PWA-Meteo

Pier Francesco Uliana, Pietro Puozzo, Enoc Ke

Documentazione per il progetto per corso di Programmazione di Sistemi Embedded 2024-2025

Abstract
PWA-Meteo è una applicazione innovativa che sfrutta le potenzialità offerte dalla tecnologia delle Progressive Web App. Essa unisce dunque la flessibilità e il range di copertura delle app web e i vantaggi in termini di funzionalità e integrazione delle applicazioni native. Difatti PWA-Meteo è realizzata con lo scopo di funzionare come una applicazione cross-platform, da cui deriva la denominazione 'Progressive', unita con le funzionalità di una applicazione installata localmente.

Contents

1 Introduzione

Panoramica sul design e progettazione di PWA-Meteo

1.1 Spunti di riferimento

Come progetto per il corso di Programmazione di Sistemi Embedded il nostro gruppo ha scelto la traccia di "Cross-Platform Development", implementando una web app che possa funzionare su dispositivi diversi e anche in assenza di internet. L'opzione migliore era realizzare una web app con la tecnologia delle Progressive Web App che permetteva l'utilizzo di tecnologie moderne come il service worker e il web manifest. Come prompt d'inizio, abbiamo posto come obiettivo quello di sviluppare un servizio completo e autosufficiente, in grado di fornire un reale valore d'uso oltre la semplice dimostrazione tecnologica. Prendendo spunto dalle applicazioni più utilizzate nella vita quotidiana, la nostra scelta è ricaduta sulla applicazione meteo, dove erano già disponibili diverse API che ne permettessero l'implementazione, come ad esempio quella di `Open-Meteo`.

La selezione dell'API Open-Meteo come base del progetto è derivata da un'approfondita valutazione di requisiti tecnici e vincoli di implementazione. I criteri di giudizio includono:

- Disponibilità dei dati senza richiesta di API key
- Copertura geografica
- Formato di risposta ottimizzato per applicazioni web
- Libertà di utilizzo (open-source)

L'architettura PWA è stata conseguentemente progettata per massimizzare l'utilizzo di queste caratteristiche, con particolare attenzione alla gestione offline dei dataset meteorologici.

1.2 Perché una PWA

Le **Progressive Web Apps (PWA)** sono applicazioni web che combinano il meglio del web e delle app native, offrendo esperienze affidabili, veloci e coinvolgenti. Come indicato nel sito di Google Developers web.dev, le PWA possono guidare il successo aziendale migliorando l'engagement degli utenti, aumentando le conversioni e riducendo i costi di sviluppo. Nel nostro caso, abbiamo scelto la modalità di sviluppo delle applicazioni progressive proprio per sperimentare con i vantaggi e le possibilità che ci offre, naturalmente su una scala minore e non aziendale. Tuttavia, questo non ci ha impedito di sfruttare molte delle funzionalità delle PWA per rendere la nostra applicazione web reattiva e corredata di funzionalità tipiche delle app native, particolarmente rilevanti per servizi dipendenti da aggiornamenti frequenti come le previsioni meteo. Di seguito ne elenchiamo alcuni.

1.2.1 Accessibilità Universale

L'ingegnere di Google Chrome Alex Russell, uno degli inventori del termine 'PWA', nel suo articolo "Progressive Apps: Escaping Tabs Without Losing Our Soul" spiega come le PWA mantengono l'essenza del web garantendo accesso immediato senza barriere di installazione. Per un'app meteo significa:[1]

- Consultazione istantanea da qualsiasi browser
- Adattabilità a formati di visualizzazione diverso

1.2.2 Performance in Rete Limitata

La presenza di un `service worker`, come spiegato nell'articolo The Offline Cookbook[2] di Jake Archibald, abilitano funzionalità essenziali per una applicazione meteo:

- Visualizzazione di dati cached durante indisponibilità di rete
- Aggiornamenti differiti quando la connettività ritorna
- Riduzione del consumo dati per utenti mobili

1.2.3 Installazione Progressiva

Il modello PWA, grazie alla presenza di un webapp manifest, permette :

- Aggiunta alla home screen dopo l'uso (senza app store)
- Esperienza app-like con display standalone

1.3 Installazione

L'installazione sui dispositivi delle PWA può variare molto da dispositivo a dispositivo. In generale, sui browser e dispositivi supportati viene generata una finestra di dialogo/finestra pop up/icona di installazione. Interagendo con il pulsante `Installa` viene inizializzato automaticamente il processo di installazione.

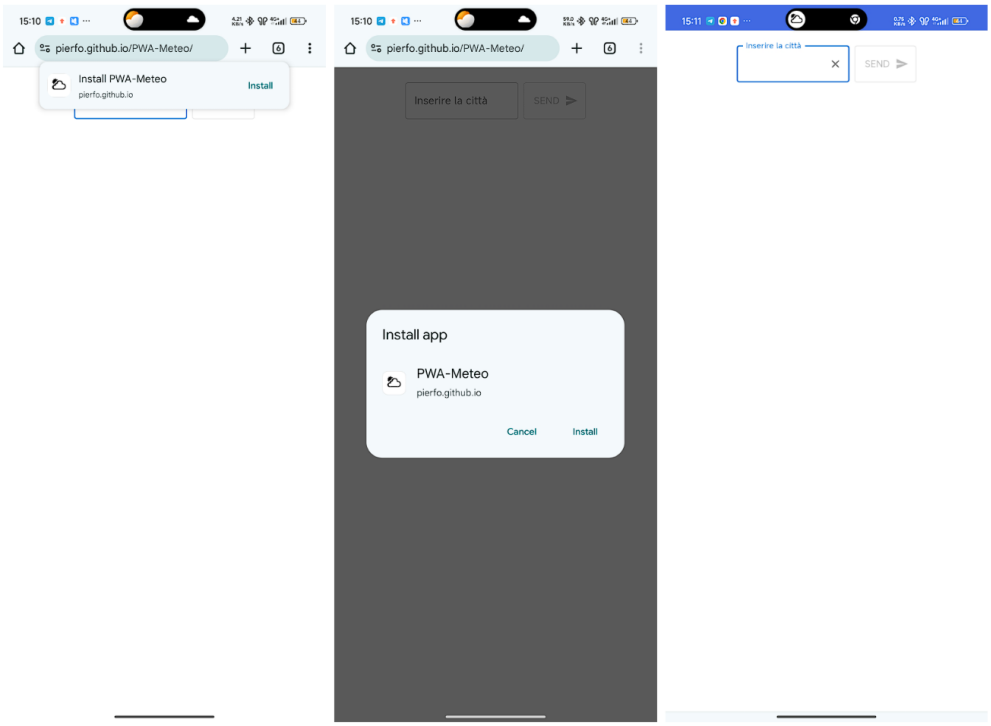
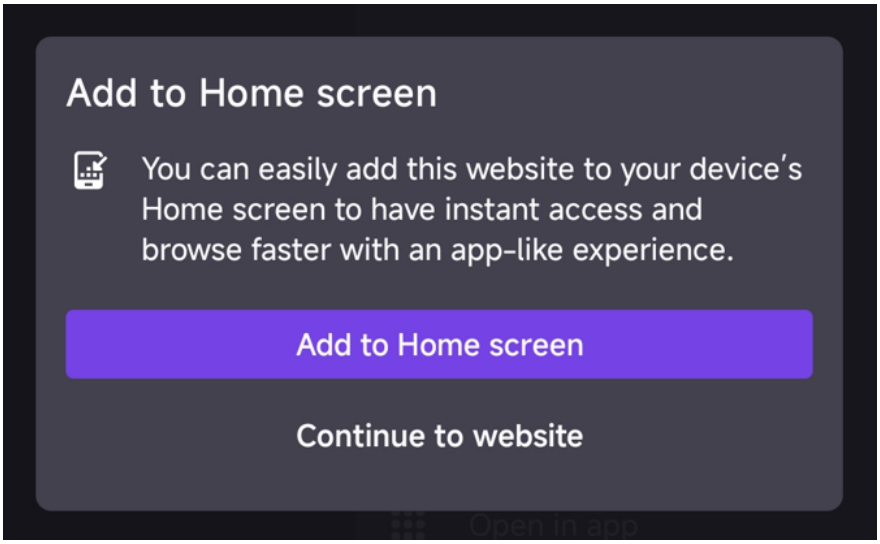


Figure 1: Schermata di installazione da dispositivo mobile

Sui dispositivi o browser non supportati è sempre possibile installare manualmente l'applicazione tramite la funzione `Aggiungi alla schermata Home` accessibile generalmente dal menu del browser.



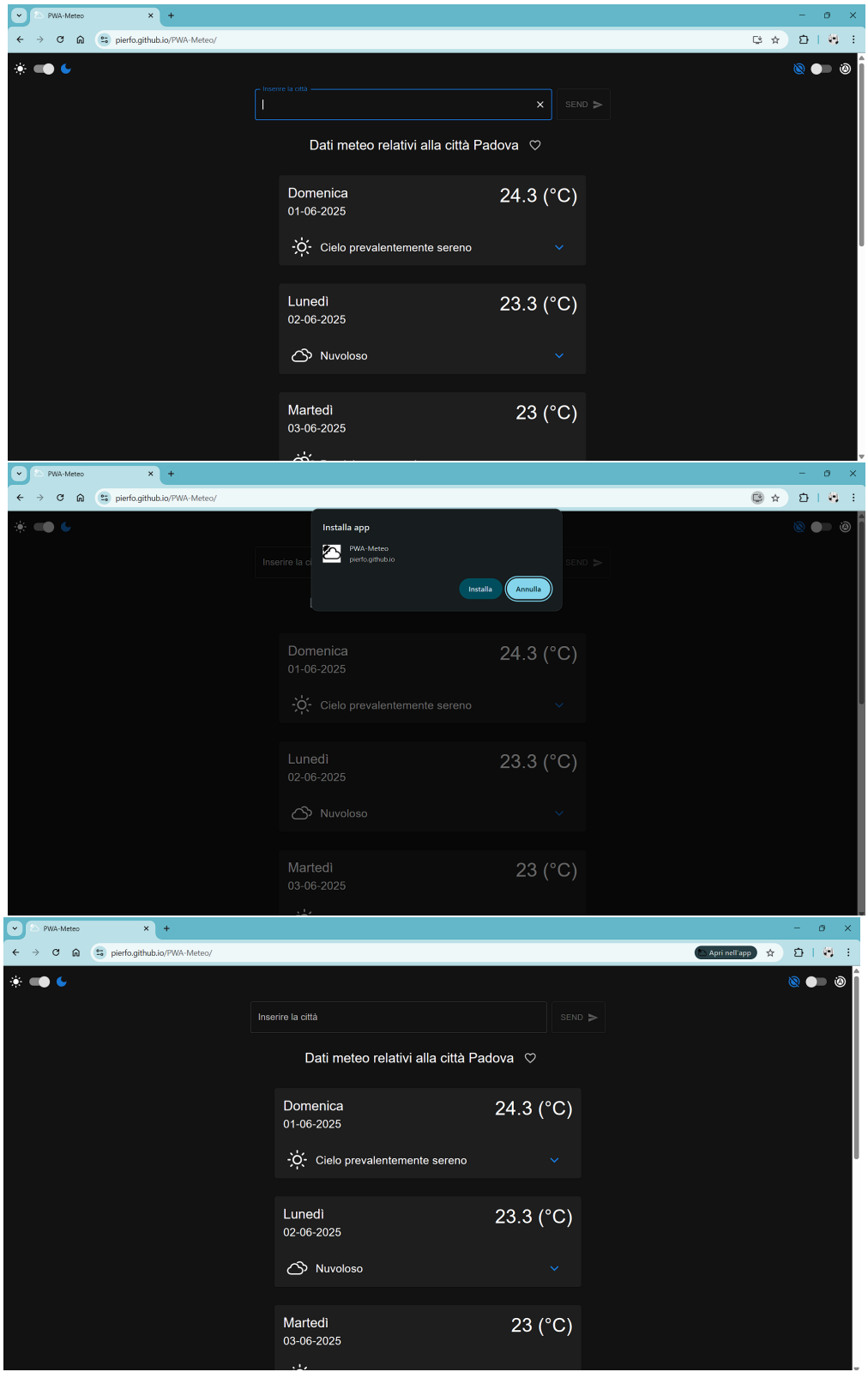


Figure 2: Schermata di installazione da desktop

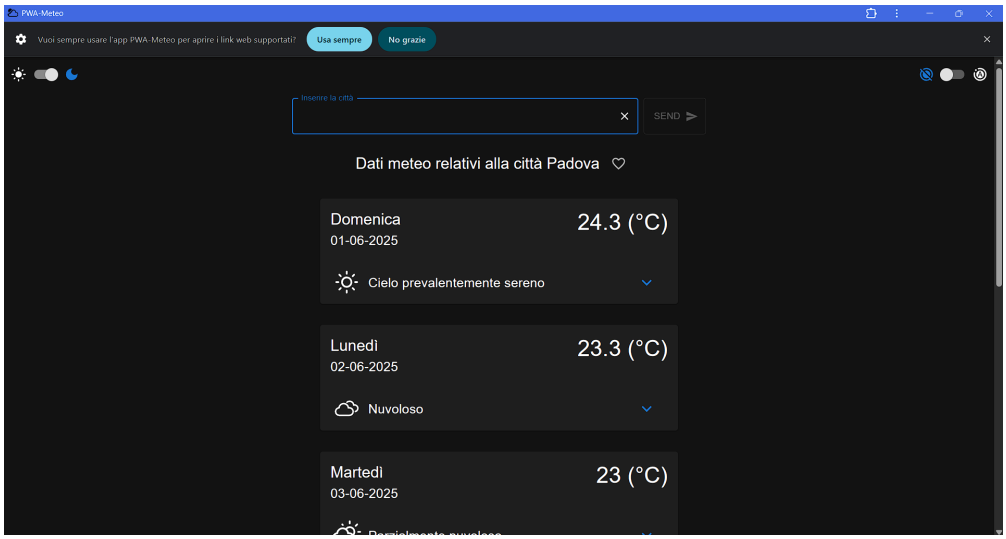


Figure 3: Una volta installato verrà visualizzata una finestra come qui sopra

1.4 Dipendenze

Per più dettagli vedere sezione 3 "Dipendenze"

- **React**: Framework che permette di dare dinamicità alla pagina web e di utilizzare i component.
- **Material-Ui React UI**: Libreria per il framework di React che contiene component utili per lo sviluppo.
- **Open-Meteo** : API open-source che fornisce dati meteo in base alle città.
- **Vite.js** : Framework che permette il testing su server locale delle pagine web.

1.5 UML

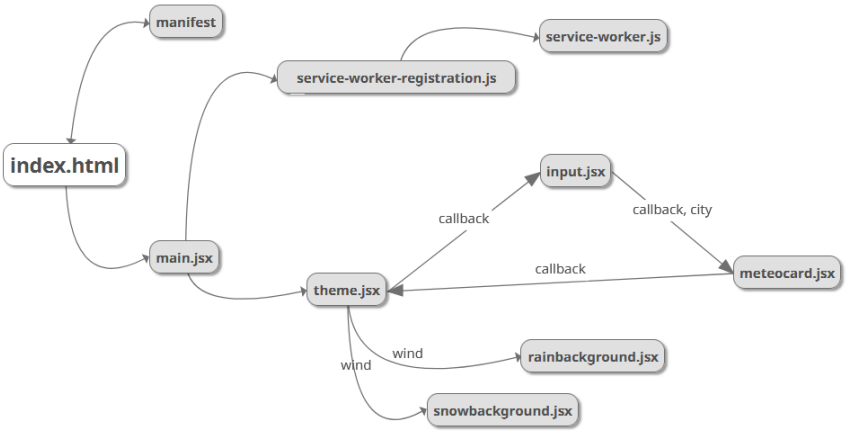


Figure 4: Diagramma rappresentante le dipendenze tra le classi

2 Manifest

Scopi e utilizzi del Manifest

2.1 Scopo Manifest

Il file `app.webmanifest` è un componente fondamentale di una Progressive Web App (PWA), in quanto fornisce al browser le informazioni necessarie per installare l'applicazione e renderla accessibile come una vera e propria app nativa. Il manifest viene solitamente scritto con il linguaggio Javascript e nel caso di una PWA assume l'estensione `.webmanifest`

Di seguito viene mostrato come esempio il contenuto del file `app.webmanifest` dell'applicazione **PWA-Meteo**.

```
{
  "name": "PWA-Meteo",
  "short_name": "PWAM",
  "lang": "it",
  "start_url": "/PWA-Meteo/",
  "id": "m0",

  "theme_color": "#4169E1",
  "background_color": "#ADD8E6",
  "display": "standalone",

  "icons": [
    {
      "src": "/PWA-Meteo/icon_144x144.png",
      "type": "image/png",
      "sizes": "144x144"
    },
    {
      "src": "/PWA-Meteo/icon_512x512.png",
      "type": "image/png",
      "sizes": "512x512"
    },
    {
      "src": "/PWA-Meteo/maskable-icon.png",
      "type": "image/png",
      "sizes": "any",
      "purpose": "any maskable"
```

```
    }
  ]
}
```

2.2 Struttura manifest

Il manifest dunque regola attraverso campi specifici l'esperienza degli utenti all'interno della PWA. Di seguito analizzeremo per campi alcune informazioni che possiamo trasmettere tramite il file manifest e le loro implicazioni.

2.2.1 Informazioni generali

```
"name": "PWA-Meteo",
"short_name": "PWAM",
"lang": "en",
"id": "m0",
```

- **name**: `"PWA-Meteo"` è il nome completo dell'applicazione. Questo valore viene utilizzato dal sistema operativo quando lo spazio disponibile non è un problema, ad esempio nella schermata delle informazioni.
- **short_name**: `"PWAM"` è una versione abbreviata del nome dell'app, utilizzata quando lo spazio è limitato, ad esempio nella schermata home o nella barra delle applicazioni.
- **lang**: `"it"` specifica la lingua predefinita del contenuto dell'app, in questo caso l'italiano.
- **id**: `"m0"` è un identificatore univoco per l'applicazione. Anche se opzionale, può essere utile per distinguere versioni o moduli specifici.

2.2.2 Aspetto e comportamento

```
"start_url": "/PWA-Meteo/",
"theme_color": "#4169E1",
"background_color": "#ADD8E6",
"display": "standalone",
```

- **start_url**: `"/PWA-Meteo/"` indica la risorsa da caricare all'avvio dell'app. Questa è la pagina iniziale quando l'app viene lanciata dalla schermata home.
- **theme_color**: `"#4169E1"` definisce il colore principale dell'interfaccia utente, che può essere visibile nella barra superiore del browser o nel contorno dell'app.
- **background_color**: `"#ADD8E6"` stabilisce il colore di sfondo utilizzato durante il caricamento iniziale dell'app.

- **display**: `"standalone"` specifica che l'app deve essere eseguita in modalità indipendente, simulando il comportamento di un'app nativa e nascondendo la UI del browser.

2.2.3 Icone

L'attributo `icons` è una lista di oggetti che specificano le immagini da utilizzare come icona dell'applicazione.

```
"icons": [
    {
      "src": "/PWA-Meteo/icon_144x144.png",
      "type": "image/png",
      "sizes": "144x144"
    },
    {
      "src": "/PWA-Meteo/icon_512x512.png",
      "type": "image/png",
      "sizes": "512x512"
    },
    {
      "src": "/PWA-Meteo/maskable-icon.png",
      "type": "image/png",
      "sizes": "any",
      "purpose": "any maskable"
    }
]
```

- **src**: percorso dell'immagine, può essere sia un URL o la directory dell'immagine
- **type**: `"image/png"` indica il formato dell'immagine.
- **sizes**: `"64x64"` specifica le dimensioni dell'icona.

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

2.3 Linking del manifest

Per comunicare al browser l'esistenza del manifest della PWA, viene inserito un collegamento alla pagina html, in questo caso `index.html`, attraverso la seguente linea di codice:

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

2.4 Testing

Per il testing della compatibilità del manifest abbiamo usato lo strumento Lighthouse di Google Chrome DevTools. Questo test, oltre a dare informazioni sulla performance della web app, ci permette di capire l'accessibilità della nostra PWA e se ha eventuali problemi che impediscono di essere riconosciuta come tale dal browser. Di solito la valutazione di Lighthouse per Performance, Accessibilità, Best Practices e Ottimizzazione per Motori di Ricerca non dipendono solo dal manifest (ad esempio guardano anche al service worker), tuttavia ci permettono di avere un'idea su quali campi sono necessari per un corretto funzionamento come web app.

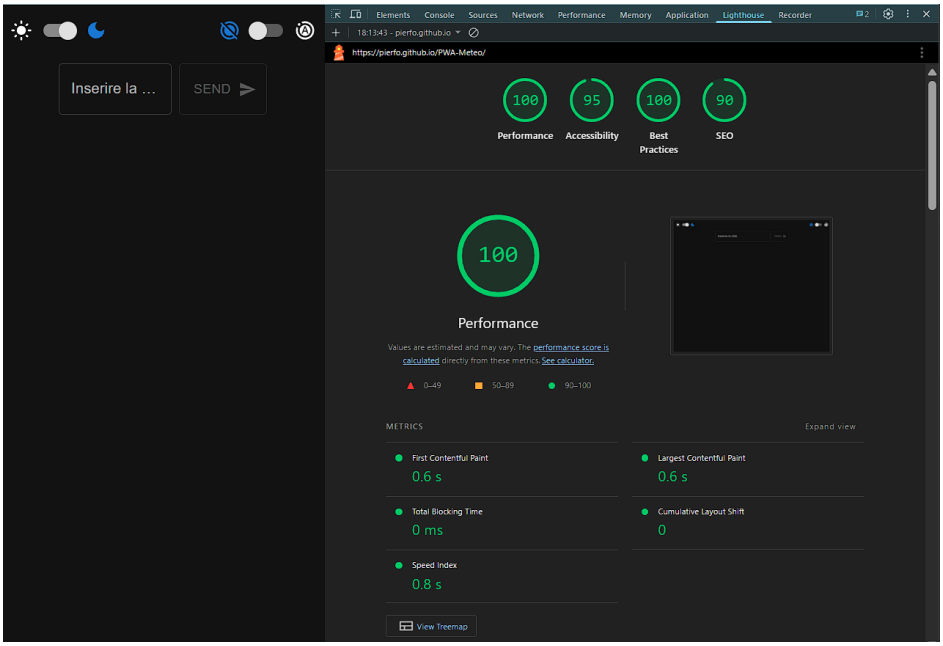


Figure 5: Ultimo test Lighthouse eseguito su PWA-Meteo

2.5 Problemi incontrati

Durante la stesura del file `app.webmanifest` abbiamo incontrato problemi con la compilazione del codice. Per individuare il problema abbiamo analizzato le potenziali problematiche riguardanti l'installabilità della Web App e gli eventuali errori di formattazione e abbiamo riscontrato che talvolta non è possibile aggiungere commenti al file `app.webmanifest`.

3 Framework

Panoramica delle librerie e framework utilizzati

3.1 React

`React` è una libreria JavaScript open-source sviluppata da Meta (ex Facebook) per costruire interfacce utente (UI) moderne e reattive. Si basa su component riutilizzabili, che permettono di creare applicazioni web dinamiche in modo efficiente.

Caratteristiche principali:

- **Component**: Blocchi modulari e riutilizzabili che gestiscono la propria logica e rendering.
- **JSX**: Sintassi simile a HTML che permette di scrivere la struttura UI direttamente nel codice JavaScript.
- **Gestione dello Stato**: React aggiorna dinamicamente la UI in base ai cambiamenti dei dati (state e props).
- **Virtual DOM**: Ottimizza le prestazioni aggiornando solo le parti necessarie della pagina.
- **Ecosistema Integrabile**: Funziona con altre librerie (es. React Router, Redux) e framework (Next.js).

React è ideale per single-page applications (SPA) e interfacce complesse, grazie alla sua flessibilità e alla vasta community.

3.2 MUI

`Material UI` è una libreria React open-source che implementa il Material Design di Google. Include una collezione di component pronte all'utilizzo che supportano lo standard di Material Design 2.

I vantaggi principali di MUI sono:

- **Prontezza all'uso**: MUI contiene una vasta gamma di component creati da numerosi contributori in tutto il mondo, pronti a molti casi d'uso.
- **Adattabilità**: MUI permette la modifica e adattamento di template customizzabili in base al proprio bisogno

3.3 React Vite

`Vite.js` è un tool di build e ambiente di sviluppo (dev server) progettato per essere veloce, leggero e intuitivo. Creato da Evan You (il fondatore di Vue.js), è ottimizzato per progetti moderni basati su JavaScript e framework come React, Vue, Svelte e altri.

I principali motivi per cui abbiamo scelto vite.js sono:

- **Performance**: Vite ci ha permesso di visualizzare la pagina web che abbiamo creato su un server locale in pochi secondi, risparmiando tempo e costi durante la progettazione.
- **Semplicità d'uso**: la configurazione di Vite è molto intuitivo e minimale con un avvio pressoché istantaneo

3.4 Open-meteo

`Open Meteo` è un servizio che fornisce dati meteorologici globali gratuiti attraverso un'API semplice e veloce. A differenza di molti servizi a pagamento, Open-Meteo è open-source, non richiede chiavi API per l'uso base e offre dati ad alta risoluzione provenienti da modelli meteorologici affidabili, tra cui:

- Dati in tempo reale e previsioni (fino a 16 giorni)
- Dati storici (dal 1940 a oggi)
- Copertura globale con aggiornamenti frequenti
- Formato JSON leggero e facile da integrare

Punti di forza principali:

- Gratuito per uso personale e non commerciale
- Nessuna registrazione necessaria (senza chiave API per le richieste base)
- Supporta vari modelli meteo (ECMWF, GFS, ICON, ecc.)
- Dettagli su precipitazioni, temperatura, vento, umidità e altro

3.5 OpenStreetMap

`OpenStreetMap` (OSM) è un progetto collaborativo per creare una mappa libera e modificabile del mondo, i cui dati geografici sono distribuiti con licenza `Open Database License` (ODbL). Nato nel 2004, OpenStreetMap è basato su dati provveduti da utenti da tutto il mondo. La maggior parte di questi dati provengono da dispositivi GPS o da immagini aeree, che sono poi modellate tramite software. A differenza delle mappe tradizionali, i dati di OpenStreetMap possono essere liberamente utilizzati, modificati e condivisi da chiunque, rendendolo un'ottima scelta per il nostro progetto PWA-Meteo dove viene impiegata per ricavare le coordinate geografiche a partire dal nome della città.

4 Funzionalità

Panoramica su alcune delle funzionalità "nascoste" dell'app *PWA-Meteo*

4.1 Salvataggio dello stato

Lo stato dell'app `PWA-Meteo` è composto da:

- contenuto della barra di ricerca
- dati meteo relativi all'ultima città cercata

Siccome l'app è stata sviluppata con tecnologie cross-platform, non è stato possibile accedere alle componenti specifiche di Android, come il `Bundle savedInstanceState` o la funzione `onSaveInstanceState()`; per risolvere il problema, sono stati utilizzati gli strumenti del web: il contenuto della barra di ricerca è salvato mediante *Local Storage*, mentre il nome dell'ultima città cercata è salvato mediante un *Cookie* della durata di tre ore.

In fase di avvio, l'app controlla se in Local Storage è presente la entry "`searched-item`" e, in caso affermativo, carica il suo valore nella barra di ricerca; in seguito, controlla se è ancora attivo il Cookie "`last-searched`" e, in tal caso, avvia la ricerca, passando come parametro il nome contenuto in quest'ultimo. Se il Cookie non è presente allora l'app richiederà i dati meteo relativi alla città salvata come preferita, se presente.

Quando viene caricata la pagina iniziale, è necessario conoscere il motivo del caricamento: in particolare, c'è il bisogno di sapere se la pagina è caricata perché l'utente ha appena avviato l'app oppure se è caricata perché l'utente ha eseguito il refresh del sito. In base alla situazione, il programma deve mostrare contenuti differenti: nel primo caso deve caricare lo stato salvato, nel secondo deve svuotare il contenuto della barra di ricerca e mostrare i dati relativi alla città salvata come preferita (se presente). La funzione `isAtStartup()`, definita in `/src/input.jsx`, sfrutta *Session Storage* per distinguere fra i due casi: la funzione verifica la presenza dell'entry `"session"` all'interno di Session Storage e in caso affermativo restituisce `false`, mentre in caso negativo salva una stringa in Session Storage e restituisce `true`. Quando l'utente avvia l'app, Session Storage è vuoto e pertanto `isAtStartup()` restituirà `true`; ai successivi refresh dell'app, invece, il dato salvato dalla prima invocazione di `isAtStartup()` rimarrà in memoria e pertanto la funzione restituirà sempre `false`.

Una limitazione dovuta dall'uso di tecnologie cross-platform è il fatto che non è possibile distinguere fra quando l'utente esce dall'app premendo il pulsante "`home`" (situazione in cui si vuole salvare lo stato) e quando esce premendo "`back`" (situazione in cui, invece, non si vuole salvare lo stato): l'app `PWA-Meteo` salverà lo stato in ogni caso. Inoltre, mentre in un'app nativa Android lo stato viene eliminato, dopo un certo intervallo di tempo, per risparmiare risorse, nel caso della web app si è riusciti a fornire solo un'illusione parziale di "recupero delle risorse": grazie ai Cookie, è stato possibile salvare l'ultima ricerca effettuata, assicurandosi anche che essa sia eliminata dopo tre ore di inutilizzo dell'app. Non è stato

possibile usare Cookie per il salvataggio del contenuto della barra di ricerca: infatti, siccome questa componente dello stato dev'essere aggiornata a ogni carattere aggiunto o rimosso, usare Cookie significherebbe che, a ogni modifica del testo, il server deve inviare un nuovo Cookie al client, appesantendo così il traffico. Per memorizzare questa componente dello stato si ha dunque deciso di utilizzare Local Storage, con l'effetto collaterale che il contenuto della barra di ricerca sarà salvato in modo permanente e dunque sarà ricaricato anche se l'utente non avrà utilizzato l'app per giorni.

4.2 Salvataggio delle preferenze

L'app `PWA-Meteo` permette all'utente di:

- definire una città preferita (mediante la `checkbox` a forma di cuore che compare all'inizio dei dati meteo)
- scegliere un tema preferito (mediante lo `switch` in alto a sinistra)
- scegliere se usare lo sfondo animato o quello statico (mediante lo `switch` in alto a destra)

Anche in questo caso, l'utilizzo di tecnologie cross-platform ha impedito l'accesso alle componenti specifiche di Android, come `sharedPreferences`; tuttavia, è stato comunque possibile ottenere un risultato equivalente sfruttando Local Storage: le tre preferenze sono dunque salvate rispettivamente nelle entries "`favourite-city`", "`preferred-theme`" e "`prefers-animations`" di Local Storage. Per quanto riguarda le ultime due preferenze, è stato necessario definire dei valori di default (nel caso in cui l'utente non abbia ancora scelto una delle due opzioni): di default, il tema è scuro e le animazioni sono disattivate.

4.3 Caching

L'app `PWA-Meteo` adotta un approccio *cache-first*: ogni volta che il programma richiede una risorsa, il *Service Worker* verifica se quest'ultima è presente in cache e, in caso affermativo, propone il contenuto salvato, altrimenti effettua la ricerca sul web e restituisce la risposta ottenuta dal server remoto, salvandola in cache per futuri utilizzi.

Questo approccio ha il vantaggio di permettere di ottenere massime prestazioni, però presenta anche le seguenti problematiche:

- non permette di ricevere dati aggiornati: se l'utente cerca una città, allora il relativo `json` dei dati meteo è memorizzato in cache; poi, a qualunque ricerca successiva di quella stessa città, il service worker continuerà a riproporre il `json` salvato anziché fornire dati aggiornati
- può causare un'eccessiva occupazione di memoria: a ogni ricerca, i dati meteo sono salvati in cache per poi non essere più cancellati

Per risolvere il primo problema, si adotta la seguente strategia: ogni volta che un `json` dei dati meteo viene salvato in cache, si inserisce in una seconda cache l'istante temporale in cui è

stato effettuato il salvataggio. Poi, se il `json` viene chiesto nuovamente, si confronta l'istante attuale con l'istante di salvataggio e, se la loro differenza supera una certa soglia (un'ora), si considera il `json` come obsoleto e lo si sostituisce con una nuova versione proveniente dal web.

Per risolvere il secondo problema, invece, si effettua una pulizia periodica della cache, in cui sono eliminati tutti i `json` del meteo che verrebbero considerati obsoleti secondo il criterio descritto prima. Tale pulizia viene effettuata all'avvio dell'app e dopo ogni sua ora di utilizzo.

Tutte le altre risorse (componenti dell'app e `json` restituiti da `OpenStreetMap`) sono, invece, salvate in cache in maniera permanente: non si prevede, infatti, che queste possano cambiare col tempo; inoltre, salvare i risultati di OpenStreetMap non dovrebbe portare a problemi di spazio, vista la loro dimensione ridotta.

Per via dell'utilizzo del framework `react`, non è stato possibile inserire gli asset dell'app in cache già in fase di installazione o di attivazione del Service Worker: infatti, durante la compilazione del progetto, le varie componenti dell'applicativo subiscono hashing, impedendo, così, di conoscere a priori quale sarà il loro l'URL, necessario per poterle salvare in cache; gli asset saranno memorizzati man mano che l'app li richiederà dal web. Alla luce di ciò, è possibile che, per rendere l'app *offline ready*, sia necessario ricaricare la pagina.

4.4 Ricerca placeholder

Nell'eventualità in cui le API `OpenStreetMap` o `Open-Meteo` dovessero essere temporaneamente non disponibili, è stata costruita una repository di file `json` da cui l'app può continuare a ricevere dati meteo: questa seconda modalità di ricerca non fornisce dati aggiornati e serve solamente a continuare a utilizzare il programma anche in caso di interruzione di servizio da parte di una delle API citate prima.

Per inoltrare la richiesta al "server" placeholder è necessario aggiungere alla propria ricerca la flag "`-p`", separata da uno spazio (ad esempio, anziché cercare "`Padova`" si cerca "`Padova -p`").

La repository placeholder contiene un numero limitato di file `json`: in particolare, contiene i dati meteo relativi a tutte e sole le città che possono essere suggerite dall'`AutoComplete` della barra di ricerca (tutte le città salvate nel file `/src/cities_italy_100.json`). La ricerca placeholder permette anche di visualizzare i file `json` relativi a tutte le condizioni meteo riconosciute dall'app (tutti i possibili output della funzione `getWeatherDescription(weatherCode)` definita in `/src/MeteoCard.jsx`): se, ad esempio, si cerca "`Rovesci intensi -p`", allora sarà restituito un `json` in cui, per tutta la settimana, ci saranno rovesci intensi. Quest'ultima funzionalità è utile per visualizzare tutti le possibili condizioni meteo.

5 Conclusione

PWA-Meteo rappresenta un caso di studio, seppur limitato, sull'implementazione di una Progressive Web App nel dominio meteorologico. Come spiegato in questa documentazione, il progetto ha cercato di sfruttare appieno i vantaggi delle PWA: dall'**accessibilità universale** garantita dal modello web, alle **performance offline** abilitate dai service worker, fino all'**installabilità** tramite web app manifest. Tramite l'integrazione con API open-source come Open-Meteo e OpenStreetMap si è riuscito a realizzare un servizio di meteo completo senza dipendenze da soluzioni proprietarie, mentre l'uso di React e MUI ha contribuito ad un'interfaccia utente reattiva e coerente su tutti i dispositivi. Le soluzioni tecniche adottate per il caching intelligente dimostrano come le PWA possano garantire affidabilità anche in scenari di rete limitata. Questo progetto conferma che la tecnologia PWA, quando combinata con una progettazione attenta alle esigenze degli utenti, può competere con le applicazioni native tradizionali nel fornire una user-experience completa e performanti.

References

[1] *Learn PWA, web.dev website.*

[2] "Jake archibald - the offline cookbook, url: https://web.dev/offline-cookbook/."