

Progressive Web App

Introduzione

Prof. Federico Dossena



Storia

- Il concetto di **webapp** (applicazione web) non è niente di nuovo
- Una webapp è sostanzialmente un'**applicazione che funziona all'interno del browser**, sviluppata utilizzando le **tecnologie web**, che **non richiede installazione**, e generalmente **salva i dati su un server remoto**
- Esistono dagli anni '90, ma con l'evoluzione delle tecnologie web sono cambiate moltissimo

Storia

- In origine, la stragrande maggioranza del codice di una webapp era eseguito sul server
- Il codice HTML delle pagine veniva generato dinamicamente dal server in base agli input dell'utente
- Il browser non era altro che un "terminale glorificato"

Storia

- Nei primi anni 2000, con l'introduzione di **XMLHttpRequest** e il concetto di **AJAX** (Asynchronous Javascript And XML), c'è stata una graduale separazione dei compiti
- Il browser non è più un "terminale", ma esegue un'interfaccia grafica per la webapp sviluppata con HTML, CSS e JS
- Il server fornisce delle **API** che vengono utilizzate dall'interfaccia grafica per interagire con l'applicazione sul server

Storia

- Le moderne webapp portano questo concetto all'estremo
- Il browser esegue la **shell**, ossia l'interfaccia grafica dell'applicazione, **solitamente una singola pagina HTML** con una montagna di codice JavaScript che implementa tutta la logica client-side dell'applicazione
- Il server fornisce tutte le API necessarie per l'applicazione comunicando in un formato "comodo" (solitamente JSON)

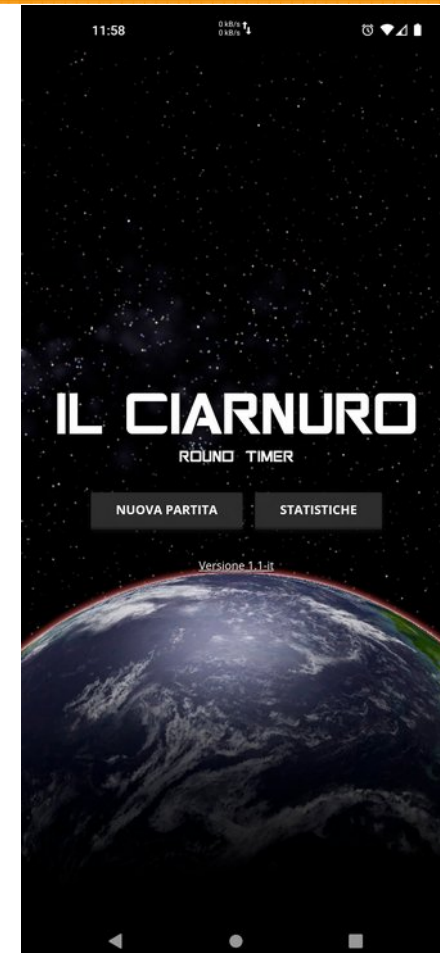
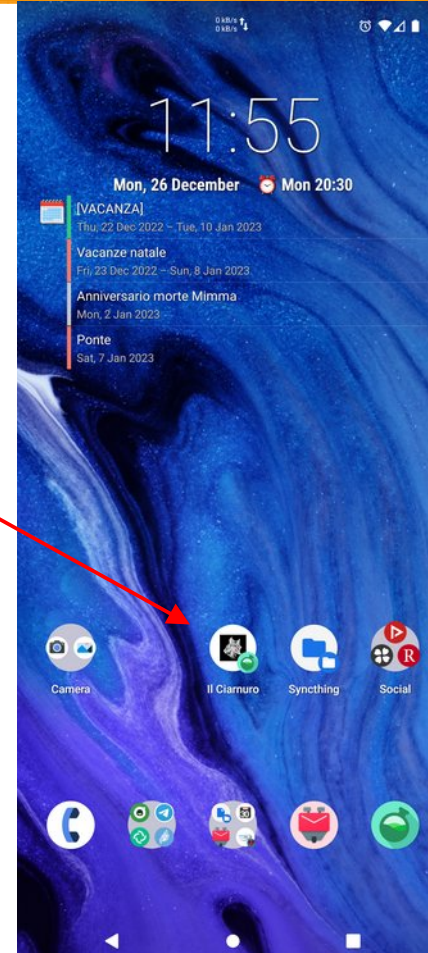
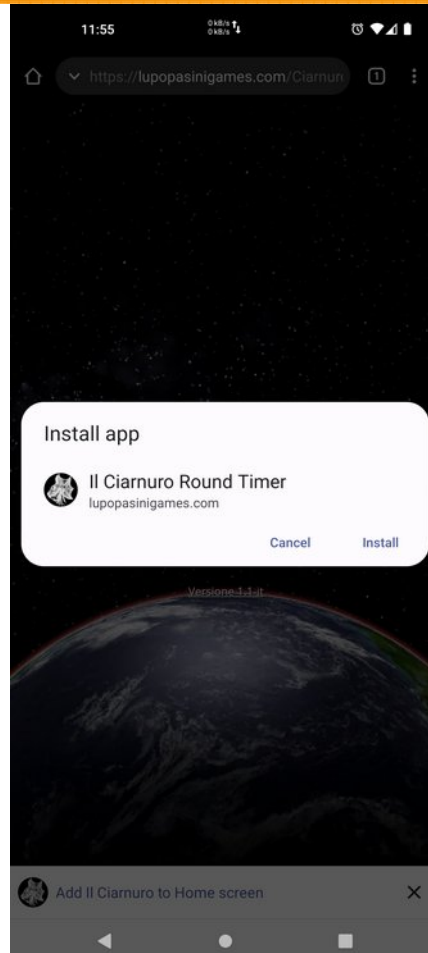
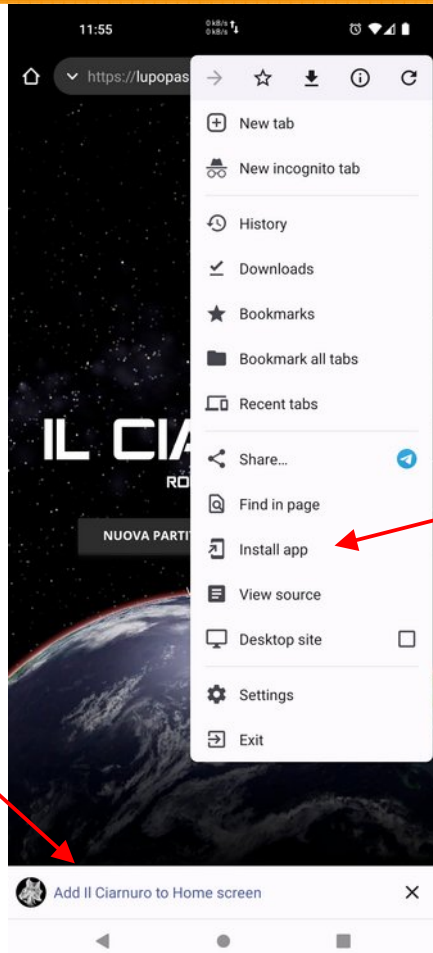
Cos'è una PWA

- Il termine **PWA (Progressive Web App)** è stato coniato nel 2015 da Frances Berriman e Alex Russel, due dipendenti di Google
- Una PWA è una webapp che fa uso delle nuove tecnologie presenti nei browser come i **Service Worker** e i **Web App Manifests** e che **può essere „installata“ nel sistema**, appearing all'utente come una qualsiasi applicazione nativa installata nel dispositivo, pur essendo eseguita da un browser
- Il supporto nei browser principali è arrivato intorno al **2019**, quindi è una **tecnologia molto recente e poco matura** (aspettatevi problemi)

Cos'è una PWA

Provala tu:

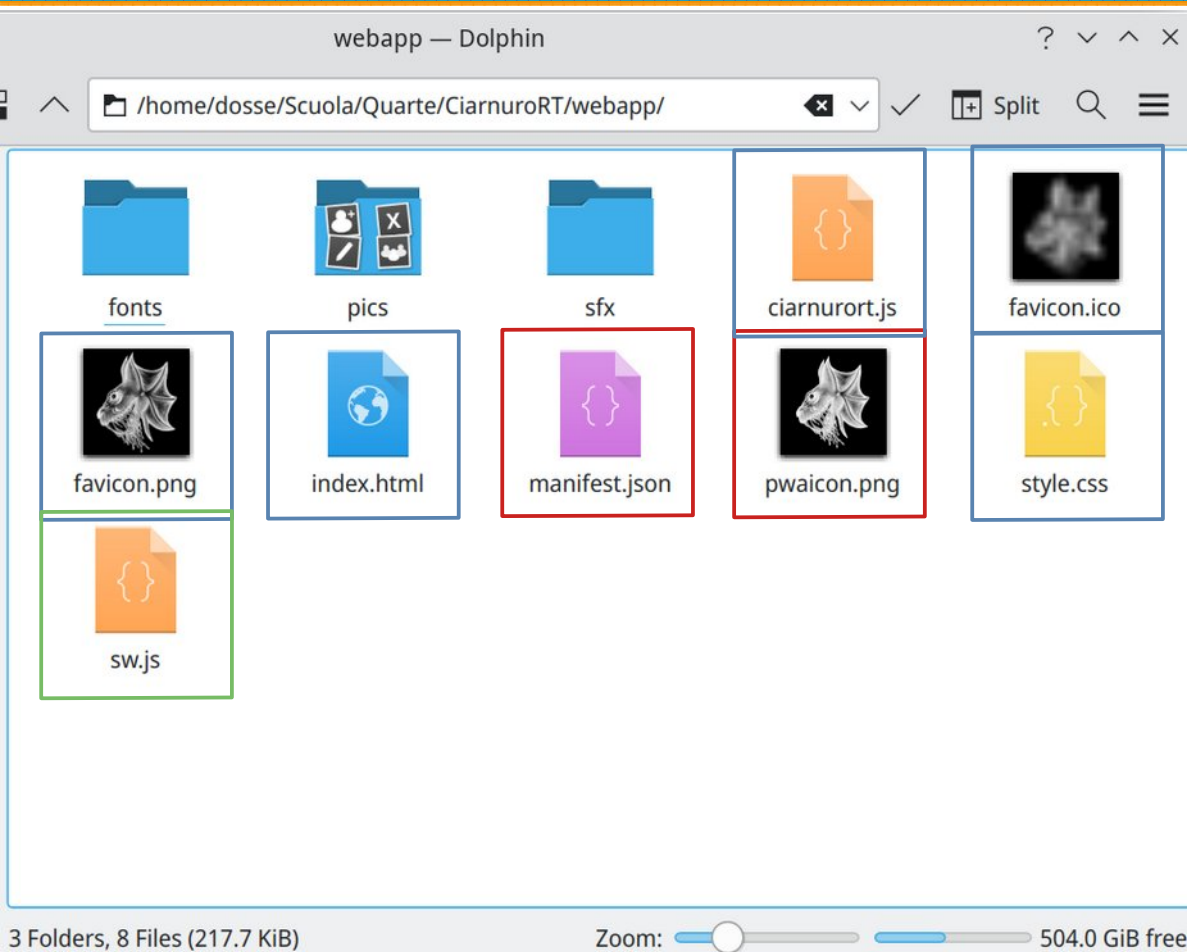
<https://lupopasinigames.com/CiarnuroRT/webapp>



Cos'è una PWA

- Quando ci si collega con il browser a una PWA, è **possibile utilizzarla direttamente nel browser**
- Se il browser in uso lo supporta, **ci offrirà di installarla**
- Una volta installate, **molte PWA sono in grado di funzionare anche senza connessione ad Internet, poichè scaricano tutti i file necessari in una cache**
- Il supporto nei browser è piuttosto buono:
 - Chrome, Edge, Opera e altri **browser basati su Chromium supportano le PWA** sia su PC che su Mobile
 - Firefox e altri **browser basati su di esso supportano le PWA solo su Mobile** per ora
 - **Safari supporta le PWA** sia su iOS che su macOS (anche se mancano funzionalità critiche)

Struttura di una PWA



- **Web App Manifest:** contiene informazioni sull'app come il nome, l'icona, la pagina iniziale, ecc.
- **File della shell:** uno o più file HTML, CSS e JS che implementano l'interfaccia grafica dell'applicazione
- **Service Worker:** gestisce eventi come l'installazione e il caching della risorse
- **Altri file dell'applicazione**

Web App Manifest

```
{
  "name": "Il Ciarnuro Round Timer",
  "short_name": "Il Ciarnuro",
  "id": "ciarnurortpwa",
  "description": "App companion per il gioco di ruolo da tavolo Il Ciarnuro di Matteo Lupo Pasini",
  "icons": [
    {
      "src": "pwaicon.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable any"
    }
  ],
  "start_url": "index.html",
  "scope": "/CiarnuroRT/webapp/",
  "display": "standalone",
  "theme_color": "#000000",
  "background_color": "#000000"
}
```

- **name:** nome completo dell'applicazione
- **short_name:** nome usato per l'icona dell'app
- **id:** breve nome per identificare l'app sul dispositivo
- **icons:** array di icone almeno 192x192
- **start_url:** nome del file HTML principale della shell
- **scope:** percorso sul server in cui si trova la PWA
- **display:** il tipo di UI che deve avere la PWA una volta installata (standalone = sembra una app nativa)
- **theme_color:** colore del bordo della finestra (solo Chromium)
- **background_color:** sfondo dell'icona (solo Firefox)

- Il manifest è un **file JSON** che **contiene informazioni essenziali sulla PWA**
- È un semplice elenco di coppie proprietà-valore

La shell

- La shell è l'**interfaccia utente dell'applicazione**
- Implementa **tutta la logica client-side**
- Generalmente è composta da un **singolo file HTML** che si collega al manifest e a **uno o più file CSS e JS**
- Il codice JavaScript della shell si occupa di comunicare con le API fornite dal server (se necessario) e di modificare la pagina per mostrare contenuti diversi man mano che l'utente interagisce con l'applicazione
- Solitamente si usano framework come [Bootstrap](#) per velocizzare lo sviluppo, ma non è obbligatorio ed è sconsigliato ai novizi

La shell

- L'HTML deve **obbligatoriamente** contenere alcuni elementi per essere identificato come PWA

```
<!DOCTYPE html>
<html lang="it">
  <head>
    <meta charset="utf-8" />
    <title>Esempio PWA 1</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
    <script src="app.js" type="text/javascript"></script>
    <link rel="manifest" href="manifest.json" />
    <link rel="icon" type="image/png" href="favicon.png" />
    <meta name="viewport" content="width=device-width" />
  </head>
  <body>
    <div id="barra">
      
    </div>
  </body>
</html>
```

Identificano il file come HTML5 standard

Collegamento al file CSS e JS della shell

Collegamento a manifest e icona

Informa il browser che la pagina è responsive (si adatta alla dimensione dello schermo)

La shell

- Per ragioni di sicurezza, le PWA non possono essere testate localmente semplicemente aprendo il file index.html nel browser
- **Lo standard richiede obbligatoriamente che sia caricata su un server e servita via HTTPS**
- In caso contrario, il browser non identifica la webapp come una PWA, non permetterà di installarla e non caricherà il Service Worker
- È una buona idea inserire un **redirect automatico** nel codice (o sul server)

```
//Redirect HTTP to HTTPS
if(location.protocol=="http:"){
    location.href="https"+location.href.substring(4);
}
```

Il Service Worker

- Una delle caratteristiche principali che distinguono una PWA da una semplice webapp è la presenza di un Service Worker
- **Un Service Worker è un file JavaScript che riceve alcuni eventi dal browser** come l'avvio dell'applicazione, l'arrivo di notifiche, la richiesta di un file da parte dell'applicazione, ecc.
- Nel caso di semplici applicazioni, la sua funzione principale è gestire il caching della shell, per consentire all'app di funzionare offline

Il Service Worker

- Il Service Worker deve essere **caricato dal codice JavaScript della shell** (solitamente all'inizio del file)
- Di solito il nome del file è **sw.js**, ma possiamo usare il nome che preferiamo

```
//Register PWA service worker
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('sw.js');
}
```

Il Service Worker

- Nel codice del Service Worker devono essere implementati obbligatoriamente questi 3 handler di eventi:
 - **install**: questa funzione deve cachare tutti i file necessari per il funzionamento offline della shell. Viene chiamata al primo avvio dell'applicazione, non all'installazione come farebbe intuire il nome
 - **fetch**: chiamata quando la shell richiede un file (ad esempio un'immagine). Deve decidere se servirla dalla cache o dalla rete
 - **activate**: chiamata quando l'applicazione viene avviata
- A seconda delle funzionalità che implementiamo potrebbero esserci anche altri eventi, come l'arrivo di notifiche push

Il Service Worker

- Nella documentazione troviamo del codice di riferimento per il service worker che possiamo riutilizzare e adattare

```
const cacheName='pwaes1';
const appFiles=[
  'index.html',
  'app.js',
  'favicon.png',
  'manifest.json',
  'pwaicon.png',
  'style.css',
  'sw.js',
  'immagini/logo.png',
  'immagini/background_light.jpg',
  'immagini/background_dark.jpg'
];
```

```
// Caches all the PWA shell files (appFiles array) when the app is launched
self.addEventListener('install', (e) => {
  console.log('[Service Worker] Install');
  const filesUpdate = cache => {
    const stack = [];
    appFiles.forEach(file => stack.push(
      cache.add(file).catch(_=>console.error(`can't load ${file} to cache`))
    ));
    return Promise.all(stack);
  };
  e.waitUntil(caches.open(cacheName).then(filesUpdate));
});
```

```
// Called when the app fetches a resource like an image, caches it automatically
self.addEventListener('fetch', (e) => {
  e.respondWith(
    (async () => {
      const r = await caches.match(e.request);
      console.log(`[Service Worker] Fetching resource: ${e.request.url}`);
      if (r) {
        return r;
      }
      const response = await fetch(e.request);
      const cache = await caches.open(cacheName);
      console.log(`[Service Worker] Caching new resource: ${e.request.url}`);
      cache.put(e.request, response.clone());
      return response;
    })()
  );
});
```

```
// Called when the service worker is started
self.addEventListener('activate', (e) => {
  console.log("[Service Worker] Activated");
});
```

Il Service Worker

- L'approccio tipico è quello di fare un array con l'elenco dei file che compongono la shell e farli cachare tutti quando viene lanciato l'evento **install**
- Per quanto riguarda l'evento **fetch**, l'approccio tipico è controllare se il file richiesto è già presente nella cache e in tal caso servirlo all'applicazione, altrimenti recuperarlo dalla rete, copiarlo in cache e servirlo all'applicazione
 - Potenzialmente si possono creare strategie di caching più sofisticate ma le **Cache API sono incredibilmente acerbe, piene di bug e imprevedibili**
 - La cache può essere cancellata automaticamente dal browser senza preavviso
- Per quanto riguarda l'evento **activate**, generalmente è vuoto

Tecnologie tipicamente usate nelle PWA

- **XMLHttpRequest:** permette all'applicazione di caricare file dal codice JS e di interagire con API tramite semplici richieste GET e POST
- **localStorage:** permette di salvare stringhe che persistono dopo la chiusura del browser. Utile per salvare impostazioni, salvataggi di giochi, ecc.
- **Geolocation API:** permettono di ottenere la posizione dell'utente
- **Web Audio API:** un vero e proprio sintetizzatore di suoni, player e mixer
- **Push API e Notification API:** permettono all'applicazione di ricevere notifiche push da un server. Troppo complicate da usare per il tempo che abbiamo
- Framework e librerie varie per JS

Tecnologie tipicamente usate nelle PWA

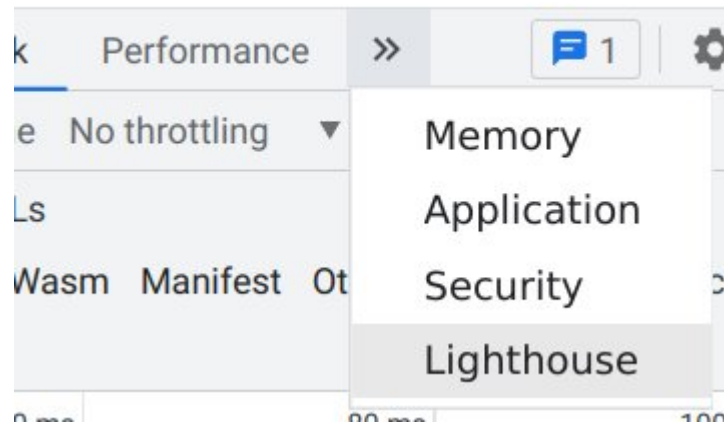
- Purtroppo non abbiamo ancora a disposizione PHP e tecnologie simili, per cui non possiamo ancora implementare logica dell'applicazione lato server
- In altre parole, non possiamo ancora fare nulla che richieda un database online, tipo un negozio o una chat
- Ma questo va bene, ci permette di focalizzarci di più sul fatto che le PWA possono funzionare anche offline
- **L'integrazione dell'app con il sistema è molto limitata** (ad esempio, non può essere associata a tipi di file, per ora)

Debuggare le PWA

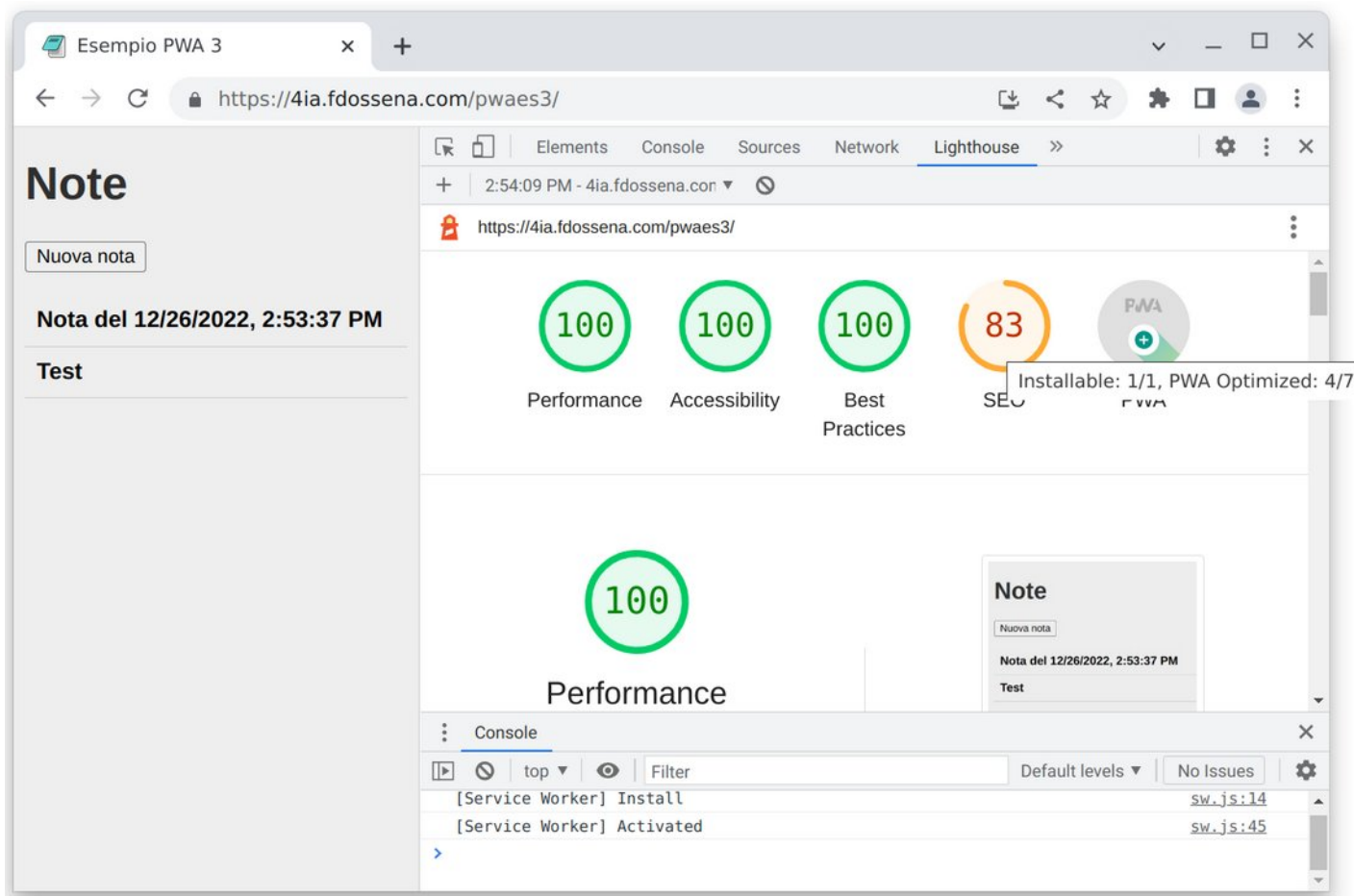
- Per il debugging delle PWA abbiamo a disposizione, oltre ai soliti strumenti di sviluppo del browser (tasto **F12**):
 - Google Lighthouse (specifico per le PWA)
 - Gli strumenti di sviluppo remoti (per debuggare problemi su mobile)
- Questi strumenti purtroppo sono presenti **solo nei browser basati su Chromium** (Chrome, Edge, Opera, ...)

Google Lighthouse

- Nei browser basati su Chromium è disponibile tra gli strumenti di sviluppo del browser una funzione specifica per le PWA che si chiama Lighthouse
- Lighthouse analizza approfonditamente la pagina per identificare **problemi che impediscono l'installazione** (ad esempio errori nel manifest o nel Service Worker), **problemi di accessibilità** e **problemi di performance**
- Per usarlo basta **premere F12 e selezionare Lighthouse**
- È un po' strict...



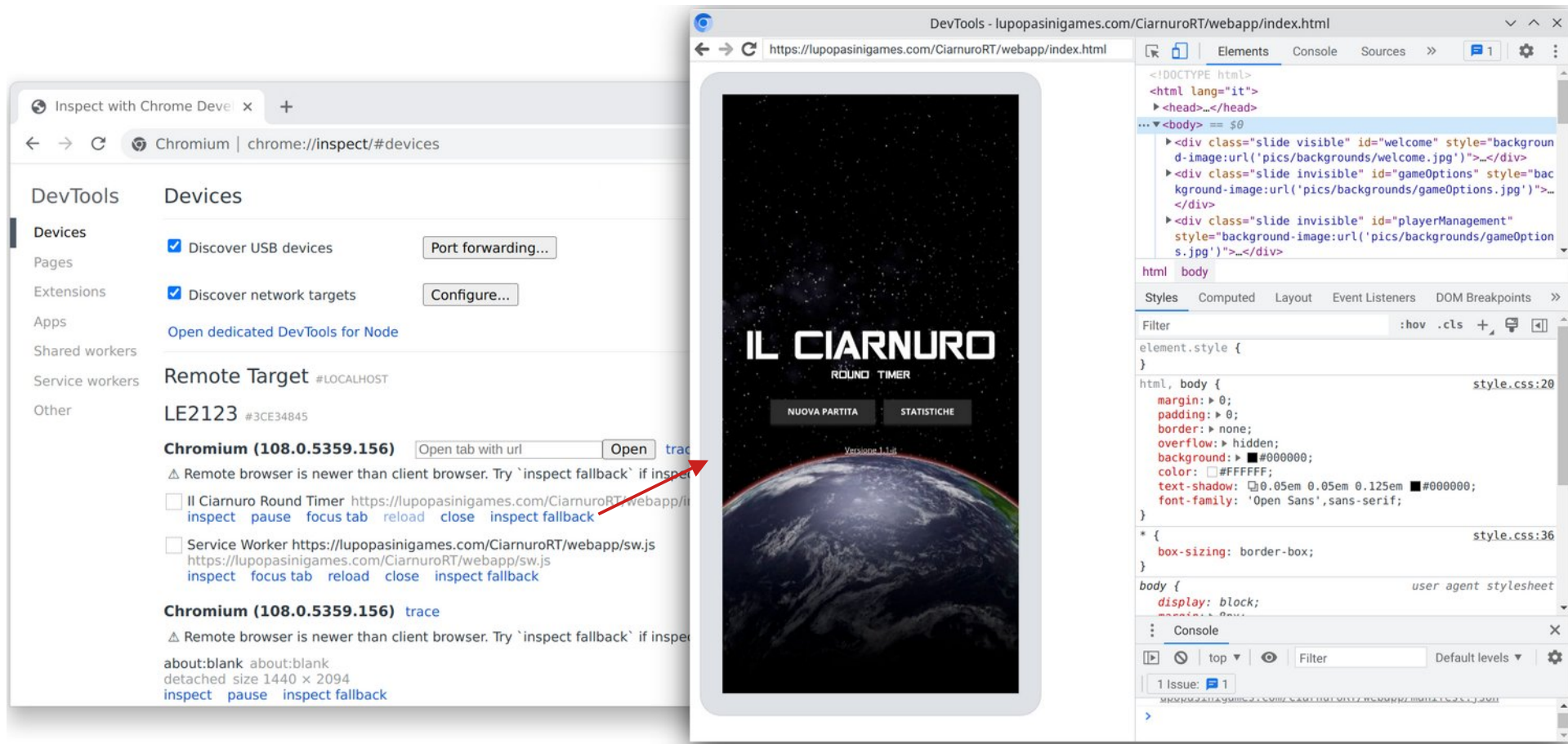
Google Lighthouse



Strumenti di sviluppo remoti

- Se un bug si presenta solo su mobile può essere difficile da debuggare data l'assenza degli strumenti di sviluppo nel browser mobile
- I browser basati su Chromium possono collegarsi al browser del telefono e fornirci tutti gli strumenti di sviluppo normalmente disponibili su PC, incluso Lighthouse
- Per utilizzarlo, bisogna attivare il **Debug USB** nelle impostazioni di sviluppo di Android, **collegare il telefono al PC** (su Windows bisogna installare il driver ADB) e visitare <chrome://inspect/#devices> sul PC

Strumenti di sviluppo remoti



Esempi di PWA

Esempi

- Assieme a queste slide trovate alcuni esempi che mostrano funzionalità e alcune limitazioni delle PWA
 - **pwaes1**: una semplice PWA che mostra varie funzioni, come capire se l'app è installata o se è in esecuzione nel browser, il tipo di connessione dell'utente, la posizione e salvataggio nel localStorage
 - **pwaes2**: app per disegnare tipo paint, può salvare nel localStorage o scaricare i file tramite la libreria FileSaver.js
 - **pwaes3**: app per prendere appunti di testo che salva nel localStorage

Esempi

- **pwaes4**: una sorta di piccolo spotify. Quando ascolti una canzone, viene salvata nella cache ed è ascoltabile anche offline. Funziona piuttosto male per via delle Cache API buggate
- **pwaes5**: mostra l'uso di XMLHttpRequest verso un server di terze parti (che accetta connessioni da qualsiasi origine, solitamente non è così) per ottenere delle informazioni tramite un'API
- **CiarnuroRT/webapp**: la webapp de Il Ciarnuro vista negli screenshot. È una PWA completa sviluppata nel 2020 e semplificata leggermente per usarla come esempio

Esempi

- **pwatemplate**: un template con una PWA vuota che potete usare come base per la vostra PWA facendone una copia e configurandolo opportunamente:
 - Nel manifest, bisogna impostare name, short_name, id, description e scope
 - Nel service worker bisogna impostare l'id della PWA nella variabile cacheName e aggiornare l'elenco di file da cachare nell'array appFiles

Progetto

Progetto

- Sviluppare una PWA per risolvere un problema senza tempo: „dove ho parcheggiato la macchina stamattina?“
- L'applicazione deve avere:
 - Un pulsante per ottenere la **posizione corrente** e memorizzarla
 - Un elenco di tutte le **posizioni precedenti**, cancellabili
 - Cliccando una delle posizioni precedenti deve **mostrare una mappa** di OpenStreetMap (non usiamo Google Maps perchè richiede una API key)
 - La possibilità di copiare il **link in formato geo:** della posizione, così da poterlo condividere con qualcuno
Esempio: <geo:45.3363,9.6986>

Progetto

- Per poter testare il vostro lavoro, **potete caricarlo online** utilizzando [FileZilla](#) (o altro client FTP) e le credenziali fornite dal docente
- **È utile cancellare spesso la cache del browser** tra una prova e un'altra per evitare che pezzi di codice vecchio rimangano in memoria. Per farlo rapidamente, basta premere Ctrl+Shift+Canc

Risorse utili

- Per tutto quello che riguarda HTML, CSS e JS, **w3schools** è una risorsa eccellente di documentazione semplice da capire e esempi facili da riutilizzare
- La documentazione su **MDN** è più tecnica ma estremamente utile e copre anche le tecnologie più moderne (controllate sempre la compatibilità con i vari browser)
- Il prof