

Progressive Web App

Pier Francesco Uliana, Pietro Puozzo, Enoc Ke

Progetto per corso di Programmazione di Sistemi Embedded 2024-2025



Abstract

Con l'avvento delle applicazioni web e la diffusione dei dispositivi mobili, ci fu una sempre maggiore richiesta di applicativi web con funzionalità più avanzate, simili alle controparti native, e allo stesso tempo flessibili come le applicazioni web che potevano girare semplicemente su un web browser. Per rispondere a queste crescenti esigenze, Google nel 2016 introdusse le Progressive Web App, nate per offrire un'esperienza d'uso simile a una applicazione nativa e allo stesso tempo in grado di mantenere la semplicità d'installazione degli applicativi web. In questo report vengono presentate le tecnologie che hanno permesso di realizzare queste Progressive Web App, una panoramica sulle sue funzionalità e sui motivi per cui è conveniente scegliere questa soluzione di sviluppo al giorno d'oggi, nonché i componenti necessarie per una sua semplice implementazione.

Contents

1 Progressive Web App	3
1.1 PWA	3
1.2 Storia ed Evoluzione	5
1.2.1 Le Origini (2007-2015)	5
1.2.2 La Nascita delle PWA (2015-2019)	5
1.2.3 Maturità e Integrazione con gli Ecosistemi (2020-Oggi)	5
1.2.4 Sfide e Futuro	6
1.3 Tecnologie	6
1.4 Vantaggi	7
1.5 Casi d'uso nel mercato	8
1.6 TWA	10
1.6.1 Trusted Web Activities (TWA)	10
1.6.2 Osservazioni	10
2 Implementazione	12
2.1 Premessa	12
2.1.1 Design	12
2.2 Cookies	14
2.2.1 Implementazione	15
2.3 Local e session storage	16
2.3.1 Implementazione	16
2.4 IndexedDB	17
2.4.1 Implementazione	18
2.5 Service Worker	20
2.6 Caching	21
2.6.1 Implementazione	22
2.7 Installazione	24
2.7.1 Prompt di installazione	24
2.7.2 Installazione su Desktop	24
2.7.3 Installazione su iOS	26
2.7.4 Installazione su Android	27
2.8 Web App Manifest	30
2.8.1 Campi principali	31
2.8.2 Icône	31
2.8.3 Linking del manifest	34
2.9 Strumenti di Testing per PWA	35
2.9.1 Cross-platform Testing	35
2.10 Lighthouse	37
2.10.1 Metodologia di Analisi	37
2.10.2 Audit Specifici per PWA	37
2.10.3 Modalità d'Uso	37
2.10.4 Altri Strumenti Raccomandati	37
2.10.5 Interpretazione dei Risultati	38
3 Conclusione	39

1 Progressive Web App

Panoramica sulle tecnologie e vantaggi delle PWA

1.1 PWA

Le Progressive Web Apps (PWA) rappresentano un paradigma evolutivo nello sviluppo digitale, dove i confini tra web tradizionale e applicazioni native si fanno sempre più sfumati. Come evidenziato dalla documentazione ufficiale di [web.dev](#), questa tecnologia non costituisce una semplice alternativa tecnica, ma una vera strategia business in grado di rivoluzionare l'engagement degli utenti e i modelli di conversione. La peculiarità che distingue le PWA risiede proprio nel loro carattere *progressivo*, una qualità che merita un'analisi approfondita per comprenderne appieno il potenziale.

Le PWA hanno come obiettivo da un lato quello di portare all'utente un'esperienza simile ad una applicazione nativa come il funzionamento offline, l'elevata prestazione ed integrazione con il sistema, il salvataggio di dati localmente e un maggior accesso delle funzioni del dispositivo. D'altro canto, essa cerca di preservare la portabilità di una web app tradizionale e la facilità di installazione assieme alla semplicità di accesso alla versione più recente in tempo reale. Infatti una PWA è progettata per essere universalmente accessibile, garantendo una funzionalità di base su qualsiasi browser o dispositivo, indipendentemente dalle sue capacità tecniche.

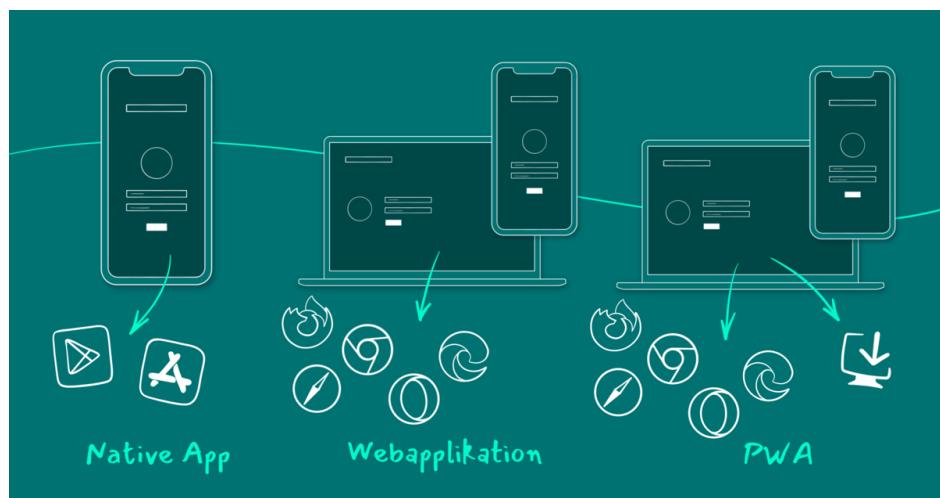


Figure 1: Confronto tra app nativa, web app e PWA

Una seconda caratteristica 'progressiva' delle PWA è la capacità dinamica di sfruttare le funzionalità avanzate dei dispositivi moderni laddove disponibili. Infatti una PWA quando installata può avere una icona sul desktop o barra applicazioni, se su PC, o sulla home screen, se su dispositivi mobili e questo permette di accedere in modo intuitivo e semplice l'applicazione proprio come una applicazione nativa. In aggiunta, come una applicazione nativa, quando vengono aperti file compatibili con la PWA, è possibile scegliere se usare la PWA in alternativa al browser.

Un'altra caratteristica delle PWA è l'uso della tecnologia del *service worker*, delle notifiche push e la possibilità di salvare localmente i dati prodotti nell'app. Ciò offre la possibilità di operare con la PWA anche in modalità offline, di mantenere l'accesso ad eventuali account collegati all'interno e di accedere a funzioni

hardware.

Infine, la natura progressiva si manifesta nel percorso di adozione stesso. Uno sviluppatore può iniziare con un sito web tradizionale e, passo dopo passo, aggiungere i componenti necessari per qualificarlo come PWA completa: l'implementazione di un web app manifest, l'attivazione del service worker, e la garanzia del protocollo HTTPS. Talvolta, una PWA può essere caricata su App StoreTWA delle diverse piattaforme se essa soddisfa i requisiti di installazione, e ricevere aggiornamenti periodici dagli sviluppatori. Tematiche a questuesto riguardo verranno sviluppate ulteriormente nella sezione "TWA". Questo approccio incrementale riduce significativamente i rischi e i costi di sviluppo.

1.2 Storia ed Evoluzione

La storia della nascita delle PWA deve gli suoi origini alla diffusione delle web app popolarizzate all'inizio del millennio. Ora vedremo in breve come queste si sono evolute nel corso del tempo e sono diventate uno strumento potente per molte aziende al giorno odierno.

1.2.1 Le Origini (2007-2015)

- **2007-2011:** Con l'avvento degli smartphone, le *web app mobili* (basate su HTML5) emersero come alternativa alle app native. Queste pagine web dinamiche presentavano funzionalità interattive, ma allo stesso tempo erano limitate nelle performance e funzionalità. Fu in questo periodo che Apple introdusse il concetto di "*Home Screen Web Apps*" per Safari, che permetteva l'aggiunta di siti alla home screen, ma con supporto limitato alle API device.^[1]
- **2015:** Google pubblica il "*Service Worker API*", tecnologia chiave per abilitare cache offline e notifiche push nelle web app.^[2]

1.2.2 La Nascita delle PWA (2015-2019)

- **Giugno 2015:** Il termine "**Progressive Web App**" viene coniato da *Alex Russell* (Google) e *Frances Berriman*. Loro definirono quelli che sarebbero diventati successivamente i principi base dello sviluppo delle PWA:^[3]
 - *Responsive* dunque adattabile ad ogni forma di schermo
 - *Indipendenza dalla connessione*, la PWA doveva essere utilizzabile anche in offline
 - *Funzionamento simile ad una app nativa*, quindi doveva avere una interfaccia ed esperienza di navigazione simile alle app native
 - *Fresh*, cioè sempre aggiornata all'ultima versione grazie al service worker che automatizza gli update
 - *Sicura* perché il service worker forzava l'utilizzo delle TLS per evitare rischi di snooping
 - *Discoverable*, perché facilmente reperibili da motori di ricerca online
 - *Re-engageable* grazie all'accesso al sistema operativo e di conseguenze delle funzioni ad esse collegate come la possibilità di inviare notifiche push
 - *Installabilità* tramite Web App Manifest.
 - *Linkabile* e condivisibile in modo veloce e semplice tramite un URL
- **2016-2019:** Google Chrome diventa il primo browser a supportare pienamente le PWA, seguito da Firefox (che però è stato abbandonato parzialmente nel 2020, supportando solo la versione Android). Successivamente anche Microsoft annuncia il supporto alle PWA in Windows 10, integrabili nel Microsoft Store. Nello stesso periodo Apple inizia a sperimentare i Service Worker in Safari, ma con molte limitazioni e soltanto nel 2018 vengono supportate pienamente i web Manifest.^[1]
- **2018:** Twitter lancia "*Twitter Lite*" come PWA, dimostrando scalabilità e performance superiori all'app nativa.^[4]

1.2.3 Maturità e Integrazione con gli Ecosistemi (2020-Oggi)

- **2020:**
 - Google introduce le *Trusted Web Activities (TWA)*, permettendo di pubblicare PWA sul Play Store come app Android.
 - Adobe e Starbucks migrano alle PWA, riducendo i costi di sviluppo e migliorando l'engagement.^{[5] [6]}
- **2021-2023:**
 - Apple migliora il supporto a Service Worker e Web Push in Safari, sebbene con restrizioni.

- Framework come *React*, *Angular* e *Vue.js* integrano tool dedicati per lo sviluppo PWA (es. [angular pwa](#)).^[6]

- **2024:**

- Le PWA diventano lo standard per progetti cross-platform, con casi d'uso in e-commerce, fintech e app enterprise.
- Nuove API (es. *File System Access*, *Web Bluetooth*) espandono le capacità delle PWA.^[6]

1.2.4 Sfide e Futuro

Lo sviluppo delle PWA è in continuo aumento con ogni anno nuove API e Framework al servizio di esse. I limiti attuali stanno nella difficile integrazione nei sistemi iOS dovuto al supporto disomogeneo e accesso difficoltoso a funzionalità del sistema operativo iOS. Questo problema persiste tutt'ora alla stesura del report per alcune funzionalità più rischiose dell'hardware che il sistema operativo blocca anche nei dispositivi Android, come l'NFC. Gli sviluppi futuri delle PWA si allineano agli trend del mondo dei software come l'integrazione dell'*intelligenza artificiale*, degli apparecchi connessi all'*Internet of Things* (IoT), strumenti di *realtà virtuale* e integrazione più in profondità con il sistema operativo tramite le tecnologie di *WebAssembly* che permettono capacità di calcolo più potenti e client-side.

1.3 Tecnologie

La diffusione delle PWA deve il suo successo anche agli strumenti che le hanno portato prestazioni simili alle applicazioni native. Il corretto funzionamento di una PWA dipende fortemente dall'interazione delle sue componenti, di seguito ne elenchiamo alcuni:

- **Web App Manifest**

Il manifest è un file JSON che definisce metadati essenziali per l'installazione della PWA, tra cui nome, icone, colore del tema e orientamento dello schermo. Consente all'app di integrarsi nell'ambiente del dispositivo come un'app nativa, provvedendo le informazioni necessarie per la corretta visualizzazione sia nei browser che nelle diverse icone dei dispositivi.

- **Service Worker**

Il service worker è uno script che opera in background, abilitando funzionalità critiche come la cache offline, la gestione delle notifiche push e il precaricamento delle risorse. Agisce da intermediario tra l'app e la rete, garantendo prestazioni affidabili anche con connessioni instabili.



Figure 2: Componenti essenziali delle PWA

- **Framework e strumenti moderni**

L'introduzione di nuove piattaforme hanno portato anche nuovi strumenti dedicati alla realizzazione di UI o funzionalità sempre più complesse ed efficienti, come librerie, API e framework a servizio delle web app. Alcune API di particolare importanza sono quelle che gestiscono dati, come le **cache API**, salvataggio dei **cookies** o **IndexedDB** per archiviazione in strutture dati. Oltre a quest API sono

disponibili in circolazione framework per migliorare l'esperienza utente come quelle per la gestione delle notifiche push o **React**^[7] le sue librerie dedicate (ad esempio MUI) per l'implementazione di interfacce utente gradevoli e secondo i principi di design più recenti. Infine altri strumenti come **Workbox** usati per la gestione semplificata dei Service Worker e framework come **Angular** e **Vue.js** offrono librerie e modelli preconfigurati per accelerare lo sviluppo.

1.4 Vantaggi

Riassumendo, i principali vantaggi offerti delle Progressive Web App sono:

- **Un codice, più piattaforme**

Le PWA non richiedono di riscrivere il codice per ogni piattaforma specifica, bensì può operare su più dispositivi mantenendo lo stesso codice base. Questo è possibile perché di base PWA possono funzionare anche solo nei browser web come dei siti web. Questo riduce significativamente la quantità di codice prodotto e di conseguenza i costi di sviluppo e manutenzione successiva. ^{[8] [9]}

- **Portabilità e lightweight**

Le PWA essendo essenzialmente delle pagine web, possono essere indicizzate più facilmente dai motori di ricerca. Oltre a ciò, esse possono essere avviate o installate direttamente da una pagina web. Gli utenti inoltre ricevono sempre la versione più recente senza dover scaricare manualmente gli aggiornamenti da una App store dedicata. ^[2]

- **Sicurezza**

Soltanente le web app per soddisfare i requisiti di installazione, utilizzano protocolli end-to-end HTTPS. Ciò garantisce sicurezza ai dati trasmessi e previene attacchi man-in-the-middle. ^[8]

- **Esperienza simile alle app native**

Offrono un'interfaccia full-screen, navigazione fluida e accesso alle funzionalità del dispositivo come notifiche push, geolocalizzazione, l'accesso alla fotocamera, sensori e altre API device-specific. ^{[9] [8]}

- **Funzionamento offline**

Grazie ai Service Worker, le PWA possono funzionare senza connessione o con rete limitata, memorizzando i dati nella cache. ^[8]

- **Performance**

Caricamento rapido grazie al caching strategico e pre-caricamento delle risorse. Le PWA consumano meno dati rispetto alle app native grazie alla cache e alla compressione delle risorse. Questo permette alle PWA di occupare meno spazio rispetto alle app native, poiché non richiedono un'installazione pesante. ^[9]

1.5 Casi d'uso nel mercato

Molte aziende hanno beneficiato dei vantaggi derivanti dalla migrazione a PWA, di seguito ne citiamo alcuni più famosi:

1. Twitter Lite

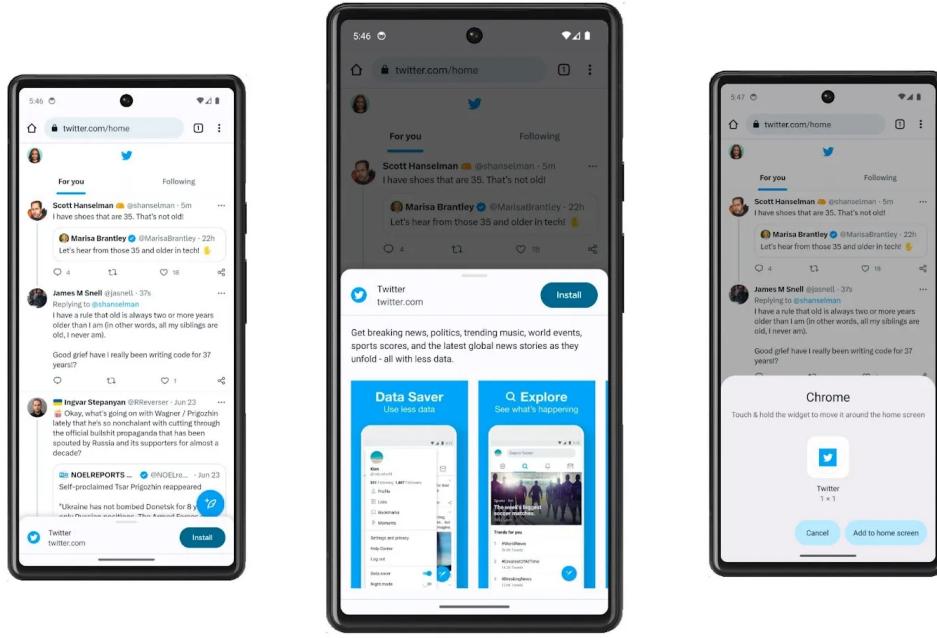


Figure 3: Twitter Lite^[4]

- Risultati: Riduzione del 20% del tempo di caricamento, aumento del 75% dei tweet pubblicati e del 65% delle pagine visitate per sessione. Anche la dimensione è stata ridotta da 31 MB dell'app nativa a circa 600 KB, permettendo anche a dispositivi con spazio di archiviazione ridotta di installare Twitter. Per rendere Twitter Lite ancora più reattiva, il team di sviluppo ha adottato un codice route-based e tecniche di bundle-splitting per permettere di caricare solo il minimo indispensabile del carico JavaScript in base al view selezionato dall'utente, invece di caricare tutto pagina web. Infine opzioni come 'Risparmio dati' ha permesso un caricamento più veloce delle immagini che ora vengono solo visualizzate in qualità minima prima che l'utente opti per una qualità migliore.



Figure 4: Esempio di immagine caricata con 'Risparmio dati'

- **Vantaggi:** Minore consumo di dati e miglioramento dell'esperienza utente in aree con connettività limitata. La possibilità di aggiungere Twitter Lite a Google Play Store ha aiutato anche alla diffusione dell'app e permesso ai nuovi utenti di confrontarsi con altri utenti tramite le recensioni. [4]

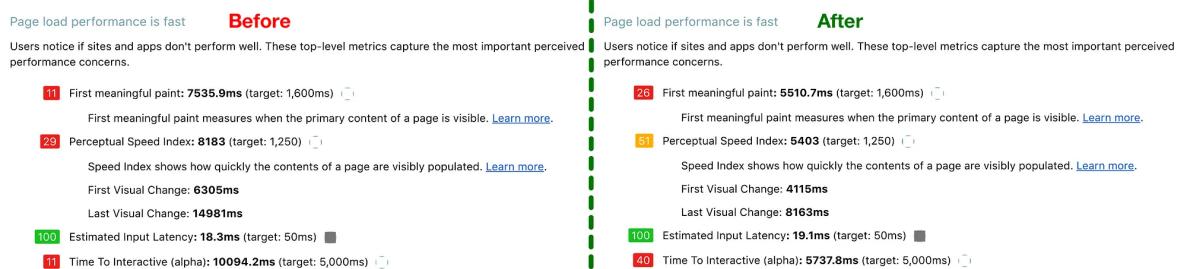


Figure 5: Confronto Vantaggi passaggio a PWA di Twitter [4]

2. Starbucks

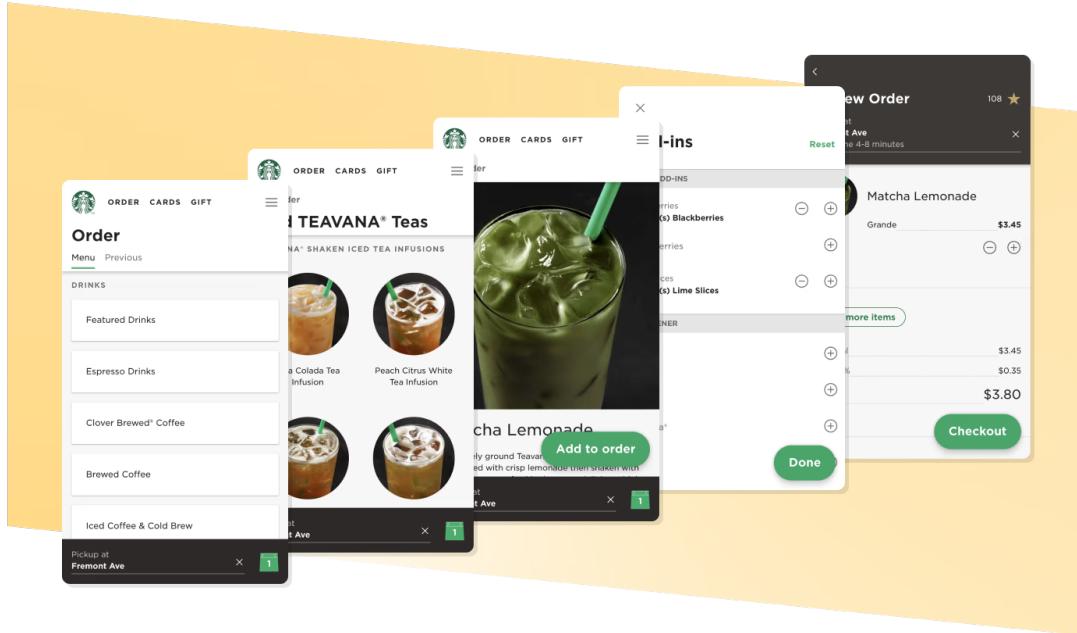


Figure 6: Interfaccia web app di Starbucks [10]

- **Risultati:** La PWA ha raddoppiato gli ordini giornalieri rispetto all'app nativa, con un footprint ridotto. Infatti il codice della PWA occupa meno di 240 KB, 99.84% più piccola rispetto alle 148 MB dell'app nativa. [10]

- **Vantaggi:** Accessibilità immediata senza download, specialmente per utenti con dispositivi meno potenti. Anche la velocità di caricamento dei vari menu è ottimizzato grazie ad un caricamento ottimizzato della UI della PWA che li dispone tutti in una stessa schermata. Questa interfaccia user-friendly ha portato una maggiore facilità d'uso e una migliore esperienza per utenti su dispositivi mobili. [11] [10]

3. Clipchamp
 - Risultati: Aumento del 97% delle installazioni nei primi 5 mesi e incremento del 9% degli utenti che hanno riutilizzato il servizio dopo una prima esperienza.
 - Vantaggi: Aumento performance del 230% rispetto alla web app originale.^[5]
4. Pinterest
 - Risultati: Incremento del 60% del tempo di interazione e del 44% delle entrate pubblicitarie.
 - Vantaggi: Performance paragonabili all'app nativa, con un tasso di conversione più elevato.^[11]
5. Uber
 - Risultati: La PWA funziona su dispositivi low-end con tempi di caricamento inferiori a 3 secondi, raggiungendo un pubblico globale più ampio.
 - Vantaggi: Nessun vincolo di spazio di archiviazione e compatibilità con browser obsoleti.^[11]

1.6 TWA

1.6.1 Trusted Web Activities (TWA)

Le **Trusted Web Activities (TWA)** sono una strategia introdotta da Google per integrare Progressive Web Apps (PWA) con il sistema Android, consentendo la distribuzione attraverso il Play Store. Una TWA esegue una PWA all'interno di un APK Android in modalità full-screen, senza mostrare l'interfaccia del browser. Per pubblicare una TWA è necessario:

1. Verificare che la PWA soddisfi i requisiti base (offline operation, HTTPS, performance sotto i 5s, ecc.) e ottenga un punteggio minimo di 80/100 su Lighthouse.
2. Stabilire un collegamento tra il sito web e l'APK tramite **Digital Asset Links**, che certifica l'appartenenza allo stesso sviluppatore.
3. Generare l'APK utilizzando **Android Studio**, seguendo le linee guida di Google.
4. Superare i controlli del Play Store per la pubblicazione.

Anche se una TWA essenzialmente è molto simile a una PWA, le prime presentano alcune differenze come la modalità di installazione e reperibilità. Di seguito ne listiamo alcuni più evidenti dal punto di vista dell'esperienza di un utente finale.

Caratteristica	PWA	TWA
Distribuzione	Browser web	Play Store
Installazione	Tramite browser	Come APK Android
Accesso	Launcher del dispositivo	Launcher del dispositivo
Supporto offline	Sì (con Service Worker)	Sì, ma primo caricamento richiede connessione
IDE consigliato	qualsiasi editor	Android Studio
Compatibilità	Tutti i browser (limiti su iOS)	Solo Android (da versione 4.4+)
Esperienza utente	Simile a nativa	Identica a PWA, ma con maggiore integrazione OS

Table 1: Tabella esemplificativa delle principali differenze e similarità tra PWA e TWA

1.6.2 Osservazioni

Le PWA rappresentano un'alternativa promettente per lo sviluppo multipiattaforma, unificando desktop e mobile. Le TWA offrono vantaggi significativi per gli sviluppatori che desiderano raggiungere utenti Android attraverso il Play Store, mantenendo i benefici delle PWA (come l'aggiornamento senza ricompilazione). Tuttavia, restano sfide come la compatibilità con iOS, la maturità dell'ecosistema e la dipendenza da browser che la supportano completamente come Chrome e queste limitazioni ne riducono l'adozione universale. Come lavoro futuro, gli autori del paper **Computer Science CACIC (2019)** citato in precedenza, propongono di

estendere l'analisi comparativa includendo altri approcci (ibridi, cross-compilati) e valutare l'impatto energetico delle PWA.^[6]

2 Implementazione

Come progettare una PWA

2.1 Premessa

In questa sezione parleremo di come possiamo implementare una PWA a partire da una pagina web preesistente. Dunque, per rendere una web app 'progressive', applicheremo il miglioramento progressivo per portare all'utente una user experience affidabile e simile a una app nativa. Quindi vedremo le componenti che rendono una web app installabile, capace di interagire con il sistema operativo, sempre aggiornata e indipendentemente funzionante, anche in assenza di connessione internet. Ergo, una volta installata, una PWA deve

presentare almeno le seguenti caratteristiche:

- Integrazione con il sistema operativo
- Funzionamento offline
- Interfaccia indipendente dalla UI del browser
- Presente nella lista delle app, predisposta di nome e icona e indicizzabile tramite ricerca nel sistema.

In questa sezione si discuterà anche delle varie strategie con cui le Web App possono immagazzinare dati nel browser dell'utente, in modo da potervi accedere anche con connessione internet debole o assente.

2.1.1 Design

Oltre ai requisiti funzionali ci sono caratteristiche tecniche e stilistiche riguardanti l'adattabilità della pagina web in schermi e finestre di dimensione diversa, qualità delle immagini in schermi con risoluzione maggiore o adeguamento alle più nuove regole di stile come Material Design 3. Questi aspetti di design dipendono dai casi d'uso della web app e dalle preferenze degli utenti finali. Ad esempio in base al tipo di schermo

possiamo indirizzare la PWA a diverse UI adatte alle dimensioni attraverso il seguente codice in css.

```
/* apertura in un browser */
@media (display-mode: browser) {
}

/* apertura come app indipendente */
@media (display-mode: standalone) {
}

/* apertura come app in caso generale */
@media (display-mode: standalone), (display-mode: fullscreen), (display-mode: minimal-ui) {
```

Oltre a come viene aperta la PWA, possiamo anche controllare la navigazione e le interazioni dell'utente all'interno della PWA come ad esempio i colori, temi, area di display in caso di notch sporgente nell'area dello schermo, font dei caratteri o i gesti di selezione, aggiornamento, trascinamento. (Vedi figura 7)

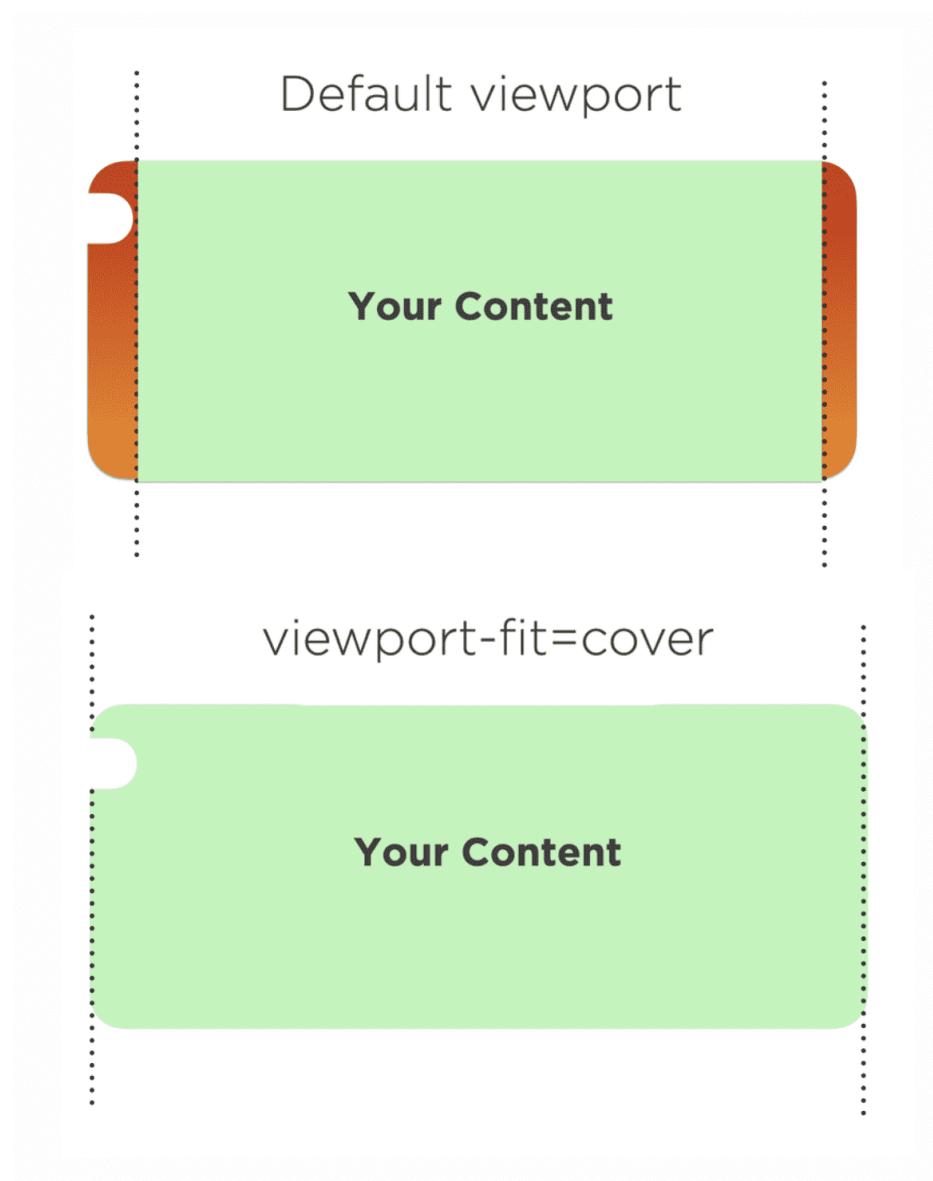


Figure 7: Gestione delle aree di 'notch' e 'chin' in schermi con maggiore copertura di area touchscreen

2.2 Cookies

Come buona parte delle pagine web, anche le PWA fanno uso di Cookie: un ***Cookie*** è una stringa di testo che viene inviata dal server web al client, il quale avrà poi il compito di memorizzarla e rinviarla al server, senza modifiche, ogni volta che accede al dominio web di appartenenza del Cookie (o a sue porzioni) ^[12].

I Cookie hanno il grande vantaggio di permettere di definire uno stato in un protocollo che di per sé è *stateless*: ad esempio, per servizi che richiedono autenticazione, è possibile definire uno stato "logged-in", che consente all'utente di navigare nel sito dovendo fare il login solo una volta. Senza Cookie, invece, sarebbe necessario ripetere l'accesso ogni volta che si accede a una nuova pagina all'interno del dominio ^[12].

I Cookie hanno una dimensione ridotta: infatti, dato che il browser può dover inoltrare anche centinaia di Cookie durante la navigazione, delle dimensioni eccessive provocherebbero danni alle prestazioni. Essi sono inviati attraverso specifici header del protocollo *HTTP*: nel caso di *HTTP response* viene usato l'header **Set-Cookie** mentre per *HTTP request* si usa **Cookie** ^[12]. A un Cookie viene associata inoltre una data di scadenza, oltre la quale non viene più considerato valido ^[12].

I Cookie possono avere diversi scopi: possono coprire aspetti necessari al corretto funzionamento del sito (in tal caso si parla di *cookie tecnici*), possono raccogliere dati in forma anonima a fini statistici (*cookie statisticci*) oppure possono tracciare la navigazione dell'utente, con lo scopo di costruire un profilo personalizzato dello stesso utile a fornire annunci mirati (*cookie pubblicitari*) ^[12].

L'ultima categoria di Cookie è stata oggetto di diverse discussioni in ambito legale, in quanto un suo abuso potrebbe costituire una minaccia alla privacy degli utenti; per questo motivo, il loro utilizzo è normato da svariate leggi in diversi stati: in Europa il *GDPR* (*General Data Protection Regulation*) ha stabilito che tutti i siti internet che fanno uso di Cookie pubblicitari di terze parti sono obbligati ad avvertire l'utente di ciò all'inizio della sua navigazione nella pagina web ^[13]. Tale avviso deve essere effettuato attraverso un banner che ostacoli la visualizzazione della pagina e al cui interno sono elencati tutti i Cookie pubblicitari presenti nel sito, specificando anche chi elaborerà i dati raccolti e con quali finalità ^[13]; è inoltre obbligatorio fornire la possibilità di negare il consenso dei singoli Cookie, non è sufficiente permettere all'utente di negare/consentire tutti i Cookie in blocco ^[13]. Un Cookie non potrà essere attivato se non dopo il consenso esplicito da parte dell'utente, il quale sarà registrato in opportuna documentazione a testimoniare alle autorità che l'autorizzazione è stata concessa ^[13]. Il consenso a un Cookie, inoltre, deve essere fornito mediante un'azione non equivoca (come ad esempio cliccare sul pulsante "Acconsento"); azioni come continuare la navigazione sul sito non devono essere considerate come permesso per installare i Cookie ^[13]. In aggiunta, si deve fornire la possibilità di modificare le proprie scelte in un secondo momento.

All'inizio della navigazione, tutti i Cookie di terze parti e non strettamente necessari al corretto funzionamento del servizio devono essere preventivamente disattivati in attesa della scelta dell'utente. Una volta confermata la scelta, il *Cookie di consenso* si occuperà di attivare tutti i soli Cookie inserzionistici per il quale l'utente ha dato esplicito consenso ^[13].

Diversi siti internet fanno uso dei Cookie come identificatore di sessione univoco dell'utente (utile ad esempio per permettere al cliente di mantenere l'accesso senza dover fare continuamente login): questo può costituire un rischio per la sicurezza, in quanto un utente malintenzionato può rubare il Cookie di qualcun altro e sfruttarlo per impersonare la vittima. Questo problema può essere risolto sfruttando soli Cookie con la flag "**Secure**" (che consente l'invio dello stesso solo attraverso protocollo criptato *HTTPS*) ^[12].

In DevTools (accessibile su buona parte dei browser premendo il tasto F12) è possibile avere una visuale dettagliata di tutti i Cookie installati, permettendo all'utente anche di eliminare quelli indesiderati: per browser Chromium-based (come Google Chrome o Microsoft Edge) basta cliccare su "**Application**" (nella riga in alto, a volte il pulsante è nascosto dal bottone ">>") e poi "**Cookies**" nel menù a sinistra, per Firefox invece "**Storage**" e poi "**Cookies**".

2.2.1 Implementazione

I Cookie hanno un'interfaccia molto primitiva: non sono definite, infatti, delle funzioni per l'aggiunta, rimozione o la modifica di essi. L'unico modo per inserire, modificare, leggere ed eliminare Cookie è mediante l'attributo `document.cookie`^[14].

Per inserire un nuovo Cookie basta assegnare una nuova stringa a `document.cookie`^[14]: tale stringa dovrà presentarsi nel formato^[14]

```
"name=value; optionalField1=optionalValue1; optionalField2=optionalValue2;..."
```

La coppia "name=value" deve essere specificata, altrimenti l'inserimento fallisce silenziosamente; tutte le coppie successive sono invece opzionali. Qui sotto è riportata una lista di alcuni dei campi opzionali:

- `"expires"`: definisce la data di scadenza del Cookie come stringa UTC; per esprimere una data in tale formato è possibile usare il metodo `toUTCString()` della classe `Date` definita in JavaScript^[14].
- `"max-age"`: specifica la durata del Cookie in secondi^[14]. Se non sono specificati né `expires` né `max-age` allora il Cookie scadrà al termine della sessione^[14], mentre se sono specificati entrambi allora `max-age` ha la priorità su `expires`.
- `"secure"`: indica che il Cookie può essere trasmesso solo attraverso il protocollo criptato HTTPS^[14].
- `"partitioned"`: indica che il Cookie deve essere memorizzato in memoria partizionata^[14]. Un Cookie partizionato impedisce il tracciamento *cross-site* dell'utente, meccanismo che viene ad esempio usato dagli inserzionisti per costruire un profilo personalizzato dell'utente^[15]. Si Supponga di accedere a un sito A che carica un annuncio da un sito di terze parti. Al momento del caricamento, quest'ultimo imposta un Cookie sul dispositivo dell'utente*. Si supponga ora di spostarsi a un sito B, che carica lo stesso annuncio di prima. Se il Cookie non è partizionato, allora il sito di terze parti sarà in grado di accedere al Cookie definito precedentemente, difatti, in questo caso, la chiave del Cookie è definita solo dall'host che lo ha impostato; il dominio da cui proviene l'inserzione è pertanto in grado di capire che l'utente ha visitato sia A che B. Se, invece, il Cookie è partizionato, allora il sito di terze parti non sarà in grado di accedere al Cookie in quanto, in questo caso, la sua chiave è definita dalla coppia host + sito in cui è caricato il contenuto^[15].
- `"domain"`: specifica il dominio a cui il Cookie potrà essere inviato; se non inserito allora assume un valore di default che coincide con l'host della pagina. Il Cookie è visibile ai sottodomini solo quando questo parametro è esplicitato^[14].
- `"path"`: specifica il percorso del dominio in cui il Cookie è visibile; il Cookie potrà essere inviato solo alla porzione indicata da `path` all'interno del dominio di appartenenza, incluse anche eventuali sottodirectory^[16]. Se non inserito allora assume il valore di default "/" (la *root directory*). `path` e `domain` definiscono assieme l'ambito di visibilità del Cookie^[12].
- `"samesite"`: definisce quando inviare il Cookie al server^[14]. Esso può assumere valore `lax` se il Cookie può essere inviato solo in occasione di *same-site requests* e *top-level navigation requests*^{†[14]} (in questo secondo caso, però, il Cookie può essere inviato solo attraverso *safe requests*, come `GET` o `HEAD` ma non `POST`^[17]), `strict` se si vuole impedire l'invio del Cookie attraverso cross-site requests^[14], `none` se non si applica alcun vincolo^[14] (in quest'ultimo caso è però richiesto che sia esplicitato il parametro `secure`^[16]). Si supponga, ad esempio, che l'utente stia navigando in un sito A e che clicchi un link che lo porta al sito B: essendo questa una top-level navigation request, se il Cookie ha parametro `samesite=lax` allora esso sarà inviato a B, se, invece, il Cookie ha parametro `samesite=strict`, allora esso non sarà inoltrato^[17].

Una volta inserito un Cookie, esso non può essere modificato direttamente; è possibile solo sovrascrivere quest'ultimo con un altro: per farlo basta assegnare a `document.cookie` un nuovo Cookie con lo stesso nome di quello da sovrascrivere^[18].

*supponendo che l'utente abbia acconsentito a ciò

†cioè una navigazione a un altro sito che porta alla modifica del contenuto della barra degli indirizzi^[17]

Per quanto riguarda l'eliminazione dei Cookie, l'unica strategia disponibile è quella di sostituire il Cookie con un altro avente scadenza già passata^[18].

L'attributo `document.cookie` può anche essere acceduto in lettura: in tal caso si ottiene una lista delle sole coppie nome-valore di tutti i Cookie salvati in quel momento^[14].

2.3 Local e session storage

Local storage e **Session storage** sono strumenti con cui le Web App possono immagazzinare piccole quantità di dati nel dispositivo dell'utente: essi possono essere usati, ad esempio, per salvare le preferenze dell'utente, in maniera tale che queste possano continuare a essere applicate a qualunque successiva apertura dell'app.

Local e session Storage sono entrambi oggetti di tipo **Storage**: in quanto tali, essi permettono di salvare dati in memoria locale sotto forma di coppie chiave-valore (entrambe stringhe)^[19].

I due oggetti differiscono per visibilità e durata: Local storage serve a contenere dati permanenti (cioè che rimangono in memoria indefinitamente[‡], salvo esplicita rimozione da parte dell'utente o del programmatore) e condivisi fra le varie schede, per cui schede distinte del medesimo browser connesse allo stesso sito accedono allo stesso Local storage^[20]. D'altro canto, Session storage è progettato per contenere dati relativi alla singola sessione di navigazione: esso, quindi, non è condiviso fra schede e viene rimosso automaticamente alla chiusura della pagina (viene però conservato quando la pagina è ricaricata)^[21].

DevTools fornisce la possibilità di visualizzare il Local storage e il Session storage di una pagina: per browser Chromium-based è sufficiente cliccare su "Application" e poi "Local Storage" o "Session Storage", mentre per Firefox si deve cliccare su "Storage", quindi "Local Storage" o "Session Storage".

2.3.1 Implementazione

Local storage e Session storage sono accessibili rispettivamente mediante gli oggetti `window.localStorage` e `window.sessionStorage`; essendo entrambi istanze di **Storage**, essi condividono la stessa API, composta dai metodi

- `setItem(key, value)`, che inserisce una nuova coppia chiave-valore se la chiave non è presente, altrimenti sostituisce il valore associato a `key` con `value`^[20].
- `getItem(key)`, che restituisce il valore associato a `key` o `null` se `key` non è presente^[20].
- `removeItem(key)`, che rimuove la coppia con chiave `key`, se presente^[20].
- `clear()`, che svuota l'intero **Storage**^[20].
- `key(n)`, che restituisce il nome della `n`-esima chiave oppure `null` se `n` è negativo o maggiore o uguale al numero di entries^[20].

È definito anche l'evento "storage", che viene lanciato quando il contenuto dello **Storage** subisce delle modifiche: questo evento può essere catturato da tutte le schede connesse allo **Storage** diverse dalla pagina che ha modificato il contenuto. Per ascoltare l'evento basta aggiungere un opportuno `eventListener` all'oggetto `window`^[19].

`localStorage` e `sessionStorage` non sono disegnati per contenere grandi quantità di dati: un oggetto di tipo **Storage**, infatti, ha una capienza massima di 10MB. Se è necessario memorizzare grandi moli di dati bisogna ricorrere a *IndexedDB*^[22].

[‡]Esistono però condizioni che possono spingere il browser a eliminare dati che dovrebbero essere permanenti, come spiegato nelle sezioni successive

2.4 IndexedDB

Gli **IndexedDB** (*Indexed database*) sono una strategia con cui le Web App possono immagazzinare dati strutturati nel browser dell’utente. Un IndexedDB implementa un *Object Oriented Database Management System (OODBMS)* e, in quanto tale, memorizza coppie chiave-valore^[22].

A differenza di Local Storage e Session Storage, un Indexed database non presenta limiti di spazio (in questo caso la capienza massima è definita dal singolo browser[§]), inoltre, chiavi e valori possono essere un qualunque oggetto, non per forza stringhe. Come per local storage, i dati salvati nel database sono permanenti, salvo esplicita rimozione da parte dell’utente o del programmatore; se, però, l’utente sta navigando in modalità privata (o modalità incognito), allora tutti i dati relativi alla navigazione (inclusi IndexedDB) saranno rimossi automaticamente al termine della sessione^[22].

Mentre in un *RDBMS* (*Relational Database Management System*) i dati sono memorizzati come righe all’interno di una tabella, in un OODBMS essi sono salvati come oggetti contenuti in *Object Store*: all’interno di un Object Store ciascun record sarà identificato da una chiave univoca (come nei RDBMS), che può essere sfruttata per ottenere il riferimento al valore associato^[22]. L’insieme degli Object Store in un database ne definisce lo *schema*^[23].

Oltre che con le chiavi, gli OODBMS possono essere interrogati anche attraverso indici: un *indice* è un Object Store ausiliario che si riferisce a un altro Object Store, e che serve a effettuare ricerche su quest’ultimo^[22]. Un Object Store può avere uno o più indici a esso associati, purché i valori in esso contenuti siano oggetti e non dati primitivi. Questo è dovuto dal fatto che le ricerche effettuate da un indice si basano sui campi dei valori salvati^[23]: ad esempio, se si ha costruito un database di studenti e per ciascuno studente si ha deciso di salvare nome, cognome, matricola e voto di laurea, è possibile costruire un indice che effettua ricerche in base al voto di laurea o in base alla matricola.

All’interno di un indice, ciascun record ha valore che coincide con la chiave di un altro record contenuto nell’Object Store riferito; inoltre, ogni volta che il contenuto di quest’ultimo subisce una modifica, l’indice viene aggiornato automaticamente^[22].

Gli indici sono anche utili per imporre dei vincoli sugli Object Store riferiti^[23]: tornando all’esempio di prima, per mezzo di indici si può imporre il vincolo di unicità per la coppia (“nome”, “cognome”).

Quando si effettua una ricerca tramite indice, il risultato che viene restituito non è una lista di record bensì un *cursore*, oggetto ausiliario negli OODBMS che permette di iterare fra record di un Object Store^[22].

Gli indexed database applicano un modello basato su transazioni: una *transazione* è una sequenza di operazioni che va a costituire un’unità logica di lavoro^[24]. Qualunque operazione eseguita sul database (sia essa di sola lettura o anche di scrittura) dev’essere eseguita all’interno di una transazione^[23]. Le transazioni sono *atomiche*, ciò vuol dire che esse possono o essere completate nella loro interezza o non avere alcun effetto; dunque, se una delle operazioni della sequenza fallisce, tutte le modifiche effettuate dalle operazioni precedenti saranno annullate^[24]. Il fatto che le transazioni siano atomiche permette di mantenere l’integrità dei dati nel caso di accessi concorrenti al database^[24].

Gli IndexedDB applicano una *same-origin policy*, il che vuol dire che un’app può accedere a tutti e soli gli IndexedDB presenti nella sua stessa origine (con anche lo stesso port e lo stesso protocollo)^[22].

La indexedDB API presenta comunque delle limitazioni, fra cui il fatto che è progettata solamente per l’amministrazione di server *Client-Side*, non sono forniti strumenti per la gestione *Server-Side*^[22]. Per visualizzare gli IndexedDB salvati in un determinato dominio è sufficiente aprire DevTools e poi cliccare su

[§]vd. ”Caching”

”Application” e poi ”IndexedDB” (per browser Chromium-based) o su ”Storage” e poi ”IndexedDB” (su Firefox).

2.4.1 Implementazione

La IndexedDB API è principalmente asincrona, ciò vuol dire che la maggioranza delle operazioni sulla base di dati è eseguita per mezzo di *richieste*, rappresentate mediante oggetti di tipo `IDBRequest`^[23]: esse presentano i campi `onSuccess` e `onError`, che permettono di definire delle callback functions da eseguire rispettivamente in caso di successo o errore^[23]. L'esito di una richiesta è definito dal parametro `type`, che può avere valore ”`success`” o ”`error`”^[23].

È possibile creare una richiesta di apertura della connessione con il database per mezzo della funzione `IDBFactory.open(name, version)`, dove `name` è il nome del database (che deve essere unico all'interno del dominio) e `version` (opzionale) è la sua versione^[23]: il metodo lancia una `IDBOpenDBRequest` che si avverrà con l'apertura del database^[25]. Una volta aperto, la base di dati è accessibile mediante il campo `IDBOpenRequest.result` all'interno dell'handler `onsuccess`^[25].

Aprire un database con lo stesso nome di uno già esistente ma numero di versione maggiore viene interpretato come la necessità di effettuare un aggiornamento del database stesso, causando il lancio dell'evento `upgradeneeded`: per gli oggetti di tipo `IDBOpenDBRequest`, è presente il campo `onupgradeneeded` che permette di definire che funzione eseguire in questa situazione^[23]. Aprire un database con lo stesso nome di uno già presente nel dominio ma numero di versione inferiore provoca invece il lancio dell'errore `VER_ERR`^[23].

Lo schema del database può essere modificato solo nella callback function da eseguire come risposta all'evento `upgradeneeded`: tale modifica può comprendere solamente l'aggiunta o la rimozione di Object Store, se si intende invece alterare la struttura di un Object store l'unico modo per farlo è eliminarlo e ricostruirlo^[23].

Gli Object Store possono essere creati per mezzo di `IDBDatabase.createObjectStore(name, options)`, dove `name` è il nome e `options` (facoltativo) definisce delle opzioni riguardo le chiavi: questo secondo parametro è un oggetto che può avere i campi ”`keyPath`”, che specifica quale proprietà del valore inserito usare come chiave (dunque qualunque valore inserito nell'Object Store non dovrà essere un dato primitivo e dovrà disporre del campo specificato), o ”`autoIncrement`”, che specifica che la chiave sarà un intero autoincrementante generato automaticamente^[23]. Se non è specificato alcun valore per `options`, allora, a ogni inserimento, sarà necessario specificare la chiave^[23].

Per creare un indice si usa la funzione `IDBObjectStore.createIndex(indexName, keyPath, options)`, in cui ”`indexName`” è il nome dell'indice, ”`keyPath`” è il nome della proprietà su cui l'indice effettuerà le query e ”`options`” è un oggetto che definisce delle opzioni aggiuntive (contiene ad esempio il campo ”`unique`” che permette di imporre il vincolo di unicità su `keyPath`)^[25].

Il codice seguente crea un nuovo database sulla base dell'esempio mostrato prima: la base di dati avrà un Object Store ”`Lauree`”, contenente i voti di laurea per ciascuno studente, e un indice ”`voti`”, che permette di effettuare query sui voti

```
//Si suppone che ogni studente sia rappresentato da un oggetto contenente i campi
// "matricola", "nome", "cognome", "voto_laurea"
const request = window.indexedDB.open("studenti", 1);

request.onerror = (event) => {
    //Gestisce errore
};

request.onupgradeneeded = (event) => {
```

```

    const db = event.target.result;

    const objectStore = db.createObjectStore("Lauree", {keyPath: "matricola"});

    objectStore.createIndex("voti", "voto_laurea", {unique: false});
};

}

```

Come è stato detto precedentemente, qualunque operazione all'interno di un database può essere effettuata solamente entro il contesto di una transazione: questa può essere definita per mezzo del metodo `IDBDatabase.transaction(storeNames, mode)`^[25], dove `storeNames` è l'array dei nomi degli Object Store in essa coinvolti (lo *scope* della transazione), mentre `mode` (opzionale) definisce il tipo di transazione. Il secondo parametro può assumere i valori `"readonly"`, se la transazione è di sola lettura, o `"readwrite"`, se la transazione può anche modificare il contenuto degli Object Store^[23]. Esiste anche un terzo tipo di transazione che è `"versionchange"`, riservata all'handler `onupgradeneeded`^[23].

Una volta definita una transazione, si può ottenere il riferimento a uno degli Object Store nel suo scope mediante il metodo `IDBTransaction.objectStore(name)` ("`name`" è il nome dell'Object store interessato)^[25], sul quale è possibile

- Inserire un dato mediante `IDBObjectStore.put(item, key)` (non è necessario specificare `key` nel caso si usi `keyPath` o `autoIncrement`)^[25].
- Ottenere il valore associato a una chiave mediante `get(key)` (questa funzione restituisce solo una richiesta, il risultato della query è accessibile dal parametro dell'handler `onsuccess`)^[25].
- Rimuovere un record mediante `delete(key)`^[25].

Questi metodi sono validi anche per gli indici.

In seguito è presente una continuazione del codice precedente che crea una transazione che popola il database e poi lo interroga alla ricerca degli studenti che si sono laureati con 110

```

// "dati" è un array contenente le informazioni sugli studenti da inserire
request.onsuccess = (event) => {
    const db = event.target.result;

    const transaction = db.transaction(["Lauree"], "readwrite");

    const Lauree = transaction.objectStore("Lauree");

    dati.forEach((element) => {
        Lauree.put(element);
    });

    // Ottengo il riferimento all'indice "voti"
    const voti = Lauree.index("voti");

    console.log(voti.getAll(110));
};

```

Un *cursor* è un oggetto che permette di iterare sugli elementi di un Object Store. Per crearne uno è necessario invocare `IDBObjectStore.openCursor(query, direction)`^[25], dove `query` è un oggetto di tipo `IDBKeyRange` che definisce il range di chiavi su cui si può iterare mentre `direction` è una stringa che definisce

la direzione con cui si muove il cursore: essa può assumere valore ”`next`”, se il cursore parte dall’inizio del range e si muove verso la fine, o ”`prev`”, se si muove nella direzione opposta; sono definiti anche i valori ”`nextunique`” e ”`prevunique`” che saltano eventuali duplicati. Entrambi i parametri sono opzionali^[25]. La funzione restituisce una `IDBRequest` e il cursore potrà essere acceduto dal parametro `result` nell’handler `onsuccess`^[23]. Una volta ottenuto il riferimento al cursore, sono definiti i campi `key`, per leggere la chiave dell’elemento puntato, e `value`, per leggerne il valore. Il metodo `continue()` permette, invece, di avanzare al record successivo^[25].

Il seguente codice crea un cursore sull’Object store ”Lauree” che stampa tutte le matricole salvate (continuazione dell’handler `request.onsuccess`)

```
const cursorRequest = Lauree.openCursor();

cursorRequest.onsuccess = (event) => {
    const cursor = event.target.result;

    if(cursor) {
        console.log(cursor.key);

        //Sovrascrive il valore di cursor con un nuovo cursore che punta all'elemento
        //successivo, se si ha raggiunto la fine dell'object store allora
        //sovrascrive il valore di cursor con "undefined"
        cursor.continue();
    }
}
```

2.5 Service Worker

Il **Service Worker** è un tipo di *Worker*[¶] che funge da intermediario per la comunicazione della web app con la rete: il suo scopo principale è quello di intercettare le *network requests* effettuate dal sito e decidere che contenuto servire in risposta a queste ultime^[27]. Esso è anche in grado di inviare notifiche push ed effettuare la sincronizzazione dei contenuti in background^[27].

L’esecuzione del Service Worker avviene in un thread separato rispetto a quello della pagina web, pertanto è possibile che questo lavori anche ad app chiusa^[28]. In ogni caso, per risparmiare risorse, il Service Worker viene automaticamente bloccato dal browser se rimasto inattivo per alcuni secondi, per poi essere riattivato quando serve^[28]. Il fatto che il Service Worker giri su un thread secondario gli impedisce di accedere ad API sincrone come Local e Session Storage^[27].

Il codice del Service Worker dev’essere definito all’interno di un file JavaScript a sé stante. La directory in cui questo file è inserito ne determina lo *scope*, cioè il suo ambito di lavoro: il Service Worker potrà infatti controllare tutte e sole le pagine collocate nella cartella dove si trova il suo script o in sue sottocartelle^[28]. Gli asset governati dal Service Worker prendono il nome di *clients*^[28].

Per ragioni di sicurezza, i Service Worker possono essere utilizzati solo su comunicazioni HTTPS: comunicazioni non criptate come HTTP sono, infatti, maggiormente soggette ad attacchi informatici, che potrebbero risultare ancora più dannosi se fosse garantito loro accesso alla *Service Worker API*^[27].

[¶]Un *Worker* rappresenta un’azione in background che può essere creata tramite script^[26]

Per entrare in servizio, il Service Worker dev'essere esplicitamente registrato dall'app mediante la funzione `serviceWorkerContainer.register(scriptURL)`, dove `scriptURL` è il percorso del file contenente il suo codice. La funzione restituisce una JavaScript `Promise`[¶] che si avvera con il download del suo script^[30].

A un Service Worker è associato un determinato *ciclo di vita*, che comincia col verificarsi degli eventi `install` (che viene lanciato quando il suo codice è compilato con successo) e `activate`, entrambi i quali possono avvenire una sola volta^[30]. Anche se il Service Worker è attivo, esso non è immediatamente in grado di controllare la navigazione nel suo scope: per garantirgli il controllo sarà necessario ricaricare il sito. Questo meccanismo è implementato per permettere di avere consistenza all'interno dell'app: infatti, se la pagina web è stata caricata senza Service Worker, allora tutti gli altri contenuti saranno caricati in questo modo^[30]. È possibile bypassare questo ostacolo chiamando, in fase di attivazione, la funzione `clients.claim()`, mediante la quale il Service Worker prende il controllo di tutti i client non governati^[30].

Ogni volta che l'utente naviga verso una pagina contenuta nello scope, il browser confronta byte per byte lo script del Service Worker in esecuzione con il codice di quello disponibile online e, se vengono trovate delle differenze, interpreta l'evento come la disponibilità di una nuova versione^[30]. Questa nuova versione verrà registrata e installata in background, poi entrerà nello stato di `waiting`, in cui attende che il Service Worker corrente non controlli più alcun client per poter così prendere il suo posto. Una volta superato questo stadio, la nuova versione sarà finalmente attiva^[30]. Questo comportamento serve a evitare che l'utente acceda a due versioni diverse dell'app in contemporanea, cosa che può causare gravi instabilità al servizio; un altro vantaggio di questo meccanismo è che se l'installazione del nuovo Service Worker fallisce, l'utente non percepisce alcuna interruzione di servizio, in quanto la versione corrente continuerà a eseguire senza problemi^[30]. È comunque disponibile la funzione `ServiceWorkerGlobalScope.skipWaiting()`, che permette al nuovo Service Worker di rimpiazzare il precedente non appena esso ha finito di installare^[30].

Il Service Worker può ascoltare gli eventi `install` e `activate`, all'interno dei quali può eseguire operazioni aggiuntive come preparare la cache o rimuovere quella associata alla versione precedente; entrambi gli eventi sono sottoclassi di `ExtendableEvent`, il quale dispone del metodo `waitFor(promise)`, che si occuperà di estendere l'evento, mettendo in coda tutti quelli successivi, fino a quando la Promise passata per argomento non si sarà conclusa^[27]. Una volta attivo, il Service Worker può intercettare le `Request` effettuate dalle pagine nel suo scope ascoltando l'evento `fetch`, il quale possiede il metodo `respondWith(response)`, che permette di definire il contenuto da fornire in risposta alla richiesta; tale contenuto è rappresentato come istanza della classe `Response`^[27]. Se la risposta non è immediatamente disponibile, è possibile fornire, come argomento di `respondWith()`, una Promise che si avverrà in una `Response`^[27].

In DevTools è possibile visualizzare la lista dei Service Worker attivi o in attesa cliccando su "Application" e poi "Service worker" (Valido sia per Firefox che per browser Chromium-based). Su browser Chromium-based è possibile anche simulare la mancanza di connessione (senza scollegare la macchina dalla rete) spuntando l'opzione "offline". Per gli sviluppatori è disponibile anche la funzione "update on reload", la quale permette di aggiornare il Service Worker ogni volta che viene ricaricata la pagina (senza dover chiudere e riaprire l'app).

2.6 Caching

La **Cache Web** è un'area nello spazio di archiviazione del dispositivo dell'utente usata per immagazzinare asset e altri dati, cosicché, quando necessario, questi ultimi possano essere recuperati dalla memoria anziché dalla rete. Il suo utilizzo ha il vantaggio di ridurre i tempi di caricamento (infatti reperire una risorsa dalla cache anziché dal server remoto richiede meno tempo) e di diminuire il traffico della rete. La cache è una tecnologia largamente utilizzata nelle web app, in quanto permette a queste ultime di continuare a interagire con l'utente anche in assenza di connessione a internet: i dati salvati in cache, infatti, sono reperibili anche quando il dispositivo è offline, per cui il Service Worker può rispondere alle network request con i dati

[¶]Una JavaScript `Promise` è un oggetto che rappresenta un evento asincrono: essa può essere *pending*, se l'evento non si è ancora concluso, *fulfilled*, se l'evento è stato completato con successo (il metodo `then()` permette di definire la callback function da eseguire in tal caso), oppure *rejected*, se si è verificato un errore (che può essere gestito mediante il metodo `catch()`)^[29].

contenuti in cache^[31].

Come per le altre forme di WebStorage viste precedentemente, anche la cache è organizzata come insieme di coppie chiave-valore, con il vincolo che la prima dev'essere un oggetto di tipo *Request* e il secondo dev'essere un'istanza di *Response*.

La cache non presenta limiti di spazio; inoltre, i dati in essa salvati sono permanenti. Tuttavia, come per qualunque altra forma di Web Storage, esistono delle situazioni in cui il suo contenuto può essere automaticamente rimosso: la maggior parte dei browser, infatti, definisce un tetto massimo allo spazio occupato complessivamente dai vari domini, il quale coincide con metà dello spazio disponibile sul dispositivo dell'utente^[32]. Se un'operazione di salvataggio dei dati causa il superamento di questo limite, allora il browser elimina completamente tutti i dati immagazzinati da una determinata origine (scelta con politica *Least Recently Used*), finché lo spazio liberato non è sufficiente a soddisfare tale operazione^[32]. Come accennato prima, questa politica non si applica solo alle cache ma a tutte le forme di Web Storage (inclusi Local Storage, Session Storage e IndexedDB).

Quando l'app richiede una risorsa proveniente da un'altra origine, essa verrà memorizzata in cache come *opaque response*, il che vuol dire che il browser non sarà in grado di determinare la dimensione della risorsa e pertanto la stima dello spazio occupato differisce dal valore reale^[31].

In DevTools è possibile visualizzare e manipolare il contenuto delle cache gestite da un determinato sito cliccando su "Application" e poi "Cache storage" (per browser Chromium-based) oppure su "Storage" e poi "Cache Storage" (in Firefox).

2.6.1 Implementazione

La *Cache Storage API* è l'interfaccia che permette la gestione di tutte le cache aperte in una determinata origine**^[33]; essa può essere acceduta sia dal thread dell'app che da quello del Service worker (o di qualunque altro Worker)^[31]. Siccome le operazioni su cache sono asincrone, la Cache Storage API è basata su Promise.

L'API è composta da due classi: *CacheStorage*, che fornisce un'interfaccia per la gestione di tutte le cache nel proprio dominio^[33], e *Cache*, che permette invece di gestire il contenuto di una singola cache^[34].

Cache dispone dei metodi

- **add(request)**, che crea una Promise che si avvera con l'esecuzione della request passata per parametro e il salvataggio della relativa risposta in cache, usando come chiave la request stessa^[34]
- **addAll(requests)**, che crea una Promise che si avvera con il salvataggio in cache di tutte le risposte relative alle richieste presenti nell'array **requests**^[34]
- **delete(request)**, che crea una Promise che si avvera con la rimozione dell'elemento associato a **request** (nel caso in cui l'elemento non è trovato, si avvera senza eseguire alcuna modifica)^[34]
- **keys()**, che crea una Promise che si avvera con la restituzione di un array contenente tutte le chiavi presenti in cache^[34]
- **match(request)**, che crea una Promise che si avvera con la restituzione dell'elemento associato a **request** (la Promise si avvera anche nel caso in cui la chiave non sia trovata, in tal caso restituisce **undefined**)^[34]
- **put(request, response)**, che crea una Promise che si avvera con l'inserimento di **response**, associandolo alla chiave **request**^[34]

CacheStorage, invece, dispone dei metodi

**anche la cache, come gli IndexedDB, segue una *same-origin policy*^[31]

- `open(cacheName)`, che crea una Promise che si avvera con l'apertura della connessione con la cache specificata da `cacheName` (creandola nel caso non fosse già presente) e con la restituzione di un oggetto di tipo `Cache` contenente il riferimento alla cache aperta^[33]
- `delete(cacheName)`, che crea una Promise che si avvera con l'eliminazione della cache specificata (se la cache non è stata trovata, si avvera senza eseguire niente)^[33]
- `has(cacheName)`, che crea una Promise che si avvera restituendo `true` se è presente una cache col nome specificato, `false` altrimenti^[33]
- `keys()`, che crea una Promise che si avvera con la restituzione di un array contenente i nomi di tutte le cache presenti in quel dominio^[33]
- `match(request)`, che crea una Promise la quale ispeziona tutte le cache alla ricerca di una chiave che coincida con `request`: se trovata allora la Promise si avvera restituendo la Response a essa associata, altrimenti si avvera restituendo `undefined`^[33]

2.7 Installazione

L'installazione di una PWA avviene solitamente in modo diverso da una applicazione nativa. Malgrado il criterio di installazione vari da browser a browser, la maggior parte dei requisiti vengono descritti da un file JSON chiamato **Web App Manifest**.

2.7.1 Prompt di installazione

Il prompt di installazione viene solitamente generato automaticamente, però è possibile gestirlo con l'evento `beforeInstallPromptEvent`. Di default il browser avvia il prompt se vengono soddisfatti i seguenti requisiti:

- Un service worker attivo (per abilitare il funzionamento offline).
- Un manifest file valido (con metadati come nome, icona e tema).
- Navigazione su HTTPS (per garantire sicurezza).
- Interazione dell'utente con il sito (per evitare prompt intrusivi).

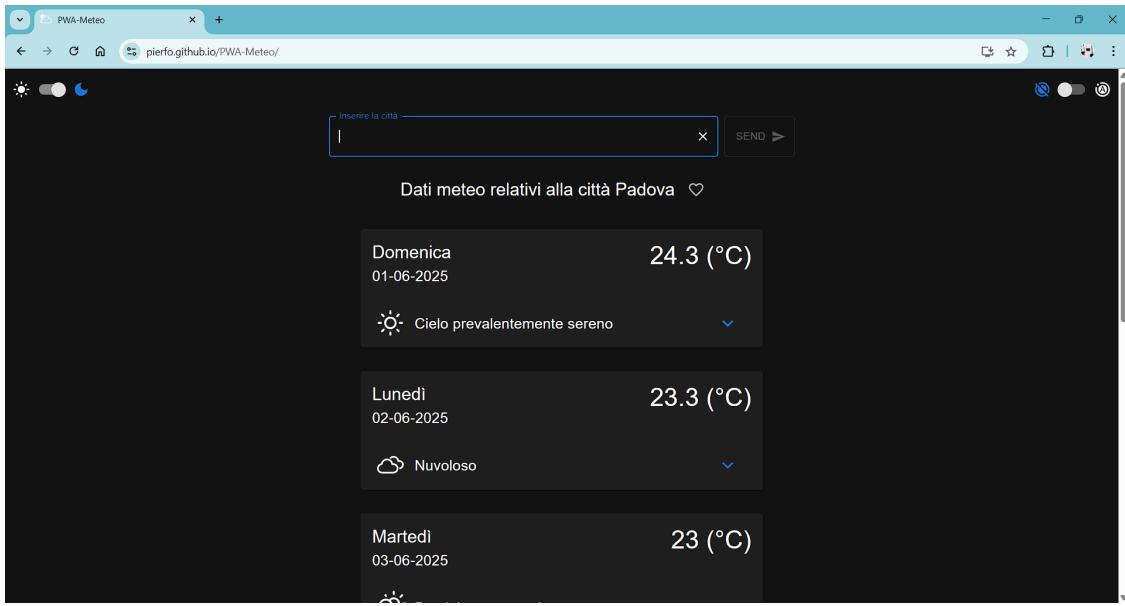
Come spiegato prima, è possibile intercettare l'evento `beforeinstallprompt` con un `listener` nel service worker e mandarlo in azione quando si vuole, ad esempio quando un utente interagisce con un tasto di installazione customizzato. Di seguito mostriamo un esempio tratto da `web.dev`.^[35]

```
// This variable will save the event for later use.
let deferredPrompt;
window.addEventListener('beforeinstallprompt', (e) => {
  // Prevents the default mini-infobar or install dialog from appearing on mobile
  e.preventDefault();
  // Save the event because you'll need to trigger it later.
  deferredPrompt = e;
  // Show your customized install prompt for your PWA
  // Your own UI doesn't have to be a single element, you
  // can have buttons in different locations, or wait to prompt
  // as part of a critical journey.
  showInAppInstallPromotion();
});
```

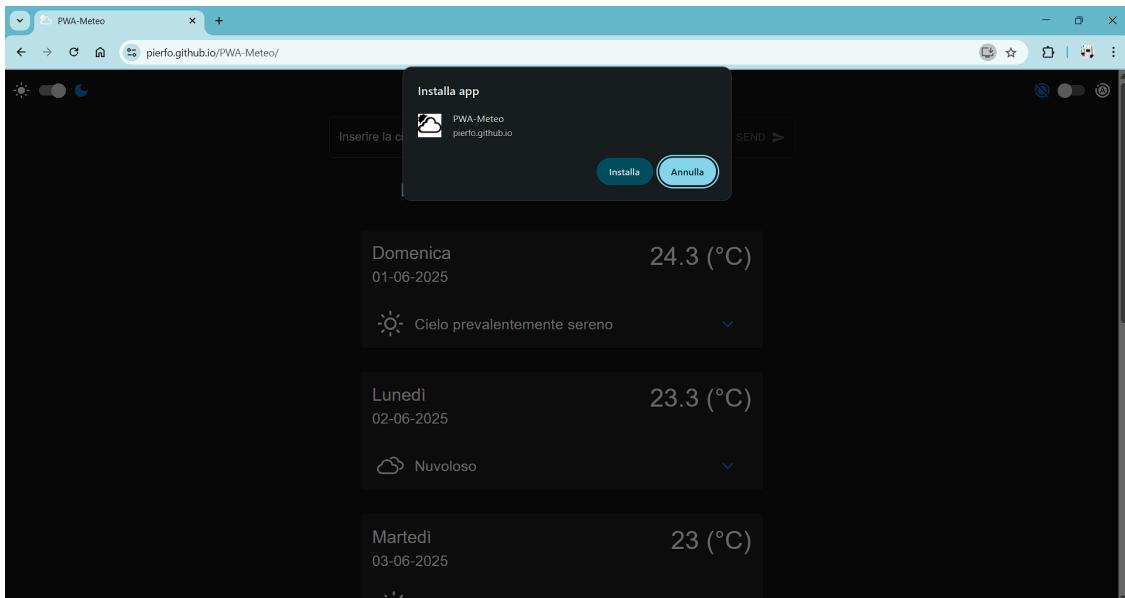
Suddivideremo ora l'installazione di una PWA nei casi di dispositivo desktop, iOS e Android.

2.7.2 Installazione su Desktop

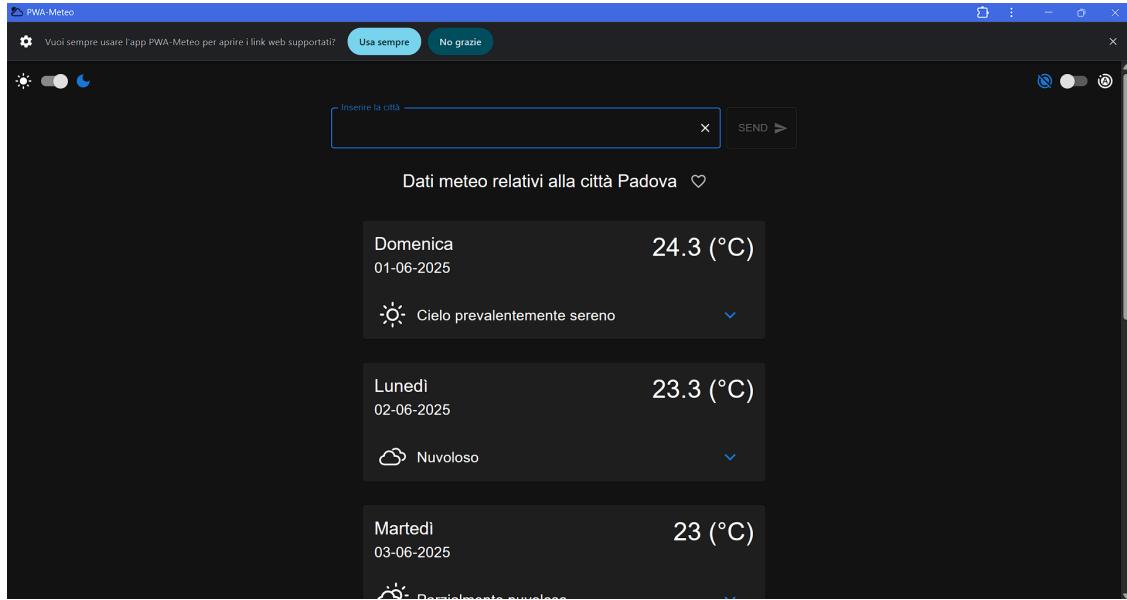
Se una PWA raggiunge i requisiti minimi di installazione, il browser permette la creazione di una piccola icona sulla barra di ricerca (nel caso di Chrome) e interagendo con essa possiamo installare la PWA. In seguito vedremo alcuni esempi presi dalla PWA del progetto, PWA-Meteo.



Una volta selezionata l'icona di installazione, viene visualizzato un prompt che chiede all'utente di confermare l'installazione.



Dopo la finalizzazione la PWA viene visualizzata come un programma installato localmente nella barra di navigazione e nella lista delle applicazioni di Windows.



Grazie al Web App Manifest, la PWA installata possiede un ID univoco e dunque potrà essere indirizzata nella ricerca delle applicazioni del sistema. Questo comporta anche che non sarà possibile installare la stessa PWA due volte consecutivamente dallo stesso browser, senza prima disinstallare la prima copia.

2.7.3 Installazione su iOS

Nel caso di iOS, il browser non permette la creazione di un prompt d'installazione, però è sempre possibile aggiungerle alla schermata Home tramite il menu 'Share'. Queste PWA installate su iOS hanno funzionalità limitate e compaiono solo nella lista delle applicazioni della App Library e Home screen mentre non sono presenti nella lista di App Gallery.

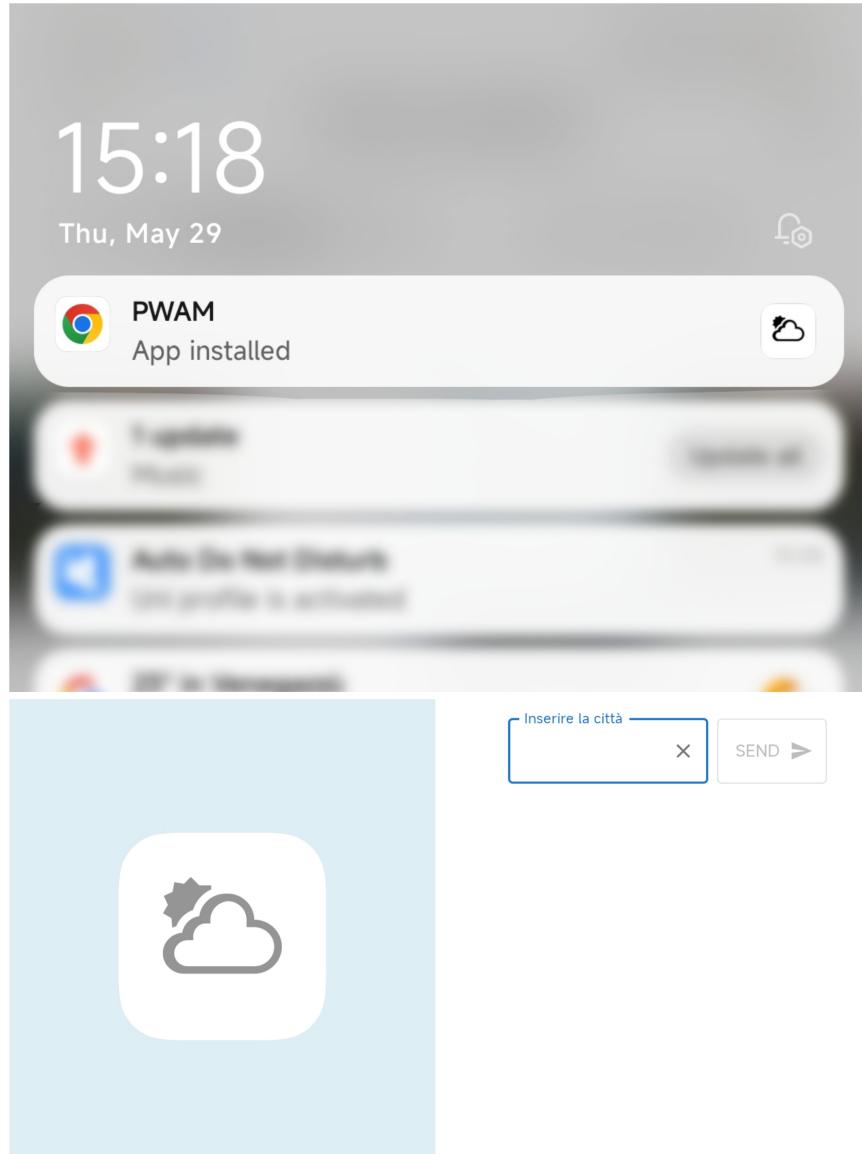


2.7.4 Installazione su Android

Nel browser Chrome per Android, se una PWA soddisfa i requisiti di installazione, un prompt appare come notifica pop up nella parte superiore dello schermo (immagine 1 sotto).



Premendo 'Install' viene visualizzata una finestra pop up che chiede conferma per l'installazione (immagine 2 sopra). Se viene confermata la scelta, l'applicazione viene installata e un messaggio di installazione viene trasmesso in basso allo schermo che indica che il sistema operativo sta installando la PWA (immagine 3 sopra).



Una volta finalizzata la PWA, il browser invia una notifica all’utente annunciando l’avvenuta installazione. Aprendo la PWA dalla notifica viene visualizzata una pagina di caricamento che di default mostra l’icona e il colore del tema scelto come sfondo. Una volta caricata l’app per la prima volta, è possibile visualizzare i contenuti salvati localmente anche in modalità offline.

La modalità di installazione in verità dipende da dispositivo in dispositivo e anche dal firmware del sistema operativo. Quando una PWA viene installata tramite Google Chrome o su dispositivi Samsung, Pixel, OnePlus e Xiaomi, il browser genera automaticamente una WebAPK tramite il trusted provider indicato di default (In questo caso Google Mobile Services) che si occupa di firmare e ‘impacchettare’ (minting) la PWA in una pacchetto Android. Questi pacchetti android vengono considerati ‘sicuri’ dal sistema operativo in quanto generati dal trusted provider e dunque non necessitano il processo di verifica che avviene quando si installano applicazioni direttamente da APK non provenienti da Google Play Store.

Nel caso di browser che non hanno accesso diretto a un trusted provider, come nel caso di Firefox, Microsoft Edge, Opera, Brave, and Samsung Internet su dispositivi non Samusung, allora il broser cerca di creare un ‘collegamento’ sulla schermata home. Anche in caso di Chrome, se la PWA non soddisfa i requisiti di installazione, viene utilizzato questo metodo.

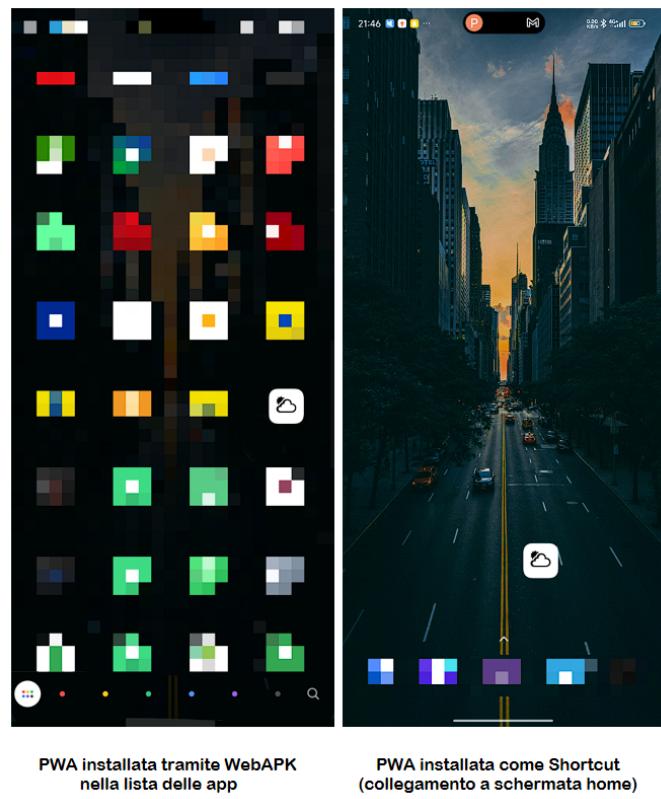


Figure 8: Confronto tra WebAPK e shortcut

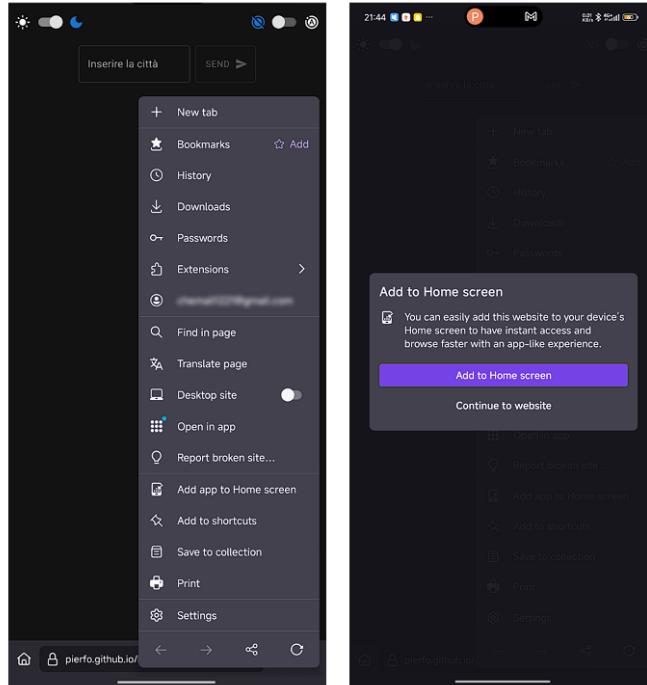


Figure 9: Installazione tramite Firefox Android con la funzione Add app to homescreen

2.8 Web App Manifest

Come spiegato precedentemente, una PWA utilizza un file JSON chiamato Web app manifest per gestire il funzionamento della PWA e il riconoscimento da parte del browser come tale. Il manifest di solito contiene informazioni come nome dell'app e come si comporterà come applicazione una volta installata. Inoltre definisce metadati come nome, colore del tema e modalità di visualizzazione, il colore del background durante la schermata di caricamento e l'icona in varie situazioni di visualizzazione nel sistema operativo. Questo file viene spesso salvato con l'estensione `.webmanifest`, ma può avere anche l'estensione `.json` come ogni file JSON. Di seguito vedremo una panoramica dei principali campi contenuti nel manifest e come interagiscono con l'installabilità e corretto funzionamento della PWA.

```

1  {
2      "name": "My Progressive Web App",
3      "short_name": "MyPWA",
4      "description": "A sample PWA for demonstration purposes",
5      "start_url": "/",
6      "display": "standalone",
7      "display_override": ["minimal-ui"],
8      "background_color": "#ffffff",
9      "theme_color": "#3367D6",
10     "icons": [
11         {
12             "src": "icons/icon-192.png",
13             "sizes": "192x192",
14             "type": "image/png"
15         },
16         {
17             "src": "icons/icon-512.png",
18             "sizes": "512x512",
19             "type": "image/png"
20         }
21     ],
22     "scope": "/",
23     "orientation": "portrait",
24     "lang": "en-US",
25     "dir": "ltr",
26     "categories": ["productivity", "utilities"],
27     "screenshots": [
28         {
29             "src": "screenshots/home.png",
30             "sizes": "1280x720",
31             "type": "image/png"
32         }
33     ]
34 }
```

2.8.1 Campi principali

- **name:**
Il nome completo dell'applicazione, utilizzato nell'install prompt e nello splash screen.
- **short_name:**
Una versione abbreviata del nome, usata quando lo spazio è limitato. A volte viene utilizzato nella barra delle applicazioni o nella schermata home del dispositivo.
- **start_url:**
Specifica la pagina che viene caricata all'avvio dell'app (es. "/" o "/index.html").
- **background_color:**
Colore di sfondo utilizzato durante il caricamento e nello splash screen (deve essere un valore HEX, es. "#ffffff").
- **theme_color:**
Definisce il colore della barra degli strumenti e della UI del browser (deve essere un valore HEX).
- **display:**
Definisce la modalità di visualizzazione dell'app. I valori possibili sono:
 - "fullscreen" (nasconde tutta l'UI del browser)
 - "standalone" (simula un'app nativa, nascondendo la barra degli indirizzi)
 - "minimal-ui" (mostra alcuni controlli di navigazione)
 - "browser" (comportamento standard del browser)
- **display_override:**
Permette di usare una modalità display preferito, però se non è possibile usare questa modalità di visualizzazione viene utilizzata quella di default definita da **display**^[36]
- **scope:**
Definisce l'ambito di navigazione dell'app. Se l'utente naviga fuori da questo percorso, l'app si comporta come un normale sito web.
- **orientation:**
Forza un orientamento specifico ("portrait", "landscape", o "any").
- **lang:**
Specifica la lingua principale dell'app (es. "en-US").
- **dir:**
Definisce la direzione del testo ("ltr" per sinistra-a-destra, "rtl" per destra-a-sinistra).
- **categories:**
Un array di categorie per classificare l'app (es. ["productivity", "business"]).
- **screenshots:**
Array di screenshot mostrati negli store delle PWA (Chrome Web Store, Microsoft Store, etc.).

2.8.2 Icone

L'attributo **icons** è un array di oggetti che definisce le icone dell'app. Ogni icona deve includere:

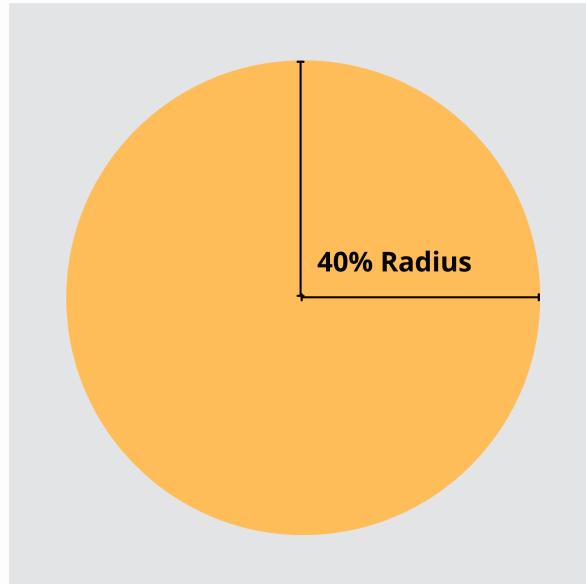
- **src** (percorso dell'immagine)
- **sizes** (dimensioni in pixel, come ad esempio 48x48, 72x72, 96x96, 144x144, 168x168, 256x256, 512x512, 1024x1024...)
- **type** (formato, es. "image/png")

L'icona viene utilizzata in diverse situazioni, come nella schermata iniziale, nel launcher dell'app o nei task manager del sistema operativo.

Per adattare l'icona alle esigenze del sistema operativo o del browser, nel manifest si possono specificare

anche icone con diversi utilizzi. Uno di questi campi è **purpose** che indica come viene visualizzata l'icona. Per esempio può essere:

- **monochrome** icona con background di colore in base la material ui theme del sistema operativo o con sfondo ripieno.
- **maskable** icona con possibilità di applicarci maschere. Di solito le immagini di queste immagini hanno una zona sicura centrale (con il logo o simbolo icona) e il resto può essere nascosto senza compromettere con l'aspetto desiderato.



A volte è necessario modificare l'immagine per renderla adatta all'attributo **maskable** senza influenzare l'aspetto finale. Le icone **maskable** devono essere almeno di 512x512 di dimensione. [37] [35]



Figure 10: Cattivo esempio di icona maskable

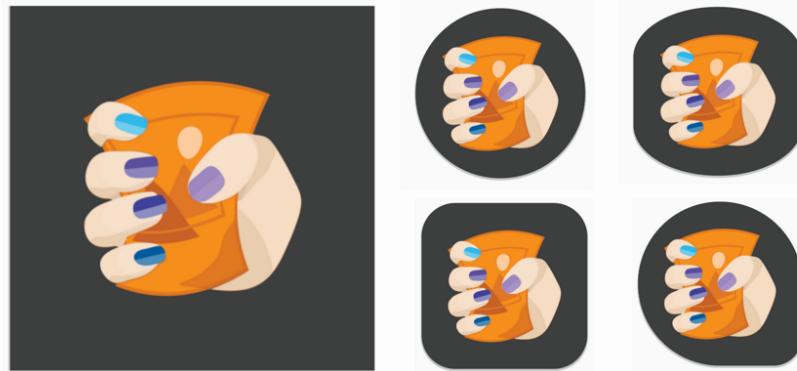


Figure 11: Buon esempio di icona con padding abbastanza largo da permettere il ritaglio

- `any` valore di default, può essere utilizzato in ogni contesto.

I campi di purpose possono essere aggiunti uno dopo l'altro per indicare più casi di utilizzo:

```
{  
  "icons": [  
    {  
      "src": "/icons/144.png",  
      "type": "image/png",  
      "sizes": "144x144"  
    },  
    {  
      "src": "icons/512.png",  
      "sizes": "512x512",  
      "type": "image/png"  
      "purpose": "monochrome"  
    },  
    {  
      "src": "icons/512-m.png",  
      "sizes": "512x512",  
      "type": "image/png"  
      "purpose": "any maskable"  
    }  
  ]  
}
```

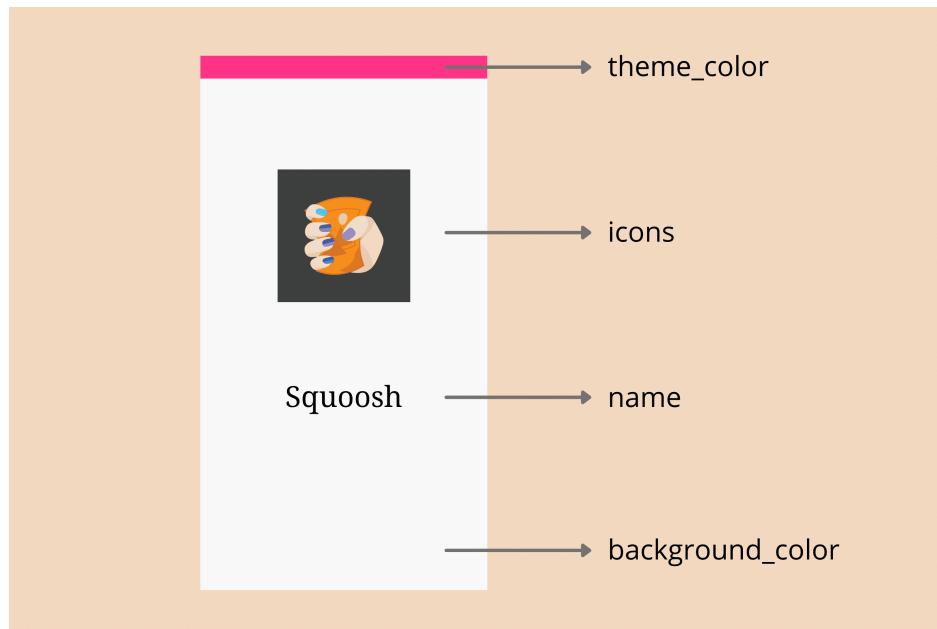


Figure 12: Componenti grafiche nella pagina di caricamento di default quando una PWA viene aperta

2.8.3 Linking del manifest

Il file manifest viene collegato alla pagina html della pagina web principale attraverso la seguente linea di codice:

```
<link rel="manifest" href="/app.webmanifest">
```

2.9 Strumenti di Testing per PWA

Gli strumenti per il testing delle Progressive Web App permettono di verificare il corretto funzionamento delle componenti fondamentali delle PWA:

- **Installabilità:** Verifica della presenza e correttezza del Web App Manifest
- **Funzionalità Offline:** Test del Service Worker e delle strategie di caching
- **Performance:** Analisi dei tempi di caricamento e interattività

2.9.1 Cross-platform Testing

Uno dei modi per testare il funzionamento in diverse piattaforme è attraverso il testing diretto di come vengono visualizzate le pagine della PWA in browser diversi e formati di schermo variabili. Questo può essere raggiunto con simulatori come in Chromium DevTools con `Device mode` o in Firefox e Safari con `Responsive Design Mode`.^[35]

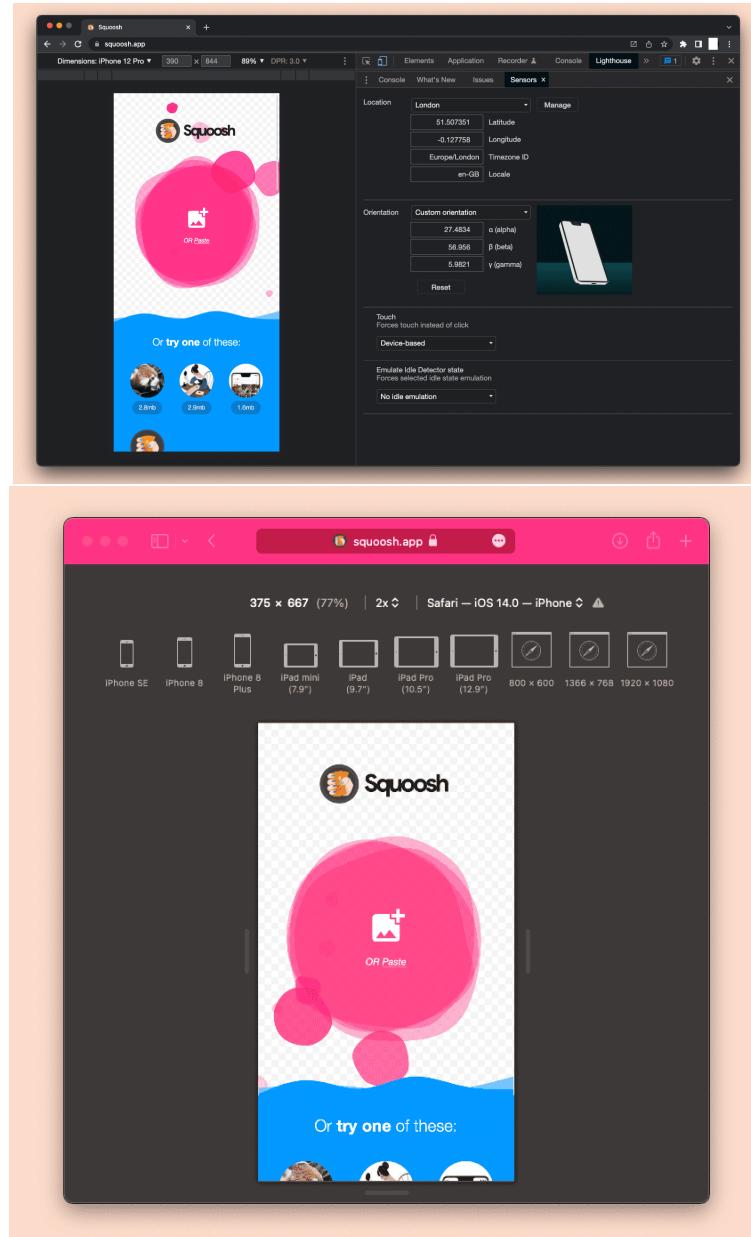


Figure 13: Safari Web Inspector: Responsive Design Mode^[35]

Un'altra categoria di strumenti disponibili sono gli emulatori di sistemi operativi dove possiamo verificare il funzionamento non solo in browser e schermi diversi, ma anche in ambiente nativo. Esistono vari emulatori per iOS come Xcode e altrettanti emulatori android come quelli offerti da Android Studio o dall'Android SDK: SDK Manager e AVD Manager.

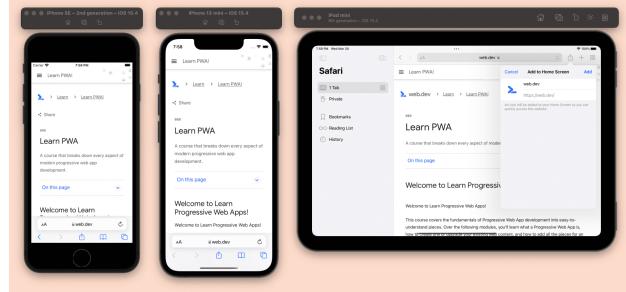


Figure 14: iOS emulator^[35]

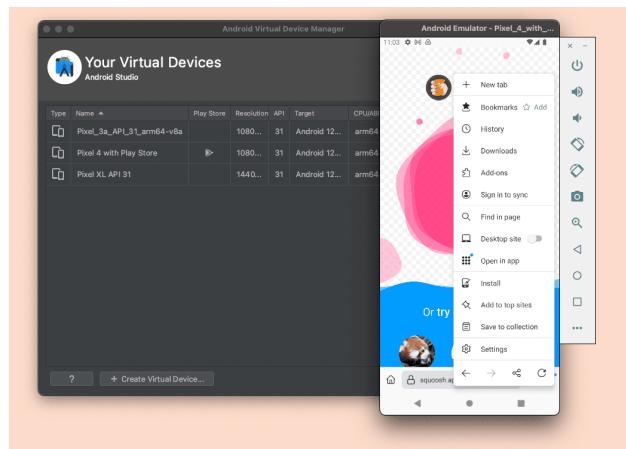


Figure 15: Android emulators^[35]

2.10 Lighthouse



Lighthouse è uno strumento open-source automatizzato sviluppato da Google per il miglioramento della qualità delle applicazioni web. Fornisce audit specifici per PWA attraverso una serie di controlli:

2.10.1 Metodologia di Analisi

Lighthouse esegue una valutazione strutturata in 5 categorie:

1. **Performance:** Metriche come First Contentful Paint (FCP) e Time to Interactive (TTI)
2. **Accessibilità:** Compatibilità con screen reader e contrasto colori
3. **Best Practices:** Uso di HTTPS, corretta gestione dei cookie
4. **SEO:** Ottimizzazione per motori di ricerca
5. **PWA:** Verifica dei requisiti fondamentali

2.10.2 Audit Specifici per PWA

Per le Progressive Web App, Lighthouse controlla:

- Presenza del file manifest con metadati essenziali (nome, icone, theme_color)
- Registrazione corretta del Service Worker
- Funzionamento in condizioni di rete non affidabile
- Dimensioni delle icone (minimo 192px)
- Utilizzo di HTTPS

2.10.3 Modalità d'Uso

Lighthouse è disponibile in tre varianti:

- **Chrome DevTools:** Integrato direttamente nel browser (tab "Lighthouse")
- **Node.js CLI:** Per integrazione in pipeline CI/CD **Estensione Chrome:** Per test rapidi durante lo sviluppo

2.10.4 Altri Strumenti Raccomandati

- **Workbox:** Libreria per debug avanzato del Service Worker
- **WebPageTest:** Analisi delle performance in condizioni di rete simulata
- **DevTools Application Panel:** Verifica manuale del Manifest e della Cache

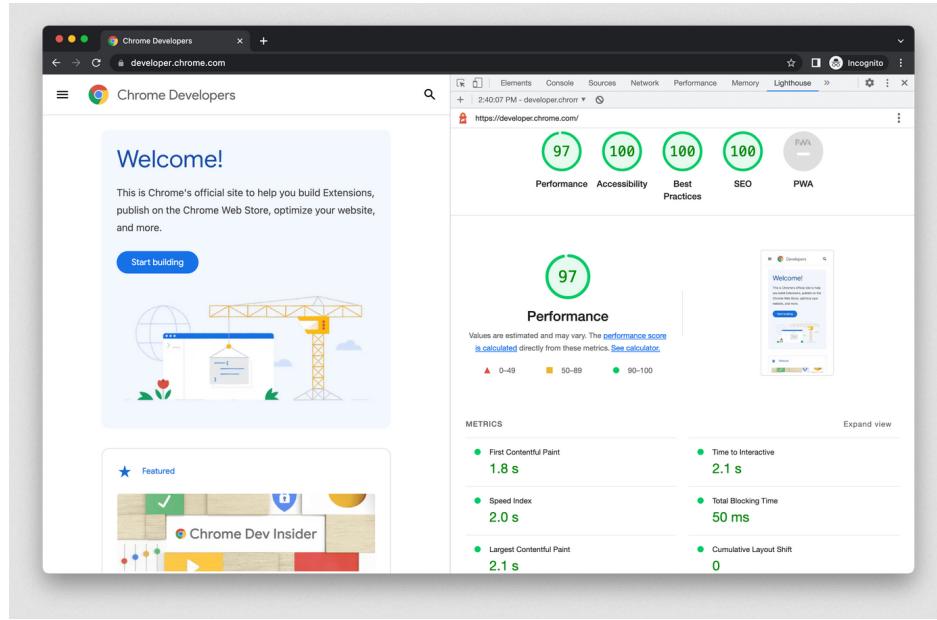


Figure 16: Esempio di risultato di test Lighthouse

2.10.5 Interpretazione dei Risultati

I report di Lighthouse utilizzano un sistema a punteggio (0-100) con:

- **Verde (90-100)**: Implementazione ottimale
- **Arancio (50-89)**: Miglioramenti consigliati
- **Rosso (0-49)**: Problemi critici da risolvere

Per le PWA, un punteggio superiore a 90 nella sezione dedicata garantisce compatibilità con i principali store PWA (Microsoft Store, Google Play).^[38]

3 Conclusione

Le PWA combinano tecnologie web avanzate per offrire esperienze simili alle app native, con vantaggi tangibili in termini di performance, accessibilità e riduzione dei costi. Aziende leader hanno sfruttato queste caratteristiche per raggiungere un pubblico più ampio e migliorare metriche chiave, dimostrando la scalabilità e l'efficacia del modello PWA in settori diversificati e dando prova di essere una soluzione efficace per lo sviluppo cross-platform. Dunque i casi studio analizzati (Twitter, Starbucks, Uber) confermano che le PWA non sono più solo prototipi accademici, ma strumenti produttivi adottati da industry leader. Difatti l'integrazione con gli ecosistemi mobile attraverso le Trusted Web Activities ne ampliano ulteriormente il potenziale.

Questo lavoro dimostra che le PWA costituiscono oggi la scelta ottimale per progetti che richiedono:

- Rapido time-to-market
- Ampia copertura di dispositivi
- Budget contenuti

Con l'evoluzione degli standard web e il miglioramento del supporto browser, le PWA sono destinate a diventare il modello dominante per lo sviluppo di applicazioni consumer e business.

References

- [1] “Wikipedia - progressive web app, url: https://en.wikipedia.org/wiki/Progressive_web_app.”
- [2] “Google developers - from pwa to twa: Adding pwa to google’s play store, url: <https://developers.google.com/codelabs/pwa-in-play/>.”
- [3] “Alex russell - progressive web apps: Escaping tabs without losing our soul, url: <https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/>.”
- [4] “web.dev - twitter lite pwa significantly increases engagement and reduces data usage, url: <https://web.dev/case-studies/twitter>.”
- [5] “web.dev - progressive web apps, url: <https://web.dev/explore/progressive-web-apps>.”
- [6] V. Aguirre, L. Delía, P. Thomas, L. Corbalán, G. Cáseres, and J. F. Sosa, “Pwa and twa: Recent development trends,” in *Computer Science – CACIC (2019) URL: https://link.springer.com/chapter/10.1007/978-3-030-48325-8_14*, 2019.
- [7] “React - la libreria per le interfacce utente web e native, url: <https://it.react.dev>.”
- [8] “Microsoft - overview of progressive web apps (pwas), url: <https://learn.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium/>.”
- [9] “Techradar - definition of progressive web app (pwa), url: <https://techradar.softwareag.com/technology/progressive-web-apps/>.”
- [10] “Nearform - starbucks ordering and store locator progressive web app, url: <https://nearform.com/work/starbucks-progressive-web-app/>.”
- [11] “Codewave - successful examples of progressive web apps, url: <https://codewave.com/insights/examples-progressive-web-apps-success-stories/>.”
- [12] “Cookie, url: <https://it.wikipedia.org/wiki/Cookie>.”
- [13] “GDPR Cookies, Consent and Compliance, url: <https://www.cookiebot.com/en/gdpr-cookies/>.”

-
- [14] “Document: cookie property, url: <https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie>.”
 - [15] “Cookies Having independent Partitioned State (CHIPS), url: https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Privacy_sandbox/Partitioned_cookies.”
 - [16] “Using HTTP cookies, url: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies>.”
 - [17] “Prima risposta al thread *”What is the difference between SameSite=”Lax” and SameSite=”Strict””* di Stack Overflow, url: <https://stackoverflow.com/questions/59990864/what-is-the-difference-between-samesite-lax-and-samesite-strict>.”
 - [18] “Javascript cookies, url: https://www.w3schools.com/js/js_cookies.asp.”
 - [19] “Using the Web Storage API, url: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API.”
 - [20] “Window: localStorage property, url: <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>.”
 - [21] “Window: sessionStorage property, url: <https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>.”
 - [22] “IndexedDB key characteristics and basic terminology, url: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Terminology.”
 - [23] “Using IndexedDB, url: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB.”
 - [24] “Database transaction, url: https://en.wikipedia.org/wiki/Database_transaction.”
 - [25] “IndexedDB API, url: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.”
 - [26] “Worker, url: <https://developer.mozilla.org/en-US/docs/Web/API/Worker>.”
 - [27] “Service Worker API, url: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.”

-
- [28] “Service workers, url: [https://web.dev/learn/pwa/service-workers.](https://web.dev/learn/pwa/service-workers/)”
 - [29] “Promise, url: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.”
 - [30] “The service worker lifecycle, url: <https://web.dev/articles/service-worker-lifecycle>.”
 - [31] “Caching, url: <https://web.dev/learn/pwa/caching>.”
 - [32] “Browser storage limits and eviction criteria, url: https://udn.realityripple.com/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria.”
 - [33] “CacheStorage, url: <https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage>.”
 - [34] “Cache, url: <https://developer.mozilla.org/en-US/docs/Web/API/Cache>.”
 - [35] “web.dev - learn pwa, url: <https://web.dev/learn/pwa/>.”
 - [36] “Manifest incubations - draft community group report 30 may 2025, url: <https://wicg.github.io/manifest-incubations/#onappinstalled-attribute>.”
 - [37] “Mdn - mozilla developer web docs, url: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.”
 - [38] “Introduzione a lighthouse, url: <https://developer.chrome.com/docs/lighthouse/overview?hl=it>.”