Università degli Studi di Udine

AudioGuard

CdL Internet of Things, Big Data & Machine Learning

A. Gritti - G. Palma

Internet of Things

Docente: Ivan Scagnetto

Marzo 2024

Contenuti

1 AudioGuard

1.1 Cos'è AudioGuard

AudioGuard è un sistema di *sorveglianza audio* progettato per ambienti domestici. Attraverso la *classificazione audio*, è in grado di riconoscere e analizzare suoni sospetti o attività anomale in tempo reale, informando gli utenti mediante notifiche sull'applicazione mobile dedicata, in modo che possano reagire immediatamente a eventuali minacce nella loro abitazione.

1.1.1 Il logo di AudioGuard

Figure 1.1: *Logo di AudioGuard*

Il logo di AudioGuard usa elementi simbolici per comunicare il concetto di protezione attraverso la sorveglianza audio. Le **onde sonore** stilizzate simboleggiano la tecnologia audio al cuore del sistema, mentre l'uso dello **scudo** offre un'immediata sensazione di sicurezza e difesa, trasmettendo tutta l'affidabilità di AudioGuard.

1.2 State of the art dell'audiosorveglianza

Il campo dell'audiosorveglianza, sebbene ancora molto acerbo, negli ultimi anni ha vissuto notevoli progressi tecnologici. Un significativo salto riguarda il miglioramento della qualità dell'audio attraverso tecniche come la **riduzione del rumore**, che permette l'isolamento delle voci e di altri suoni rilevanti da ambienti rumorosi, migliorando la qualità dell'audio catturato.

Parallelamente, l'analisi dell'audio basata sull'**intelligenza artificiale** ha rivoluzionato completamente il settore in generale, spesso accantonato in favore dell'analisi dei video. Un esempio è il *riconoscimento vocale*, che si avvale di queste tecnologie per aumentare l'accuratezza dei sistemi per il riconoscimento del parlato.

Inoltre, la **miniaturizzazione** dei dispositivi ha reso possibile la produzione di microfoni e

componenti molto più piccoli e discreti, migliorando la furtività e consentendo di raccogliere informazioni senza destare sospetti.

Tuttavia, nonostante i progressi, è importante notare che le soluzioni sul mercato per la sorveglianza audio sono ancora limitate. Alcune delle attuali applicazioni si concentrano su attività come *spionaggio*, *intercettazioni* e *microspie*, e non sull'effettiva sorveglianza, come AudioGuard promette di fare.

1.3 Vantaggi e svantaggi dell'audiosorveglianza

V A N T A G G I

1. **Basso costo:** l'audiosorveglianza può offrire un'alternativa più economica rispetto ai sistemi di videosorveglianza, è caratterizzata infatti dalla semplicità dei componenti, facilitando l'installazione e la manutenzione.
2. **Discrezione:** l'audiosorveglianza offre un monitoraggio discreto, ideale per situazioni in cui la presenza di telecamere potrebbe essere considerata invasiva o indesiderata.
3. **Complementarità con videosorveglianza:** si integra perfettamente con i sistemi di videosorveglianza esistenti, fornendo un livello aggiuntivo di sicurezza.

S V A N T A G G I

1. **Sensibilità alle interferenze:** l'audiosorveglianza può essere sensibile alle interferenze esterne, come rumori ambientali o altri suoni non rilevanti, portando a possibili falsi allarmi.
2. **Tecnologia acerba:** nonostante i progressi, l'audiosorveglianza è considerata ancora una tecnologia in evoluzione, con margini di miglioramento nella precisione del rilevamento.
3. **Ambiguità:** la mancanza di contesto visivo può rendere l'interpretazione dei dati audio più complessa e rendere più difficile distinguere tra situazioni normali e potenziali minacce.

1.3.1 Differenze rispetto la videosorveglianza

Le differenze tra audiosorveglianza e videosorveglianza sono significative e vanno ad influenzare direttamente il modo in cui questi sistemi vengono impiegati per la sicurezza e il monitoraggio.

Copertura L'audiosorveglianza eccelle nella copertura grazie alla capacità di monitorare ampie aree senza dover essere direzionata. Nella videosorveglianza invece il campo visivo è limitato dalla direzione delle telecamere e può essere ostacolato da barriere fisiche.

Detezione L'audiosorveglianza si distingue per la sua abilità nel rilevare eventi specifici tramite il suono, come vetri rotti o passi, anche al di fuori del campo visivo di una telecamera, mentre la videosorveglianza fornisce immagini limitate agli eventi all'interno del suo campo visivo.

Discrezione In termini di discrezione, l'audiosorveglianza è meno invasiva rispetto alla presenza evidente delle telecamere.

Costi L'audiosorveglianza comporta costi inferiori e una più semplice implementazione rispetto alla videosorveglianza, grazie alla minor necessità di infrastruttura fisica complessa.

Influenzabilità L'audiosorveglianza può essere influenzata da rumori ambientali, che possono aumentare il rischio di falsi allarmi, mentre la videosorveglianza è più vulnerabile alle variazioni di luminosità, tuttavia, la sua capacità di fornire dettagli visivi aiuta a minimizzare i falsi positivi.

1.4 Panoramica delle Componenti Principali

- **Dispositivo principale (Raspberry Pi):** il cuore di AudioGuard. Il *Raspberry Pi* funge da hub centrale per raccogliere e processare dati audio dall'ambiente circostante utilizzando il microfoni collegato. È responsabile della trasmissione di dati al cloud e di interfacce locali per la configurazione e il controllo.
- **Applicazione mobile (Dart e Flutter):** l'applicazione mobile offre agli utenti un'interfaccia intuitiva per monitorare e gestire il sistema AudioGuard. Realizzata con *Dart* e *Flutter*, fornisce un'esperienza cross-platform uniforme su dispositivi Android e iOS. Gli utenti possono ricevere notifiche in tempo reale, visualizzare lo storico degli eventi e personalizzare le impostazioni direttamente dall'applicazione.
- **Classificazione audio (TensorFlow Lite):** *TFLite* consente al sistema di identificare e reagire a suoni specifici scelti dall'utilizzatore mediante l'applicazione. Ad esempio, può riconoscere allarmi, suoni di rotture, bisbigli o altri eventi rilevanti.
- **Cloud (Firebase):** *Firebase* consente di archiviare e recuperare i dati audio raccolti dal dispositivo, nonché le informazioni utente e sui dispositivi stessi.

1.5 A chi si rivolge?

AudioGuard attualmente si rivolge esclusivamente al mercato dei **privati**, offrendo una (e a quanto pare unica) soluzione avanzata per la sicurezza di casa. Potendosi concentrare sugli utenti privati, il sistema fornisce un monitoraggio efficace degli accessi e protezione contro intrusioni indesiderate nella propria abitazione. L'interfaccia *user-friendly* dell'applicazione e le funzionalità integrate rendono AudioGuard ideale per chi desidera proteggere la propria casa sfruttando una soluzione accessibile e immediatamente utilizzabile per un pubblico non specializzato. Sebbene al momento il target sia limitato ai privati, potrebbe essere considerata un'espansione futura verso altri settori in base all'evoluzione del prodotto.

2 Dispositivo

2.1 Livelli dell'IoT e architettura di rete

I **livelli dei dispositivi IoT** sono una categorizzazione che considera le caratteristiche e le funzionalità dei dispositivi connessi in rete. Questa categorizzazione descrive in che modo il sistema è strutturato, se i dispositivi comunicano con il cloud, in che modo interagiscono con esso e soprattutto come vengono gestiti i dati.

In tale contesto, in base agli obiettivi di AudioGuard, il prodotto può essere classificato come dispositivo IoT di livello **quattro**. I dispositivi IoT di livello quattro si distinguono per la presenza di capacità di analisi e di processamento dati direttamente in loco. Questo implica che tali dispositivi sono in grado di analizzare ed elaborare i dati senza doverli trasmettere a un server remoto.

Nel caso specifico, il sistema può essere composto da diversi dispositivi interconnessi che raccolgono dati audio, che vengono analizzati tramite l'utilizzo della rete neurale *YAMNet*, finalizzato alla categorizzazione dell'audio. I risultati delle analisi infine vengono memorizzati in cloud attraverso la piattaforma *Firebase*.

Questo approccio consente un trattamento rapido ed efficace dei dati audio, offrendo un'analisi dettagliata e tempestiva direttamente sul dispositivo senza la necessità di ricorrere a risorse esterne. Tale configurazione risulta particolarmente vantaggiosa in ambiti in cui la tempestività e l'efficienza dell'analisi rivestono un ruolo cruciale, come nel caso di AudioGuard.

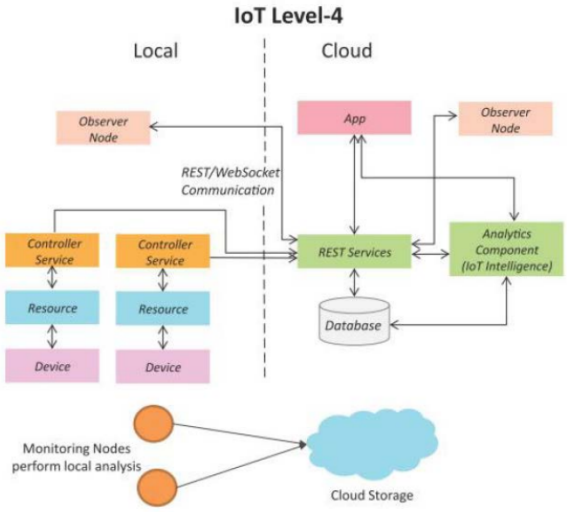


Figure 2.1: *Schema di rete di un dispositivo IoT di livello 4*

2.2 Comunicazioni e protocolli usati

2.2.1 Bluetooth Low Energy

Il **Bluetooth Low Energy** (comunemente chiamato BLE) è uno standard di rete wireless progettato per il mondo IoT e dispositivi a basso consumo. Esso è indipendente dal Bluetooth classico ma nonostante ciò, molte interfacce Bluetooth di ultime generazioni supportano entrambe le tecnologie. Sviluppato originariamente da Nokia nel 2006 sotto il nome di *Wibree*, fu integrato definitivamente all'interno della specifica *Bluetooth 4.0* nel luglio 2010.

Dettagli tecnici

La differenza principale rispetto al classico Bluetooth è il minor consumo energetico che varia in un range di $0.01 - 0.50$W in base al suo utilizzo e ad un picco di consumo pari a 15mA rispetto i 30mA del Bluetooth classico, questo ne permette l'utilizzo su sistemi embedded che spesso necessitano di essere alimentati tramite una batteria. A causa di ciò, il throughput (a livello di application-layer, detto anche "goodput") è nel range di $0.27 - 1.37$ Mbit/s rispetto ai $0.7 - 2.1$ Mbit/s del classico Bluetooth. Nonostante ciò, il suo utilizzo è prevalentemente quello di lettura/scrittura di dati di piccole dimensioni, di fatti, l'ATT Payload contenente i dati riguardanti l'attributo, è di soli 20 bytes (ad oggi, dalla specifica Bluetooth 4.2, l'ATT Payload è di 244 bytes).

Sicurezza

Nei pacchetti BLE la sicurezza viene implementata in due modi. La sicurezza base, a livello di pacchetto, avviene tramite il classico **Cyclic Redundancy Check** (CRC) che controlla l'integrità del pacchetto stesso. A livello di applicazione, il BLE (dalla specifica Bluetooth 5.2) permette di utilizzare **Encrypted Advertising Data** (EAD), un sistema crittografico basato sulla doppia chiave asimmetrica. I dispositivi BLE trasmettono le proprie chiavi pubbliche a tutti i dispositivi nelle vicinanze, queste chiavi vengono successivamente utilizzate per criptare le informazioni all'interno dei ATT payload. L'utilizzo della crittografia EAD viene indicato dai 4 bytes successivi all'header.

Generic Attribute Profile (GATT)

Tutti i dispositivi BLE usano il **G**eneric **Att**ribute Profile (GATT). Esso è l'API offerto dal Bluetooth Low Energy per l'utilizzo del protocollo **ATT** da parte dei dispositivi. Gli elementi principali sono:

Attributo È l'entità base, astratta, che viene scambiata durante la comunicazione, esso è composto da un identificativo, un handle (un valore che definisce lo specifico attributo), permessi (lettura e/o scrittura) ed infine i dati. Il concetto di attributo viene definito dal protocollo ATT che viene sfruttato dal GATT per descrivere le successive entità e svolgere tutte le sue operazioni.

Server (Peripheral) È il dispositivo che fornisce i **services**. Ogni service è suddiviso in **characteristics**. Tali characteristics possono essere richiesti dal client tramite operazioni di lettura e/o scrittura. Ogni characteristics può avere un numero illimitato di **descriptor**, che aggiunge informazioni al characteristics. Ogni service e ogni characteristic è concettualmente un attributo, quindi ognuno di essi viene identificato da un **UUID-128** (Universally Unique Identifier a 128 bit).

Client (Central) Il client (o central) è il dispositivo che ricerca altri dispositivi peripheral ed utilizza le characteristics a lui utile.

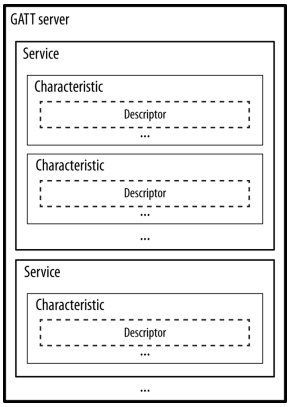


Figure 2.2: *Illustrazione di un server GATT*

Utilizzo del BLE all'interno del dispositivo AudioGuard

Il BLE viene utilizzato dal sistema AudioGuard per la configurazione iniziale del Wi-Fi. Tale soluzione, oltre ad essere ampiamente usata da molti dispositivi di questo genere, ci permette di usare tutte le funzionalità offerta dal Raspberry Pi, sfruttando il suo modulo Bluetooth. Una soluzione alternativa sarebbe stata quella dell'utilizzo di un SoftAP, cioè la creazione di hotspot temporaneo che poteva essere usato dal client per fornire le informazioni di *SSID* e *password* del Wi-Fi; tale soluzione però necessitava di due interfacce Wi-Fi nel dispositivo, una per la ricerca delle reti Wi-Fi nelle vicinanze, e l'altra per il SoftAP.

Implementazione Il sistema operativo usato dal dispositivo AudioGuard è Raspberry Pi OS Lite (ex Raspbian Lite), un sistema operativo linux, headless, sviluppato per Raspberry Pi. L'implementazione del protocollo Bluetooth non è presente nel kernel Linux, viene quindi implementata dal software **BlueZ**. La comunicazione con il software BlueZ avviene tramite il sistema di comunicazione tra processi D-Bus. Per facilità di sviluppo, viene utilizzata la libreria `bluez-peripheral` che permette di creare un GATT server sfruttando BlueZ estraniando lo sviluppatore dall'utilizzo diretto di D-Bus. Il dispositivo mette a disposizione due services: `WifiConfigurator`, che comprende le characteristics per configurare i dati Wi-Fi da parte dell'app e il service `WifiScanner`, che viene usato per ricevere la lista di reti Wi-Fi accessibili dal dispositivo. La configurazione del server GATT può quindi essere riassunta da questa tabella.

Service		**Characteristic**		
Nome	Handle	Nome	Handle	Permessi
WifiConfigurator	0x180D	State	0x2A37	READ, WRITE
		SSID	0x2A38	WRITE
		Password	0x2A39	WRITE
		Security	0x2A3A	WRITE
WifiScanner	0x180E	State	0x5BA0	READ, WRITE
		AP List	0x5BA1	READ

2.2.2 HTTP, REST API e comunicazione con Firebase

Il dispositivo AudioGuard utilizza come sistema backend in cloud il servizio Firebase, offerto da Google. La comunicazione tra il dispositivo e il sistema Firebase avviene tramite chiamate HTTPS alle diverse REST API offerte dal sistema stesso. Questo permette all'applicativo di comunicare direttamente con il database, il sistema real-time e il sistema di autenticazione.

3 Dart e Flutter

3.1 Introduzione a Dart e Flutter

Dart e **Flutter** sono due tecnologie sviluppate da Google che permettono lo sviluppo di applicazioni cross-platform per sistemi mobile, web e desktop. Non un concetto nuovo certamente (vedasi *React Native* ad esempio), ma la caratteristica fondamentale che ha portato alla loro rapida diffusione è l'offrire performance spesso identiche a quelle delle controparti *native*, distinguendosi così da altri strumenti di sviluppo cross-platform che spesso peccano (e di molto) in termini di efficienza e reattività. Per questi motivi, Dart e Flutter sono stati scelti per lo sviluppo dell'applicazione dedicata ad **AudioGuard**.

(a) (b)

Figure 3.1: *Loghi di* (a) *Dart e* (b) *Flutter*

Dart language Dart è un linguaggio di programmazione orientato agli oggetti, annunciato per la prima volta nel 2011 con lo scopo di sostituire *JavaScript* nello sviluppo web. Non riuscitoci, è passato in sordina per parecchi anni, fino a quando nel 2018 ha guadagnato popolarità in quanto scelto come linguaggio principale per lo sviluppo di applicazioni con il framework Flutter.

È progettato per essere veloce ed efficiente, una delle peculiarità di Dart (e Flutter) è infatti l'*hot-reload*, che permette di visualizzare all'istante i cambiamenti apportati all'applicazione senza la necessità di riavviarla.

Flutter framework Flutter è un framework open-source per la creazione di applicazioni cross-platform che utilizza il linguaggio Dart. Implementa interfacce utente basate su *Material Design* di Android e *Cupertino Style* per iOS, gestendo il rendering e le animazioni delle componenti mediante la libreria Skia, un motore grafico sviluppato sempre da Google.

La principale caratteristica di Flutter è l'approccio basato su widget per la costruzione dell'interfaccia utente, dove l'intera app è costruita combinando e annidando questi elementi, andando a rappresentare sia la struttura che l'aspetto visivo dell'applicazione.

Listing 3.1: *Esempio di codice Dart/Flutter*

```dart
class MyHelloWorld extends StatefulWidget {
  @override
  _MyHelloWorldState createState() => _MyHelloWorldState();
}

class _MyHelloWorldState extends State<MyHelloWorld> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Hello World App'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            print('Hello, World!');
          },
          child: Text('Premi per salutare'),
        ),
      ),
    );
  }
}
```

3.1.1 Come mai Dart e Flutter?

La decisione di adottare Dart e Flutter per sviluppare l'applicazione di AudioGuard si basa su diversi punti chiave:

1. **Unica code base per più piattaforme:** questo approccio ha semplificato la manutenzione e accelerato lo sviluppo, offrendo un notevole risparmio di tempo e risorse.

2. **Alte prestazioni:** Dart trasforma il codice in applicazioni che girano come se fossero native, assicurando rapidità e fluidità.

3. **Vasta gamma di widget:** Flutter mette a disposizione una ricca libreria di widget per creare interfacce utente non solo intuitive e funzionali, ma anche gradevoli esteticamente e al passo con lo stile.

4. **Supporto dalla community:** Dart e Flutter sono supportati da una vasta community di sviluppatori che forniscono risorse, documentazioni, pacchetti e aiuti continui, facilitando lo sviluppo.

3.2 Architettura dell'applicazione

L'architettura su cui è basata l'applicazione di AudioGuard è la **clean architecture**, un approccio modulare che si basa sul principio della *separation of concerns*, e consiste nella divisione del software in layer per semplificare lo sviluppo e la manutenzione del sistema.

3.2.1 Struttura della clean architecture

La clean architecture è composta da tre principali layer: **presentation layer**, **domain layer** e **data layer**. Ogni layer ha la sua funzione specifica e i componenti che lo compongono.

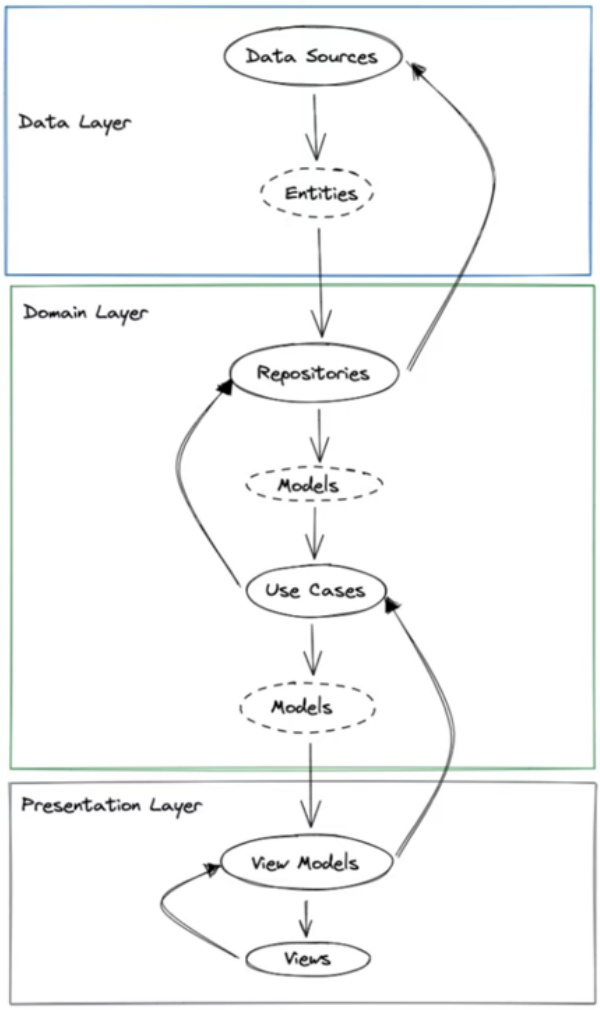


Figure 3.2: *Struttura a livelli della clean architecture*

1. **Presentation layer:** il presentation layer è responsabile di presentare il contenuto dell'app all'utente e gestire gli eventi che modificano lo stato dell'applicazione. È diviso in tre parti:
   - *Pages:* rappresenta le diverse pagine o schermate dell'applicazione. Ogni pagina può contenere widget e logica specifica per quella particolare schermata.
   - *State management:* si occupa di gestire lo stato dell'applicazione in modo da riflettere i cambiamenti nell'interfaccia utente. Vengono utilizzate librerie come *Bloc* o **Riverpod** per gestire lo stato.
   - *Widgets:* include i widget specifici che vengono utilizzati nelle pagine dell'applicazione.

2. **Domain layer:** il domain layer è il nucleo della clean architecture e non ha dipendenze da altri layer. Contiene:

   - *Entità:* le entità rappresentano i tipi di dati o le classi che vengono utilizzati in diverse parti del software. Sono oggetti che contengono la logica di business dell'applicazione e non dipendono da nessuna implementazione specifica del framework.
   - *Interfacce di repository:* le interfacce di repository definiscono i contratti o le interfacce per l'accesso ai dati nel sistema. Questo permette una separazione tra la logica di business e l'implementazione dei dettagli di persistenza dei dati.
   - *Casi d'uso:* i casi d'uso rappresentano le regole di business specifiche dell'applicazione. Ogni caso d'uso rappresenta un'interazione dell'utente con il sistema.

3. **Data layer:** il data layer è responsabile del recupero dei dati, che può avvenire attraverso chiamate API o database locali. È suddiviso in tre parti:

   - *Repository:* contiene le implementazioni effettive delle interfacce di repository definite nel Domain Layer. I repository sono responsabili di coordinare i dati provenienti da diverse fonti di dati e fornire i metodi per l'accesso ai dati.
   - *Data sources:* rappresentano le diverse origini da cui è possibile ottenere i dati. Ci possono essere sorgenti di dati remote, come le chiamate API, e sorgenti di dati locali, come il caching o i database interni.
   - *Models:* i modelli sono le rappresentazioni delle strutture dati, come JSON, che vengono utilizzati per interagire con le data sources.

3.2.2 Vantaggi nell'adozione della clean architecture

La clean architecture grazie alla sua struttura modulare favorisce la scalabilità, consentendo l'aggiunta di nuove funzionalità senza impattare il resto del sistema, e il principio di separation of concerns aiuta nella leggibilità del codice, agevolando la comprensione e la navigazione nel progetto. Inoltre, la manutenibilità è migliorata grazie ai layer ben definiti, permettendo modifiche specifiche senza influire su altre parti dell'applicazione.

3.3 Funzionalità principali dell'applicazione

L'applicazione dedicata ad AudioGuard mira a fornire agli utenti una soluzione completa per il monitoraggio e la sicurezza di ambienti domestici o professionali. Di seguito sono elencate le funzionalità principali per gestire il sistema.

1. **Avviso dello stato del sistema:**

   - L'home page dell'applicazione fornisce un immediato riscontro visivo dello stato di sicurezza corrente, con indicatori che segnalano se sono stati rilevati pericoli, se non ne sono presenti o se il sistema è attualmente offline. Questa rapida visualizzazione permette agli utenti di avere una immediata panoramica della situazione, assicurando che possano reagire prontamente in caso di allarme.

2. **Ascolto dell'ambiente da remoto:**

   - AudioGuard permette agli utenti di ascoltare in tempo reale l'audio catturato dai dispositivi di sorveglianza, indipendentemente dalla loro posizione.

3. **Storico delle rilevazioni:**

   - Il sistema conserva uno storico dettagliato di tutte le rilevazioni effettuate, permettendo agli utenti di consultare gli eventi passati. Ogni evento rilevato dal sistema viene accompagnato da informazioni dettagliate, come la data, l'ora, il tipo di suono rilevato e la possibile ubicazione all'interno dell'area monitorata.

4. **Informazioni relative ai dispositivi registrati:**

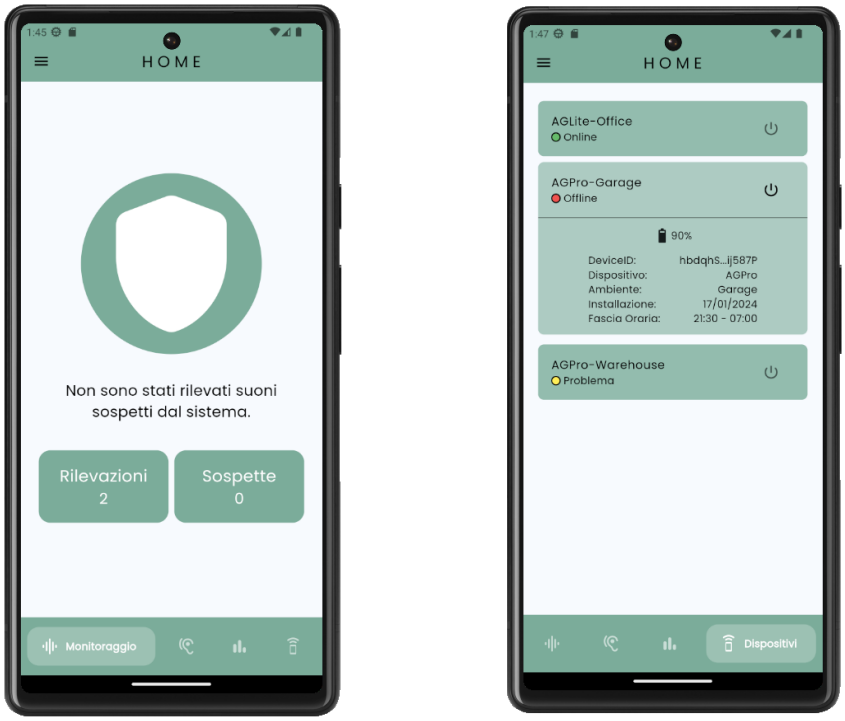
   - La sezione dedicata ai dispositivi fornisce una panoramica completa degli apparecchi registrati sul sistema AudioGuard, come il modello, la data di installazione, e l'ubicazione. Gli utenti possono così gestire facilmente i dispositivi, verificandone lo stato o aggiornandone le configurazioni all'occorrenza.

5. **Configurazione di un nuovo dispositivo:**

   - Gli utenti possono configurare nuovi dispositivi in modo semplice settando soltanto alcune informazioni necessarie.

6. **Scelta delle notifiche:**

   - Gli utenti hanno la possibilità di personalizzare i dispositivi su cui ricevere le notifiche e selezionare quale suono considerare una minaccia e quale no, assicurando che siano allertati solo per gli eventi ritenuti più rilevanti.



(a) (b)

Figure 3.3: *Le schermate di* (a) *monitoraggio e* (b) *dispositivi*

4 Firebase

4.0.1 Che cos'è Firebase?

Firebase è un sistema backend in cloud offerto da Google ideato principalmente per lo sviluppo di app mobile, webapp e videogiochi. Firebase nasce dalla startup **Envolve**, fondata da James Tamplin e Andrew Lee nel 2011. Essi hanno messo a disposizione un API per permettere agli sviluppatori di utilizzare il loro sistema di chat. Contrariamente dall'utilizzo previsto, gli sviluppatori utilizzarono tale API come sistema di comunicazione real-time tra i propri dispositivi, questo portò Tamplin e Lee a separare il sistema di chat da quello di comunicazione real-time, che prese, appunto, il nome di Firebase. Nel 2014 Firebase fù acquisita dalla Google, ed ora fa parte dell'ecosistema **Google Cloud**.

Figure 4.1: *Logo di Firebase*

4.0.2 Servizi offerti da Firebase

Nel corso degli anni Firebase implementò diversi servizi, tra cui:

Realtime Database Il Realtime Database fu il primo servizio offerto da Firebase. Esso è un database cloud-hosted NoSQL che memorizza dati in formato JSON e li sincronizza in realtime con ogni client connesso al servizio. Tutti i client, hanno accesso sicuro al database, condividendo la stessa istanza del Realtime Database e automaticamente ricevono aggiornamenti dei nuovi dati. I dati persistono localmente nel client, che possono essere usati qual'ora esso risulti offline. Quando il dispositivo riacquista la connessione, il Realtime Database sincronizza le modifiche ai dati locali con gli aggiornamenti remoti che si sono verificati mentre il client era offline, gestendo automaticamente eventuali conflitti.

Il Realtime Database viene utilizzato dal dispositivo AudioGuard per segnalare, in tempo reale, eventuali rilevazioni direttamente all'app. In questa maniera l'utente è in grado di ricevere automaticamente aggiornamenti dai propri dispositivi.

Firestore Database Firestore Database è un database NoSQL a cui i client possono accedere direttamente tramite SDK nativi o tramite API REST e RPC.

A seguito della tipologia del database NoSQL, i dati vengono archiviati in documenti contenenti il mapping dei campi ai valori. Questi documenti sono memorizzati in collezioni, che permettono di organizzare facilmente i dati e costruire query. I documenti supportano molti tipi di dati diversi, da semplici stringhe e numeri, a oggetti nidificati complessi. È inoltre possibile creare sotto raccolte all'interno dei documenti e costruire strutture di dati gerarchiche che si ridimensionano man mano che il database cresce. Questo sistema permette di creare query superficiali per recuperare i dati a livello di documento senza dover recuperare l'intera collezione. L'accesso ai dati dell'utente è protetto da Firebase Authentication e **Cloud Firestore Security Rules**.

Firestore Database viene usato nel progetto come database principale per memorizzare tutte le informazioni riguardanti gli utenti, i loro dispositivi e le rilevazioni registrate da essi. Il database è composto da una raccolta principale contenente i documenti per ogni utente. Gli utenti sono identificati da un *UID* (fornito da Firebase Authentication) e caratterizzati da username, nome e cognome, e dalle impostazioni utente usate nell'app. Per ogni utente è presente una sotto raccolta contenente i dispositivi da loro registrati. Essi sono caratterizzati dalle statistiche, dalle informazioni di sistema e dalle impostazioni del dispositivo configurate dall'utente. Per ogni utente, inoltre, è presente una sotto raccolta contenente le rilevazioni dei dispositivi associati, questa soluzione permette di semplificare le query per la lettura e l'inserimento di nuovi dati. Gli utenti, grazie al servizio Firebase Authentication, posso accedere unicamente ai loro dati, tale protezione viene fornita direttamente da Firestore Database.

Listing 4.1: *Regola usata per proteggere i dati di Firestore*

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /user/{userId}/{documents=**} {
      allow read, write: if request.auth != null &&
                        request.auth.uid == userId &&
                        request.auth.token.email_verified == true
    }
  }
```

Come si può notare, le richieste di lettura e scrittura, da parte di ogni utente, ai documenti nel percorso `/user/{userId}/` sono valide solo se l'utente è autenticato, l'email dell'utente è stata verificata e l'id di autenticazione corrisponde all'id del documento richiesto.

Firebase Storage Firebase Storage è un servizio cloud di archiviazione di oggetti. Tale servizio permette l'upload e il download di file di grandi dimensioni. Molto simile a servizi come AWS S3, esso si integra perfettamente con gli altri servizi offerti da Firebase, permettendo quindi di utilizzare Firebase Authentication per gestire l'accesso degli utenti ai file e Firestore per collegare tali file ai corrispettivi dati d'interesse.

Con AudioGuard, gli utenti hanno la possibilità di memorizzare e ascoltare l'audio registrato dal dispositivo durante le rilevazioni. Quando il dispositivo riconosce il suono come pericoloso (cioè che rientra tra le tipologie di audio scelte dall'utente), esso memorizza in un file i dati relativi al suono e lo carica utilizzando il servizio Firebase. Allo stesso modo, l'app permette di scaricare i file audio, permetto all'utente di avere sempre accesso ai suoni segnalati come pericolosi. I file all'interno di Firestore vengono mantenuti per 30 giorni, successivamente, i file audio relativi alle rilevazioni non presenti tra i preferiti dell'utente, vengono cancellate.

Firebase Authentication Firebase Authentication è un servizio cloud offerto da Firebase per la gestione degli utenti e dei relativi permessi. Firebase Authentication mette a disposizione diversi metodi di accesso, chiamati provider, supporta l'autenticazione tramite email/password, numeri di telefono, provider di identità popolari come Apple, Google, Facebook e altro ancora. Oltre a ciò, grazie all'integrazione con gli altri servizi, Firebase Authentication permette di gestire i permessi degli utenti sui vari servizi offerti da Firebase come già visto nel codice 4.1

Firebase Authentication permette ai dispositivi AudioGuard e all'app utente di accedere in maniera sicura ai propri dati. Tutto il processo di autenticazione viene gestito interamente dal servizio stesso, non necessitando di servizi esterni e, sfruttando un ecosistema già sicuro e ben collaudato, permettendo l'utilizzo diretto da parte dei client. Gli stessi permessi vengono gestiti da Firebase, proteggendo i dati da eventuali infrazioni client-side. L'utilizzo di SDK dedicate e REST API, facilita notevolmente la gestione degli utenti, senza intaccare in alcun modo la sicurezza del sistema.

5 Classificazione Audio

5.1 Cos'è?

La **classificazione audio** sfrutta l'*intelligenza artificiale* e il *machine learning* per identificare e categorizzare i segmenti sonori all'interno di file audio o di flussi in tempo reale. Questo processo è in grado di distinguere tra vari tipi di suoni, come discorsi, musica, rumori ambientali o specifici eventi sonori (ad esempio, vetri che si rompono nel caso di AudioGuard). Per implementare la classificazione audio nel dispositivo, è stato utilizzato il framework **TFLite** e il modello di rete neurale **YAMNet**.

5.2 Tensorflow Lite

TensorFlow Lite (*TFLite*) è una versione ottimizzata del framework **TensorFlow**, sviluppato da Google per offrire un insieme di strumenti per consentire l'esecuzione di modelli di machine learning su dispositivi *mobili*, *embedded* ed *edge*.

Figure 5.1: *Logo di TensorFlow Lite*

5.2.1 Caratteristiche principali di TFLite

Ottimizzazione TFLite è stato progettato specificamente per dispositivi mobili e sistemi embedded. Offre un'architettura leggera e modulare che minimizza l'uso della memoria e migliora l'efficienza computazionale sui dispositivi con risorse hardware limitate.

Quantizzazione La quantizzazione è una delle tecniche chiave utilizzate in TFLite per ridurre le dimensioni dei modelli (ad esempio, riducendo la precisione dei pesi), consentendo un'importante riduzione del consumo energetico e delle risorse computazionali necessarie.

Real-Time TFLite è ottimizzato per garantire bassa latenza e alta efficienza nell'esecuzione di operazioni che richiedono lo scambio di informazioni in tempo reale.

5.3 Classificazione audio con YAMNet

YAMNet (***Y**et **A**nother **M**obile **Net**work*), è un modello di rete neurale pre-addestrato, sviluppato da Google, specificamente progettato per classificare suoni ambientali ed eventi audio in oltre 500 categorie basate sull'insieme di dati **AudioSet** (sempre appartenente a Google). Si basa sull'architettura **MobileNetV1**, ottimizzata per dispositivi mobili e sistemi embedded con risorse limitate, caratterizzata da layer separabili, che riducono il numero di parametri e il carico computazionale, mantenendo allo stesso tempo elevate prestazioni di classificazione. Il processo di classificazione con YAMNet inizia con la pre-elaborazione dell'audio di ingresso, dove il segnale viene convertito in una **trasformata di Fourier a breve termine** ($STFT$) per generare spettrogrammi. Questi spettrogrammi sono poi utilizzati come input per il modello.

AudioSet AudioSet è un ampio dataset composto una collezione di oltre 2 milioni di clip audio tratte da video di **YouTube**, che coprono un'ampia gamma di suoni provenienti da diverse situazioni. Ogni clip audio è stata accuratamente etichettata da un vocabolario di circa 632 classi di suoni, che includono categorie come musica, parlato umano, veicoli, animali e molti altri.

5.4 Accuratezza del modello

L'accuratezza del modello è stata gestita da *Google* internamente, che ha intrapreso un'approfondita valutazione per assicurare l'esattezza delle annotazioni all'interno di AudioSet, la quale ha rilevato che, a causa di una serie di discrepanze, molte classi audio presentavano una bassa *precision* nelle etichette attribuite. Per risolvere questi problemi e migliorare la qualità dei dati, Google ha avviato un processo di revisione e ottimizzazione (detto *rerating*) per le classi di qualità inferiore, implementando istruzioni più dettagliate e raggruppando i segmenti audio in cluster più omogenei.

Il processo, al momento dell'ultima comunicazione da parte di *Google*, è stato portato a termine per circa il 50%, e data l'ampia estensione del dataset, il rerating è stato applicato a un massimo di 1000 segmenti per ogni classe, selezionati in modo indipendente per ciascuna etichetta. Tuttavia, per le classi che comprendono un numero totale di segmenti significativamente superiore a 1000, la qualità dei dati nel set non bilanciato può mostrare notevoli differenze rispetto ai set bilanciati.

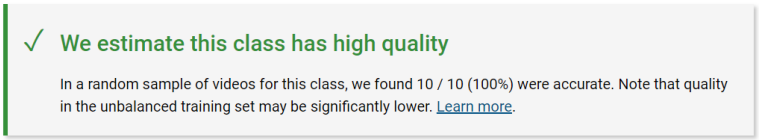


Figure 5.2: *Esempio di qualità della classe siren*

Questo esempio illustra la qualità della classe *siren*, dove nella valutazione di un campione casuale di video rappresentativo per questa classe, si è riscontrato che il 100% dei video è stato etichettato in modo accurato.

5.4.1 Accuratezza delle classi di AudioGuard

In forma riassuntiva, qui sono elencate le percentuali di accuratezza di AudioGuard

Table 5.1: *Accuratezza per classe*

Classe	**Accuratezza**
Glass	90%
Breaking	80%
Siren	100%
Shot	100%
Explosion	100%
Alarm	80%
Honk	60%
Car	100%
Voices	100%
Whistle	90%
Breathing	80%
Whispers	90%

5.5 Integrazione con Raspberry Pi

Il Raspberry Pi è un single-board computer di dimensioni ridotte sviluppato dalla Raspberry Pi Foundation nel Regno Unito. Il suo obiettivo principale è la creazione di progetti IoT, basati su sitemi Linux. Il Raspberry Pi è diventato estremamente popolare grazie alla sua versatilità, alle sue dimensioni ridotte e al suo prezzo accessibile. I single-board computer (SBC) sono dispositivi completi di elaborazione basati su una singola scheda di circuito stampato. Sono caratterizzati dalla presenza di CPU, RAM, memoria di archiviazione, porte di input/output (General Purpose Input/Output, GPIO) e altri componenti essenziali, tutti integrati su una singola scheda. I SBC sono utilizzati in sistemi embedded, prototipazione rapida, educazione informatica, automazione domestica, robotica, IoT e molto altro ancora. Con l'aumento di interesse dell'utilizzo del machine learning nei dispositivi IoT, i SBC stanno diventando sempre più potenti e ideali per sistemi di edge computing (come il caso di AudioGuard).

Per il dispositivo AudioGuard, è stato utilizzato un Raspberry Pi 4 Model B nella configurazione da 4GB di RAM. Questo Raspberry Pi monta un processore aarch64 Broadcom BCM2711 Quad-Core a 1.5 GHz.

Integrazione di TensorFlow Lite con il Raspberry Pi offre la possibilità di eseguire modelli di machine learning direttamente sul dispositivo, senza la necessità di una connessione a internet o di una potenza di elaborazione esterna.

L'utilizzo di TensorFlow Lite avviene tramite la libreria Python `tflite-runtime`. Tale libreria è una versione notevolmente più leggera e semplificata rispetto alla libreria completa `tensorflow`, nonostante ciò, mette a disposizione dello sviluppatore tutti gli strumenti necessari per utilizzare modelli `.tflite` (nel nostro caso `lite-model_yamnet_classification_tflite_1.tflite`).

YAMNet/classification, il modello YAMNet utilizzato da TensorFlow Lite, è un modello di classificazione audio che accetta come input frame un tensore o un array numpy 1-D float32 di lunghezza 15600 contenente una forma d'onda di 0,975 secondi rappresentata come campioni mono da 16 kHz. Lo stream audio, tramite l'utilizzo della libreria `pyaudio`, viene registrato con un sample rate di 16 MHz (16000 valori al secondo) e suddiviso in frame (o chunk) di 15600 valori float32. Per ogni frame, il modello restituisce un tensore di forma 2-D float32 (1, 521) contenente i punteggi previsti per ciascuna delle 521 classi presenti nel dataset AudioSet.