event handler install il service worker passa allo stato 'installed' e sarà in grado di gestire i successivi eventi inviati dallo user agent, nello specifico l'evento successivo è l'evento 'activate'. **[49, 50, 51]**
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
Nella nostra implementazione del service worker, per la fase di installazione abbiamo realizzato un modulo di installazione, al cui interno sono presenti delle procedure che svolgono le operazioni di precaricamento delle risorse, o precaching.
In particolare la procedura 'swInstall(Event event, boolean isPrecaching, precachingListConfig)', prende come argomenti l'evento 'install', un booleano 'isPrecaching' che determina se effettuare il pre-caching delle risorse e una lista di oggetti, che contengono i nomi degli asset da pre-caricare e il nome della cache in cui vanno copiati,

permettendoci di gestire più cache se lo vogliamo. Di default tutti gli assets statici vengono inseriti nella cache principale 'cache_MAIN'.
Inizialmente la procedura per il precaching era stata pensata per copiare in cache i file principali di Angular (main.js, polyfills.js, ecc.) che permettono alla nostra web app di poter essere eseguita.
Sfortunatamente ci siamo resi conto che ciò non era possibile in fase di precaching, in quanto in produzione i nomi dei file di Angular come main, polyfills, ecc. si presentano con un suffisso composto da caratteri alfanumerici, che ad ogni "build" del progetto viene modificato per ogni file e abbiamo dovuto gestire la cosa in modo diverso ripiegando su un algoritmo che individua le risorse solo in seguito all'installazione e attivazione del service worker.

File bundle di Angular dopo il build di progetto:
"/main.3bc4i09c810e556bbb0af.ts"
"/polyfills.2bc7i010c10e556bag0af.ts"
"/styles.3bc4i09c810e556bbbdsad.css"
"/runtime.2aa4i09c8123213bbb0af.ts"
"/vendor.3ss4i09232356bbdb0af.ts "

A seguito della fase di installazione, abbiamo la fase di attivazione del service worker.
Non appena il service worker ha terminato l'esecuzione dell'event handler 'oninstall', lo stato del service worker cambia da stato 'installing', cioè in fase di installazione, a stato 'install', cioè che la fase di installazione è appena terminata. A questo punto, quando il service worker cambia il suo stato da 'install' a 'activating', l'evento 'activate' viene spedito al service worker, che verrà gestito dall'event handler 'onactivate'.
Tipicamente in questa fase si effettua una pulitura delle cache, popolata da una versione precedente del service worker. Come per l'event handelr 'oninstall' anche in questo caso si impiega il metodo 'waitUntil' per estendere il tempo di vita dell'evento 'activate' e permetterci di gestire le operazioni richieste, finche la Promise passata al metodo waitUntil non si è risolta con successo, impedendo al browser di terminare il service worker inaspettatamente ancor prima che abbia concluso la fase di attivazione.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
Estendere il tempo di vita dell'event handler 'onactivate' permette di evitare che degli eventi funzionali vengano inviati al service worker mentre sta effettuando la rimozione delle risorse deprecate nelle cache sotto la sua origine. Questo previene che delle risorse vengano inserite quando ancora il service worker sta spazzando la memoria cache.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#wait-until-method)
Nella nostra implementazione abbiamo creato all'interno del modulo di attivazione una procedura 'swActivate(Event event, List cacheWhiteList)', con due parametri, l'evento 'activate', rappresentato da 'event', e una lista di nomi di cache, che rappresenta l'insieme delle cache del service worker che non devono essere spazzate. All'interno della funzione svolgiamo due operazioni. Una la conosciamo già, ed è l'eliminazione delle cache obsolete 'clearCacheIfDifferent()', l'altra operazione invece permette di abilitare il service worker alla ricezione degli eventi fetch lanciati dal browser, dopo che l'event handler 'onactivate' è terminato, senza dover attendere il refresh della pagina.
Tramite la chiamata al metodo 'waitUntil', comunichiamo che il browser deve attendere che la Promise delle

due funzioni ritornino con successo, prima che il service worker possa gestire eventi 'fetch' o altri eventi funzionali.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)

12.3.5.2 GESTIONE DELL'EVENTO FETCH

Conclusasi con successo la fase di attivazione, di default il service worker non inizia immediatamente a gestire gli eventi funzionali ma va in uno stato di attesa. Noi non abbiamo mantenuto questo tipo di comportamento di default, ma abbiamo abilitato l'attivazione immediata, per permettere al service worker di ricevere fin da subito eventi funzionali.
[48] (W3C Working Draft, Service Workers - W3C First Public Working Draft 08 May 2014, https://www.w3.org/TR/2014/WD-service-workers-20140508/#concepts)
Nell'attivare immediatamente il service worker non si sono notati grossi vantaggi.
Non tutte le risorse vengono inserite in cache in quanto durante il tempo di installazione del SW gli elementi che vengono fetchati non passano attraverso il SW. Quindi in ogni caso bisogna rifare una seconda chiamata per avere tutti i file principali necessari ad avere la nostra web app offline fin dal primo avvio.
Inoltre bisogna considerare che una volta installato, il SW è permanente. almeno finché non è l'utente stesso a rimuoverlo. Quindi sebbene non si riesca ad avere attivo il service worker nella prima istanza, nelle istanze successive esso sarà in grado di fare lo storing di tutte le risorse e metterle nella cache, ma mettere le risorse nella cache subito dopo l'installazione non ci fa risparmiare tempo, se messo a confronto con la durata complessiva di un service worker. Va detto che però così l'assistenza offline è fallace in quanto c'è sempre il rischio di perdere delle risorse se non si esegue il refresh della pagina specifica, nel nostro caso siccome abbiamo una Single Page Web App, risulta più difficile che delle risorse vegano perse durante la fase di inizializzazione del SW, poiché si lavora su una pagina singola.

12.3.5.2.1 MODULO FETCH

Il modulo Fetch, contiene tutte le operazioni che il service worker deve svolgere quando intercetta un evento di tipo fetch.
Quando un evento fetch request viene lanciato dal client verso il web server possiamo usare il service worker per anteporci tra i due e intercettare l'evento fetch request, garantendoci il controllo sul tipo di risposta da restituire per la request.
Per rispondere agli eventi l'API ServiceWorker fornisce l'interfaccia 'ServiceWorkerGlobalScope'. Per rispondere ad un evento fetch è stata implementata la proprietà 'onfetch', che è un gestore di eventi fetch che permette al service worker di mettersi in ascolto.
Ogni volta che si verifica un'evento fetch, l'event handler 'onfetch' viene lanciato dal 'ServiceWorkerGlobalScope'. È possibile assegnare alla proprietà 'onfetch' la propria funzione di gestione di un evento fetch specificandola all'interno del service worker. Grazie a questo meccanismo ogni fetch request fatta dal cliente al server deve passare attraverso la nostra strategia di caching **[52]**.

La prima tecnica di caching implementata per testare il service worker nella nostra applicazione è stata di tipo Cache First, come abbiamo visto questa tecnica predilige le risorse in cache rispetto a quelle caricate dal network.
Nell'implementazione di questa tecnica è però sorto un problema di memorizzazione e utilizzo di risorse deprecate se non vengono aggiornate. In quanto 'Twipping App' è soggetta a modifiche regolari dei file assets, abbiamo impiegato una tecnica più idonee per il nostro caso di studio, la tecnica Network First then Cache Update.
Questa seconda tecnica ha risposto meglio alle nostre necessità di fornire all'utente sempre le risorse più aggiornate nella cache, che verranno impiegate solo quando l'utente non è connesso. Quello che si perde è sicuramente la velocità di caricamento delle pagine in quanto la necessità di prendere le risorse dalla cache si limita solamente ad una situazione in cui il dispositivo dell'utente non è in grado di fornire una connessione al server o in un secondo caso meno ovvio, quando il server stesso è offline o non è raggiungibile dal dispositivo che è regolarmente connesso.

12.3.6 CACHING DELLE RISORSE

Nella fase di caching delle risorse, non tutte le risorse, devono essere prese dal network e inserite nella cache, ma è necessario implementare un meccanismo che permetta al service worker di identificare unicamente quelle risorse che vogliamo siano recuperate dalla memoria cache della PWA quando l'utente è offline.
Nella nostra implementazione abbiamo sviluppato due meccanismi molto diversi tra loro per come le risorse vengono recuperate dal server e identificate come idonee ad essere inserite in memoria.

CACHING CASO 1

Nel primo caso abbiamo sviluppato un algoritmo in grado di richiamare una funzione dal server, che prende come input la risposta precedentemente recuperata dal service worker dal network tramite fetch. Quindi la funzione elabora la risposta recuperando per il service worker la politica di caching più adatta per trattare la suddetta risorsa. La politica specifica se la risorsa deve essere inserita in cache o se necessita di essere aggiornata. Il corpo della nostra funzione viene determinato in fase di runtime dal service worker tramite una chiamata al server che ritorna il corpo della funzione in base alla valore della richiesta.

CACHING CASO2

Nel secondo caso che è anche quello che abbiamo deciso di inserire come strategia di base nella gestione delle risorse, andiamo impiegando un semplice stratagemma che ci permette di controllare le risorse da aggiungere in cache direttamente da lato client. Quello che facciamo è introdurre per ogni richiesta da aggiungere alla cache, un parametro chiave nell'header della request. Quando il service worker identifica la presenza di una chiave, allora la risorsa deve essere aggiunta in cache, inoltre per le richieste che usano il metodo POST, impieghiamo il parametro chiave anche come contenitore per la parola che identifica univocamente la coppia (request, response) nella cache.

CACHING FILE BUNDLE DI ANGULAR

In entrambi i casi dobbiamo gestire i file principali di Angular. In questo primo caso di studio ci affidiamo ad un semplice algoritmo di riconoscimento. Per ogni request intercettata dal service worker controlliamo se il nome del file contenuto nell'url della richiesta è compatibile con uno dei nomi dei file da noi cercati, cioè: 'main','runtime','polyfills','styles','vendor'
Poiché i file angular della nostra applicazione, vengono passati nell'url come 'http://...../main-254324252235453535.js' ed ad ogni ricompilazione del codice server cambiano suffisso si fa uso di un'espressione regolare per controllare la corrispondenza.
Questo metodo di controllo risulta però problematico, infatti i nomi main', 'runtime', 'polyfills', 'styles', 'vendor', specialmente 'main' possono essere impiegati per altri file quindi è possibile intercettare una risorsa che usa una di queste cinque parole. Sotto questo aspetto per evitare di inserire in cache le risorse indesiderate, risulta più comodo impiegare le funzionalità messe a disposizione da Angular per il service worker.

Codice

Presentiamo la funzione che controlla l'url di una richiesta ricevuta dal service worker per determinare tramite una espressione regolare se la risorsa è una file di Angular che deve essere copiato nella memoria della cache.

```
1.  /*
2.  Controlla se il file è un file bundle angular e se ha estensione js.
3.  */
4.  function isSpecialRequest(request, specialList) {
5.      let response = false;
6.      // Array 'specialList' contiene i nomi dei file bundle Angular da copiare in
7.          cache.
8.      // Controlla se gli elementi in specialList devono essere controllati e se
9.          la 'SpecialList' non è vuota.
10.     if (Array.isArray(specialList.toCache) && specialList.toCache.length > 0) {
11.         // Prende il nome dell'url della request
12.         let fileName = getFileName(request.url);
13.         let extension = 'js';
14.         // Per ogni nome in SpecialLit...
15.         for (let index in specialList.toCache) {
16.             // Crea espressione regolare che ritorna un riscontro positivo
17.             // con l'ultima parola dell'url della request se la parola inzia
18.             // con main, vendor, ecc. e ha come suffisso caratteri alfanumerici
19.             // e termina con estensione 'js'
20.             let regExp = getMatchFileNameRegularExpression(specialList.toCache[index
    ], extension);
21.             if (fileName.search(regExp) == 0) {
22.                 response = true;
23.                 break;
24.             }
25.         }
26.     }
27.     return response;
28. }
```

Figura 19 *Controlla se il file è un file bundle di Angular: main, vendor, ecc.*

Inseriamo anche le funzioni ausiliarie impiegate per spezzare la stringa dell'url e per creare l'espressione regolare.

```
1. /* Prende l'ultima parola dell'url */
2. function getFileName(url){
3.     // spezza la stringa dell'url dove incontra una backslash.
4.     let splitedUrlArray = url.split("/");
5.     // copia l'ultimo elemento (cioè il nome del file) nell'array di elementi
6.     // spezzati dall'url.
7.     let myString = splitedUrlArray.pop();
8.     return myString;
9. }
```

Figura 20 *Spezza la stringa dell'url della request e recupera l'ultima parola che è il nome della risorsa*

```
1. /*
2.    Crea l'espressione regolare che utilizziamo per controllare se nell'ultima
3.    parola dell'url è contenuto il nome di un file angular speciale
4. */
5. function getMatchFileNameRegularExpression(name, extension){
6.     let strRegExp;
7.     // Se l'estensione è stata specificata allora...
8.     if(extension != '')
9.         // parola deve inziare per 'nome', continuare con qualsiasi valore,
10.        // e terminare con stringa '.js'
11.         strRegExp = `^(${name})[^.]*\.(${extension})`;
12.    // Se l'estensione non è stata specificata allora...
13.    else
14.        // parola deve inziare per 'nome' e continuare con qualsiasi valore,
15.         strRegExp = `^(${name})[^.]*`;
16.    return new RegExp(strRegExp);
17. }
```

Figura 21 *crea l'espressione regolare*

12.3.6.1 ANALISI CACHING CASO 1: ALGORITMO DI CACHING

12.3.6.1.1 SETTARE LA FASE DI TESTING

Per testare il nostro codice è stato realizzato un semplice server. Il server si occupa di creare e ritornare una funzione al service worker, impiegata per ricavare la politica di gestione di una risorsa.
Il server si mette in ascolto su una porta da noi specificata (porta 5000), quando il service worker fa una chiamata ad un URL specifico, (http://localhost:5000/function') il server in ascolto recupera la funzione "analyze()" per la gestione delle risorse e la restituisce al service worker in formato json, contenente i parametri e il corpo della funzione.

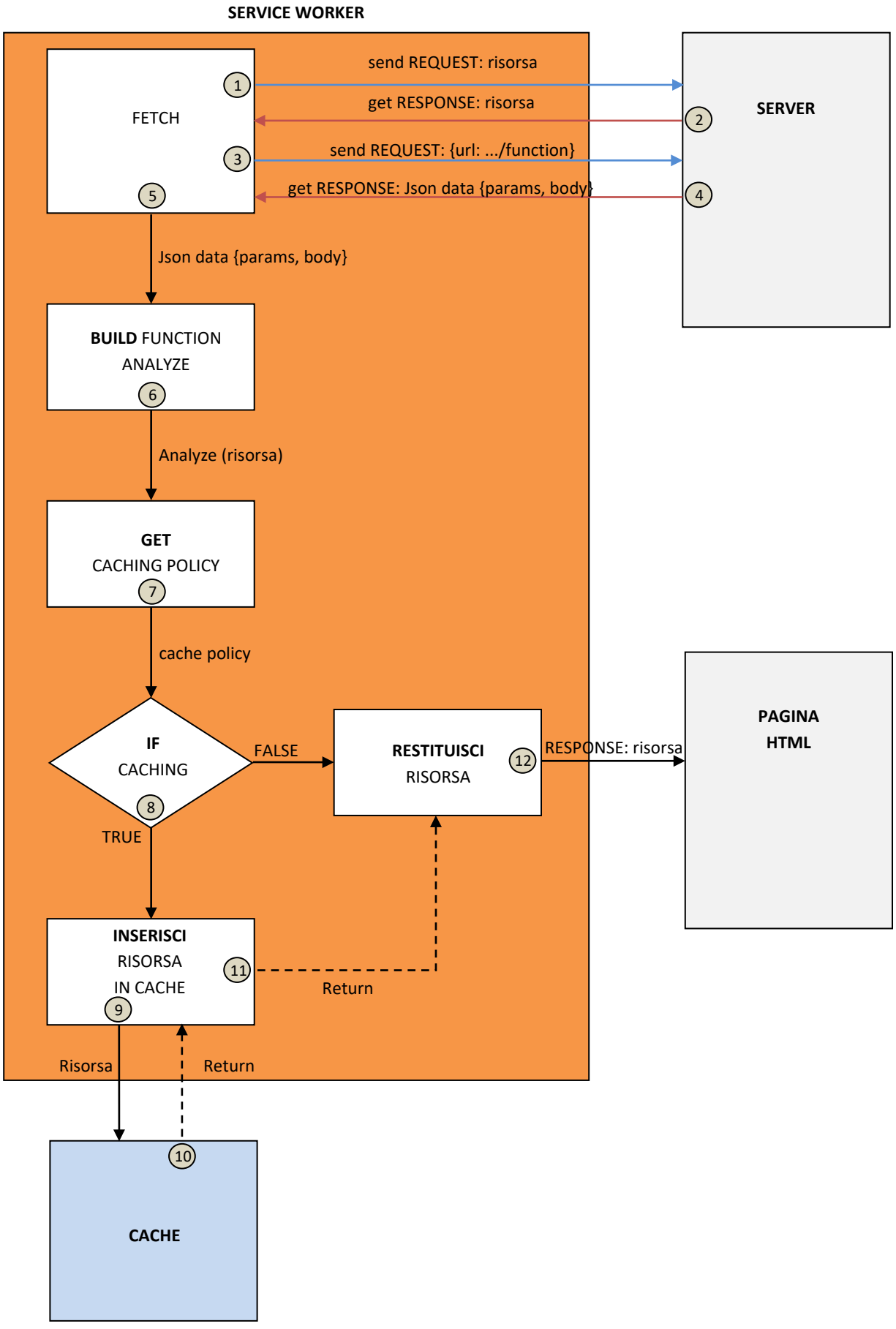
SERVICE WORKER
FETCH
1
send REQUEST: risorsa
2
get RESPONSE: risorsa
SERVER
3
send REQUEST: {url: .../function}
4
get RESPONSE: Json data {params, body}
5
Json data {params, body}
BUILD FUNCTION
ANALYZE
6
Analyze (risorsa)
GET
CACHING POLICY
7
cache policy
IF
CACHING
8
FALSE
RESTITUISCI
RISORSA
12
RESPONSE: risorsa
PAGINA
HTML
TRUE
INSERISCI
RISORSA
IN CACHE
9
11
Return
Risorsa
Return
10
CACHE

Figura 22 *Caching request con algoritmo "Analyze"*

SERVER

Riportiamo qui sotto le sezioni più interessanti del codice del nostro server.
Funzione "analyze" è la funzione ritornata dal server al service worker che verrà impiegata per determinare se una risorsa deve essere inserita in cache, o aggiornata. Attualmente nella funzione non è stato specificato alcun algoritmo per determinare quali risorse devono essere selezionate per essere copiate nella memoria e quali vengono scartate. Di default ritorna un oggetto di prova, presettato sia per la politica di cache che per l'aggiornamento delle risorse.

```
1. // Funzione analyze è la funzione che permette di decidere la logica di
2. caching per le risorse.
3. // Deve essere passata dal server al client per decidere se la risorsa
4. richiesta deve essere: inserita in cache/ fatto l'update.
5.
6. function analyze(response){
7.
8.     // corpo della funzione per determinare la politica di gestione di
9.       una risorsa deve essere implementato qui.
10.
11.          // la funzione ritorna la politica di gestione di un risorsa
12.          return {
13.              cache: true,
14.              update: false
15.          }
16.      }
```

Figura 22.1 Metodo "analyze()" da file "server.js"
Funzione "functionToObject()" viene usata per convertire la funzione in un oggetto, con un formato simile al JSON, che risulta adatto all'interscambio di dati fra applicazioni client/server.

```
1.  // Funzione che trasforma la funzione analyze in un oggetto da inviare(ritornare)
2.  // al client (service worker) nella fase di fetch.
3.  function functionToObject(f){
4.      //trasformo la funzione in una stringa.
5.      stringf = f.toString();
6.      // Spezzo la stringa e prendo i valori dei parametri e il corpo della funzione.
7.      // Il metodo map() crea un nuovo array con i risultati della chiamata di una funzione for
    nita su ogni elemento dell'array chiamante.
8.      // Il metodo trim() rimuove gli spazi bianchi da entrambi i lati di una stringa.
9.      return {
10.         params: stringf.slice(stringf.indexOf('(')+1, stringf.indexOf(')'))
11.         .split(',').map(element => {
12.             // removes whitespace from both sides of a string.
13.             return element.trim();
14.         }),
15.         body: stringf.slice(stringf.indexOf('{')+1, stringf.lastIndexOf('}')).trim()
16.     }
17. }
```

Figura 22.2 *Metodo "functionToObject()" da file "server.js"*

Il server permette di intercettare la richiesta utilizzando node.js e express.

```
1. var express = require('express');
2. // Istanza di express.
3. var app = express();
4.
5. // Il server intercetta la richiesta inviata dal service worker all' URL: "'http://l
   ocalhost:5000/function'",
6. // e ritorna al service worker l'oggetto {params: string, body: string} in formato j
   son.
7. // Definiamo il route di base con il metodo GET di richiesta HTTP.
8. app.get('/function', (req, res)=>{
9.     res.json(f2Obj(analyze));
10. })
```

Figura 22.3 *Routing di base da file "server.js"*

12.3.6.1.2 ALGORITMO DI CACHING

Dopo che il service worker, tramite una chiamata al server, ha recuperato la risposta è richiesto di far intervenire l'algoritmo di caching che stabilisce se sia necessario o meno copiare la risorsa in cache.
Mostriamo lo pseudocodice e il codice dell'algoritmo di caching.

Pseudocodice

```
1. Algoritmo di Caching (richiesta, risposta, URLrecuperoFunzione, DizionarioAggiornaRi
   sorse)
2. {
3.     Se (Non è ancora stata recuperata una funzione di caching)
4.     {
5.        Effettua una chiamata ad "URLrecuperoFunzione" per il recupero della funzione
6.        di caching dal server.
7.
8.        Ricomponi la funzione dai dati in formato JSON recuperati dalla chiamata.
9.     }
10.
11.    Passa la "risposta" alla funzione di caching per conoscere la politica di
12.    gestione della risorsa.
13.
14.    In base alla politica di gestione della risorsa:
15.        1. Inserisci la risorsa in cache
16.        2. Non inserire la risorsa in cache
17. })
```

Figura 22.4 *Pseudo codice algoritmo di caching*

Codice per il recupero della funzione di caching dal server

```
 *
     Se la funzione functionLogic non esiste
     oppure
     l'url della response non contiene la chiave di riconoscimento 'function'
     allora
     il sw fa una richiesta al server all'indirizzo
     'http://localhost:5000/function' che gli ritorna la funzione
     sottoforma di json,
     con il valore ritornato dalla chiamata inizializziamo analyze creando la
     nuova funzione.
*/
async function buildLogicConstructor (functionLogic, urlLogic, response){
    if( !functionLogic || response.url.split('/').indexOf('function')== -1){
        return await fetch(urlLogic).then(response => response.json());
    }
    return null;
}
```

Figura 22.5 *Codice per recuperare la funzione di caching dal server, da modulo "md_swCacheLogic.js"*

Codice che impiega la funzione precedente per implementare l'algoritmo di caching

```
async function logicManager(request, functionLogic, urlLogic, response){
    let funLogicManager = functionLogic;
    let policy = null;
    let logicConstructor = await buildLogicConstructor(functionLogic, urlLogic, response.clon
e());
    if( logicConstructor ){
        funLogicManager = new Function (logicConstructor.params, logicConstructor.body);
    }
    /*
        Se ad "funLogicManager" è stata assegnata la funzione di gestione della logica di
        caching (quindi non è nulla),
        allora ricaviamo la politica di gestione della risorsa contenuta in "response"
        passando la "response" alla funzione appena creata.
        NOTA: attualmente (18-10- 2019) la politica di gestione è un semplice
        oggetto di testing con i seguenti attributi:
              policy:
              {
                  cache:  boolean,    // se true la risorsa va inserita in cache.
              }
    */
    if(funLogicManager) {
        _logicManagerSettings.functionLogic = funLogicManger;
        policy = funLogicManager(response.clone());
    }
    let responseToCache = null;
    /*
        se esiste una politica di gestione della response allora:
        - se la "policy.cache" è settata a true allora la risorsa
          deve essere clonata e successivamente inserita in cache.
    */
    if(policy){
        const responseToCheck = response.clone();
        if(policy.cache){
            responseToCache = responseToCheck.clone();
        }
    }
    // Controllo se la risposta è da inserire in cache e la inserisce finalmente nella
```

```
37.         cache.
38.     if(responseToCache!=null){
39.         cacheManager(request, response.clone(), _cacheSettings.operations.addToCache);
40.     }
41.     return response;
42. }
```

Figura 22.6 *Ccodice algoritmo di caching da modulo "md_swCacheLogic.js"*

12.3.6.2 ANALISI CACHING CASO 2: ATTRIBUTO CHIAVE IN REQUEST HEADER

12.3.6.2.1 GESTIRE LE RICHIESTE CON ATTRIBUTO CHIAVE

Nel secondo caso invece di implementare una funzione che legge la risorsa e ci fornisce le informazioni utili a stabile se la risorsa deve essere copiata in memoria, introduciamo nell'header della richiesta (fig.23) il nuovo parametro "cacheKey". Se il parametro è presente quando la richiesta viene intercettata dal service worker, all'operazione di recupero della risposta dal server, segue la copia della risorsa in cache.

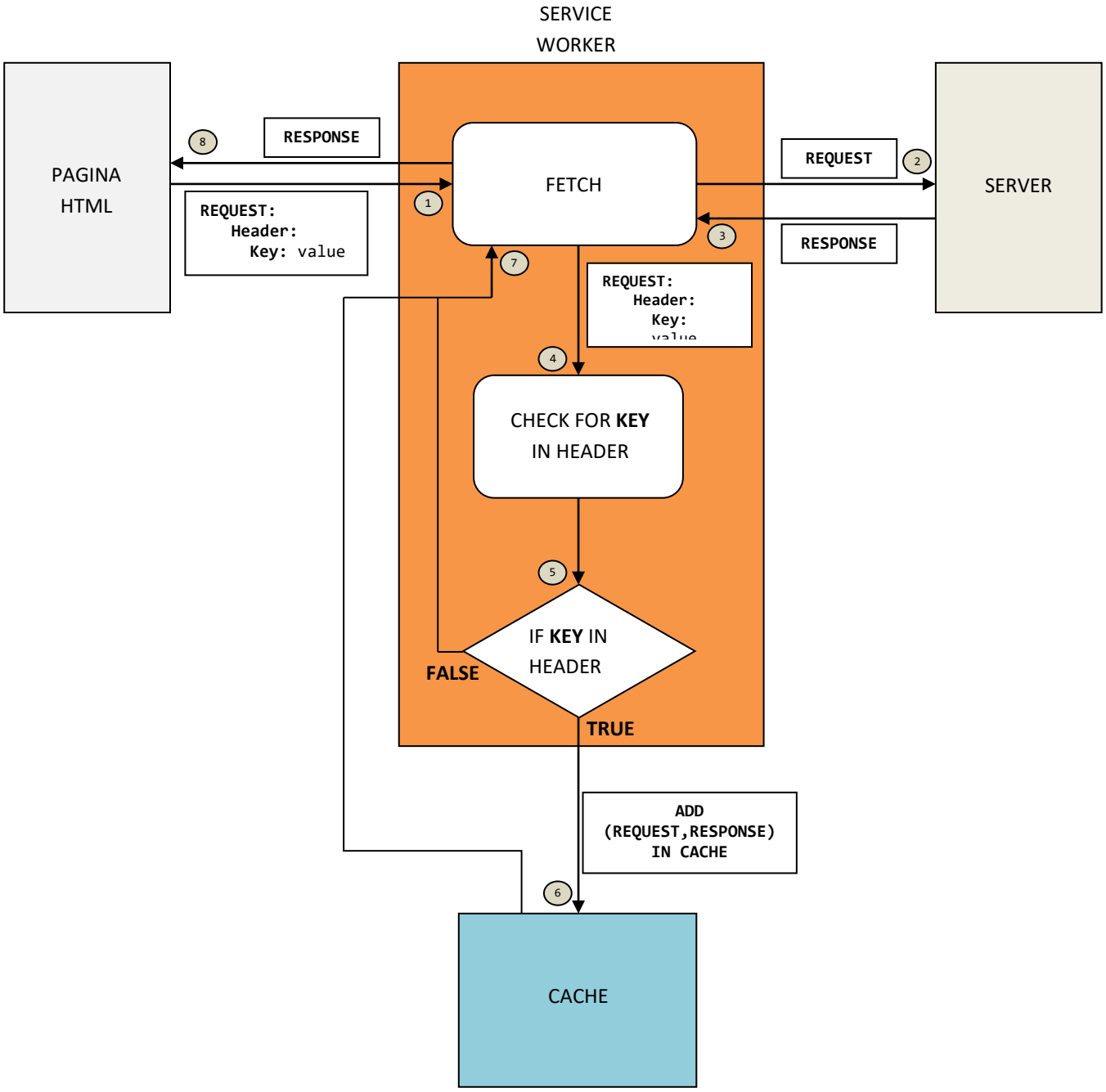

Figura 23 *Caching di una request con l'attributo chiave nell'header*

Questo metodo può tornare utile per inserire in cache tutte le risorse ad alta priorità facili da individuare, la cui permanenza nella cache è necessaria.

Pseudocodice

```
1. GESTORE DEGLI EVENTI FETCH
2. {
3.     Esegui il fetch della "richiesta"
```

```
4.
5.      Attendi una "risposta" dal server.
6.
7.      SE (fetch ha successo)
8.      {
9.          SE( Header della "richiesta" contiene il parametro "chiave" ) ALLORA
10.         {
11.             Copia la risorsa in cache
12.         }
13.     }
14. }
```

Figura 23.1 *copia in cache di una richiesta con attributo chiave*

12.3.6.2.2 GESTIRE LE RICHIESTE GET E POST CON ATTRIBUTO CHIAVE IN HEADER

Sappiamo che le risorse copiate nella memoria cache, che si presentano come una coppia (key, value), vengono identificati univocamente dal valore key (da non confondere con l'attributo chiave dell'header).
Il valore key è sempre rappresentato dall'url di una request o dall'intero oggetto request. Nel nostro caso specifico abbiamo detto che di tutte le request intercettate dal service worker, dobbiamo copiare in cache solo le richieste il cui metodo è GET o POST. Questi sono due casi che vanno analizzati separatamente per poter capire come vengono gestiti nella memorizzazione in cache.
Il metodo GET viene impiegato tipicamente per recuperare informazioni da un server tramite un URL, apponendo magari una query string all'interno dell'URL.
Come si è già detto di default l'API Cache, permette di copiare solo oggetti request con metodo GET. Quindi una risorsa che viene richiesta tramite metodo GET, se copiata in cache, è univocamente identificabile dall'oggetto request o dall'url della request che si presenta composto da due possibili parti:

- l'URL della risorsa
- una query string opzionale anche detto payload

Ad esempio un URL di una request con metodo GET si può presentare con questa struttura:

```
https://domain-name /path
```

oppure se impiega una query string con questa struttura: 'key2 = value&key1= value' , che è una query string e serve a specificare un set di dati che verranno elaborati dal server per determinare la risposta.

```
https://domain-name /path?key2 = value&key1=value1
```

come si può notare la query string è in chiaro ed è inserita nell'URL.
Un elemento costituito dalla coppia (request, response) con request con metodo GET viene copiata in cache senza problemi.
Invece gli oggetti con metodo POST non possono essere copiati in cache.
Il metodo POST viene impiegato per inviare dati al server. I dati sono codificati nel corpo della richiesta e non in una query string anteposta al nome del percorso URL come per le richieste con metodo GET e non sono accessibili dagli script Javascript.
Inoltre anche supponendo di poter accedere ai dati nel payload di una request POST, per creare una query string da anteporre all'URL della request come visto per le richieste GET, non identificherebbe univocamente

una risorsa, poiché alcuni valori contenuti nel paylod di una stessa richiesta sono dinamici.
La soluzione consiste nell'impiegare sempre l'attributo key nell'header della request, assegnandogli un valore univoco per distinguere due richieste POST diverse. Il valore key nell'header potrà essere anteposto all'url della request, come visto per la request GET:

```
https://domain-name /path?key = value
```

Questo ci permetterà di memorizzare la coppia (key, value) impiegando l'URL della request come valore chiave univoco, che ci permetterà di recuperare la risorsa dalla cache:

```
(request.url, response) = ( https://domain-namepath?key=value , resource)
```

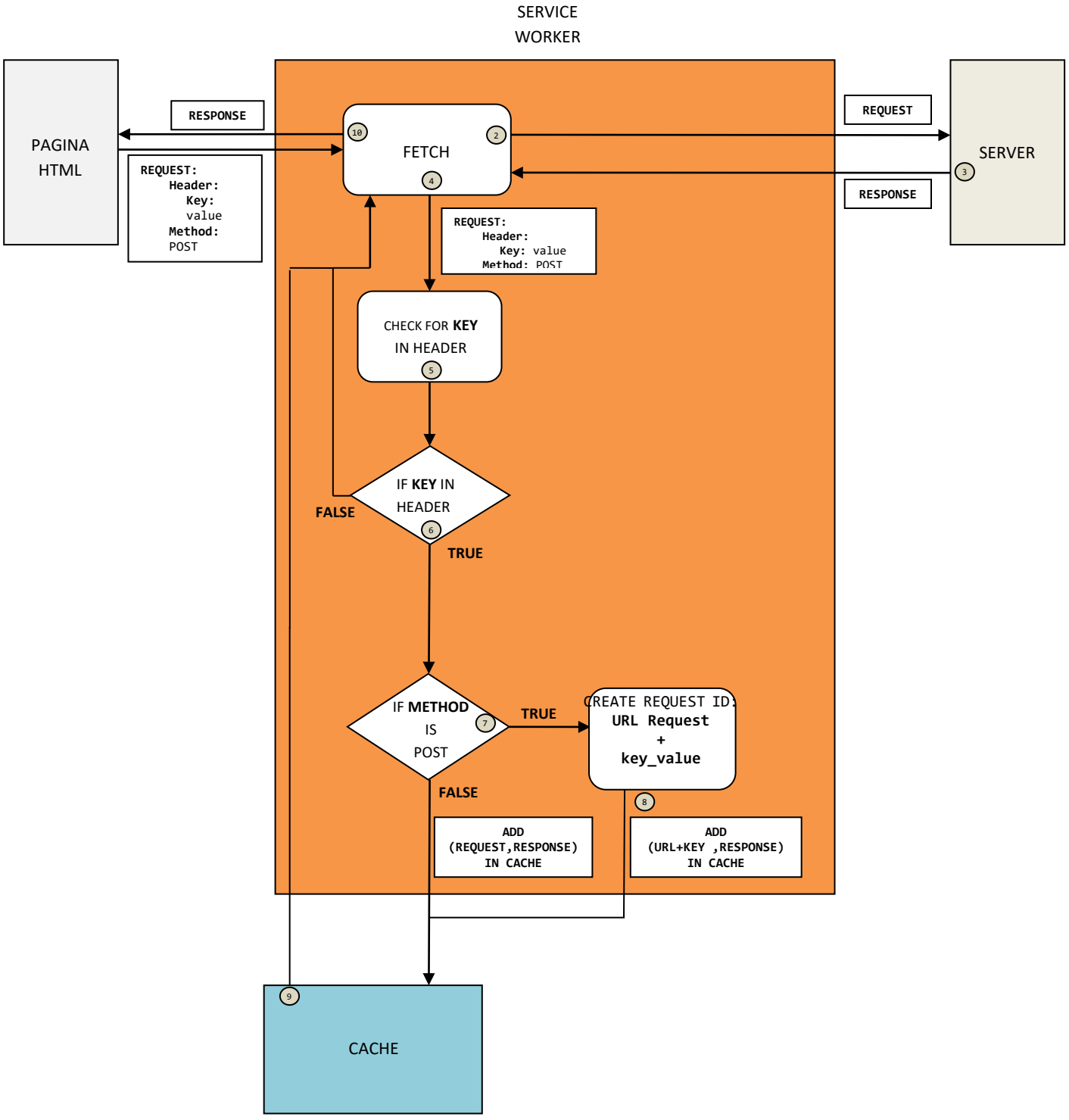

Figura 24 *Caching di una request con l'attributo chiave nell'header e con metodo POST*

Pseudocodice

Una variante all'implementazione del codice in Figura 23.1, è il seguente (Figura 24.1), in questa sezione di codice gestiamo in modo diverso una request in base al metodo POST e GET.

```
1.  // Gestire casi particolari di "richieste" con metodo POST.
2.  GESTORE DEGLI EVENTI FETCH
3.  {
4.      ...
5.      ...
6.      SE( fetch ha successo ) ALLORA
7.      {
8.          ...
9.          ...
10.         (Sulla base di uno dei due meccanismi di caching proposti)
11.         Inserire risorsa in Cache :
12.             Controlla se il metodo della richiesta è GET o POST:
13.                 SE( "richiesta" ha metodo GET )
14.                 {
15.                     Copia la risorsa ("richiesta", "risposta") in cache
16.                 }
17.                 ALTRIMENTI SE ("richiesta" ha metodo POST)
18.                 {
19.                     Rendi URL "richiesta" univoco: URL "richiesta" = URL "richiesta" + valore "chiave"
20.                     Copia la risorsa (URL "richiesta", "risposta") in cache
21.                 }
22.     }
23.     ...
24.     ...
25. }
```

Figura 24.1 copia in cache di una richiesta con attributo chiave effettuata con metodo POST

12.3.7 IMPLEMENTAZIONE DEL SERVIZIO OFFLINE

La funzione principale del nostro service worker è di mantenere il servizio attivo e funzionante della nostra web app anche a fronte dell'assenza di connettività. Per garantire questo servizio ci appoggiamo alla tecnica di caching. Le risorse che costituiscono la base della nostra web app vengono copiate nella cache e aggiornate periodicamente, nella fase di caching, e recuperate dal service worker quando si verifica una delle seguenti due condizioni:

1. il client è offline.
2. Il web server è offline

Client offline: Agli occhi del service worker il client risulta offline quando il browser rileva che l'utente non è connesso. Per controllare se la nostra applicazione è online, alcuni browser implementano una proprietà booleana "onLine" che ritorna "true" se l'applicazione è online e "false" se l'applicazione è offline.
La proprietà si aggiorna ogni volta che il browser non è più in grado di connettersi al network quando un utente segue un link o quando uno script richiede una pagina remota.
[53] (MDN web docs, Online and offline events ,
https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/Online_and_offline_events)

Web server offline: Il server risulta offline quando una fetch valida effettuata dal service worker non ottiene nessuna risposta dal server, nonostante l'utente sia connesso alla rete.

12.3.7.1 LOGICA DI IMPLEMENTAZIONE

Nella figura 25.1 viene riportato lo pseudocodice che presenta i diversi passaggi da seguire per l'implementazione offline.
Siccome seguiamo una politica di caching "**Network first and Cache Update"** daremo priorità al network, cercando sempre di reperire la risorsa prima dal network. Alternativamente per rendere la PWA più veloce si potrebbe sia effettuare una chiamata al server ma anche una alla cache. La chiamata può essere effettuata in contemporanea o solo se il server è lento a rispondere alle richieste del client.
In ogni caso è necessario controllare prima della chiamata al network se il browser connesso alla rete.
Tramite le API del browser verifichiamo se lo stato è connesso, si presentano due casi.

Primo caso:
Se il browser è connesso, il service worker invia la richiesta del client al server e il server ritorna una risorsa, se la risorsa è presente in cache allora va aggiornata, altrimenti se risorsa non è in cache, allora si determina se la risorsa deve essere copiata tramite la politica di caching. Una volta terminata la fase di caching, il service worker restituisce la risorsa più recente al client.

Secondo caso:
Se il browser non è connesso oppure la chiamata fetch fallisce, deve essere attivato il servizio offline per l'applicazione. Nel servizio offline viene controllata la cache per vedere se la risorsa è stata copiata in cache precedentemente, se la risorsa viene trovata allora viene anche restituita come risposta al client, altrimenti viene restituita una risposta nulla.

Pseudocodice

```
1.  Offline Service(richiesta)
2.  {
3.      SE(client è online)
4.      {
5.          Effettua la fetch della richiesta
6.          SE(fetch ha successo cioè il server è online)
7.          {
8.              SE(risorsa in cache)
9.                  Aggiorna risorsa in cache
10.             ALTRIMENTI SE(politica di caching)
11.                 Inserisci nuova risorsa in cache
12.             ...
13.             ...
14.             Ritorna risposta al client
15.         }
16.         ALTRIMENTI
17.         {
18.             Attiva servizio offline
19.         }
20.     }
```

```
    ALTIRMENTI
    {
        Attiva servizio offline
    }

    SE(servizio offline attivo)
    {
        Controlla cache:
            SE(risorsa è in cache)
            {
                Ritorna risorsa al client
            }
            ALTIMENTI
            {
                Ritorna risposta nulla
            }
    }
}
```

Figura 25.1 *Implementazione del servizio offline per una PWA*

Codice

```
Offline service con strategia di caching "Network First than Cache Update"

Funzione principale.

--------------------------------------------------------------------------------
```

```
function swFetch(event){
    if('caches' in self)
    {
        // controlla se il client è online
        if(navigator.onLine)
        {
            // SE il client è online...
            // effettua la fetch della richiesta (network first)
            event.respondWith(fetch(event.request).then((fetchResponse) => {
                // Controlla se il server ha risposto
                if(fetchResponse.clone()) {
                    // Se la risposta dal server non è nulla...
                    // controlla se la risorsa è in cache
                    // aggiorna la risorsa in cache.
                    // inserisci la risposta nella cache.
                    cachingStrategy(chooseStrategy, mainCache, request, fetchResource.clone())
                    return fetchResponse;
                }
                else
                {
                    // servizio offline
                    // controlla se risorsa in cache e ritornala
                    return takeCacheResource(mainCache, request)
                }
            }).catch(error){
                // Gestisci le request che causano un
                // errore con il metodo fetch.
                // restituisci una risposta nulla.
                let response = null;
                console.error('errore fetch: ', error, request);
                let init = { "status" : 200 , "statusText" : "I am a custom service worker response!" };
                return new Response(null, init);
            });
        }
```

```
        else
        {
            //servizio offline
            // apri la cache in scrittura e lettura
            event.respondWith(caches.open(mainCache).then((cache) => {
                // controlla se la risorsa è in cache
                return cache.match(getRequestToMatch(event.request)).then((cacheResponse) => {
                    //restituisci la risorsa altrimenti restituisci un valore null
                    return cacheResponse;
                })
            }));
        }
    }
    else
    {
        console.error("browser doesn't support cache API");
    }
}

```

Funzione che determina la strategia di caching da adottare tra due possibili scelte.

```
function cachingStrategy(chooseStrategy, mainCache, request, fetchResource){
    switch(chooseStrategy)
    {
        case 'KEY':
            // controlla se la request ha un attributo chiave
            if(isRequestKey(request))
            {
                // aggiorna risorsa in cache
                UpdateCacheResource(mainCache, request, fetchResource.clone());
            }
            // controlla se la request è un bundle javascript
            else if(isSpecialRequest(request, _fetchSetting.specialList))
            {
                // aggiorna bundle in cache
                UpdateCacheBundle(mainCache, request, fetchResource.clone())
            }
            break;

        case 'LOGIC_MANAGER':
            ...
            ...
            break;
    }
}

```

Controlla se nell'header della richiesta c'è un valore chiave.

Se presente la risorsa deve essere messa in cache.

```
function isRequestKey(request){
    return request.headers.get(_fetchSetting.cacheKey)
}

```

Aggiorna la risorsa file in cache se presente eliminando quella vecchia.

Altrimenti la copia semplicemente in cache

```
function updateCacheResource(mainCache, request, fetchResource)
{
    // apri la cache
    return caches.open(mainCache).then((cache) => {
        // controlla se la risorsa è in cache
        return cache.match(getRequestToMatch(event.request)).then((cacheResponse) => {
            if (cacheResponse) {
                console.log("HIT: ", event.request.url);
                // rimuovi vecchia risorsa
                cache.delete(request);
                // aggiungi risorsa aggiornata in cache
                cache.add(request, fetchResource);
            }
        });
    });
}

```

Aggiorna la risorsa bundle in cache se è diversa ed esiste già.

Se la risorsa bundle non esiste in cache si limita ad aggiungerla.

```
function updateCacheBundle(mainCache, request, fetchBoundle)
{
    // apri la cache
    return caches.open(mainCache).then((cache) => {
        // controlla se la risorsa è in cache
        return cache.match(getRequestToMatch(event.request)).then((cacheResponse) => {
            if (cacheResponse) {
                console.log("HIT: ", event.request.url);
                // non va aggiornato il file bundle in quanto due versioni di
                // un file bundle hanno nomi differenti.
                // rimuovi vecchia risorsa
            }
            else {
                // dobbiamo cercare se in memoria è presente un file bundle
                // prendi nomeBundle per esteso request es. 'main-31231231rrdsda121.bundle.js'
                let fileName = getFileName(request.url);
                // prendo il nomeBundle ridotto es. 'main'
                let bundleName = getBundleName(fileName);
                // cerca il bundleName es. 'main' nella cache
                // se lo trova elimina il file e inserisce la nuova risorsa.
                removeBundleInCache(cache, request, fetchBundle.clone());
            }
        });
    });
}


```

Rimuove la veccia risorsa bundle in cache e la sostituisce con la nuova versione

---

```
function removeBundleInCache(cache, request, bundleName, fetchBundle){
        // controlla se la risorsa bundle è in cache
        return cache.keys().then((cacheKeys) => {
            if (cacheKeys) {
                for(let index in cacheKeys)
                {
                    // guarda se il bundleName è presente nell'url di ogni cacheKeys[index]
                    let keyName = getFileName(cacheKeys[index]);
                    // crea l'espressione regolare
                    let regExp = getMatchFileNameRegularExpression(bundleName, '');
                    if(keyName.search(regExp) == 0) {
                        // se il nome della chiave nella cache coincide con il nomeBundle allora...
                        // rimuovi la vecchia coppia (request, respone) dalla cache
                        cache.delete(cacheKeys[index]);
```

```
                        // aggiungi la nuova coppia (request, fetchBundle)
                        cache.add(request, fetchBundle.clone());
                        // esci dal ciclo for
                        break;
                    }
                }
                return cacheResponse;
            }
        });
    });
}


Prende la risorsa dalla cache e la restituisce all'app web


--------------------------------------------------------------------------------


function takeCacheResource(mainCache, request)
{
    return caches.open(mainCache).then((cache) => {
        // controlla se la risorsa è in cache
        return cache.match(getRequestToMatch(event.request)).then((cacheResponse) => {
            return cacheResponse;
        });
    });
}


Se la request usa il metodo POST trasforma la request per essere inserita in cache


------------------------------------------------------------------------------------------


function getRequestToMatch(request){
    return request.method === 'POST'? makePOSTrequest(request): request;

}


Crea una request di tipo POST da inserire nella cache


--------------------------------------------------------------------------


function makePOSTrequest(request) {
    let key = request.headers.get(_fetchSetting.cacheKey);
    if ( key && key !== '' && !isSpecialRequest(request, _fetchSetting.specialList)){
        return request.url + '?' + key;
    }
    return request.url;
}


Controlla se la risposta è valida


-----------------------------------------


function isResponseValid(response){
    return response && isResponseStatus(response.status) && isResponseType(response.type);
}
function isResponseStatus(status){
    // API Cache accetta solo risposte con status 2xx range
    return status >= 200 && status < 300 ;
}
function isResponseType(type){
    return type !== 'error';
}
```

Figura 25.2 *Servizio offline con strategia di caching "Network First than Cache Update"*

12.4 SECONDO CASO DI STUDIO - GESTIONE DEI FILE BUNDLE DI ANGULAR

Questo secondo caso di studio differisce da quello precedente solo per la fase di registrazione e memorizzazione dei file bundle di Angular 8. Infatti implementiamo il service worker del nostro progetto appoggiandoci al pacchetto Angular Service Worker, che genera e configura per noi un service worker in grado di lavorare a stretto contatto con gli altri file di configurazione di Angular e memorizzare in cache per noi i file bundle di Angular(Figura # CASO 2).

Per realizzare questo service worker impieghiamo sempre il service worker visto nel nostro primo caso di studio, il file 'service-worker.js', che chiamiamo custom service worker,a cui aggiungiamo il nuovo codice javascript generato da Angular 8. Infatti tralasciando i file di configurazione di Angular service worker, il service worker generato in Angular è semplicemente un file javascript che può essere copiato all'interno del nostro custom service worker. Copiando il codice nel nostro service worker e modificando i file di configurazione di Angular service worker affinchè puntino al nostro file 'service-worker.js' per la registrazione, possiamo ancora impiegare la strategia di caching da noi implementata e avvantaggiarci dei benefici che offre un service worker gestito da Angular 8.

Con il service worker di Angular 8 non è più necessario inserire il codice javascript 'service-worker-registration.js' nella pagina 'index.html' dell'applicazione per permettere l'avvio della fase di registrazione del service worker quando un utente visita o effettua il refresh del documento.
Inoltre la gestione dei file bundle main.js, vendor.js, polyfills.js, styles.js non richiederanno più di essere gestiti tramite un algoritmo nella fase di fetching delle richiesta, che sia in grado di individuare la risorsa quando viene richiesta dall'applicazione web, per poi essere copiata in cache. Infatti la copia dei file citati in cache avviene nella fase di precaching cioè durante l'installazione del service worker e viene gestita tramite i file di configurazione del service worker di Angular.

12.4.1 SCRIPT ANGULAR SERVICE WORKER

Il service worker generato da Angular non differisce molto dalla nostra implementazione del service worker.
Esattamente come per il nostro custom service worker, anche il file javascript service worker generato da Angular gestisce gli eventi impiegando i gestori di eventi già visti in precedenza. Per questo è possibile modificare, con una certa accortezza, il contenuto del file agendo esclusivamente sul codice che si occupa di mettere il service worker in ascolto degli eventi 'install', 'activate', 'fetch'. Nella personalizzazione del file, andremo unicamente a modificare la funzione che si occupa di gestire gli eventi fetch, importando parte della nostra logica di caching all'interno della funzione di gestione di un evento fetch.
Vediamo ora quali sono i passaggi seguiti per implementare il nostro Angular service worker.

12.4.2 ANGULAR SERVICE WORKER IMPLEMENTAZIONE

Il pacchetto di Angular service worker può essere aggiunto alla nostra cartella di progetto sempre tramite la CLI di Angular tramite una semplice linea di comando.
Comando in CLI per aggiungere il service worker ad un progetto già esistente:

```
ng add @angular/pwa --project *project-name*
```

[29] (Angular, Getting started with service workers,
https://angular.io/guide/service-worker-getting-started#adding-a-service-worker-to-your-project)
Tramite il comando vengono svolte le seguenti azioni:

1. il pacchetto '@angular/service worker' viene aggiunto al progetto
2. abilita il supporto Angular built-in nella CLI per il service worker
3. crea il file di configurazione del service worker 'ngsw-config.json' che specifica quali file e dati URLs devono essere copiati in cache dal service worker e come devono essere aggiornati.
4. importa e registra il service worker nell'AppModule e setta nel file di configurazione di angular 'angular.json' il service worker come abilitato per il progetto.
5. Il file 'index.html' viene aggiornato includendo un link al file manifest.json che fornisce metadati riguardanti la web app, utilizzati per l'installare dell'applicazione web sul dispositivo.

[29] (Angular, Getting started with service workers,
https://angular.io/guide/service-worker-getting-started#adding-a-service-worker-to-your-project)
[54] (Angular, Service worker configuration,
https://angular.io/guide/service-worker-config#service-worker-configuration)
Una volta che è stato scaricato il pacchetto service worker e che i file di Angular 'app-module.ts', 'angular.json' sono stati configurati automaticamenti, e il file di configurazione del service worker 'ngsw-config.json' è stato creato, per creare anche il file javascript del service worker, cioè 'ngsw-worker.js' è necessario assemblare il service worker in produzione, in quanto l'Angular service worker può essere testato solo in produzione e non in sviluppo.
Attraverso i l comando assembliamo la nostra web app per la produzione:

```
ng build --prod
```

Nella cartella di build di Angular, abbiamo tutti i file e cartelle che possono essere ospitati sul server e tra questi file grazie al fatto che abbiamo installato il pacchetto del service worker verrano automaticamente creati anche il file service worker 'ngsw-worker.js' e il file di configurazione a runtime del service worker 'ngsw.js' che è stato creato sulla base del file di configurazione 'ngsw-config.js'.
Vediamo quindi come vengono configurati i file del progetto quando creiamo il service worker con Angular.

ANGULAR.JSON

Il file 'angular.json' si trova nel livello radice del nostro workspace e fornisce una configurazione di default per gli strumenti di sviluppo forniti da Angular CLI per il progetto.
[55] (Angular, Angular Workspace Configuration,
https://angular.io/guide/workspace-config#angular-workspace-configuration)
Nel file di configurazione della CLI 'angular.json' viene settato il flag 'service worker' a 'true' che abilita il supporto al service worker nella CLI e viene anche specificato il file di configurazione del service worker.

```
1.  "apps": [
2.      {
3.        "root": "src",
4.        "outDir": "dist",
5.        "assets": [
6.          ...
7.        ],
8.        "index": "index.html",
9.        "main": "main.ts",
10.       "polyfills": "polyfills.ts",
11.       "test": "test.ts",
12.       "tsconfig": "tsconfig.app.json",
13.       "testTsconfig": "tsconfig.spec.json",
14.       "prefix": "app",
15.       "styles": [
16.         "styles.css"
17.       ],
18.       ...
19.       ...
20.       },
21.       "serviceWorker": true,
22.       "ngswConfigPath": "ngsw-config.json"
23.     }
24.   ]
```

Figura 27 *File di configurazione 'angular.json' che abilita il service worker nella CLI build*

Impostando il service worker a "true" nel file di configurazione di 'angular.json', nella fase di build del progetto vengono creati i due file javascript 'ngsw-worker.js' e 'ngsw-config.json'.

12.4.3 APP MODULE E REGISTRAZIONE DEL SERVICE WOKER

Ogni applicazione creata con Angular ha almeno un modulo. Il modulo radice usato per lanciare l'applicazione è chiamato 'AppModule' ed è rappresentato dal file 'app.module.ts' che descrive come le parti di un'applicazione devono essere messe assieme. All'interno di questo file abbiamo indicato un decorator '@NgModule' che identifica AppModule come una classe modulo. Tutte le librerie e i moduli in Angular sono indicati come classi 'NgModules'. La classe '@NgModule' dice ad Angular come compilare e lanciare l'applicazione.

[56] (Angular, NgModules, https://angular.io/guide/ngmodules#angular-modularity)

[57] (Angular, Bootstrapping, https://angular.io/guide/bootstrapping#prerequisites)

```
src/app/app.modules.ts

----------------------------------------------------------------

1. import { NgModule }      from '@angular/core';
```

```
2. import { BrowserModule } from '@angular/platform-browser';
3. import { AppComponent }  from './app.component';
4.
5. @NgModule({
6.   imports:      [ BrowserModule ],
7.   declarations: [ AppComponent ],
8.   bootstrap:    [ AppComponent ]
9. })
10.      export class AppModule { }
```

Figura 28 *File App Module*

Di rilevante importanza è l'imports array. Nell'imports array vengono inseriti i moduli di cui l'applicazione necessita in esecuzione, ed è tramite l'import dell'AppModule che registriamo il nostro Angular service worker nel browser (Figura 29).

```
src/app/app.modules.ts

------------------------------------------------------


1.  @NgModule({
2.
3.    imports: [
4.      BrowserModule,
5.      AppRoutingModule,
6.      ServiceWorkerModule.register('/ngsw-worker.js', { enabled: environment.production })
7.    ],
8.    declarations: [AppComponent],
9.    bootstrap: [AppComponent]
10. })
11. export class AppModule {}
```

Figura 29 *Configurazione di AppModule per la registrare del service worker*

Come si vede il codice 'ServiceWorkerModule.register() registra il 'file 'ngsw-worker.js' nel browser. Impiegando il modulo del service worker 'ServiceWorkerModule' e il metodo 'register()' viene richiama la funzione 'navigator.serviceWorker.register()' utilizzata in precedenza anche da noi nel nostro file di registrazione del service worker.

12.4.3.1 REGISTRARE UN SERVICE WORKER PERSONALIZZATO

Per registrare il nostro service worker con Angular dobbiamo prima inserire il nostro file 'service-worker.js', cioè il service worker personalizzato, nel root del progetto, 'src/service-worker.js'. Per copiare il file nel root del progetto possiamo procedere come indicato nel primo caso di studio, l'importante è che nel file di

configurazione di 'angular.json' il file sia specificato alla voce 'assets' (Figura 30) per includere il file nel build del progetto.

angular.json

```
-------------------------------------------------------

1.  "apps": [
2.      {
3.        "root": "src",
4.        "outDir": "dist",
5.        "assets": [
6.        ...
7.        "src/service-worker.js"
8.        ],
9.        "index": "index.html",
10.       "main": "main.ts",
11.       "polyfills": "polyfills.ts",
12.       "test": "test.ts",
13.       "tsconfig": "tsconfig.app.json",
14.       "testTsconfig": "tsconfig.spec.json",
15.       "prefix": "app",
16.       "styles": [
17.         "styles.css"
18.       ],
19.       ...
20.       ...
21.       },
22.       "serviceWorker": true
23.     }
24.   ]
```

Figura 30 *Il file service worker viene registrato come una risorsa asset di Angular nel file ' angular.json'*

Importiamo quindi il codice javascript di Angular Service Worker 'ngsw-worker.js' nel nostro file 'service-worker.js', come abbiamo fatto con i moduli del service worker precedentemente esposti.

```
1.  service-worker.js
2.  ------------------------------------------------------------------------
3.      // moduli da caricare nello script principale del service worker.
4.      const _modules = {
5.          ngsw: './ngsw-worker.js',
6.          install: '/assets/js/service-worker-modules/md_swInstall.js',
7.          activate: '/assets/js/service-worker-modules/md_swActivate.js',
8.          fetch: '/assets/js/service-worker-modules/md_swFetch.js',
9.          notification: '/assets/js/service-worker-modules/md_swNotification.js',
10.     };
11.
12.     // * Importa codice del Service Worker di Angular nel nostro service worker * //
```

```
13.     improtScripts(_modules.ngsw);
14.     // * Gestione Altri Eventi * //
15.
```

Figura 31 *Importare il codice di Angular service worker 'ngsw-worker.js' nel nostro service worker file*

In AppModule dobbiamo sostituire la registrazione del service worker 'ngsw-worker.js' con il percorso del nostro codice javascript 'service-worker.js' (Figura 32).

```
src/app/app.modules.ts

-----------------------------------------------------

12. @NgModule({
13.
14.   imports: [
15.     BrowserModule,
16.     AppRoutingModule,
17.     ServiceWorkerModule.register('/service-worker.js',{enabled:environment.production})
18.   ],
19.   declarations: [AppComponent],
20.   bootstrap: [AppComponent]
21. })
22. export class AppModule {}
```

Figura32 *Registrazione del service worker personalizzato 'service-worker.js' in AppModule*

Con l'ultimo passo il nostro service worker verrà caricato sul browser quando l'utente utilizza la nostra applicazione web e potremmo ancora usufruire dei vantaggi di un Angular service worker, e allo stesso tempo avere la libertà di personalizzare la gestione delle risorse in cache e della memoria della PWA.

12.4.4 PRECACHING DEI FILE BUNDLES CON 'ngsw-config.json' e 'ngsw.js'

Con angular service worker l'inserimento dei file bundle in memoria nella fase di preload delle risorse avviene tramite i file di configurazione a runtime del service worker 'ngsw.js'. La gestione dei file javascript bundle sarà automatizzata da Angular che creerà per noi i file di configurazione a runtime del Service Worker Angular con i file bundle già taggati per essere inseriti in cache quando il service worker viene registrazione dalla pagina principale 'index.html' della nostra PWA.
Il file di configuarazione 'ngsw.config.json' impiegato dall'Angular service worker ci permette di impostare le risorse da inserire in cache, nella così detta fase di precaching e inoltre consente di configurare altre opzioni del service worker molto utili che però non tratteremo. Come per il primo caso di studio anche Angular Service Worker inserisce le risorse nella Cache Storage impiegando la Cache API.
Quando il ServiceWorkerModule chiama l'istruzione 'navigation.serviceWorker.registration()' indirettamente aziona anche il caricamento delle risorse specificate.

ngsw-config.json

```
{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app-shell",
      "installMode": "prefetch",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/index.html",
          "/pwa-manifest.json",
          "/assets/images/favicons/favicon.ico",
          "/assets/js/*.js",
          "/*.css",
          "/*.js",
          "!/*-es5*.js"
        ],
        "urls": [
          "https://fonts.googleapis.com/**",
          "https://fonts.gstatic.com/s/**"
        ]
      }
    },
    ...
    ...
}
```

Figura 33 *File 'ngsw.config.json'*

```
interface AssetGroup {
  name: string; // identifica il gruppo di asset
  installMode: 'prefetch' | 'lazy'; // politica di caching inziale
  updateMode: 'prefetch' | 'lazy'; // politica di caching per update
  resources: {  // risorse da copiare in cache
    files: string[]; // file di progetto
    urls: string[]; // risorse URLs
  };
}
```

Figura 34 *Struttura di un AssetGroup*

Il file 'ngsw-config.json' contiene la voce 'assetGroups' un array che definisce un set di risorse e come vengono copiate in cache 'installMode', che specifica se le risorse devono essere copiate nella fase di prefetch (precaching) o solo quando viene fatta una richiesta esplicita della risorsa da parte dell'applicazione web.
[54] (Angular, Service worker configuration, https://angular.io/guide/service-worker-config#assetgroups)

[54] (Angular, Service worker configuration, https://angular.io/guide/service-worker-config#installmode)

Quando viene effettuato il build della nostra applicazione tramite comando della CLI:

```
ng build --prod
```

L'applicazione viene assemblata e all'interno della cartella 'dist/assets' vengono generati i file del nostro progetto, tra questi c'è anche il file 'ngsw.json' questo file è un file di configurazione runtime assemblato sulla base del file di configurazione 'ngsw-config.js' e contiene le informazioni necessarie al service worker per memorizzare i files in cache a runtime. All'interno di questo file vengono specificati anche i nomi dei file javascript bundle che vengono copiati in cache insieme al file 'index.html' e gli altri file indicati dallo sviluppatore.

```
1. ngsw.json
2. ----------------------------------------------------
3. {
4.   "configVersion": 1,
5.   "index": "/index.html",
6.   "assetGroups": [
7.     {
8.       "name": "app",
9.       "installMode": "prefetch",
10.            "updateMode": "prefetch",
11.            "urls": [
12.              "/favicon.ico",
13.              "/index.html",
14.              "/inline.5646543f86fbfdc19b11.bundle.js",
15.              "/main.3bb4e08c826e33bb0fca.bundle.js",
16.              "/polyfills.55440df0c9305462dd41.bundle.js",
17.              "/styles.1862c2c45c11dc3dbcf3.bundle.css"
18.            ],
19.            "patterns": []
20.          },
21.          {
22.            "name": "assets",
23.            "installMode": "lazy",
24.            "updateMode": "prefetch",
25.            "urls": [],
26.            "patterns": []
27.          }
28.        ],
29.        "dataGroups": [],
30.        "hashTable": {
31.          "/inline.5646543f86fbfdc19b11.bundle.js": "1028ce05cb8393bd5370606
   4e3a8dc8f646c8039",
32.          "/main.3bb4e08c826e33bb0fca.bundle.js": "ae15cc3875440d0185b46b4b7
   3bfa731961872e0",
33.          "/polyfills.55440df0c9305462dd41.bundle.js": "c3b13e2980f9515f4726
   fd2621440bd7068baa3b",
34.          "/styles.1862c2c45c11dc3dbcf3.bundle.css": "3318b88e1200c77a5ff691
   c03ca5d5682a19b196",
35.          "/favicon.ico": "84161b857f5c547e3699ddfbffc6d8d737542e01",
```

```
36.           "/index.html": "cfdca0ab1cec8379bbbf8ce4af2eaa295a3f3827"
37.         }
38.       }
```

Figura 35 *File 'ngsw.json' contenente i file bundle da inserire nella memorica cache*

I file bundle vengono inseriti in cache nella fase di prefatch come specificato dal valore dell'attributo 'installMode' in figura 35. Prefetch dice all'Angular Service Worker di fare il fetch di ogni singola risorsa nella lista anche se non è stata ricevuta alcuna richiesta per la risorsa.
Ogni volta che viene fatto il production build dell'applicazione il contenuto del file 'ngsw.json' viene aggiornato con i nuovi nomi dei file javascript bundle, diversamente dal primo caso di studio in cui era richiesta un'espressione regolare per gestire questa situazione.

12.4.5 MODIFICARE IL FILE 'ngsw-worker.js'

Per modificare il contenuto del service worker di Angular 'ngsw-worker.js' e adattarlo alle nostre esigenze di caching dobbiamo modificare i gestori a eventi nel nostro caso specifico il gestore degli eventi fetch.

ngsw-worker.js

```
1.  .
2.  .
3.  .
4.  // The install event is triggered when the service worker is first installed.
5.  this.scope.addEventListener('install', (event) => {
6.      // SW code updates are separate from application updates, so code updates are
7.      // almost as straightforward as restarting the SW. Because of this, it's always
8.      // safe to skip waiting until application tabs are closed, and activate the new
9.      // SW version immediately.
10.     event.waitUntil(this.scope.skipWaiting());
11. });
12.
13. // The activate event is triggered when this version of the service worker is
14. // first activated.
15. this.scope.addEventListener('activate', (event) => {
16.
17. // Handle the fetch, message, and push events inside Angualr Service Worker.
18. this.scope.addEventListener('fetch', (event) => this.onFetch(event));
19. this.scope.addEventListener('message', (event) => this.onMessage(event));
20. this.scope.addEventListener('push', (event) => this.onPush(event));
21. this.scope.addEventListener('notificationclick', (event) => this.onClick(event));
```

Figura 36 Gestori di eventi in angular service worker script 'ngsw-worker.js'.

Come si può vedere il gestore dell'evento fetch richiama una funzione 'onFetch(event)' che possiamo modificare. All'interno del corpo della funzione 'onFetch()' Angular gestisce tutte le request che vengono fatte dall'applicazione web al server. Una possibile modifica può consistere nel lasciare che Angular gestisca solo le risorse specificate nel file 'ngsw-config.js' mentre le risorse che vogliamo gestire personalmente senza

interferenze da parte di Angular, possono essere identificabile dall'attributo chiave già impiegato nel caso precedente.

Pseudocodice

```
funzione onFetch()

-----------------------------------------------------------------------
```

```
function onFetch(evento)
{
    var richiesta

    SE (richiesta CONTIENTE l'attributo chiave)
    {
        questa richiesta me la gestisco io
    }
    ELSE
    {
        questa richiesta la gestisce la logica dell'Angular Service Worker
    }
}
```

Figura 37 *Pseudocodice funzione 'onFetch()' determina se gestire la richiesta manualmente o tramite AngularSW*

Codice

```
funzione onFetch()

-----------------------------------------------------------------------
```

```
/**
* The handler for fetch events.
*
* This is the transition point between the synchronous event handler and the
* asynchronous execution that eventually resolves for respondWith() and waitUntil().
*/
onFetch(event) {
    const req = event.request;
    const scopeUrl = this.scope.registration.scope;
    const requestUrlObj = this.adapter.parseUrl(req.url, scopeUrl);
    // Se nell'header della request è presente l'attributo chiave gestisci
    // la fetch manualmente
    if(req.headers.get('key'))
    {
        // Crea il mio spazio di memoria in cache
        // Implementa la mia logica di caching
    }
    // Altrimenti angular service worker gestisce fetch in automatico
    else
    {
        // Esegue il codice di default di OnFetch()
        ...
        ...
        ...
```

```
25.     }
26. }
```

Figura 38 *Codice funzione 'onFetch()' determina se gestire la richiesta manualmente o tramite AngularSW*

12.5 MANUAL SERVICE WORKER vs ANGULAR EXTENDED SERVICE WORKER

Nell'implementazione dei due casi di studio abbiamo tratto le seguenti conclusioni:

Manual service worker
Implementare un service worker che gestisca i file in una applicazione sviluppata in Angular 8, senza l'ausilio di del pacchetto service worker di Angular, introduce un livello computazionale aggiuntivo per la gestione dei file bundle di Angular. Per ogni chiamata della web app al server è richiesto un tempo di elaborazione della richiesta per determinare la sua natura. Inoltre nella nostra implementazione facciamo uso di una semplice espressione regolare per identificare i file bundle di Angular, che però presenta diverse incertezze. Infatti se si dovesse presentare una situazione in cui più file condividono lo stesso nome dei file bundle, rischiamo che l'espressione regole, trovi un match anche con delle risorse che non desideriamo che vengano messe in cache.

Angular extended service worker
Il service worker di Angular si è rivelato molto utile per gestire i file bundle, ma può essere impiegato anche per gestire tutti i file del progetto senza costi computazionali aggiuntivi. Il problema di impiegare un Angular service worker è che se dobbiamo estenderlo siamo vincolati a lavorare in un ambiente non familiare. Infatti dobbiamo apportare delle modifiche all'interno del codice contenuto nel file 'ngsw-worker.js' senza però sconvolgerne il funzionamento di base. Questo è un rischio in quanto, il service woker di Angular risulta per noi un'incognita, non avendo una documentazione da parte di Angular che ne approfondisce il contenuto, se non se ne conosce il funzionamento e si vogliono apportare modifiche c'è il rischio di invalidarne il funzionamento. Nel nostro caso è stato facile modificare il service worker in quanto ci bastava verificare una condizione all'interno di una struttura condizionale, ma per service worker che richidono una complessità maggiore potrebbe risultare problematico mescolare la propria logica di implementazione con quella del service worker di Angular. Inoltre il problema più grosso rimane il fatto che siccome andiamo ad inserire del codice all'interno di un file che viene ricreato ogni volta che viene effettuato il 'build' del progetto rischiamo di perdere tutto il lavoro fatto al suo interno. A questo problema esistono diverse soluzioni che però non affronteremo in questo caso di studio ma che sicuramente richiedono una maggiore ricerca.

12.6 IMPLEMENTAZIONE MODULO DI GESTIONE DELLO SPAZIO DI MEMORIA

Il modulo di gestione dello spazio di memoria occupato dalla nostra PWA in cache, svolgere le seguenti operazioni, che sono utili per controllare lo stato della nostra memoria in cache.

12.6.1 CHECK SPACE AND ADD RESOURCE

Il modulo ci permette di controlla con una funzione se lo spazio a disposizione nella cache è sufficiente a memorizzare una nuova risorsa. Nell'eventualità che la risorsa non possa essere aggiunta in cache, possiamo

consultare un algoritmo di gestione delle risorse che determina se la risorsa da aggiungere ha una priorità alta, magari attraverso una segnalazione diretta da parte nostra con l'introduzione di un parametro "priorità" nell'header della request e quali risorse che già popolano la cache possono essere sostituite, quindi rimosse dalla cache, attraverso un controllo di tutti gli elementi che la popolano. Sarebbe conveniente anche separare le risorse con alta priorità da quelle con bassa priorità in due sezioni di memoria, tramite la creazione di cache multiple, per consentire una ricerca più rapida e senza la necessità di passare tutte le risorse in cache anche quelle ad alta priorità.
Funzione che **CONTROLLA** se lo spazio disponibile in cache **PER LA RISORSA** è sufficiente:

Pseudocodice

```
SpazioDisponibilePerRisorsa(cache, risorsa, dimensioni risorsa)
{
    Raccogli le informazioni sulla memoria cache:
        - spazio allocato a PWA
        - spazio occupato

    Calcola spazio libero: spazio tot. - spazio occupato

    SE (dimensioni risorsa MINORE di spazio libero) ALLORA
        Prova a liberare spazio in memoria cache
    SE (spazio in memoria cache non è stato liberato)
        ritorna false (risorsa non deve essere inserita in cache)
    ALTRIMENTI
        ritorna true (risorsa può essere insertia in cache)
}
```

Figura 39 *Pseudocodice che definisce la struttura del codice che determina lo spazio disponibile per una risorsa da inserire in cache*

Codice

```
/* ------------------------------------------------------------------*/
/* Check Cache Free Space for the resource that must be added in cache.
/* ------------------------------------------------------------------*/
function isResourceFreeSpaceAvailable(cacheName, amountOfSpaceNeeded){
  let putInCache = false;
  // Prendi informazioni memoria cache
  let storageInfo = getPWAStorageInfo();
  // SE attributi usage e quota diversi da NaN.
  if(isStorageInfoDefined(storageInfo)){
    // SE c'è spazio per la risorsa...
    if(!(putInCache = hasFreeSpace(storageInfo, amountOfSpaceNeeded))){
      // ...libera spazio dalla cache se possibile (priorità, tempo, utilizzo).
      let hasFreeUpSpace = freeUpSpace(cacheName);
      // SE non riesco a liberare spazio per la risorsa...
      if(!hasFreeUpSpace)
        putInCache = false;
      else
        putInCache = true;
    }
```

```
20.   }
21.   else {
22.     // inserisci sempre la risorsa in cache anche se non conosciamo
23.     // le dimensioni dello spazio libero
24.     putInCache = true;
25.   }
26.   return putInCache;
27. }
```

Figura 40 *Codice principale che controlla lo spazio libero della PWA per la risorsa*

Funzioni ausiliare di supporto per determinare se una risorsa ha abbastanza spazio in cache per poter essere copiata.
La prima funzione **CONTROLLA** se c'è spazio libero in cache

```
1. /* ------------------------------------------------------*/
2. /* Has Free Space. Controlla se c'è spazio libero in cache.
3. /* ------------------------------------------------------*/
4. function hasFreeSpace(storageInfo, amountOfSpaceNeeded) {
5.   // prendi spazio libero nella PWA
6.   let freeSpace = getFreeSpace(storageInfo);
7.   // controlla se lo spazio libero è sufficiente
8.   return freeSpace >= amountOfSpaceNeeded;
9. }
```

La seconda funzione ausiliaria **CALCOLA** lo spazio libero in cache.

```
1. /* ------------------------------------------------------*/
2. /* Get Free Space. Cacola lo spazio libero.
3. /* ------------------------------------------------------*/
4. function getFreeSpace(storageInfo) {
5.   // spazio complessivo PWA - spazio usato PWA
6.   return storageInfo.quota - storageInfo.usage;
7. }
```

Figura 41 Funzioni ausiliare per il calcolo dello spazio libero in cache per la risorsa

Funzione che **AGGIUNGE** la risorsa in cache dopo aver richiamato la funzione che controlla se c'è disponibilità di spazio in cache.

Pseudocodice

```
1. AggiungiRisorsaInCache(cache, risorsa, dimensioni risorsa){
2.
3.     SE (Spazio disponibile per la risorsa) ALLORA
4.         inserisce risorsa in cache
5. }
```

Codice

```
/* ------------------------------------------------------------------------------*/
/* Add New Resource To Cache add a resource in cache if there is free space available.
/* ------------------------------------------------------------------------------*/
async function addNewResourceToCache(cacheName, resource, resourceSizeBytes){
  // SE c'è spazio per la risorsa in cache...
  if(isResourceFreeSpaceAvailable(cacheName, resource, resourceSizeBytes)){
    await caches.open(cacheName).then(function(cache) {
      // ...aggiunge la risorsa in cache.
      cache.add(resource);
    });
  }
  else{
    // SE non c'è spazio per la risorsa in cache...
    console.log(" risorsa non è stata inserita in cache, non c'è spazio nella cache");
  }
}
```

Figura 42 Codice che inserisce la risorsa in cache dopo aver controllato se c'è spazio libero in cache

12.6.2 LIBERA SPAZIO IN CACHE

Questa funzione permette di rimuovere gli assets da una cache.

Pseudocodice

```
Libera Spazio(cache, risorsa, dimensioni risorsa)
{
  Racupera tutte le risorse copiate in cache

  Confronta priorità risorsa da copiare con priorità delle altre risorse

  SE (priorità risorsa ALTA) ALLORA
      SE(risorse a BASSA priorità in cache)
         Rimuovi dalla cache le risorse a bassa priorità più vecchie e con un basso utilizzo.
      ALTRIMENTI
         Rimuovi dalla cache le risorse ad alta priorità più vecchie e con un basso utilizzo.
  }
```

Figura 43 Pseudocodice che descrive la struttura del codice che libera lo spazio in cache

12.6.3 PROGRESSIVE WEB APPLICATION STORAGE INFO

Informazioni e dettagli della memoria allocata alla PWA

Funzioni che controllano la quantità di spazio correntemente riservata per l'origine.
Le seguenti funzioni permettono di determinare le informazioni sulla memoria di una PWA indicati in tabella 9:

Attributo	Unità	Descrizione
percentage	% | NaN	percentuale di spazio usato

usage	bytes | NaN	spazio utilizzato da PWA in bytes
quota	bytes | NaN	spazio allocato dal browser alla PWA in bytes

Tabella 9 *Descrizione informazioni sulla memoria allocata alla PWA*

La funzione principale che ritorna i tre attributi in tabella 9. Impiega la funzione "storageEstimateWrapper" e la funzione "getMibStorage".

Codice recupera Info memoria PWA

```
1.  // * Get Progressive Web Application Storage Info * //
2.  /* ----------------------------------------------------------------------------------------------
3.    Funzione che ottiene le info. riguardanti lo spazio utilizzato dall'applicazione web.
4.    Pecentuale di spazio occupato,
5.  Spazio occupato in MB,
6.  Spazio riservato dal browser all'origine(PWA)
7.   ----------------------------------------------------------------------------------------------*/
8.  function getPWAStorageInfo(){
9.    // ritorna la stima dello spazio utilizzato e lo spazio totale a disposizione della PWA.
10.   storageEstimateWrapper().then(function(PWAstorageEstimation){
11.     // trasformo i bytes di spazio occupato e spazio totale libero in Megabytes
12.     //e ottengo la percentuale di spazio occupata dall'applicazione.
13.     const PWAstorageInfo = getMibStorage(PWAstorageEstimation.usage, PWAstorageEstimation.quota);
14.     return PWAstorageInfo;
15.   });
16. }
```

Figura 44 *Funzione che raccoglie le informazioni dello stato della memoria della PWA*

Funzione che seleziona una delle API per stimare le informazioni dello spazio di memoria e la utilizza per ritornare la stima dello spazio utilizzato e lo spazio totale messo a disposizione dal browser per la PWA.

Codice Stima memoria

```
1.  // * Seleziona l'API e ritorna le informazioni di stima della memoria * //
2.  async function storageEstimateWrapper(){
3.    // Se il browser corrente supporta l'API Cache Storage esegui questo
4.    if(hasCacheAPI())
5.      return await estimateCacheStorage('CacheAPI');
6.    // Se il browser non supporta Cache Storage e Storage Quota
7.    // utilizziamo due caratteristiche più vecchie ma che garantisce compatibilità.
8.    if(hasWebKitAPI())
9.      return await estimateCacheStorage('WebKitAPI');
10.   // se il browser supporta l'API Storage Quota ma non Cache Storage usiamo questo
11.   if(hasStorageQuotaAPI())
12.     return await estimateCacheStorage('StorageQuotaAPI');
13.   // If we can't estimate the values, return a Promise that resolves with NaN.
14.   return await estimateCacheStorage('');
15. }
```

Figura 45 *Codice che seleziona l'API per la stima della memoria*

Funzione che controlla se almeno una delle API Cache, WebKit o StorageQuota API per i browser che non supportano API Cache, sono disponibili per essere impiegate nel calcolo delle informazioni della memoria della PWA.

Codice controlla compatibilità browser con API

```
1.  // * Check Browser API Compatibility * //
2.  /* ---------------------------------------------------
3.    Funzione che controlla se il Browser supporta le API:
4.    CacheAPI, WebKitAPI, StorageQuotaAPI nel seguente ordine
5.   ---------------------------------------------------*/
6.  function hasCacheAPI(){
7.    return  'storage' in navigator &&
8.            'estimate' in navigator.storage;
9.  }
10. function hasWebKitAPI(){
11.   return  'webkitTemporaryStorage' in navigator &&
12.           'queryUsageAndQuota' in navigator.webkitTemporaryStorage
13. }
14. function hasStorageQuotaAPI(){
15.   return  'storageQuota' in navigator &&
16.     'queryInfo' in navigator.storageQuota;
17. }
```

Figura 46 *Codice che controlla se il browser è compatibile con le API per la stima della memoria*

Funzione che utilizza la funzione API Compatibility per calcolare la stima dello spazzio utilizzato in cache e dello spazio allocato all'origine della PWA, attraverso tre API differenti.

Codice che costruisce una stima di utilizzo momoria e quota della PWA

```
1.  // * Stima della memoria * //
2.  /* ---------------------------------------------------------------------------------
3.    Ritorna un valore "usage" e un valore "quota"
4.    "usage":
5.      reflects how many bytes a given origin is effectively using for same-origin data.
6.    "quota":
7.      reflects the amount of space currently reserved for an origin by your browser.
8.      The space that the browser devote to your web app's origin will likely change.
9.   ---------------------------------------------------------------------------------*/
10. async function estimateCacheStorage(chooseAPI){
11.   // Sceglie l'API da utilizzare tra API cache, webkit, storageQuota
12.   switch(chooseAPI){
13.     // usa API cache
14.     case "CacheAPI":
15.       console.log("WW-STORAGE-MANAGEMENT: browser supporta 'storage estimate'");
16.       return await navigator.storage.estimate().then(await function(estimate){
17.         console.log(estimate.usage, estimate.quota);
18.         // ritorna stima
19.         return estimate;
20.       });
21.       break;
22.     // usa API webKit se API cache non è supportata
23.     case "WebKitAPI":
```

```
      console.log("WW-STORAGE-MANAGEMENT: browser non supporta 'storage estimate' uso 'webkit TemporaryStorage'");
      // Return a promise-based wrapper that will follow the expected interface.
      // ritorna stima
      return new Promise(function(resolve, reject) {
        navigator.webkitTemporaryStorage.queryUsageAndQuota(
          function(usage, quota) {resolve({usage: usage, quota: quota})},
          reject
        );
      });
      break;
    // usa API storage quota se API cache e webKit non sono supportate
    case "StorageQuotaAPI":
      console.log("WW-STORAGE-MANAGEMENT: browser non supporta 'storage estimate' uso 'storage quota'");
      // ritorna stima
      return navigator.storageQuota.queryInfo("persistent");
      break;
    // se browser non supporta nessuna delle tre API,
    // stima info memoria sono indefinite
    default:
      console.log("browser non supporta nulla");
      // ritorna stima
      return Promise.resolve({usage: NaN, quota: NaN});
  }
}
```

Figura 47 *Codice che stima la quantità di memoria utilizzate e la quantità di spazio allocato alla PWA*

Codice che crea oggetto Info della memoria della PWA

```
/* ----------------------------------------------------------------------
  funzioni:
  1. Crea oggetto PWA Storage Info.
     Ritorna un'oggetto che contiene pecentuale di spazio occupato,
     spazio occupato in bytes dall'origine, spazio riservato dal browser
     all'origine(PWA) in bytes).
  2. Ottieni pecentuale di spazio occupato.
  3. Trasformo i bytes in Megabytes.
 ----------------------------------------------------------------------*/
function createPWAStorageInfo(t_usage, t_quota){
  return {
    percentage: getStoragePercent(t_usage,t_quota),
    usage_bytes: t_usage,
    quota_bytes: t_quota,
  };
}
// calcola la percentuale di dati utilizzati dall'origine.
function getStoragePercent(t_usage, t_quota){
  return Math.round(t_usage / t_quota * 100);
}
// trasforma i dati da bytes in megabytes.
function bytesToMib(t_bytes){
  return Math.round(t_bytes / (1024 * 1024));
}
```

Figura 48 *Codice che crea l'oggetto contenente le info sulla memoria*

12.6.4 SPAZIO OCCUPATO DALLE RISORSE

Abbiamo visto che nel memorizzare i file bundle di Angular main, vendor, polyfills, styles e runtime abbiamo raggiunto una soglia di spazio impiegato che si aggirava appena sotto i 10 Megabytes. Se includiamo invece anche le altre risorse della nostra applicazione web siamo arrivati all'incirca sui 90-100 Megabytes di spazio occupato in cache. Guardando ai valori massimi di spazio di allocazione permesso dai browser alle PWA, non riscontriamo alcun problema con i browser come Chrome, Opera, FireFox e Safari Desktop, invece risulta superare la soglia massima consentita dal browser iOS Safari, che mette a disposizione della PWA fino ad un massimo di 50 MB.
Possiamo notare come i file bundle siano light-weight, il più pesante è il file main che raggiunge i 3 Megabytes, perché contiene praticamente il codice di tutta la nostra applicazione. Una possibile soluzione sarebbe che Angular 8 o versioni successive, per favorire le PWA, implementino un sistema che spezzetta i bundle in pacchetti più leggeri richiamati dalla PWA solo quando richiesti. Ovviamente questo richiede una maggiore complessità nella gestione dei file bundle da parte del custom service worker, per questo converrebbe utilizzare l'Angular service worker, che dovrebbe essere capace di gestire la memorizzazione dei pacchetti automaticamente, inoltre in questa soluzione viene sollevato il problema di garantire un servizio offline completo e non solo parziale, in quanto la memorizzazione parziale dei file bundle che permettono di far funzionare la nostra applicazione può minare l'esperienza utente che dovrebbe richiamare quella di una applicazione nativa.
Nel complessivo si è però visto che tutti gli altri browser non presentano un grosso ostacolo nella quantità di risorse e di spazio messo a disposizione della nostra PWA.

13 CONCLUSIONI

Abbiamo visto come le PWA sono un'evoluzione delle semplici web app e di come migliorino quest'ultime dando accesso a quelle risorse hardware che gli permettono di implementare una gamma più ampia di servizi che agiscono in background rispetto all'utente e che rendono le web app più simili alle app native. Come confermato da molte ricerche, il passaggio da Web App a PWA sembra conveniente specialmente data l'ampia diffusione dei dispositivi mobili che sono il target a cui mirano le PWA. Abbiamo visto come quasi tutti i browser più famosi implementano un supporto ai service worker. In futuro ci potrebbero essere molte più applicazioni web che implementano la tecnologia service worker poiché il passaggio sembra necessario se si vuole accedere a maggiori funzionalità di un dispositivo. In particolare abbiamo studiato una caratteristica precisa delle PWA che è il servizio offline. Abbiamo visto come il servizio offline possa essere implementato tramite i service worker e di come effettivamente velocizza il processo di recupero delle risorse prendendole dalla cache e consegnandole subitaneamente all'utente. La memorizzare del contenuto di una web app in cache permette di ridurre effettivamente il traffico dati, concentrando gli sforzi dell'applicazione unicamente per il recupero delle risorse dinamiche e non quelle che risultano statiche o che vengono aggiornate di rado. Abbiamo visto le diverse strategie di caching e di nostro abbiamo implementato la strategia 'network first and cache update', una strategia che semplicemente recupera le risorse dalla cache se viene a mancare il servizio. Non ci è chiaro se questa strategia sia la strada più corretta da seguire per la nostra web app in quanto non sappiamo quanto la feature del sevizio offline venga realmente utilizzata dall'utente, poiché il servizio si attiva

unicamente quando si verifica un errore nel recupero della risorsa o quando non c'è connettività. È richiesto un maggiore studio sulle strategie di caching e le loro combinazioni.

In particolare abbiamo cercato di offrire un servizio offline in una app web realizzata con Angular attraverso due approcci diversi. Abbiamo visto che in entrambi gli approcci dobbiamo copiare i file bundle di Angular in cache che sono fondamentali per far funzionare la nostra applicazione anche offline. Si è visto però che la gestione manuale di questi file è particolarmente onerosa. Inoltre si è visto come implementare un service worker direttamente in Angular non solo rende più facile la gestione dei file bundle, ma fornisce maggiore controllo sull'applicazione e futuri aggiornamenti e anche una più vasta gamma di funzionalità già implementate che funzionano internamente ad Angular. In aggiunta abbiamo visto che è possibile estendere il service worker di Angular, ma si è anche notato come l'aggiunta di codice personalizzato al service worker realizzato da Angualr può portare a problemi di consistenza.

Invece a livello cache abbiamo visto come è possibile controllare lo spazio di memoria occupato dalle risorse. Abbiamo visto che generalmente quasi tutti i browser più popolari non presentano limiti di caching per le risorse della nostra PWA, eccetto iOS Safari. Nonostante lo spazio di memoria non fosse un problema, abbiamo notato che sebbene i file bundle di Angular non occupano molta memoria in cache, le risorse secondarie memorizzate della nostra applicazione richiedevano una quantità di memoria al di sopra delle nostre previsioni che si aggirava intorno a qualche MB. Invece è risultato che sono stati occupati più di 50MB. In un contesto desktop questo risultato può essere trascurato, ma per i dispositivi mobili che presentano risorse limitate, la memorizzazione di centinaia di Megabytes in cache non sembra ottimale. Ulteriori ricerche devono essere eseguite sulle memorie dei dispositivi mobili.

Bibliografia

[1] Sayali Sunil Tandel , Abhishek Jamadar. Impact of Progressive Web Apps on Web App Development. In International Journal of Innovative Research in Science, Engineering and Technology (A High Impact Factor, Monthly, Peer Reviewed Journal), vol. 7, issue 9, September 2018.
[2] William Jobe. Native Apps vs. Mobile Web Apps. iJIM – Volume 7, Issue 4, October 2013.
[3] Steiner, Thomas. (2018). What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. 789-796. 10.1145/3184558.3188742.
[4] Jesse James Garrett. (2007). Ajax: A new approach to web applications.
[5] MDN web docs. Tecnologie web per sviluppatori, Guida Web Developer, AJAX.(2008)
https://developer.mozilla.org/it/docs/Web/Guide/AJAX. visitato in data 01-01-2020.
[6] Codelabs. Your First Progressive Web App, Introduction.
https://codelabs.developers.google.com/codelabs/your-first-pwapp/#0 .visitato in data 01-01-2020.
[7] Chromium Blog. Chrome 40 Beta: Powerful Offline and Lightspeed Loading with Service Workers. (Thursday, December 4, 2014). visitato in data 01-01-2020
[8] W3C Candidate Recommendation, 19 Novembre 2019, Service Workers 1.
https://www.w3.org/TR/service-workers-1/ . visitato in data 01-01-2020
[9] Github.com. The World Wide Web Consortium (W3C). (3 Nov 2017).
https://github.com/w3c/ServiceWorker/blob/master/explainer.md#service-workers-explained .visitato in data 01-01-2020.
[10] MDN web docs. Tecnologie web per sviluppatori, Web APIs, Service Worker API.(Oct 25, 2019)
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API .visitato in data 01-01-2020.
[11] Web Hypertext Application Technology Working Group. HTML Living Standard. Web Workers. Chp.10 (19 December 2019). visitato in data 01-01-2020.
[12] Can I use?. Service Workers. https://caniuse.com/#feat=serviceworkers .visitato in data 01-01-2020.
[13] MDN web docs. Tecnologie web per sviluppatori, Web APIs, ServiceWorkerGlobalScope.(Aug 13, 2019)
https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerGlobalScope .visitato in data 01-01-2020.
[14] Matt Gaunt. Web Fundamentals. Service Worekers: an Introducution.
https://developers.google.com/web/fundamentals/primers/service-workers .visitato in data 01-01-2020.
[15] MDN web docs. Tecnologie web per sviluppatori, Web APIs, XMLHttpRequest.(Dec 12, 2019)
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest .visitato in data 01-01-2020.
[16] Web Fundamentals. Progressive Web Apps Training. Introduction to Service Worker.
https://developers.google.com/web/ilt/pwa/introduction-to-service-worker .visitato in data 01-01-2020.
[17] MDN web docs. Tecnologie web per sviluppatori, Web APIs, Web Storage API.(Aug 1, 2019)
https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API .visitato in data 01-01-2020.
[18] Lee, Jiyeon & Kim, Hayeon & Park, Junghwan & Shin, Insik & Son, Sooel. (2018). CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications. 1731-1746. 10.1145/3243734.3243867.
[19] The Chromium Projects. Service Worker Security FAQ.
https://dev.chromium.org/Home/chromium-security/security-faq/service-worker-security-faq .visitato in data 01-01-2020.

[20] MDN web docs. Tecnologie web per sviluppatori, Web APIs, Service Worker API, Using Service Workers. (Nov 12, 2019).
https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers . visitato in data 01-01-2020.
[21] MDN web docs. Tecnologie web per sviluppatori, Web APIs, Web Workers API, Using Web Workers. (Jun 30, 2019). https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers . visitato in data 01-01-2020.
[22] Dean Alan Hume. Progressive Web Apps. chp. 1. ISBN: 9781617294587
[23] Jake Archibald. Web Fundamentals. Guide. The Service Worker Lifecycle.
https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle . visitato in data 01-01-2020.
[24] MDN web docs. Learn web development, JavaScript,Client-side web APIs, Fetching data from the server. (Nov 22, 2019).
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data . visitato in data 01-01-2020.
[25] MDN web docs. Web technology for developers, Web APIs, Fetch API, Using Fetch. (Dec 27, 2019).
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch . visitato in data 01-01-2020.
[26] Angular. Workspace and project file structure. https://angular.io/guide/file-structure . visitato in data 01-01-2020.
[27] Angular. Glossary. https://angular.io/guide/glossary#glossary . visitato in data 01-01-2020.
[28] Angular. Service workers in Angular. https://angular.io/guide/service-worker-intro . visitato in data 01-01-2020.
[29] Angular. Getting started with service workers. https://angular.io/guide/service-worker-getting-started . visitato in data 01-01-2020.
[30] MDN web docs. Web technology for developers, Web APIs, Cache. (Aug 14, 2019).
https://developer.mozilla.org/en-US/docs/Web/API/Cache. visitato in data 01-01-2020.
[31] MDN web docs. Web technology for developers, Web APIs, Cache, Cache.add(). (Aug 14, 2019).
https://developer.mozilla.org/en-US/docs/Web/API/Cache/add. visitato in data 01-01-2020.
[32] MDN web docs. Web technology for developers, Web APIs, Cache, Cache.put(). (Aug 14, 2019).
https://developer.mozilla.org/en-US/docs/Web/API/Cache/put. visitato in data 01-01-2020.
[33] Mozilla. The Service Worker Cookbook. Cache only. https://serviceworke.rs/ caching-strategies.html . visitato in data 01-01-2020.
[34] Mozilla. The Service Worker Cookbook. Cache only. https://serviceworke.rs/strategy-cache-only.html . visitato in data 01-01-2020.
[35] Mozilla. The Service Worker Cookbook. Cache only. https://serviceworke.rs/
strategy-network-or-cache.html . visitato in data 01-01-2020.
[36] Mozilla. The Service Worker Cookbook. Cache only. https://serviceworke.rs/
strategy-cache-update-and-refresh.html . visitato in data 01-01-2020.
[37]] Ilya Grigorik. Web Fundamentals. Guides. HTTP Caching.
https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching . visitato in data 01-01-2020.
[38] Angular.Http Client. https://angular.io/guide/http . visitato in data 01-01-2020.

[39] MDN web docs. Web technology for developers, HTTP, HTTP caching. (Mar 21, 2019). https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching . visitato in data 01-01-2020.
[40] W3C Working Group Note. Quota Management API. (23 May 2016). https://www.w3.org/TR/quota-api/ . visitato in data 01-01-2020.
[41] MDN web docs. Web technology for developers, Web APIs, StorageManager. (Mar 18, 2019). https://developer.mozilla.org/en-US/docs/Web/API/StorageManager .visitato in data 01-01-2020.
[42] Eiji Kitamura. HTML5 Rocks. Working with quota on mobile browsers - A research report on browser storage. (January 28th, 2014). https://www.html5rocks.com/en/tutorials/offline/quota-research/ . visitato in data 01-01-2020.
[43] Addy Osama, Marc Cohen. Web Fundamentals. Guides.Offline Storage for Progressive Web Apps. https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa . visitato in data 01-01-2020.
[44] Web Fundamentals. Progressive Web Apps Training. Live Data in the Service Worker. https://developers.google.com/web/ilt/pwa/live-data-in-the-service-worker . visitato in data 01-01-2020
[45] MDN web docs. Tecnologie web per sviluppatori, Web APIs, WorkerGlobalScope. importScripts()(May 7, 2019) https://developer.mozilla.org/en-US/docs/Web/API/WorkerGlobalScope/importScripts . visitato in data 01-01-2020.
[46] Jeff Posnick . Web Fundamentals. Tweaks to cache.addAll() and importScripts() coming in Chrome 71. https://developers.google.com/web/updates/2018/10/tweaks-to-addAll-importScripts . visitato in data 01-01-2020 .
[47] Jeff Posnick . Web Fundamentals. Fresher service workers, by default. https://developers.google.com/web/updates/2019/09/fresher-sw . visitato in data 01-01-2020 .
[48] W3C Working Draft. Service Workers - W3C First Public Working Draft 08 May 2014. https://www.w3.org/TR/2014/WD-service-workers-20140508/ . visitato in data 01-01-2020 .
[49] MDN web docs. Web technology for developers, Web APIs, InstallEvent. (Aug 13, 2019). https://developer.mozilla.org/en-US/docs/Web/API/InstallEvent . visitato in data 01-01-2020.
[50] MDN web docs. Web technology for developers, Web APIs, ExtendableEnvet, waitUnitl() . (Dec 8, 2019). https://developer.mozilla.org/en-US/docs/Web/API/ExtendableEvent/waitUntil . visitato in data 01-01-2020.
[51] MDN web docs. Web technology for developers, Web APIs, ServiceWorkerGlobalScope, oninstall() . (Aug 24, 2019). https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerGlobalScope/oninstall . visitato in data 01-01-2020.
[52] MDN web docs. Web technology for developers, Web APIs, ServiceWorkerGlobalScope, onfetch() . (Mar 23, 2019). https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerGlobalScope/onfetch . visitato in data 01-01-2020.
[53] MDN web docs. Web technology for developers, Web APIs, NavigatorOnLine, Online and offline events . (Mar 23, 2019). https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/Online_and_offline_events . visitato in data 01-01-2020.
[54] Angular. Service worker configuration. https://angular.io/guide/service-worker-config . visitato in data 01-01-2020 .
[55] Angular. Angular Workspace Configuration. https://angular.io/guide/workspace-config . visitato in data 01-01-2020 .
[56] Angular. NgModules. https://angular.io/guide/ngmodules . visitato in data 01-01-2020 .

[57] (Angular. Bootstrapping. https://angular.io/guide/bootstrapping . visitato in data 01-01-2020)
[58] Arne Holst. Global smartphone sales to end users 2007-2020.
https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. (Aug 30, 2019). visitato in data 01-01-2020.
[59] Arne Holst. Smartphone users worldwide 2016-2021. (Nov 11, 2019).
https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. visitato in data 01-01-2020.