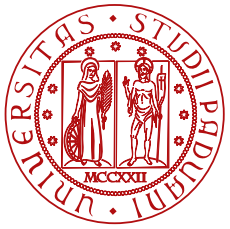

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Accesso ai sensori in applicazioni web

Relatore

Carlo Fantozzi

Università di Padova

Laureando

Giacomo Schiavo

1218658

Anno Accademico 2021/2022

Data di laurea: 21/09/2022

Alla mia famiglia
e a tutti i miei amici

Abstract

Nel mondo odierno, i dispositivi mobili sono sempre più all'avanguardia con nuove caratteristiche e funzionalità per cui i sensori, come accelerometro e giroscopio, giocano un notevole ruolo. L'accesso a questi sensori è ora presente anche nelle web app che richiedono un approccio di sviluppo indipendente dal sistema operativo, a differenza delle app native, e sono presenti, in fase sperimentale, nuove standardizzazioni di tecnologie quali NFC e Bluetooth.

In questa tesi si propone di dare una breve introduzione sul mondo delle applicazioni web e una lista di tutti i sensori e periferiche presenti in un dispositivo mobile. Successivamente verranno indicate tutte le modalità e le tecniche utilizzate per l'accesso ai dati dei sensori attraverso un semplice browser web. Per concludere verranno esposte delle considerazioni sull'estrema facilità di utilizzo e accesso dei sensori da applicazioni web e dei possibili rischi sulla sicurezza e privacy del singolo utente.

Indice

1 Introduzione

1.1 Applicazioni native e applicazioni web

In questi ultimi tempi, le applicazioni per dispositivi mobili (app) hanno ormai preso piede nella nostra vita con chat, social media, e-commerce, giochi, etc. Per sviluppare un'app ci sono diversi approcci, ognuno con i propri pregi e difetti. L'approccio nativo permette di sviluppare app a partire dagli strumenti offerti dalla piattaforma scelta e il codice per lo sviluppo è definito dalla stessa. L'approccio web permette di sviluppare app scrivendole con i linguaggi degli attuali standard web più diffusi che li rende accessibili da qualsiasi dispositivo che abbia una connessione a Internet e un web browser. Lo sviluppo ibrido permette di installare una web app come app nativa attraverso una webview, un componente nativo progettato per "ospitare" un sito web, con lo svantaggio di non poter offrire la migliore esperienza utente. Con sviluppo cross-platform si intende, invece, lo sviluppo della stessa app ma per piattaforme diverse usando sempre lo stesso codice scritto una volta sola. Gli esempi più popolari sono React Native [24], Flutter [19] e PhoneGap/Cordova [20]. In questo capitolo si andrà a discutere delle principali caratteristiche e differenze dei primi due approcci: nativo e web.

1.1.1 Sviluppo di un'app nativa

Lo sviluppo nativo è basato sulla piattaforma scelta che offre una serie di strumenti, librerie e framework per aiutare gli sviluppatori a creare la propria app nativa in modo più semplice e mantenibile. Ogni piattaforma offre un IDE (Integrated Development Environment) che permette di gestire in modo efficiente qualsiasi risorsa necessaria alla

realizzazione dell'app ed aiuta a seguire le linee guida della piattaforma di sviluppo.
Le linee guida sono basate sui modelli software e hardware della piattaforma che è necessario conoscere per la creazione di un'app e richiede sviluppatori specializzati in materia; non seguire tali regole comporta la scrittura di codice poco efficiente e poco mantenibile perché non sfrutta appieno le potenzialità offerte.
Per sviluppare un'app nativa Android, per esempio, è necessario installare Android Studio, un IDE completamente gratuito e mantenuto da Google, che contiene molte librerie e framework che aiutano gli sviluppatori nella creazione dell'app. All'interno di Android Studio sono presenti un'interfaccia grafica per realizzare una UI stabile e solida, strumenti di profiling (CPU, RAM, rete, etc.) in tempo reale, debugger e un editor di codice impostato per rendere la scrittura del codice più veloce [12].
Il vantaggio di uno sviluppo nativo è la possibilità di accedere a tutte le potenzialità del dispositivo e tutte le componenti hardware che può offrire la piattaforma; nessun altro approccio di sviluppo può avere tutta questa libertà. Con le applicazioni native è inoltre possibile creare widget, gestire i dati in memoria, offrire la migliore user experience e offrire le massime prestazioni su tutti i fronti. Inoltre l'applicazione occuperà un posto nel dispositivo rendendolo parte di esso e, se implementato, può offrire servizi anche offline. Le app vengono installate nel dispositivo attraverso lo store (Google Play Store per Android e App Store per iOS) ed è necessario pagare per il servizio di hosting. Il prezzo di mantenimento di un'app è variabile: Google Play Store permette la pubblicazione pagando una tassa di registrazione di 25$, sia che l'app sia gratuita o a pagamento [45], e trattiene una percentuale degli incassi: dal 15% per il primo milione di dollari all'anno e fino al 30% superato il milione di dollari all'anno [36]. Per pubblicare un'app su App Store è necessario avere un abbonamento "Apple Developer Program" (99$/anno) [10] sia che l'app sia gratuita o a pagamento. La tassazione sugli incassi è identica a quella imposta da Google.
Per riassumere, i vantaggi di un approccio nativo sono i seguenti:

- massime performance;
- migliore esperienza utente;
- l'app diventa parte del dispositivo;
- completo controllo di tutte le funzionalita disponibili;
- può offrire servizi offline.

I principali svantaggi sono:

- il codice deve essere riscritto per ogni piattaforma;
- è necessario avere la conoscenza dell'architettura hardware e software del sistema operativo, richiede sviluppatori specializzati;
- maggior costo di produzione.

1.1.2 Sviluppo di una web app

Per web app si intende un'applicazione accessibile da un qualsiasi web browser, sia da pc desktop che da smartphone, con una connessione a Internet. Il termine "web app" fu utilizzato per la prima volta da Steve Jobs al WWDC 2007 (Apple WorldWide Developers Conference) con l'uscita del primo iPhone [47]: il browser web permetteva la comunicazione tra web app e sistema operativo, senza dover ricorrere ad un SDK specifico, ed era necessaria la sola conoscenza degli standard web. Un anno dopo, con l'uscita dell'iPhone 3G, venne introdotto anche l'iPhone OS 2, un nuovo sistema operativo in cui comparì per la prima volta l'App Store [46] e venne reso disponibile l'SDK per sviluppare app native per iOS, pagando la licenza di sviluppatore che già ammontava a 99$/anno.
Il punto di forza delle applicazioni web è la portabilità: qualsiasi dispositivo che abbia un web browser può accedere a qualsiasi servizio offerto dall'applicazione e non richiede installazione. Di conseguenza, le applicazioni web sono facilmente condivisibili e possono funzionare sulla maggior parte dei dispositivi; l'unico limite risiede nelle performance del browser. I costi di produzione sono ridotti: è necessario pagare un servizio di web hosting, il codice è scritto una volta sola per più piattaforme contemporaneamente, è sempre aggiornato, sono necessari solo sviluppatori web e non è richiesta alcuna conoscenza sul sistema operativo delle piattaforme di destinazione. Perciò, per scrivere una web app, è richiesta solo la conoscenza degli standard web attualmente in uso: HTML, CSS e Javascript. Le performance di una web app non sono al pari di un'app nativa a causa della generalità che il web browser deve mantenere per poter essere compatibile con qualsiasi sistema operativo. Anche la mancanza del pieno accesso all'hardware (periferiche e sensori) e alle funzioni del sistema operativo riduce le funzionalità della web app, che rimane vincolata ai limiti funzionali del web browser. Essendo essenzialmente un sito web, la web app è vulnerabile a qualsiasi attacco che può essere effettuato su un sito web e perciò necessita lo sviluppo di sistemi di sicurezza per rendere il servizio sempre funzionante e sicuro. Per riassumere, i maggior vantaggi di un approccio web sono i seguenti:

- massima portabilità e condivisibilità;
- costi ridotti: il codice è scritto una volta sola;
- applicazione sempre aggiornata all'ultima versione.

I principali svantaggi di questo approccio sono i seguenti:

- non offre la miglior esperienza utente;
- performance drasticamente ridotte;
- non funziona offline;
- più vulnerabile ad attacchi informatici.

1.1.3 Approcci a confronto

Quando conviene sviluppare una web app o un'app nativa?

- Web app: il servizio da offrire deve funzionare su tutti i sistemi operativi, deve essere sempre aggiornato e non sono necessarie alte performance. I costi di produzione sono bassi e il codice viene scritto una volta sola per tutte le piattaforme. App che adottano questo approccio sono, ad esempio, Wooclap e StrawPoll.
- Nativa: l'applicazione è parte del dispositivo, il codice scritto varia per ogni piattaforma e richiede sviluppatori specializzati. I costi di sviluppo sono maggiori ma permette di offrire le massime performance, una migliore esperienza utente e l'accesso a qualsiasi funzionalità del dispositivo.
  App che adottano questo approccio sono, ad esempio, qualsiasi app preinstallata, Gmail, Maps e qualsiasi gioco.

1.2 Accesso alle periferiche e ai sensori

Come descritto nel paragrafo precedente, le app native offrono un completo accesso all'hardware del dispositivo e permettono di accedere ai dati in qualsiasi momento. Negli ultimi anni, anche i web browser possono farlo e per alcuni sensori non è nemmeno necessario chiedere l'accesso all'utente. In aggiunta, essendo il browser web presente in ogni sistema operativo, l'accesso ai sensori è uniformato, ad esempio, è possibile accedere alla geolocalizzazione del dispositivo indipendentemente dal fatto che questo

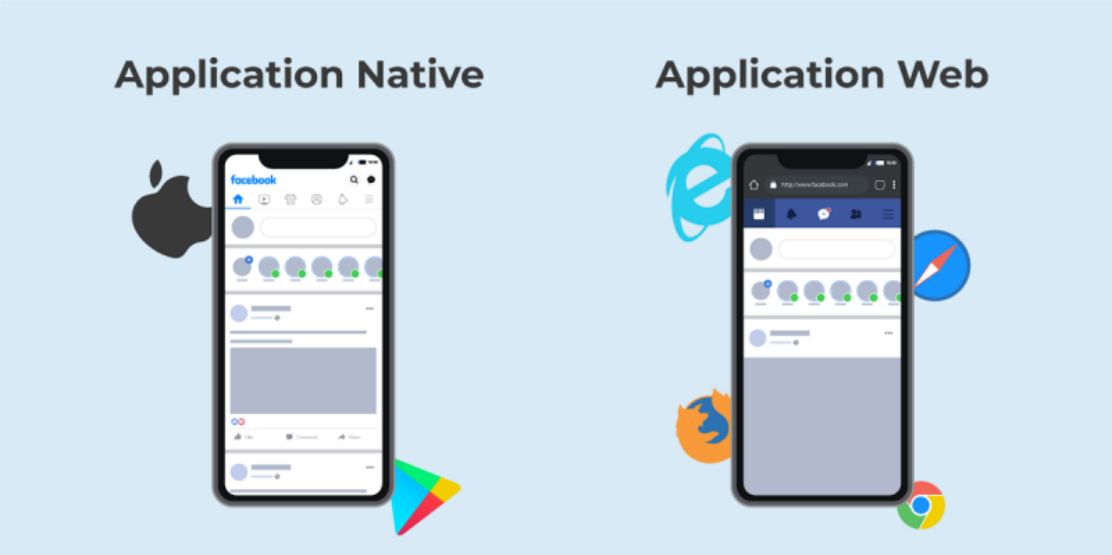

Figura 1.1: Facebook: Nativo vs Web app [5]

sia uno smartphone o un pc. Come ulteriore esempio, attraverso il Network Information API [11] è possibile accedere alle informazioni sulle connessioni a cui il dispositivo è connesso e di essere notificati nel caso di un cambio di connessione. Come visibile dalla documentazione fornita da W3C [50], queste funzionalità che accedono a informazioni legate al dispositivo sono sperimentali e alcuni web browser hanno deciso di non implementare queste funzioni principalmente per motivi legati alla privacy dell'utente. Altre funzionalità come l'accesso ai sensori di movimento e di geolocalizzazione sono definite dal consorzio W3C come "ben sviluppate", perché presenti da molto più tempo: la prime bozze risalgono al 2011 [48] [49]. Alcune tecnologie sono ancora sperimentali e includono i Generic Sensor API, Proximity Sensor, Ambient Light Sensor, Battery Status API, Geolocation (usando i sensori dell'interfaccia Sensor) e Screen Orientation [50]. Sono "Exploratory Work" tecnologie come il Web Near-Field Communications (NFC) API e Web Bluetooth, ma in questo momento solo Google Chrome ha preso in considerazione la loro implementazione [35]. Infatti W3C non fornisce un'implementazione di queste funzionalità, ma offre specifiche stabili e strutturate che possono essere implementate nel modo più adatto a seconda del browser.

1.3 Argomenti della tesi

L'obiettivo di questa tesi è descrivere il mondo dei sensori accessibili da un applicazione web: quali sono, come funzionano e come accedere ai loro dati. In conclusione

verranno discusse alcune problematiche riguardo alle modalità di accesso e sui possibili problemi di sicurezza e privacy dell'utente.
Nel **capitolo 2** verranno elencati tutti i possibili sensori presenti in un dispositivo mobile, riassumendo in modo sintetico le caratteristiche più importanti.
Nel **capitolo 3** verranno illustrate le modalità di accesso ai sensori tramite le Web API suggerite dalla W3C.
Nel **capitolo 4** verranno trattate alcune importanti considerazioni sulla privacy e sicurezza riguardo all'accesso ai dati dei sensori e sui possibili rischi legati alla privacy dell'utente.

2

Sensori

In questo capitolo sono brevemente descritti tutti i sensori presenti in un dispositivo mobile (smartphone) che sono accessibili ad uno sviluppatore (di app native). Tenendo conto che lo smartphone è un sistema embedded, i sensori sono dotati di poca risoluzione principalmente perchè la loro dimensione fisica è molto limitata, l'uso dell'energia è ridotto al minimo e una maggiore precisione non è necessaria.

2.1 Accelerometro

L'accelerometro è impiegato per la misurazione dell'accelerazione sui tre assi cartesiani: X, Y e Z (vedi figura 2.1). I valori vengono misurati in m/s^2 e si basa su un sistema inerziale: in caduta libera l'accelerazione sarà nulla mentre è pari alla forza di gravità se appoggiato su un piano (la forza esercitata dal tavolo). Questo sensore viene maggiormente utilizzato insieme ad altri sensori: per esempio può essere utilizzato con il giroscopio per ricavare la forza di gravità o per ricavare l'accelerazione lineare (accelerazione senza forza di gravità) con il magnetometro o il giroscopio. L'accelerometro può essere utilizzato singolarmente come pedometro, per il calcolo dei passi, o per determinare un'agitazione del dispositivo (shake).

2.2 Giroscopio

Il giroscopio percepisce la velocità angolare del dispositivo sui tre assi cartesiani principali: X, Y e Z; è un sistema basato sulla forza di Coriolis e le letture sono misurate

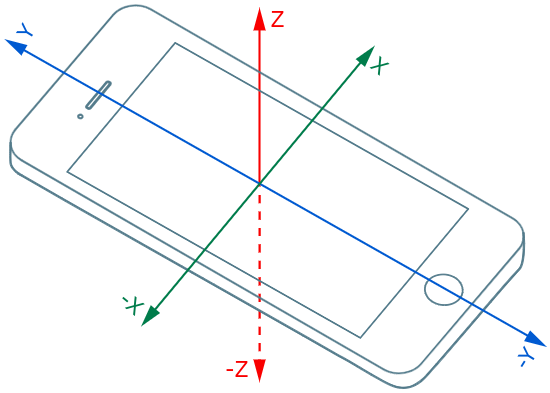

Figura 2.1: Sistema di coordinate di un accelerometro

in rad/s. Il giroscopio oscilla ad alte frequenze ed è uno dei sensori che consuma più batteria ed è anche più incline a registrare rumore, come le vibrazioni dovute allo speaker del telefono [40].

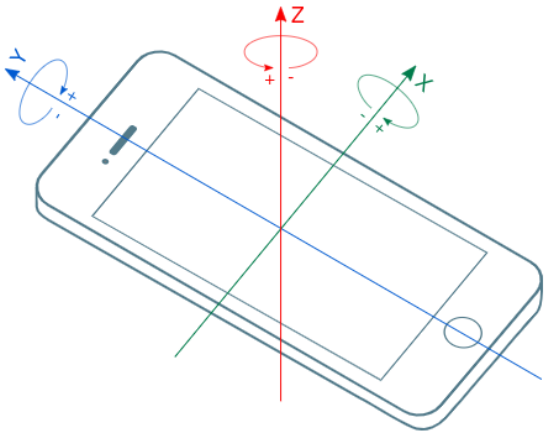

Figura 2.2: Sistema di coordinate di un giroscopio

2.3 Magnetometro

Il magnetometro è il sensore che misura la somma di tutti i campi magnetici in cui il dispositivo è immerso. Per misurare il campo magnetico si usano tre assi tra loro ortogonali (vedi figura 2.1) da cui è possibile ricavare un vettore in tre dimensioni che punta al campo magnetico più potente vicino; queste misurazioni sono indipendenti dall'orientamento del dispositivo. Per determinare come un dispositivo è tenuto in mano

è necessario conoscoscere la forza di gravità che può essere ottenuta con l'accelerometro e con il giroscopio (per risultati più precisi).

2.4 Sensore di luce ambientale

Il sensore di luce ambientale percepisce l'intensità di luce circostante, misurata in lux. Il caso di utilizzo principale è quello di regolare dinamicamente la luminosità dello schermo del dispositivo in base alla luce presente nell'ambiente: un ambiente scuro richiede una minore illuminazione dello schermo del dispositivo mentre un'ambiente luminoso richiederà una maggiore illuminazione dello schermo.

2.5 Sensore di prossimità

Il sensore di prossimità viene utilizzato per rilevare, senza alcun contatto, oggetti fisici immediatamente vicini al sensore stesso. La distanza di rilevazione dal sensore all'oggetto fisico dipende dal tipo di sensore di prossimità utilizzato: un sensore ottico rileva fino a 5 millimetri, uno induttivo fino a 8 millimetri mentre uno capacitivo ha un intervallo tra i 3 e 30 millimetri [6]. Esso emette un campo elettromagnetico, ad esempio nell'infrarosso, e rimane in ascolto per cambiamenti nel campo o nel segnale di ritorno. E' usato principalmente per bloccare o spegnere lo schermo quando l'orecchio si avvicina ad esso durante una chiamata.

2.6 Batteria

Attraverso la batteria del dispositivo è possibile ricavare le seguenti informazioni: livello di carica, se collegato ad una fonte di alimentazione, quanto tempo manca alla carica completa e quanto ne manca al completo scaricamento. Può essere usato in tutti i dispositivi mobili per regolare il consumo di risorse o salvare lo stato dei dati in caso di batteria scarica.

2.7 Fotocamera e microfono

La fotocamera è il sensore principale usato per catturare immagini e video. Nei dispositivi mobili vengono principalmente utilizzati sensori CMOS rispetto ai CCD perchè assorbono molta meno energia. Alcuni dispositivi utilizzano sensori CMOS

retroilluminati per catturare più luce, al prezzo di un costo maggiore del sensore. Negli ultimi anni non è più presente solo una singola lente a formare una fotocamera, bensì si è arrivati fino a cinque lenti, come nel modello Nokia 9 PureView, visibile nella figura 2.3. Le risoluzioni più recenti arrivano fino a 108MP, mentre per i video si può raggiungere una risoluzine fino a 8K. Il microfono è il sensore che permette di catturare i suoni dell'ambiente circostante. Nei dispositivi mobili è ottimizzato per il riconoscimento della voce, cancellando il rumore di sottofondo. Più microfoni possono essere presenti anche per migliorare la soppressione del rumore.

Figura 2.3: Nokia 9 PureView [32]

2.8 Localizzazione

Il GPS (Global Positioning System) è un sistema di oltre 30 satelliti che, orbitando attorno alla Terra, emettono continuamente dei segnali per qualsiasi ricevitore. Il ricevitore GPS su uno smartphone calcola la sua posizione sulla Terra triangolando tre o quattro segnali ricevuti dai satelliti, ovvero calcola le distanze usando come base il tempo di ricezione del segnale da parte del riceivitore. Per migliorare la precisione della posizione si possono usare anche indirizzo IP, reti Bluetooth e WiFi circostanti.

2.9 NFC

NFC sta per Near Field Communication e indica una tecnologia wireless a corto raggio che opera a 13.56MHz e permette la comunicazione tra dispositivi ad una distanza minore di 10 cm. Un tag NFC può assumere due ruoli: NFC initiator o NFC target.

Figura 2.4: come funziona il GPS [22]

Lo stato initiator è associato ad un dispositivo alimentato (attivo) ed emette un piccolo campo elettromagnetico che viene captato dal NFC target. Quest'ultimo è in uno stato passivo (non alimentato da corrente) e, subendo il campo, emette un impulso elettrico che permette la comunicazione dei dati all'initiator. In questo modo si instaura una comunicazione peer-to-peer che può essere instaurata anche tra due dispositivi attivi: essi passano periodicamente dalla modalità initiator, per cercare target, alla modalità target. Quando un target è trovato, l'initiator può scrivere/leggere dal target. Un formato standardizzato per lettura/scrittura su tag NFC da smartphone è il NDEF: la comunicazione avviene attraverso messaggi NDEF, formati da uno o più records NDEF, che possono essere inseriti in un tag NFC. Ogni record è una struttura binaria che contiene sia i dati sia le informazioni riguardanti essi come, ad esempio, la loro struttura (dimensione totale dei dati, se è sono suddivisi in più records (chunked), etc.) [41].

2.10 BLUETOOTH

Bluetooth è uno standard per la comunicazione tra dispositivi wireless a corto raggio. Il Bluetooth "classico", indicato con BR/EDR, supporta una velocità fino a circa 24Mbps mentre il Bluetooth 4.0 limita la comunicazione a 1Mbps. Il Bluetooth 4.0 introduce il concetto di "Low Energy", chiamato anche "Bluetooth Smart" o BLE, che permette ai dispositivi di spegnere il trasmettitore per la maggior parte del tempo e risparmiare energia. Oltre a implementare il vecchio modello Broadcaster-Receiver, BLE introduce due nuovi ruoli che un dispositivo può assumere: Peripheral e Central, due concetti che seguono le definizioni introdotte da GATT (Generic ATTribute profile) che definisce

un'architettura molto simile al client-server per la comunicazione tra dispositivi BLE (vedi figura 2.5). I dispositivi con il ruolo di Peripheral possono essere connessi solo ad un Central alla volta mentre i Central possono connettersi a tutti i Peripheral presenti. I dispositivi Peripheral vengono anche indicati come GATT Server che inviano risposte alle richieste emesse dai GATT Client. Un GATT Server raggruppa le proprie informazioni in Service che contengono Characteristic, l'elemento fondamentale della comunicazione. I Service primari sono quelli principalmente utilizzati per offrire funzionalità standard mentre i secondari sono utilizzati in combinazione di quelli primari come modificatori di quest'ultimi [38].

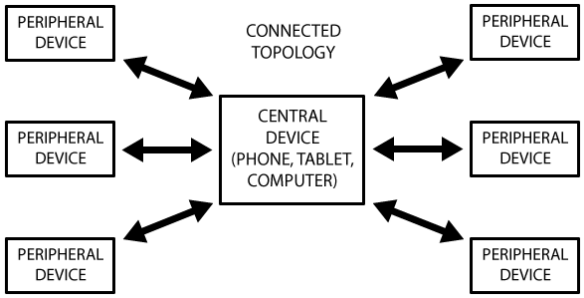

Figura 2.5: Topologia GATT [1]

3

Accesso ai sensori dal web browser

In questa sezione verranno mostrate le modalità di accesso ai sensori da web browser attraverso il linguaggio di programmazione JavaScript. Tutte le sezioni e le descrizioni presenti in questo capitolo sono tratte dalle documentazioni fornite da MDN Web Docs [28], una repository di documentazioni scritte appositamente per sviluppatori web che è ormai una guida principale per tutte le Web API. Tutte le implementazioni presenti su MDN Web Docs sono basate sulle specifiche realizzate da W3C (World Wide Web Consortium), organizzazione internazionale fondata nel 1994 da Tim Berners-Lee con lo scopo di offrire standards per il web [44]. Per le prossime sezioni si presuppone la conoscenza di Javascript con standard ES6 [42].

3.1 EventTarget

Prima di cominciare verrà introdotta l'interfaccia EventTarget [15], un elemento molto importante per la registrazione di ascoltatori ad eventi. Tutte le classi che verranno descritte avranno la propria implementazione di questa interfaccia ed è quindi meglio introdurla per prima. Essa possiede solo tre metodi principali:

- `addEventListener`: crea un ascoltatore di un evento;
- `removeEventListener`: rimuove un ascoltatore già creato;
- `dispatchEvent`: attiva l'evento stesso.

3.2 Promise

Un altro elemento importante è la classe Promise [29], rappresentante un evento asincrono. Nella creazione di una Promise viene passata una funzione (*executor*) come parametro che, a sua volta, chiede due funzioni come parametri: una funzione di risoluzione (*resolution*) e una funzione di rifiuto (*rejection*). Se la funzione principale (*executor*) si risolve senza errori, alla fine della sua esecuzione verrà eseguita la funzione *resolution* altrimenti la funzione *rejection*. Attraverso il metodo `then()` della Promise è possibile passare come parametri le funzioni *resolution* e *rejection* (entrambi opzionali), mentre con il metodo `catch()` è possibile catturare qualsiasi funzione *rejection* (utile in caso di concatenazione di Promise).

```
// risolve la Promise se un valore casuale tra 0 e 1 è maggiore di
    0.5
// il valore viene ritornato dopo un secondo
const getGuess = new Promise((resolve, reject) => {
  setTimeout(() => {
    let guess = Math.random();
    if (guess > 0.5) {
      resolve(‘Risolto con ${guess}‘);
    } else {
      reject(‘Qualcosa è andato storto... ${guess}‘);
    }
  }, 1000);
});

getGuess
  .then((resolutionMsg) => console.log(resolutionMsg))
  .catch((errorMsg) => console.log(errorMsg))
```

3.3 Sensors API

L’interfaccia Sensor [17] definisce l’interfaccia per l’implementazione delle classi Accelerometer, Gyroscope, Magnetometer, AmbientLightSensor, AbsoluteOrientation-Sensor, GravitySensor, LinearAccelerationSensor e RelativeOrientationSensor; i primi quattro rappresentano i sensori fisici, mentre gli altri sensori sono creati virtualmente dall’unione di due o più sensori fisici. Prima di poter accedere ai dati nei sensori è necessario verificare due cose molto importanti:

1. la presenza delle API per la lettura dei dati,

2. la presenza effettiva del sensore.

Per verificare la presenza di API ci sono tre metodi possibili, due dei quali verificano la presenza del costruttore del sensore nella variabile globale `window`, da cui sono accessibili tutte le proprietà, metodi ed eventi della pagina o della finestra del browser stesso.

```
// controlla direttamente che Accelerometer sia parte di window
if (window.Accelerometer) { ... }

// verifica che Accelerometer sia una proprietà dell'oggetto window
if ("Accelerometer" in window) { ... }

// verifica che Accelerometer esista e che sia una funzione
if (typeof Accelerometer === "function") { ... }
```

Una volta verificata la presenza di API è necessario verificare la presenza del sensore fisico e che non vi siano errori durante la lettura: questo processo prende il nome di "Defensive Programming" e si basa sul controllo degli errori in fase di istanziazione dell'oggetto e della lettura dei dati, per non degradare l'esperienza utente. Questo è un esempio di come può essere gestito:

```
let accelerometer = null;
// try...catch per gestire il caso in cui il sensore non sia
    disponibile
try {
  accelerometer = new Accelerometer();

  // creazione degli ascoltatori
  // in ascolto per errori
  accelerometer.addEventListener('error', (event) => { /* gestione
      degli errori */ });

  // lettura dei dati
  accelerometer.addEventListener('reading', () => { /* gestione
      della lettura dei dati */ });

  accelerometer.start();
} catch (error) {
  throw error;
}
```

Attraverso la funzione `query()` dell'oggetto `permission` (proprietà di `navigator`) è possibile verificare che i permessi per l'accesso ai sensori siano stati accettati dall'u-

tente. L'oggetto `navigator` rappresenta le preferenze e lo stato dello user-agent ed è possibile ricavare informazioni come layout della tastiera (`navigator.keyboard`), lingua predefinita impostata sul browser (`navigator.language`), capacità di memoria del dispositivo (`navigator.deviceMemory`), etc. È possibile consultare l'intera lista di proprietà e metodi al link seguente: `https://developer.mozilla.org/en-US/docs/Web/API/Navigator`.

```
navigator.permissions.query({ name: 'accelerometer' })
.then((result) => {
  if (result.state === 'denied') { ... }
  // è possibile usare il sensore
});
```

L'interfaccia `Sensor` definisce tre proprietà principali, presenti in ogni sensore:

- `activated`: indica se il sensore è attivo o meno;
- `hasReading`: indica se sono presenti delle letture;
- `timestamp`: indica il timestamp dell'ultima lettura.

I due metodi principali, `start()` e `stop()` hanno come scopo rispettivamente di attivare il sensore per la lettura o disattivarlo.

`Sensor` implementa anche l'interfaccia `EventTarget` che permette di mettersi in ascolto dei seguenti eventi.

- `activate`: il sensore è attivo, pronto per leggere.
- `error`: il sensore non riesce a dare delle letture corrette; può essere per un permesso revocato o un problema con il sensore fisico.
- `reading`: il sensore ha una lettura disponibile.

Per poter leggere i valori da un sensore che implementa Sensor è necessario mettersi in ascolto dell'evento `"reading"` ed avviare le letture con il metodo `start()`.

3.3.1 Accelerometro

Per accedere ai dati dell'accelerometro si può utilizzare la classe `Accelerometer()` che implementa l'interfaccia `Sensor`. Al costruttore della classe è possibile passare come parametro un oggetto `options` opzionale con due proprietà principali:

- `frequency`: indica il numero di letture da effettuare in un secondo; se espresso con un numero decimale indica frequenze minori di un secondo

- `referenceFrame`: indica il sistema di riferimento, può essere o "device" o "screen"; default è "device"

Per accedere ai dati basta accedere alle proprietà seguenti:

- `x`: ritorna un double contenente l'accelerazione su asse x;
- `y`: ritorna un double contenente l'accelerazione su asse y;
- `z`: ritorna un double contenente l'accelerazione su asse z.

Le proprietà sono accessibili dopo essersi messi in ascolto di un evento di lettura, ovvero "reading". Per fare ciò è necessario invocare il metodo `addEventListener` sull'oggetto `Accelerometer`.

```
let accelerometer = new Accelerometer({ frequency: 60 });
accelerometer.addEventListener('reading', () => {
  console.log(`Accelerazione su asse x: ${accelerometer.x}`);
  console.log(`Accelerazione su asse y: ${accelerometer.y}`);
  console.log(`Accelerazione su asse z: ${accelerometer.z}`);
});
accelerometer.start();
```

Per avviare la lettura dei dati è necessario richiamare il metodo `start()` per rendere il sensore attivo. In caso di errori, il sensore ritornerà "idle" e sarà necessario richiamare il metodo `start()` per rendere il sensore di nuovo attivo. Altri metodi ed eventi sono ereditati dall'interfaccia `Sensor` ed `EventTarget`.

Per più informazioni, visitare `https://developer.mozilla.org/en-US/docs/Web/API/Accelerometer`

3.3.2 Giroscopio

Per accedere ai dati del giroscopio si può utilizzare la classe `Gyroscope` che implementa l'interfaccia `Sensor`. Come per l'accelerometro, al costruttore della classe è possibile passare come parametro un oggetto `options` opzionale con due proprietà principali:

- `frequency`: indica il numero di letture da effettuare in un secondo; se espresso con un numero decimale indica frequenze minori di un secondo
- `referenceFrame`: indica il sistema di riferimento, può essere o "device" o "screen"; default è "device"

Per accedere ai dati basta accedere alle proprietà seguenti:

- `x`: ritorna un double contenente la velocità angolare sull'asse x;
- `y`: ritorna un double contenente la velocità angolare sull'asse y;
- `z`: ritorna un double contenente la velocità angolare sull'asse z.

Le modalità di lettura sono le stesse dell'accelerometro. Le proprietà sono accessibili dopo essersi messi in ascolto dell'evento "reading" utilizzando il metodo `addEventListener()` ereditato dall'interfaccia `EventTarget`.

```
let gyroscope = new Gyroscope({ frequency: 60 });
gyroscope.addEventListener('reading', () => {
  console.log(`Velocità angolare su asse x: ${gyroscope.x}`);
  console.log(`Velocità angolare su asse y: ${gyroscope.y}`);
  console.log(`Velocità angolare su asse z: ${gyroscope.z}`);
});
gyroscope.start();
```

Per avviare la lettura dei dati è necessario richiamare il metodo `start()` per rendere il sensore attivo. In caso di errori, il sensore ritornerà idle e sarà necessario richiamare il metodo `start()` per rendere il sensore di nuovo attivo. Altri metodi ed eventi sono ereditati dall'interfaccia `Sensor` ed `EventTarget`.

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/Gyroscope`

3.3.3 Magnetometro

Per accedere ai dati del magnetometro si può utilizzare la classe `Magnetometer` che implementa l'interfaccia `Sensor`. Come per l'accelerometro, al costruttore della classe è possibile passare come parametro un oggetto `options` opzionale con due proprietà principali:

- `frequency`: indica il numero di letture da effettuare in un secondo; se espresso con un numero decimale indica frequenze minori di un secondo
- `referenceFrame`: indica il sistema di riferimento, può essere o "device" o "screen"; default è "device"

Per accedere ai dati basta accedere alle proprietà seguenti:

- `x`: ritorna un double contenente il campo magnetico attorno all'asse x;

- `y`: ritorna un double contenente il campo magnetico attorno all'asse y;
- `z`: ritorna un double contenente il campo magnetico attorno all'asse z.

Le modalità di lettura sono le stesse dell'accelerometro. Le proprietà sono accessibili dopo essersi messi in ascolto dell'evento "reading" utilizzando il metodo `addEventListener()` ereditato dall'interfaccia `EventTarget`.

```
let magnetometer = new Magnetometer({ frequency: 60 });
magnetometer.addEventListener('reading', () => {
  console.log(`Campo magnetico su asse x: ${magnetometer.x}`);
  console.log(`Campo magnetico su asse y: ${magnetometer.y}`);
  console.log(`Campo magnetico su asse z: ${magnetometer.z}`);
});
magnetometer.start();
```

Per avviare la lettura dei dati è necessario richiamare il metodo `start()` per rendere il sensore attivo. In caso di errori, il sensore ritornerà idle e sarà necessario richiamare il metodo `start()` per rendere il sensore di nuovo attivo. Altri metodi ed eventi sono ereditati dall'interfaccia `Sensor` ed `EventTarget`.

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/Magnetometer`.

3.3.4 Sensore di luce ambientale

Per accedere ai dati del sensore di luce ambientale si può utilizzare la classe `AmbientLightSensor` che implementa l'interfaccia `Sensor`. Al costruttore della classe è possibile passare come parametro un oggetto `options` opzionale contenente la proprietà `frequency` che, se indicato con un numero intero, indica il numero di letture al secondo, mentre se indicato con un numero decimale, indica le frequenze minori di un secondo ($1/n$ dove n è il numero di letture in un secondo).

L'unica proprietà presente è `illuminance` che rappresenta in lux l'intensità di luce dell'ambiente circostante. `AmbientLightSensor` non ha metodi o eventi propri: sono tutti ereditati da `EventTarget` e `Sensor`.

Un piccolo esempio per come accedere al sensore:

```
const light = new AmbientLightSensor();
light.addEventListener("reading", (event) => {
    console.log("Livello di luce attuale: ", light.illuminance);
});
light.start();
```

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/AmbientLightSensor`.

3.4 Sensore di prossimità

Le interfacce che sfruttano il sensore di prossimità sono deprecate e sono presenti in questo documento a puro scopo illustrativo.

Sono presenti due tipi di eventi di prossimità: `UserProximityEvent` e `DeviceProximityEvent`. `UserProximityEvent` ha una sola proprietà che indica se il sensore ha captato un oggetto fisico nelle vicinanze: `near`. È rappresentato da un Boolean e non dà alcuna informazione riguardo alla distanza del dispositivo.

```
window.addEventListener('userproximity', (event) => {
  if(event.near) console.log("C'è un oggetto vicino!");
  else { ... }
});
```

Al contrario, per `DeviceProximityEvent` sono presenti tre proprietà che descrivono più a fondo i dati forniti dal sensore: con `max` e `min` si ricevono informazioni riguardo alla distanza massima e minima rilevabile in centimentri, mentre con `value` si ottiene la distanza in centimentri dell'oggetto più vicino.

```
window.addEventListener('deviceproximity', (event) => {
  console.log("Distanza massima rilevabile: ", event.max);
  console.log("Distanza minima rilevabile: ", event.min);
  console.log("Distanza dell'oggetto più vicino", event.value);
});
```

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/Proximity_Events`

3.5 Batteria

L'interfaccia BatteryManager viene usata per mettersi in ascolto degli eventi che notificano quando c'è un cambio dello stato della batteria (se in carica o cambio di livello) e può essere usata per regolare l'utilizzo di risorse quando la batteria è quasi scarica. Per ricavare informazioni sulla batteria è necessario ottenere un oggetto di tipo `BatteryManager` e per fare ciò si deve risolvere la Promise ritornata da `navigator.getBattery()`, che contiene un oggetto BatteryManager.

Tutti i metodi della classe `BatteryManager` sono ereditati da `EventTarget`. Una volta ottenuto l'oggetto ci si può mettere in ascolto degli eventi per ottenere informazioni (in sola lettura) dalle proprietà dell'oggetto con `addEventListener`. Qui di seguito ci sono le proprietà accessibili dall'oggetto BatteryManager:

- `charging`: indica se il dispositivo è in carica o meno;
- `chargingTime`: indica il tempo in secondi necessario alla carica completa; ritorna Infinity se non è in carica;
- `dischargingTime`: indica il tempo in secondi necessario alla scarica completa; ritorna Infinity se è in carica;
- `level`: Number tra 0 e 1 che indica il livello della batteria (0 - scarica, 1 - pieno).

Gli eventi in cui ci si può mettere in ascolto sono i seguenti:

- `chargingchange`: il valore di `charging` è aggiornato;
- `chargingtimechange`: il valore di `chargingTime` è aggiornato;
- `dischargingtimechange`: il valore di `dischargingTime` è aggiornato;
- `levelchange`: il valore di `level` è aggiornato.

Un piccolo esempio di utilizzo:

```
navigator.getBattery().then((battery) => {
  battery.addEventListener('charging', () => {
    console.log("La batteria è in carica?", battery.charging);
  });
  battery.addEventListener('levelchange', () => {
    console.log("La batteria è al ", battery.level * 100);
  });
});
```

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/BatteryManager`.

3.6 Fotocamera e microfono

Per ottenere i dati da microfono e fotocamera bisogna intercettare il flusso di dati che arriva dal dispositivo. Non sono in alcun modo gestibili le impostazioni della

fotocamera e i dati possono essere solo letti o modificati. Un flusso di dati viene implementato attraverso l'interfaccia `MediaStream` e si possono ottenere i flussi da fotocamera e microfono in tre modi: con `MediaDevices.getUserMedia()`, `MediaDevices.getDisplayMedia()` o `HTMLCanvasElement.captureStream()`. Tutti e tre i metodi ritornano una Promise che si risolve con l'oggetto `MediaStream` se il permesso è accettato dall'utente o viene rifiutata se il permesso è negato. Nei primi due metodi è possibile specificare delle opzioni minime richieste (se audio o video, dimensione della finestra, fotocamera anteriore o posteriore, etc.). Un oggetto `MediaStream` è un flusso formato da oggetti `MediaStreamTrack` che rappresentano le tracce di cui è formato il flusso, che possono essere audio o video, a seconda dell'origine (fotocamera o microfono).

```
// MediaDevices.getUserMedia
let constraints = { audio: true, video: true }
navigator.mediaDevices.getUserMedia(constraints)
  .then((stream) => { ... });

// con MediaDevices.getDisplayMedia
navigator.mediaDevices.getDisplayMedia({ audio: true, video: true
    }})
  .then((stream) => { ... });

// con HTMLCanvasElement.captureStream, è necessario il canvas
const canvas = document.querySelector('canvas');
const stream = canvas.captureStream(60); // -> 60 FPS
```

Attraverso le proprietà di `MediaStream` è possibile verificare che lo stream è attivo con `active` e visualizzare il suo UUID, una stringa di 36 caratteri che identifica univocamente il flusso, con `id`.

`MediaStream` offre metodi per: aggiungere una traccia `MediaStreamTrack`, con la funzione `addTrack(track)`, togliere una traccia con `removeTrack()`. Per ottenere le tracce si può usare il metodo `getTracks()`, che ritorna una lista di `MediaStreamTrack`, opuure selezionarla direttamente con l'id usando `getTrackById(id)`. Per ottenere tutte le tracce audio basta richiamare il metodo `getAudioTracks()` e per ottenere tutte le tracce video basta usare `getVideoTracks()`. Infine è possibile clonare il flusso con il metodo `clone()` che ne crea uno identico ma con un UUID diverso.

`MediaStream` implementa `EventTarget` per cui è possibile mettersi in ascolto dei seguenti eventi:

- `addtrack`: viene aggiunto un `MediaStreamTrack`;

- `removetrack`: viene rimosso un `MediaStreamTrack`;
- `active`: lo stato di attivo cambia;
- `inactive`: lo stato di inattivo cambia.

`MediaStreamTrack` è l'interfaccia che rappresenta invece una traccia, audio o video, presente in un flusso `MediaStream`. Ha numerose proprietà che ne descrivono il contenuto:

- `contentHint` contiene una descrizione sul contenuto;
- `enabled`: true -> la traccia è rappresentabile per il suo contenuto, false -> verranno segnati frame di silenzio per tracce audio e pixel neri per tracce video;
- `id`: GUID, id della traccia (read-only);
- `kind`: tipo della traccia: "audio" o "video";
- `label`: nome descrittivo della sorgente;
- `muted`: true -> la traccia non è in grado di fornire dati;
- `readyState`: "live" se i dati sono in real time, "ended" se non verranno più mandati alcun tipo di dati;
- `remote`: true -> lo stream deriva da una RTCPeerConnection.

Anche per `MediaStreamTrack` sono definiti alcuni metodi: con `applyCostraints(constraints)` è possibile applicare dei vincoli e ritorna una Promise che si risolve se i vincoli sono stati applicati con successo. I vincoli applicabili sono ritornati dal metodo `getCapabilities()` e possono essere frame rate, dimensioni, cancellazione del rumore, etc. Con `getConstraints()` è possibile visualizzare i vincoli che sono stati applicati, mentre con `getSettings()` è possibile visualizzare i valori presi per ciascun vincolo. Infine con `clone()` e `stop()` è possibile duplicare o fermare la traccia in riproduzione da una sorgente.

Anche `MediaStreamTrack` implementa `EventTarget` ed è possibile mettersi in ascolto dei seguenti eventi:

- `ended`: il playback della traccia è finito o non più disponibile;
- `mute`: la traccia non è in grado di mandare dati temporaneamente;
- `overconstrained`: troppi vincoli rendono la traccia incompatibile e non può essere usata (deprecato);

- `unmute`: i dati della traccia sono di nuovo disponibili.

Per ottenere una foto/frame è necessario usare il costruttore `ImageCapture()` [27] passando come parametro un `MediaStreamTrack` di tipo video (`kind` = "video"). Creato l'oggetto, è possibile richiamare due metodi per estrarre un'immagine: `takePhoto()` o `grabFrame()`. Con `takePhoto()` viene ritornata una Promise che si risolve con un `Blob`, un oggetto che rappresenta dati grezzi che possono essere letti come dati binari o convertiti in un formato a scelta. Con `grabFrame()` viene ritornata una Promise che si risolve con un oggetto `ImageBitmap`, utilizzato principalmente per disegnare su un <canvas> HTML con una latenza molto ridotta.

Per maggiori informazioni visita `https://developer.mozilla.org/en-US/docs/Web/API/MediaStream` e `https://developer.mozilla.org/en-US/docs/Web/API/MediaStreamTrack`

3.7 LOCALIZZAZIONE

Le Geolocation API permettono di conoscere la posizione dell'utente attraverso i servizi di geolocalizzazione del dispositivo e per accedervi è necessario l'oggetto `window.navigator`. Sono presenti due modi con cui è possibile accedere alla posizione dell'utente: `getCurrentPosition()` e `watchPosition()`. Entrambi richiedono una funzione callback obbligatoria in grado di gestire una `GeolocationPosition` e altre due funzioni opzionali, rispettivamente, per la gestione degli errori e per indicare altre opzioni sulla posizione. La differenza tra le due funzioni è che `watchPosition()` registra un ascoltatore che richiama la funzione callback ogni qualvolta la posizione cambia, mentre `getCurrentPosition()` ritorna la posizione una sola volta.

Una `GeolocationPosition` contiene due proprietà: `coords` che consiste in un oggetto `GeolocationCoordinates` e `timestamp`, un numero che rappresenta il tempo in cui le coordinate sono state ricavate.

Un oggetto `GeolocationCoordinates` contiene le tre proprietà principali: `latitude`, `longitude` e `altitude` che sono rispettivamente latitudine, longitudine e altitudine espresse in metri. Altre proprietà dell'oggetto includono `accuracy`, che indica l'accuratezza in metri della latitudine e longitudine, `altitudeAccuracy` per l'accuratezza dell'altitudine, `heading` e `speed` che indicano rispettivamente l'inclinazione in gradi rispetto al Nord geografico e la velocità del dispositivo in metri al secondo.

Listing 3.1: Esempio di utilizzo di `watchPosition()`

```
const options = {
  enableHighAccuracy: true,
  timeout: 2000,
  maximumAge: 0
};

function success(position) {
  const coordinates = position.coords;
  console.log(‘Latitudine misurata : ${coordinates.latitude}‘);
  console.log(‘Longitudine misurata: ${coordinates.longitude}‘);
}

function error(err) {
  console.log(err.code);
}

navigator.geolocation.getCurrentPosition(success, error, options);
```

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API`

3.8 NFC

Web NFC API permette la comunicazione NFC solo attraverso i messaggi NDEF (NFC Forum Data Exchange Format). Sono presenti tre interfacce principali: `NDEFMessage`, `NDEFReader` e `NDEFRecord`. L'interfaccia `NDEFReader` permette di leggere o scrivere dati su un tag NFC disponibile. Il construttore non richiede alcun parametro e restituisce un oggetto `NDEFReader`. Sono presenti due metodi principali.

- `scan()`: attiva la modalità di ricerca di tag NFC nelle vicinanze, ritorna una Promise che si risolve con un NFC tag letto o con un errore che può essere di tipo hardware o di permessi. Attiva un prompt per richiedere il permesso di utilizzo del sensore.
- `write()`: tenta di scrivere un `NDEFMessage` su un tag NFC, ritorna una Promise che si risolve se la scrittura avviene correttamente o la rifiuta se incontra un errore che può essere di tipo hardware o di permessi. Attiva un prompt per richiedere il permesso di utilizzo del sensore.

È possibile associare un ascoltatore per ciascuno dei seguenti eventi:

- `reading`: è disponibile una nuova lettura da dispositivi NFC;

- `readingerror`: si verifica un errore durante la lettura di un tag NFC.

Per costruire un `NDEFMessage` è necessario il costruttore definito in questo modo: `new NDEFMessage(records)` dove `records` è un array di records definiti da `NDEFRecord`.

Per accedere ai records di un `NDEFMEssage` basta accedere alla proprietà `records` che ritorna una lista `NDEFRecord`.

Un `NDEFRecord` ha un proprio costruttore, `NDEFRecord(options)` dove `options` è un oggetto che contiene tutte le proprietà del record:

- `data`: contiene i dati da trasmettere (opzionale)
- `enconding`: determina la codifica dei record (opzionale)
- `id`: identificatore del record, scelto dallo sviluppatore (opzionale)
- `lang`: indica il language tag definito da RFC 5646 (opzionale)
- `mediaType`: MIME type (opzionale)
- `recordType`: indica il tipo dei dati, deve essere uno dei seguenti: "absolute-url", "empty", "mime", "smart-poster", "text", "unknown", "URL".

Tutte le proprietà specificate qui sopra sono pubblicamente accessibili in sola lettura come proprietà dell'oggetto. L'unico metodo di `NDEFRecord` è `toRecords()` che converte `data` i dati binari in una lista di `NDEFRecord`.

Per maggiori informazioni, visita `https://developer.mozilla.org/en-US/docs/Web/API/Web_NFC_API`

3.9 Bluetooth

Web Bluetooth API è un insieme di interfacce per la comunicazione tra dispositivi Bluetooth Low Energy. Attraverso un oggetto `Bluetooth` è possibile accedere alla proprietà `referringDevice` che ritorna un `BluetoothDevice` se la pagina è stata aperta in risposta ad un'istruzione ricevuta dal dispositivo o `null` in caso contrario. Con un oggetto `Bluetooth` è possibile invocare i seguenti metodi:

- `getAvailability()`: ritorna una Promise che si risolve con un valore booleano corrispondente all'abilità dello user agent nel supportare quest'interfaccia;
- `requestDevice(options)`: options è un parametro opzionale che contiene tutte le opzioni per la ricerca di un determinato dispositivo; ritorna una Promise che si risolve con i `BluetoothDevice` trovati;

- `getDevices()`: ritorna una Promise che si risolve con un array di `BluetoothDevice` disponibili che sono già stati richiesti con `requestDevice(options)`.

È possibile registrarsi all'evento "availabilitychanged" che cambia con la disponibilità del dispositivo Bluetooth.
Un oggetto `BluetoothDevice` rappresenta un dispositivo Bluetooth. Attraverso quest'oggetto è possibile accedere all'id identificativo del dispositivo con `id` oppure al nome del dispositivo con `name` o alla reference del GATTServer con `gatt`. I metodi sono i seguenti:

- `watchAdvertisements()`: ritorna una Promise che si risolve con undefined o viene rifiutata con un errore se gli advertisements non possono essere mostrati per qualsiasi motivo;
- `unwatchAdvertisements()`: ferma la visione degli advertisements.

Con un `BluetoothRemoteGATTService` è possibile accedere alle seguenti proprietà: `device` restituisce un oggetto `BluetoothDevice` con le informazioni sul dispositivo corrente, `isPrimary` ritorna un Boolean che indica se il Service è primario (true) o secondario (false) e `uuid` ritorna l'UUID del Service. Con questo oggetto è possibile effettuare le seguenti funzioni: `getCharacteristic` ritorna una Promise che si risolve con una `BluetoothRemoteGATTCharacteristic` specifica, `getCharacteristics` ritorna una lista di `BluetoothRemoteGATTCharacteristic`.
Un oggetto `BluetoothRemoteGattCharacteristic` rappresenta una GATT Characteristic, un elemento base che provvede a dare informazioni riguardo ad un Peripheral. È possibile accedere alle seguenti proprietà: `service` ritorna il `BluetoothRemoteGATTService` a cui appartiene, `uuid` ritorna il UUID della Characteristic, `properties` ritorna un `BluetoothCharacteristicProperties` contenente le proprietà della Characteristics, `value` ritorna il valore salvato in cache della Characteristic. È possibile mettersi in ascolto dell'evento `"oncharacteristicvaluechanged"` che cambia quando i dati in Characteristic cambiano. Per leggere il valore di una Characteristic si usa il metodo `readValue` che ritorna una Promise che si risolve con una DataView che contiene un duplicato del valore. Per scrivere il valore sono presenti tre metodi: `writeValue()`, `writeValueWithResponse()` e `writeValueWithoutResponse()`. Per la gestione delle notifiche sono presenti due metodi: `startNotifications()` per verificare la presenza di qualche notifica attiva e `stopNotifications()` che avvisa con una Promise quando non sono più presenti notifiche attive. Per ottenere il Descriptor della Characteristic basta usare `getDescriptor()` o `getDescriptors()`.

Per verificare quali operazioni sono effettuabili su un `BluetoothRemoteGattCharacteristic` è necessario accedere alle sue proprietà attraverso `properties` che ritorna un `BluetoothCharacteristicProperties`. Quest'ultimo contiene delle proprietà sulle modalità di accesso: per esempio con `read` si verifica se la Characteristic permette la lettura e con `write` se è permessa la scrittura. Tutte le altre proprietà sono direttamente consultabili dalla documentazione `https://developer.mozilla.org/en-US/docs/Web/API/BluetoothCharacteristicProperties#properties`.
Un oggetto `BluetoothRemoteGATTServer` rappresenta un GATT Server in un dispositivo remoto. Attraverso la proprietà `device` è possibile accedere all'oggetto `BluetoothDevice` che rappresenta il dispositivo, mentre con i metodi `getPrimaryService()` e `getPrimaryServices()` viene ritornato un Service o una lista di Service associati al Server rispettivamente. Con `connect()` e `disconnect()` è possibile collegare o scollegare un dispositivo al Server.

Per maggiori dettagli o approfondimenti, visita `https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API`.

4

Considerazioni

Da quanto visto finora, molti sensori come accelerometro, batteria e luce ambientale, sono facilmente accessibili e non richiedono alcuna autorizzazione all'utente riguardo al loro utilizzo. Come ben sintetizzato dalla W3C [50], le implementazioni di alcuni dei sensori sopra descritti sono ancora in fase sperimentale: l'interfaccia Sensor, sensore di luce ambientale, sensore di prossimità, batteria, sensori di movimento e localizzazione, mentre altri sono in fase di esplorazione: camera e microfono, NFC e Bluetooth. Come si può osservare per il futuro, le web app saranno ben fornite di potenzialità che appartengono anche alle app native, ma è altrettanto considerata la privacy dell'utente? L'accesso indisturbato ai sensori ha portato alcuni studi a verificare che questo non comporti qualche abuso di utilizzo ed è scopo di questo capitolo riportare qui quelli più interessanti.

4.1 Browser Fingerprinting

Per cominciare, alcuni degli attacchi sono effettuati per studiare l'utente, per fare "tracking" delle sue azioni. Una delle forme più subdole di tracciamento online è il browser fingeprinting. Con questo termina si indica un modo per identificare univocamente il browser di un utente creando un'impronta univoca basata su tutte le informazioni ricavabili da script o meno e si definisce fingerprintability il livello di difficoltà nell'identificare in modo univoco uno specifico dispositivo attraverso il fingerprinting. Per riuscire a creare un'impronta più unica possibile, le informazioni raccolte devono essere il più specifiche possibile: dati come impostazioni del browser (lingua impostata, font installato, versione) ricavati passivamente e altri come risoluzione dello schermo,

indirizzo IP, sensori e timezone che richiedono script e analisi dei pacchetti inviati. L'individuazione di errori di funzionamento dei sensori può essere utile per rendere il dispositivo ancora più unico, ad esempio attraverso il Battery Status API, da cui è possibile ricavare informazioni sulla capacità della batteria. Secondo uno studio, la fingerprintability è maggiore con batterie vecchie o con ridotta capacità [31]. Ironicamente anche l'attivazione del "Do Not Track" aiuta a rendere unico il browser [7] e l'utilizzo di certe estensioni o plugin presenti in un browser aiutano a renderlo ancora più unico [34] [21]. Questa tecnica è prevalentemente utilizzata nel cross-site tracking, una forma di tracciamento dell'utente che è esercitata da compagnie di terze parti che collezionano dati da siti multipli. Se il dominio del sito non blocca l'accesso alle feature attraverso la Feature-Policy [16], il rischio è che qualsiasi iframe HTML presente nel sito principale possa accedere a qualsiasi feature. Infatti, gli iframe (in HTML, `<iframe>` permettono di "ospitare", sulla propria pagina, un'altra pagina web [33]. Altre tecniche per ridurre la fingerprintability sono definito dalla W3C come best practices con nessun tipo di obbligatorietà [39]. Un altro punto fondamentale del fingerprinting è la sua efficacia in qualsiasi condizione del browser, sia che esso sia in modalità incognito o che i cookies siano completamente disabilitati, e permette la re-identificazione dell'utente agendo come un cookie (cookie-like [39]).

4.1.1 Diffusione del fenomeno

In un importante studio del 2018 [8] sono stati rilevati dati molto importanti sui primi 100.000 siti più popolari elencati da Alexa Internet [43]:

- 3695 sfruttavano i sensors API;
- i sensori di movimento e rotazione, i più frequentemente utilizzati, erano presenti rispettivamente in 2653 e 2036 siti;
- sono stati trovati numerosi script che accedono ai dati dei sensori che poi vengono inviati in chiaro o codificati in base64 ai server remoti;
- tra gli script che accedono ai sensori, il 36.8% erano utilizzati per tracking e analytics, verificare ad impressions e distinguere i dispositivi reali da bot ed erano presenti in 1198 siti;
- il 62.7% degli script che accedono ai sensori erano utilizzati anche per fingerprinting;
- estensioni di ad blocking e liste di protezione da tracking (EasyList, EasyPrivacy e Disconnect) sono alquanto inefficaci: per esempio, dalla lista creata da Disconnect

[14], un'azienda che offre servizio di protezione da tracker [13], solo l'1.8% degli script che accedono ai sensori di movimento sono bloccati.

4.2 Attacchi tramite i sensori

In questa sezione verranno descritti una serie di attacchi alla privacy che sono avvenuti con successo usando solo i sensori accessibili da una applicazione web. I sensori coinvolti sono stati principalmente accelerometro, giroscopio e luce ambientale; nessuno di questi tre richiede un permesso di utilizzo da parte dell'utente e la lettura dei dati avviene in modalità silenziosa, ovvero a insaputa dell'utente.

4.2.1 Sensori di movimento

Nel 2016 è stato condotto un esperimento [23] utilizzando i sensori di movimento e orientamento (accelerometro, giroscopio e magnetometro). L'esperimento ha dimostrato che, con la sola lettura di tali sensori, è possibile riconoscere gestures come click, scroll, hold e zoom e ricavare un codice PIN da quattro cifre. Per la prima parte dell'esperimento è stato chiesto a undici persone di eseguire i quattro movimenti su un sito creato appositamente. Sono poi stati creati due classificatori per il riconoscimento dei gesti: il primo per distinguerli tra loro mentre il secondo per riconoscere la direzione degli scroll (up, down, left, right). L'identificazione generale tra i quattro gesti ha una confidenza del 87.39%. Per la seconda parte è stato chiesto a dodici persone di digitare una serie di PIN da quattro cifre su due dispositivi diversi, un Nexus 5 (Android) e un iPhone 5 (iOS), sempre usando come base un sito creato appositamente per la registrazione dei sensori. Infine è stato allenato un classificatore per il riconoscimento dei movimenti di ciascuno dei partecipanti e i risultati ottenuti sono i seguenti: il classificatore riesce a predire in modo corretto in almeno il 90% dei casi sul Nexus 5 e l'80% dei casi sull'iPhone 5 al terzo tentativo, probabilmente dovuto al fatto che le letture in Android sono molto più frequenti che in un dispositivo Apple. Questo esperimento dimostra inoltre la possibilità di poter fare user fingerprinting, ovvero il poter creare un'impronta basandosi solo sul modo biologico di scrivere sullo schermo.

Un altro tentativo di user fingerprinting è avvenuto in uno studio del 2018 [51]. In questo studio è stato allenato un classificatore multi-classe unificando diversi altri classificatori. Questi classificatori sono stati allenati con i dati dei sensori registranti la

pressione di dieci lettere sulla tastiera dello schermo (chiamati keystroke da qui in poi). L'esperimento è stato eseguito con meccaniche molto simili a quello precedente: venti partecipanti hanno partecipato a 25 sessioni di "scrittura" e si sono registrati un totale di 100 keystrokes per 39 caratteri diversi (26 lettere, 10 cifre e 3 caratteri speciali). Per il primo risultato, il classificatore unificato con tre classificatori allenati su 10 keystrokes ha un'accuratezza del 73% mentre con sette classificatori arriva all'85%. L'accuratezza dei modelli migliora con un dataset più grande: un classificatore unificato con tre classificatori allenati su 100 keystrokes ha un'accuratezza del 90% mentre con sette classificatori allenati arriva fino al 95%.

4.2.2 Giroscopio

Il giroscopio è un sensore molto utile nei dispositivi mobili, soprattutto per quanto riguarda l'orientamento del dispositivo, ma recenti studi hanno dimostrato che può essere utilizzato per altri scopi. In uno studio del 2017 [25] si è dimostrato che i moderni giroscopi presenti negli smartphone sono sufficientemente sensibili per misurare segnali acustici emessi nelle vicinanze del dispositivo. I segnali ottenuti dal giroscopio contengono informazioni a basse frequenze ma, processando e analizzando il segnale con algoritmi di machine learning, è possibile ricavare informazioni sulla persona che parla e, con certi limiti, a identificare un discorso. Come già visto, l'accesso al microfono richiede il permesso all'utente mentre per il giroscopio non è necessario alcun permesso e la registrazione dei dati può avvenire a completa insaputa dell'utente. In un altro recente studio [37] si è dimostrato come è possibile identificare un dispositivo attraverso fingerprinting del giroscopio, o per la precisione, della frequenza di risonanza del giroscopio. Il metodo consiste nel generare un ultrasuono e registrare la risposta del giroscopio attraverso il browser. Dopo che l'output del sensore viene preprocessato, vengono estratte delle feature sulla risonanza dopo un'analisi nel dominio delle frequenze. Queste feature saranno poi utilizzate per riconoscere i diversi dispositivi e alcune di queste sono stabili anche quando l'impugnatura del dispositivo cambia nel tempo. Sono state trovate un totale di 10 feature basate sulla frequenza di risonanza che permettono di migliorare l'accuratezza della classificazione fino al 96.5%.

4.2.3 Sensore di luce ambientale

In uno studio del 2020 [30] è stato dimostrato un tipo di attacco alla privacy usando il solo sensore di luce ambientale. Dopo questo esperimento molti browser hanno deciso di rimuovere l'accesso al sensore (Mozilla) oppure di non implementarlo (Safari) mentre

altri hanno deciso di non renderlo disponibile con le impostazioni di default (Chrome e Edge). In questo esperimento è stato usato il sensore di luce ambientale e il tag CSS `:visited` per capire se un determinato sito è stato visitato o meno e, in tal modo, carpire la cronologia dell'utente. L'idea nella realizzazione dell'attacco consiste nell'iterare una lista di siti prescelti dallo script e, sfruttando il tag CSS, di mostrare uno sfondo bianco su un sito già visitato e nero altrimenti, per poi captare la luce riflessa della luce sulla pelle umana o dell'ambiente. Tenendo conto della frequenza del sensore e del ritardo nel cambio di colore, si può arrivare a individuare la cronologia di ben 1000 siti diversi in 8 minuti e 20 secondi. Questo attacco è particolarmente efficace in ambienti scuri con luminosità del dispositivo alta. Due fattori di questo esperimento sono molto importanti: questo codice è eseguibile da iframe su un sito principale e, al tempo di scrittura dello studio, l'accesso al sensore non richiede il permesso dell'utente, come per i sensori di movimento.

4.3 Soluzioni

In questa sezione si vuole portare l'attenzione su due browser molto popolari: Mozilla Firefox e Safari (sia versione PC che mobile), che hanno deciso di non implementare o bloccare l'accesso ai sensori: più precisamente tali browser hanno deciso di non implementare l'interfaccia Sensor, come si può vedere nella figura 4.1. Entrambi i browser si sono mossi a difesa della privacy dell'utente perchè presente nei loro principi [26] [2] mentre altri browser come Google Chrome, Edge e Opera non hanno adoperato alcun comportamento in tale campo. Per qualsiasi informazione riguardo alla compatibilità delle diverse funzioni nei diversi browser, è utile consultare il sito `https://caniuse.com/` dove sono raccolte tutte le informazioni a riguardo.

4.3.1 Mozilla Firefox

In un articolo pubblicato da Mozilla Security Blog [4] è stato annunciato che dalla versione 72 (2020) è presente l'Enhanced Tracking Protection (ETP), uno strumento per proteggere gli utenti da cross-site tracking effettuato da terze parti sui siti principali; in particolar modo viene indicata la protezione da attacchi di "browser fingerprinting". Per proteggere da attacchi di questo tipo, Mozilla ha intrapreso due strade:

1. bloccare tutte le richieste che arrivano da compagnie riconosciute per fare cross-site tracking, listate da Disconnect [14];
2. rimuovere o cambiare API che contribuiscono al fingerprinting dell'utente

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
`Sensor`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`activate` event	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`activated`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`error` event	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`hasReading`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`reading` event	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`start`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`stop`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67
`timestamp`	✓ 67	✓ 79	✗ No	✗ No	✓ 54	✗ No	✓ 67	✗ No	✓ 48	✗ No	✓ 9.0	✓ 67

Figura 4.1: Compatibilità dell'interfaccia Sensor [18]. I numeri indicano le versioni dalle quali è presente il supporto. Firefox e Safari non offrono alcun supporto. Per maggiori informazioni, visita `https://caniuse.com/mdn-api_sensor`

Non è la prima volta che Mozilla è dovuta a ricorrere a queste azioni: già nel 2018 Firefox aveva rimosso anche l'accesso ai sensori di prossimità e luce ambientale (attivabili solo da impostazioni avanzate) per preservare la privacy dell'utente [3].

4.3.2 Apple Safari

Anche l'accesso ai sensori da parte di Safari cessa nel 2020, quando Apple decide di non implementare 16 web api a causa di problemi legati alla privacy dell'utente [7]. Tra questi 16 sono presenti: Web Bluetooth, Magnetometer, Web NFC, Network Information, Battery Status, Ambient Light sensor, Proximity, Web Geolocation e Web USB. I sensori di movimento (accelerometro e giroscopio) erano già stati disattivati di default nel 2018 [9] e devono essere attivati dall'utente nelle impostazioni avanzate. Come per Mozilla Firefox, Apple ha deciso di bloccare le implementazioni dei sensori per proteggere la privacy dell'utente da tracking.

Bibliografia

[1] Adafruit. *Introduction to Bluetooth Low Energy - GATT*. URL: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. (consultato il 29.08.2022).

[2] Apple. *Safari Browser*. URL: https://www.apple.com/safari/. (consultato il 2.09.2022).

[3] Bleepingcomputer. *Firefox Gets Privacy Boost By Disabling Proximity and Ambient Light Sensor APIs*. URL: https://www.bleepingcomputer.com/news/software/firefox-gets-privacy-boost-by-disabling-proximity-and-ambient-light-sensor-apis/. (consultato il 5.09.2022).

[4] Mozilla Security Blog. *Firefox 72 blocks third-party fingerprinting resources*. URL: https://blog.mozilla.org/security/2020/01/07/firefox-72-fingerprinting/?_gl=1*vqs9ci*_ga*MzEwMTgzMTgyLjE2NDAxODYxNjY.*_ga_MQ7767QQQW*MTY1NzAxMDAyOS4xLjEuMTY1NzAxMDE2MS4w. (consultato il 29.07.2022).

[5] Panda Suite Blog. *Native App vs Web App: Whats The Difference?* URL: https://blog.pandasuite.com/native-app-vs-web-app/. (consultato il 27.07.2022).

[6] William Bolton. «Chapter 2 - Instrumentation System Elements». In: *Instrumentation and Control Systems (Third Edition)*. A cura di William Bolton. Third Edition. Newnes, 2021, pp. 17–70. ISBN: 978-0-12-823471-6. DOI: https://doi.org/10.1016/B978-0-12-823471-6.00002-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128234716000022.

[7] ZDNet Catalin Cimpanu. *Apple declined to implement 16 Web APIs in Safari due to privacy concerns*. URL: https://www.zdnet.com/article/apple-declined-to-implement-16-web-apis-in-safari-due-to-privacy-concerns/. scritto il 28.06.2020 (consultato il 29.07.2022).

[8] Anupam Das et al. «The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors». In: *Proceedings of the 25th ACM Conference on Computer and Communication Security (CCS)*. ACM, ott. 2018. DOI: 10.1145/3243734.3243860. URL: https://doi.org/0.1145/3243734.3243860.

[9] Apple Developer. *Safari 12.1 Release Notes*. URL: https://developer.apple.com/documentation/safari-release-notes/safari-12_1-release-notes. (consultato il 29.07.2022).

[10] developer.apple.com. *Scegliere un abbonamento*. URL: https://developer.apple.com/it/support/compare-memberships/. (consultato il 26.07.2022).

[11] developer.mozilla.org. *Network Information API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Network_Information_API. (consultato il 27.07.2022).

[12] Google Developers. *Android Studio*. URL: https://developer.android.com/studio. (consultato il 1.09.2022).

[13] Disconnect. *Disconnect*. URL: https://disconnect.me/. (consultato il 5.09.2022).

[14] disconnectme. *disconnect-tracking-protection/services.json*. URL: https://github.com/disconnectme/disconnect-tracking-protection/blob/master/services.json. (consultato il 29.07.2022).

[15] MDN Web Docs. *EventTarget*. URL: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget. (consultato il 02.09.2022).

[16] MDN Web Docs. *Feature-Policy*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy. (consultato il 2.09.2022).

[17] MDN Web Docs. *Sensor*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Sensor. (consultato il 02.09.2022).

[18] MDN Web Docs. *Sensor APIs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs. (consultato il 29.07.2022).

[19] Flutter. *Flutter*. URL: https://flutter.dev/. (consultato il 1.09.2022).

[20] The Apache Software Foundation. *Apache Cordova*. URL: https://cordova.apache.org/. (consultato il 1.09.2022).

[21] Alejandro Gómez-Boix, Pierre Laperdrix e Benoit Baudry. «Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale». In: *WWW2018 - TheWebConf 2018 : The Web Conference, 27th International World Wide Web Conference*. Lyon, France, apr. 2018. DOI: 10.1145/3178876.3186097. URL: https://hal.inria.fr/hal-01718234.

[22] Trakkit GPS. *How GPS Works (Step-by-Step)*. URL: https://trakkitgps.com/how-gps-works/. (consultato il 29.08.2022).

[23] Maryam Mehrnezhad et al. «TouchSignatures: Identification of user touch actions and PINs based on mobile sensor data via JavaScript». In: *Journal of Information Security and Applications* 26 (2016), pp. 23–38. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2015.11.007. URL: https://www.sciencedirect.com/science/article/pii/S2214212615000678. (consultato il 29.07.2022).

[24] Inc. Meta Platforms. *React Native*. URL: https://reactnative.dev/. (consultato il 1.09.2022).

[25] Yan Michalevsky, Dan Boneh e Gabi Nakibly. «Gyrophone: Recognizing Speech from Gyroscope Signals». In: *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, ago. 2014, pp. 1053–1067. ISBN: 978-1-931971-15-7. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/michalevsky.

[26] Mozilla. *Firefox Browser*. URL: https://www.mozilla.org/it/firefox/new/. (consultato il 2.09.2022).

[27] Mozilla. *ImageCapture*. URL: https://developer.mozilla.org/en-US/docs/Web/API/ImageCapture. (consultato il 05.09.2022).

[28] Mozilla. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/. (consultato il 02.09.2022).

[29] Mozilla. *Promise*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. (consultato il 05.09.2022).

[30] Lukasz Olejnik. «Shedding light on web privacy impact assessment: A case study of the Ambient Light Sensor API». In: *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2020, pp. 310–313. DOI: 10.1109/EuroSPW51379.2020.00048. (consultato il 30.07.2022).

[31] Lukasz Olejnik et al. «The Leaking Battery». In: *Revised Selected Papers of the 10th International Workshop on Data Privacy Management, and Security Assurance - Volume 9481*. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 254–263. ISBN: 9783319298825. DOI: 10.1007/978-3-319-29883-2_18. URL: https://doi.org/10.1007/978-3-319-29883-2_18.

[32] Saggiamente. *MWC19: Nokia 9 PureView, il pentapartito fotografico*. URL: https://www.saggiamente.com/2019/02/mwc19-nokia-9-pureview-il-pentapartito-fotografico/. (consultato il 29.08.2022).

[33] W3 Schools. *HTML iframe tag*. URL: https://www.w3schools.com/tags/tag_iframe.asp. (consultato il 2.09.2022).

[34] Oleksii Starov e Nick Nikiforakis. «XHOUND: Quantifying the Fingerprintability of Browser Extensions». In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 941–956. DOI: 10.1109/SP.2017.18.

[35] Chrome Platform Status. *Feature: Web NFC*. URL: https://chromestatus.com/feature/6261030015467520. (consultato il 1.09.2022).

[36] support.google.com. *Service fees*. URL: https://support.google.com/googleplay/android-developer/answer/112622. (consultato il 26.07.2022).

[37] Junze Tian et al. «Mobile Device Fingerprint Identification Using Gyroscope Resonance». In: *IEEE Access* 9 (2021), pp. 160855–160867. DOI: 10.1109/ACCESS.2021.3131408.

[38] Kevin Townsend et al. *Getting Started with Bluetooth Low Energy - Chapter 4. GATT (Services and Characteristics)*. URL: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html. (consultato il 05.09.2022).

[39] W3C. *Mitigating Browser Fingerprinting in Web Specifications*. URL: https://www.w3.org/TR/fingerprinting-guidance/. (consultato il 25.08.2022).

[40] W3C. *Motion Sensors Explainer*. URL: https://w3c.github.io/motion-sensors/#gyroscope-sensor. (consultato il 01.09.2022).

[41] W3C. *Web NFC*. URL: https://w3c.github.io/web-nfc. (consultato il 05.09.2022).

[42] W3Schools. *Javascript ES6*. URL: https://www.w3schools.com/Js/js_es6.asp. (consultato il 02.09.2022).

[43] Wikipedia. *Alexa Internet*. URL: https://it.wikipedia.org/wiki/Alexa_Internet. (consultato il 2.09.2022).

[44] Wikipedia. *World Wide Web Consortium*. URL: https://en.wikipedia.org/wiki/World_Wide_Web_Consortium. (consultato il 02.09.2022).

[45] www.androidauthority.com. *Set up your app's prices*. URL: https://www.androidauthority.com/publishing-first-app-play-store-need-know-383572/. (consultato il 26.07.2022).

[46] www.apple.com. *Apple lancia il nuovo iPhone 3G*. URL: https://www.apple.com/it/newsroom/2008/06/09Apple-Introduces-the-New-iPhone-3G/. Press Release, 9.06.2008 (consultato il 26.07.2022).

[47] www.apple.com. *iPhone to Support Third-Party Web 2.0 Applications*. URL: https://www.apple.com/newsroom/2007/06/11iPhone-to-Support-Third-Party-Web-2-0-Applications/. Press Release, 11.06.2007 (consultato il 26.07.2022).

[48] www.w3.org. *DEVICEORIENTATION EVENT SPECIFICATION PUBLICATION HISTORY*. URL: https://www.w3.org/standards/history/orientation-event. (consultato il 27.07.2022).

[49] www.w3.org. *Geolocation API Specification Level 2*. URL: https://www.w3.org/TR/2011/WD-geolocation-API-v2-20111201/. (consultato il 27.07.2022).

[50] www.w3.org. *Sensors and Local Interactions*. URL: https://www.w3.org/2018/12/web-roadmaps/mobile/sensors.html. (consultato il 27.07.2022).

[51] Zhiju Yang, Rui Zhao e Chuan Yue. «Effective Mobile Web User Fingerprinting via Motion Sensors». In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2018, pp. 1398–1405. DOI: 10.1109/TrustCom/BigDataSE.2018.00194.