Università degli Studi di Udine

Resoconto Applicazione

CdL Internet of Things, Big Data & Machine Learning

A. Gritti - 152206

Progettazione di Applicazioni Mobili

Docente: Stefano Burigat

Giugno 2024

Contenuti

1 Schermate

Di seguito vengono descritte le schermate che compongono l'applicazione di AudioGuard, fornendo una panoramica sulle loro funzionalità e su eventuali aspetti interessanti da evidenziare.

L'applicazione ha come colore principale un **blu spento**, anche se l'utente potrà poi cambiarlo nella schermata delle impostazioni dell'app. Il font usato è il famoso **Roboto**, e sono state rispettate le direttive della **MaterialUI** fornite da Google. Non vengono usati widget proprietari di ciascun OS (**Android** e **iOS**) in modo da avere una applicazione 100% personalizzata dall'azienda (me), e le lingue disponibili alla traduzione ora sono ben **quattro**.

Per fornire adattabilità della UI ai vari formati di schermo, è stata utilizzata la classe **MediaQuery**, che fornisce informazioni sulle dimensioni e sull'orientamento del dispositivo corrente e permette di creare design responsivi che si adattano bene a una varietà di dispositivi.

1.1 Dispositivo delle Schermate

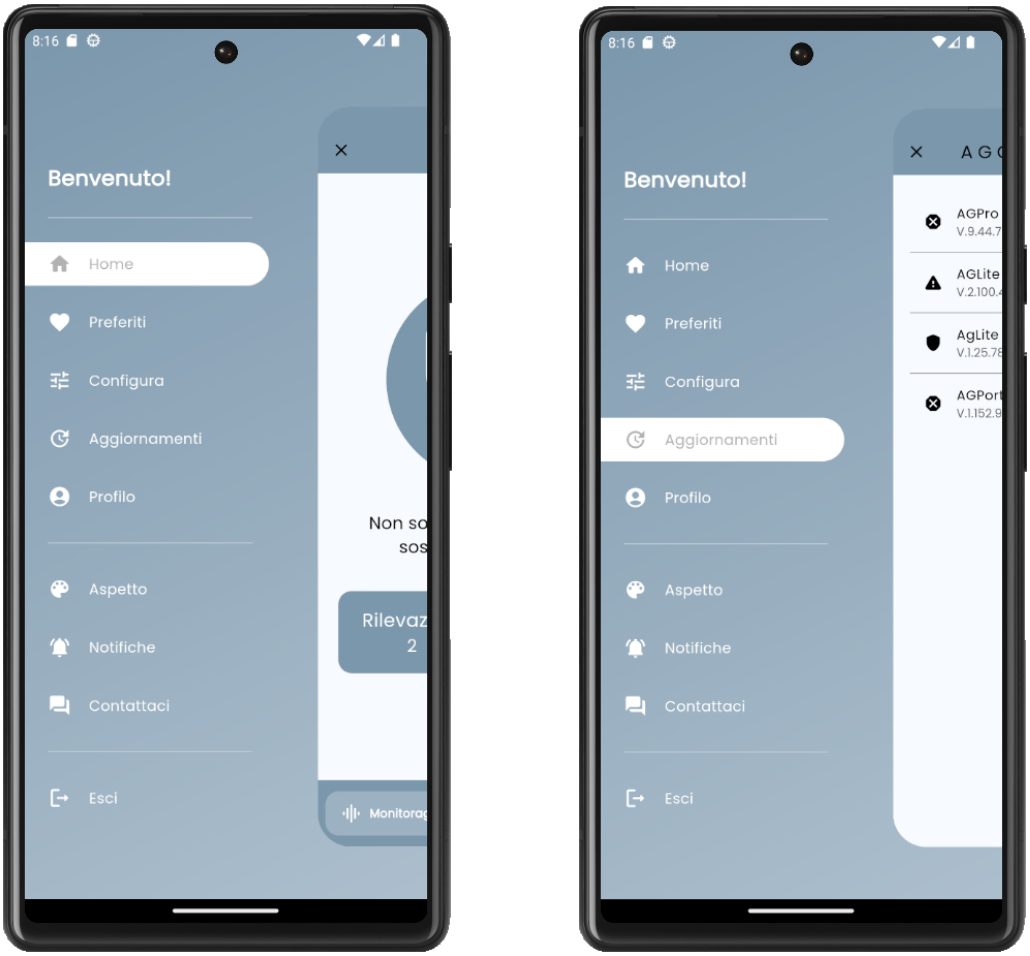
Il dispositivo usato nelle schermate seguenti è il **Google Pixel 6**:

Figure 1.1: Google Pixel 6, 2021

1.2 Drawer

Il **drawer**, accessibile mediante il menù hamburger in alto a sinistra in ogni schermata, permette all'utente di spostarsi comodamente nelle varie sezioni dell'applicazione. La pagina in cui si trova momentaneamente l'utente viene evidenziata per aiutare la navigazione. Il design non è proprietario, ma non sono stati utilizzati pacchetti per crearlo. Ciascun pulsante del drawer è funzionante, ad esclusione di "Esci".

Cambiamenti Rispetto ai mockup, le modifiche riguardano il gradiente dello sfondo, la pagina in cui si è che viene evidenziata con sfondo bianco, la schermata a destra più piccola e una cura e pulizia maggiore in generale.



(a) (b)

Figure 1.2: Nell'immagine è possibile vedere la pagina in cui si trova l'utente

1.3 Home

Sezione principale dell'applicazione, l'utente può navigare in 4 schermate principali utilizzando una comoda e intuitiva **BottomAppBar**. Le schermate disponibili sono:

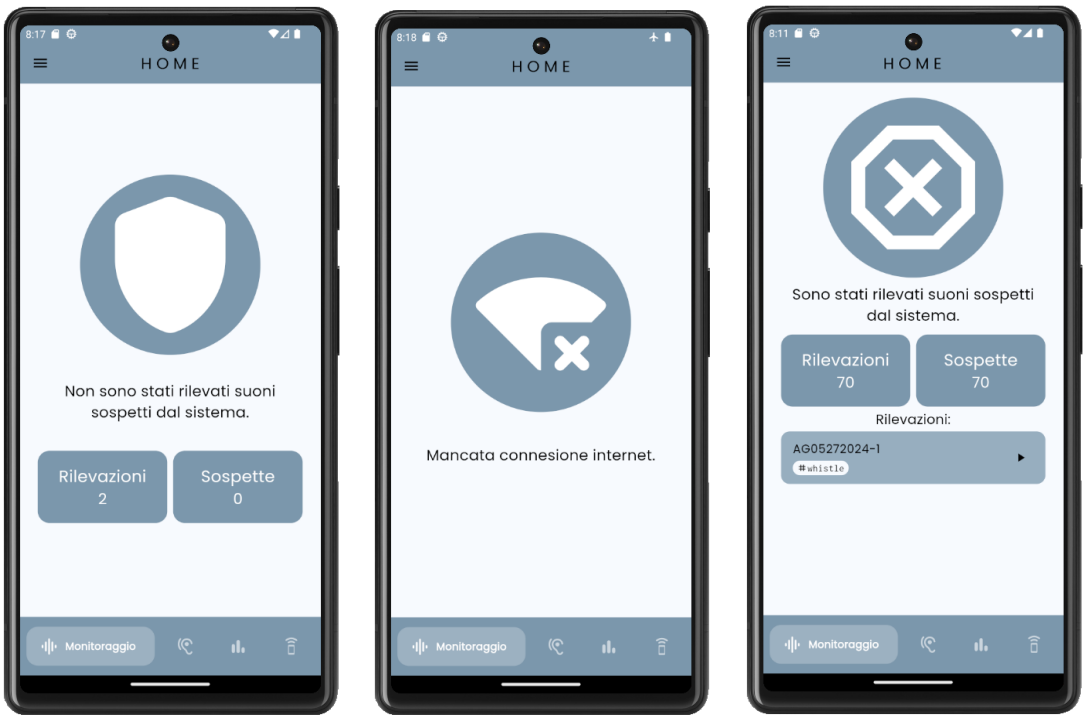
1.3.1 Monitoraggio

Divisa in 3 possibili scenari:

- **Sistema sicuro**: non vi sono rilevazioni sospette da parte del sistema. Idealmente è sempre attiva questa schermata.
- **Nessuna connessione** a internet disponibile.
- **Sistema non sicuro**: sono state rilevate attività sospette e vengono segnalate. Viene attivata nel momento in cui il sistema cattura un audio sospetto.

Le schermate di sistema sicuro e non sicuro funzionano grazie a Firebase, da dove vanno a recuperare i dati delle ultime rilevazioni. Se nella giornata corrente una nuova rilevazione viene inserita e viene classificata come pericolosa, allora la schermata si aggiorna. Non essendoci ancora un dispositivo, questa operazione può essere simulata manualmente accedendo a Firebase.

Cambiamenti Rispetto ai mockup, le icone rispecchiano la pulizia della material UI, e la BottomAppBar ora ha la pagine in cui ci si trova evidenziata.



(a) (b) (c)

Figure 1.3: (a) sistema sicuro, (b) nessuna connessione e (c) sistema non sicuro

1.3.2 Accesso Remoto

Purtroppo, per motivi riguardanti WebSocket e problematiche col dispositivo la pagina dell'**accesso remoto** è stata rimossa e sostituita con un Placeholder. Anziché essere completamente eliminata, viene lasciata per non scombussolare troppo la struttura dell'applicazione.

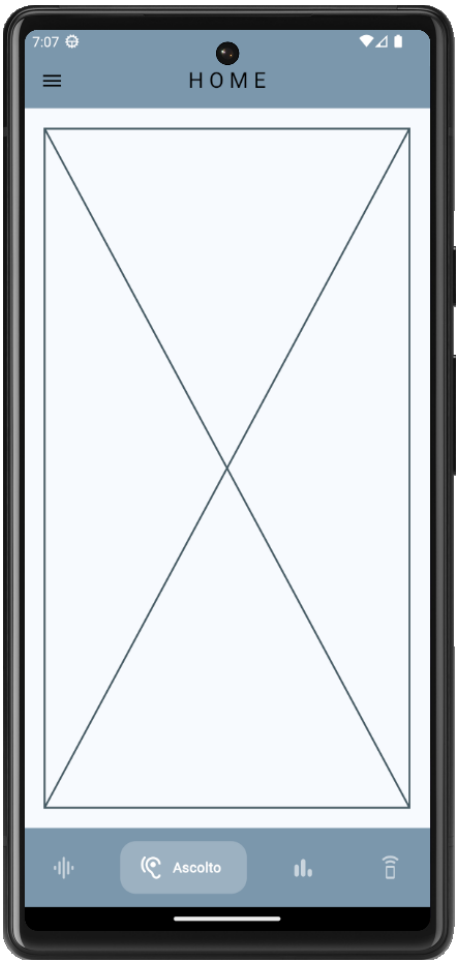
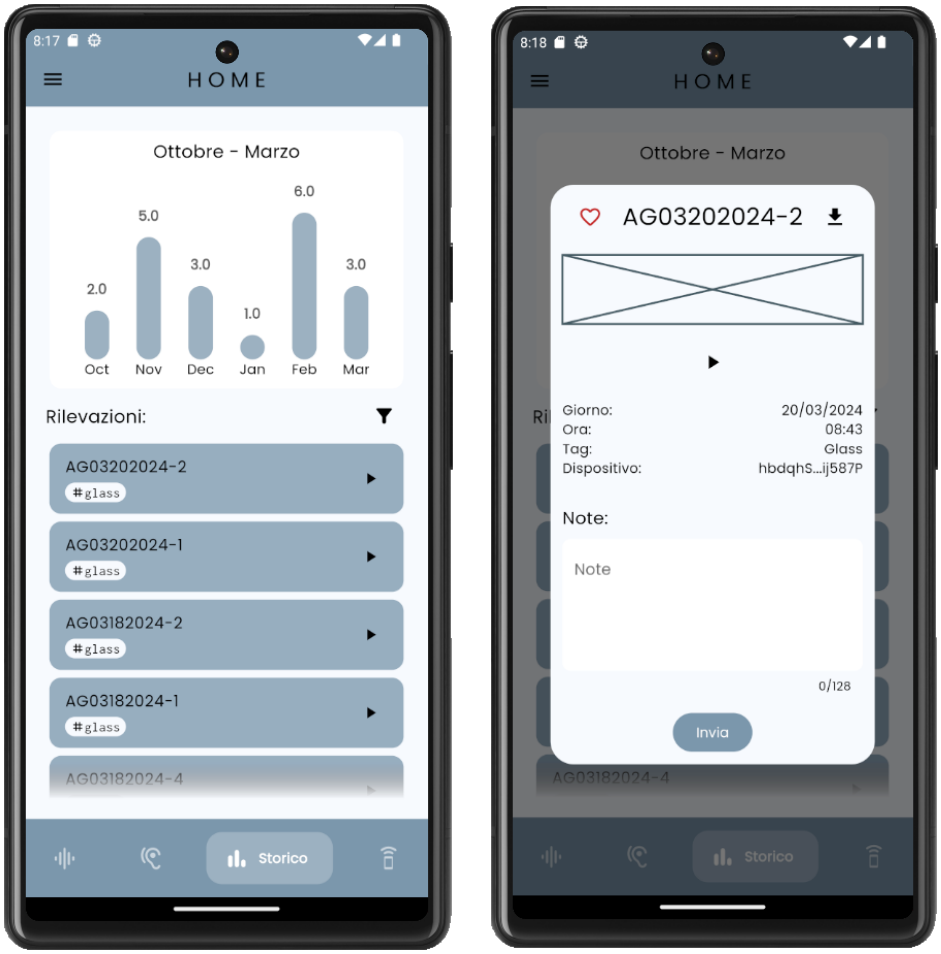


Figure 1.4: Placeholder fornito da Flutter

1.3.3 Storico

Lo **storico** mostra all'utente le varie rilevazioni pericolose che sono state captate dal sistema nel tempo e mostra un grafico che riassume un po' gli ultimi mesi, anche se al momento è hard-coded. Le rilevazioni vengono recuperate da Firebase da un database in tempo reale, infatti nel momento in cui una nuova rilevazione viene inserita, questa apparirà in cima alla lista. Cliccandoci un pop up si paleserà all'utente mostrando le informazioni della rilevazione, come il giorno, l'ora, il tag (ovvero la tipologia di suono rilevata), il dispositivo da cui è stata rilevata e le note. Cliccando su modifica (non abilitata) il pop-up si chiude mostrando una snackbar per informare delle modifiche apportate.

Cambiamenti Rispetto ai mockup, sono state solo aggiunte alcune informazioni e icone di bellezza.



(a) (b)

Figure 1.5: (a) storico e (b) pop-up delle rilevazioni

1.3.4 Dispositivi

La schermata dei **dispositivi** riassume tutti i dispositivi dell'utente e le varie informazioni collegate ad essi. Queste vengono reperite da Firebase ed è possibile accedervi cliccando sulla card del dispositivo associato. L'utente può inoltre comodamente spegnere o accendere il microfono del dispositivo facendolo passare da online e offline o viceversa. Se il dispositivo presenta un problema, viene aggiornato lo stato e segnalato all'utente, che non potrà spegnere il microfono senza prima aver risolto il problema.

Cambiamenti Rispetto ai mockup nella sezione dei dispositivi ora le informazioni sono contenute all'interno di una card espandibile.

Figure 1.6: Schermata dei dispositivi

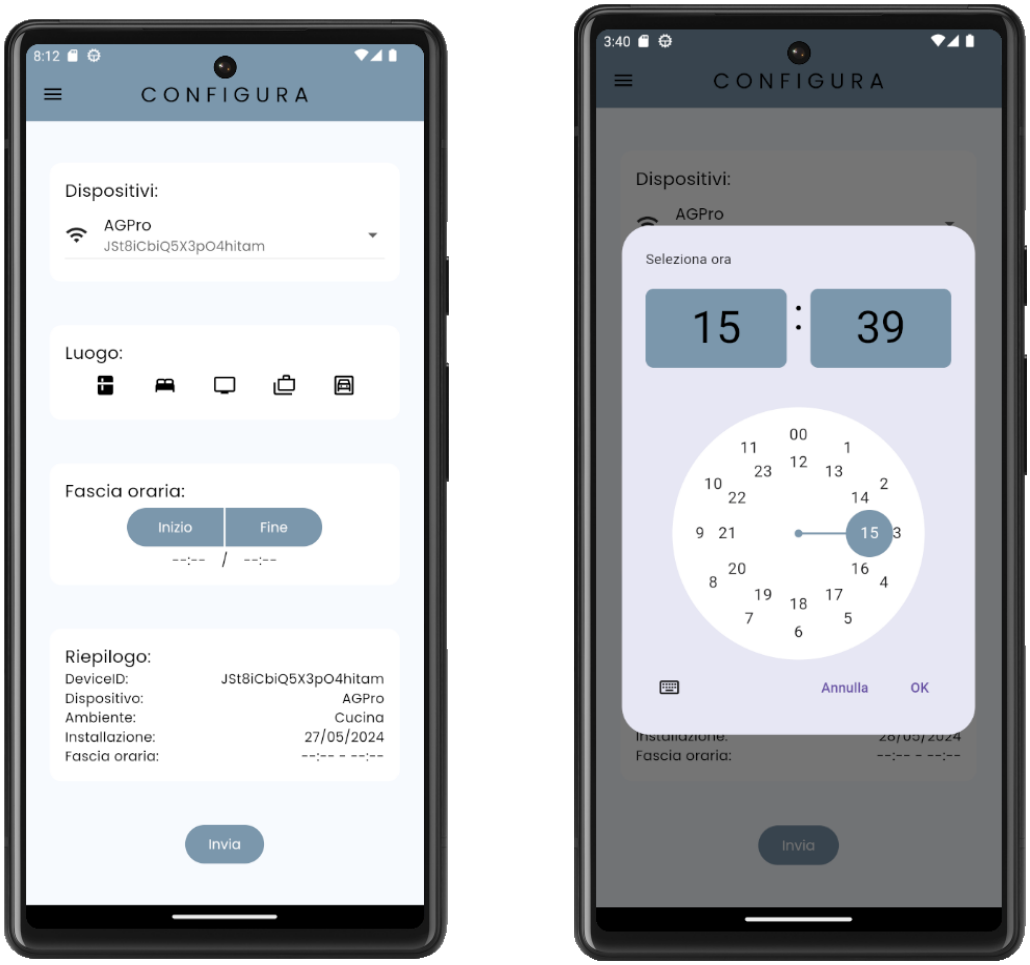
1.4 Preferiti

Figure 1.7: Schermata delle rilevazioni preferite

1.5 Configura

Da questa schermata l'utente può inserire un **nuovo dispositivo** nel sistema (non implementato in Firebase), selezionandolo dal menù a tendina tra i dispositivi trovati, può scegliere in che luogo inserirlo, la fascia oraria in cui questo deve stare attivo e infine ha un piccolo riepilogo delle informazioni del dispositivo. Cliccando invia una snackbar compare per informare l'utente.

Cambiamenti Il mockup è identico alla schermata e non ci sono stati cambiamenti.



(a) (b)

Figure 1.8: (a) schermata per la configurazione e (b) pop-up orologio per la fascia oraria

1.6 Aggiornamenti

Qui l'utente può scaricare gli **aggiornamenti** di ciascun dispositivo. Un aggiornamento può essere in 3 stati:

- **Non scaricato**: con l'icona del download grigia.
- **In download**: con la percentuale di aggiornamento scaricato.
- **Scaricato**: con l'icona del download verde.

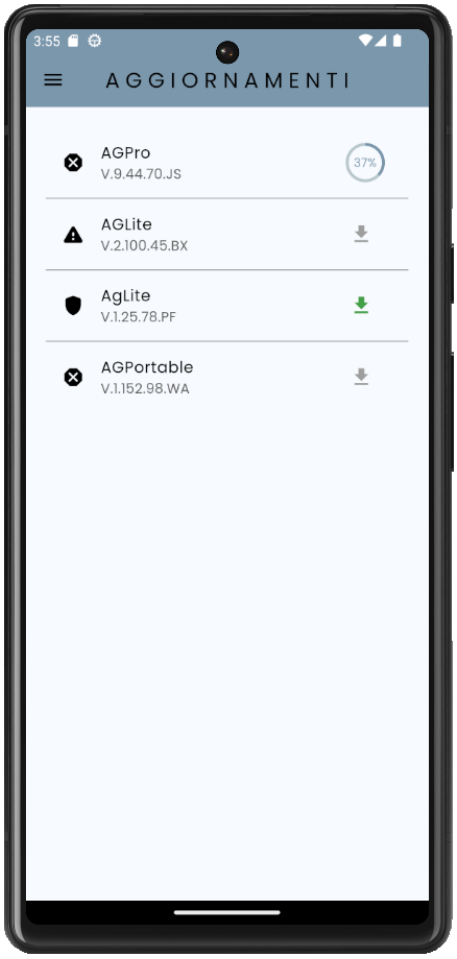


Figure 1.9: La schermata non ha subito grandi modifiche

1.7 Profilo

Questa schermata simula una eventuale schermata per il **riepilogo delle informazioni** utente recuperate eventualmente da Firebase.

Cambiamenti Rispetto ai mockup, non ci sono stati grandi modifiche.

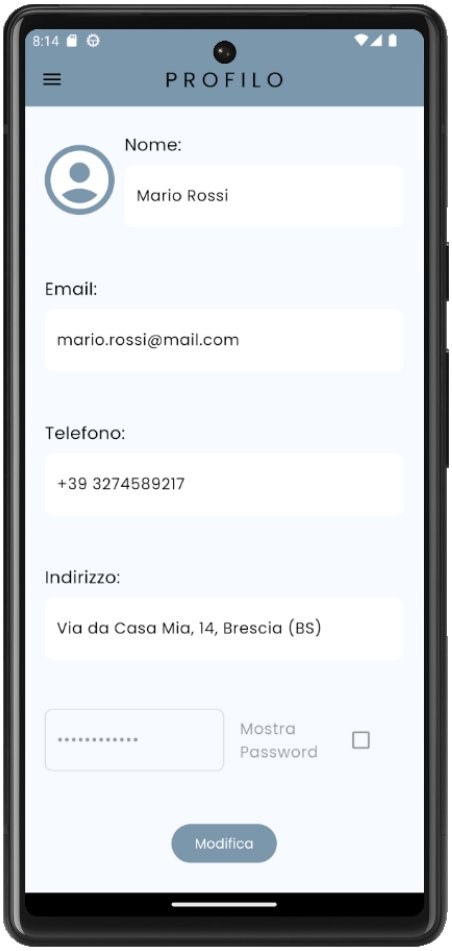


Figure 1.10: Schermata per la gestione del profilo utente

1.8 Aspetto

Da questa schermata l'utente può personalizzare l'applicazione, andando a scegliere se utilizzare la **modalità scura** (non attivata di default), il tema dell'app con una vasta scelta di **colori**, la **lingua** (italiano, spagnolo, inglese e tedesco) e il formato della **data** e dell'**ora**. Tutte le opzioni sono utilizzabili, le ultime due (data e ora) vengono ereditate direttamente nel cambiare la lingua.

Cambiamenti Rispetto ai mockup, è stata reintrodotta la selezione della lingua, ampliati i colori, e cambiato il colore dello switch della modalità scura.

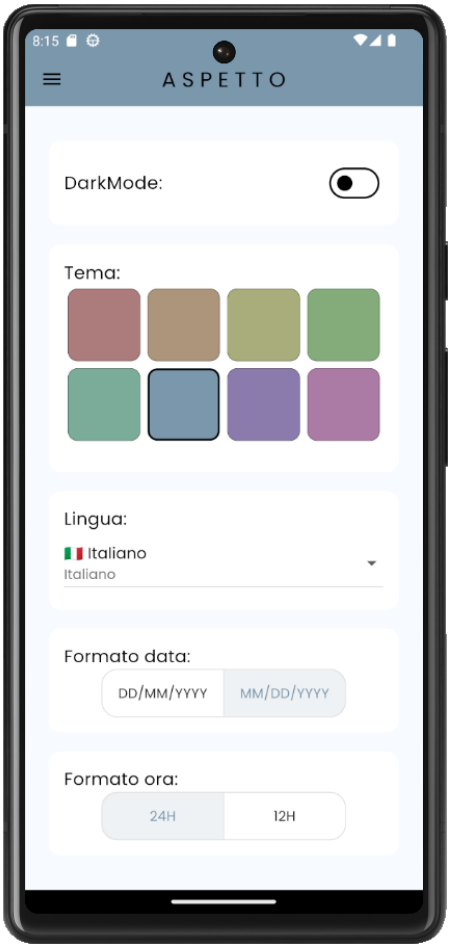
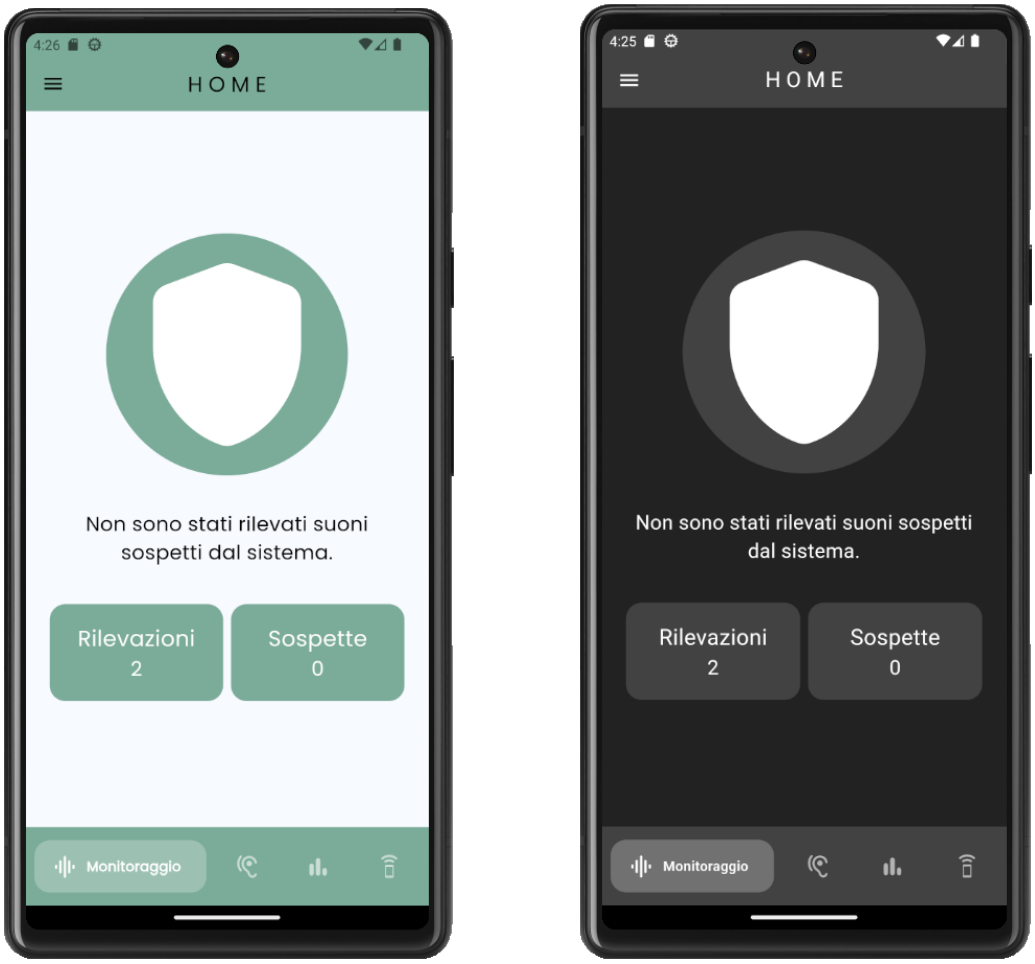


Figure 1.11: Le varie opzioni disponibili

1.8.1 Tema e Modalità Scura

Il tema e la modalità scura sono stati gestiti utilizzando **Riverpod** per la gestione dello stato e **SharedPreferences** per la memorizzazione delle preferenze utente. All'avvio, l'app carica le preferenze salvate e imposta il tema di conseguenza. Gli utenti possono passare tra modalità chiara e scura, con le preferenze aggiornate e salvate immediatamente, creando così un sistema reattivo e persistente.



(a) (b)

Figure 1.12: (a) schermata con tema verde e (b) dark mode attivata

1.8.2 Localizzazione

La localizzazione è necessaria per rendere l'app accessibile a un pubblico internazionale, ed è implementata usando il pacchetto *flutter_localizations* per supportare più lingue, tra cui italiano (it), inglese (en), spagnolo (es) e tedesco (de). Vengono così tradotti i testi dell'interfaccia utente e altri elementi in base alla lingua preferita dall'utente. Oltre alla traduzione dei testi, *flutter_localizations* gestisce anche la formattazione di date, ore e numeri, assicurando che questi elementi siano presentati nel formato corretto per ogni lingua. La localizzazione in flutter sfrutta i file *.arb* (Application Resource Bundle) contenuti nella cartella *l10n*, ovvero file JSON utilizzati per memorizzare le risorse di testo localizzate, ciascuno dei quali corrisponde a una lingua specifica e contiene le traduzioni dei testi utilizzati nell'applicazione.

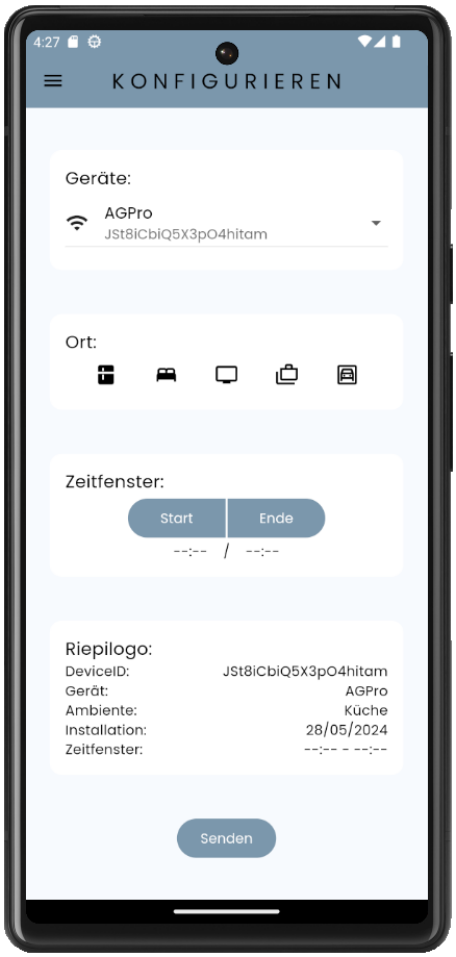


Figure 1.13: Esempio di schermata della configurazione dispositivi in tedesco

1.8.3 Shared Preferences

Per memorizzare le preferenze e i settings dell'utente, l'app utilizza il pacchetto *shared_preferences*, che consente di salvare dati in modo persistente sul dispositivo dell'utente, mediante una semplice API per leggere e scrivere coppie chiave-valore, rendendo facile salvare e recuperare i dati senza la necessità di un database complesso.

1.9 Notifiche

Figure 1.14: Scchermata delle notifiche

1.10 Contattaci

La schermata dei **contatti** permette all'utente di contattare direttamente via app l'azienda specificando la tipologia del problema e inserendo una breve descrizione della problematica, oppure di chiamare mediante il numero o presentandosi negli uffici negli orari indicati. La segnalazione inviata viene salvata direttamente in Firebase sul profilo dell'utente, così da avere uno storico delle sue segnalazioni. Premendo invia, una snackbar informerà del corretto funzionamento.

Cambiamenti Rispetto ai mockup, è stata cambiata la disposizione degli elementi.

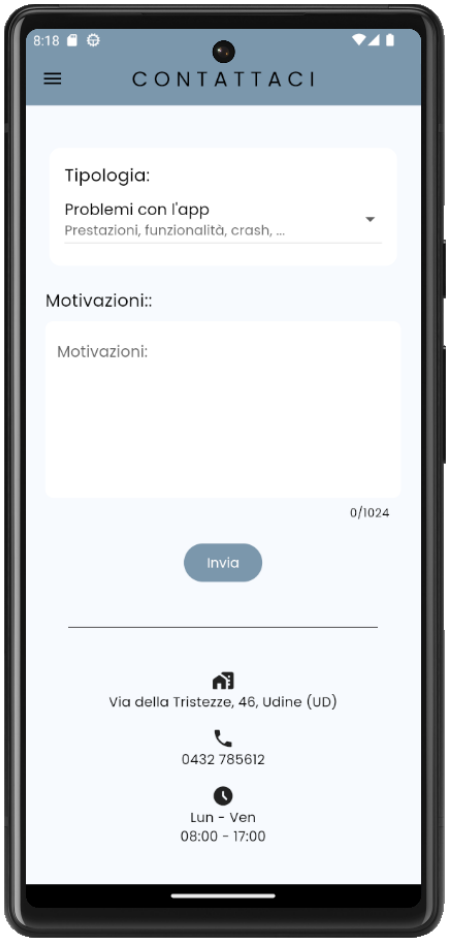


Figure 1.15: Schermata dei contatti per l'aziedna

1.11 Altro

1.11.1 Icona

Per gestire l'**icona** dell'appviene utilizzato il pacchetto *flutter_launcher_icons*. Questo pacchetto semplifica il processo di creazione e configurazione delle icone, generando automaticamente le icone di diverse dimensioni richieste per Android e iOS a partire da un'unica immagine di base, risparmiando tempo e riducendo il rischio di errori manuali.

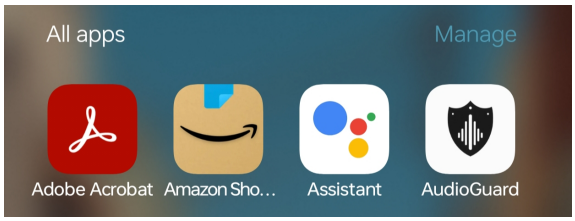


Figure 1.16: Icona dell'applicazione AudioGuard

1.11.2 Splash Screen

Per migliorare l'esperienza iniziale dell'utente, l'app include uno splash screen utilizzando il pacchetto *flutter_native_splash*. Lo splash screen è la prima cosa che l'utente vede quando avvia l'app, solitamente un'immagine o un logo che rappresenta l'applicazione, mentre vengono caricate le risorse necessarie.

(a) (b)

Figure 1.17: (a) spplash screen light e (b) splash screen dark

2 Architettura

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.

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.

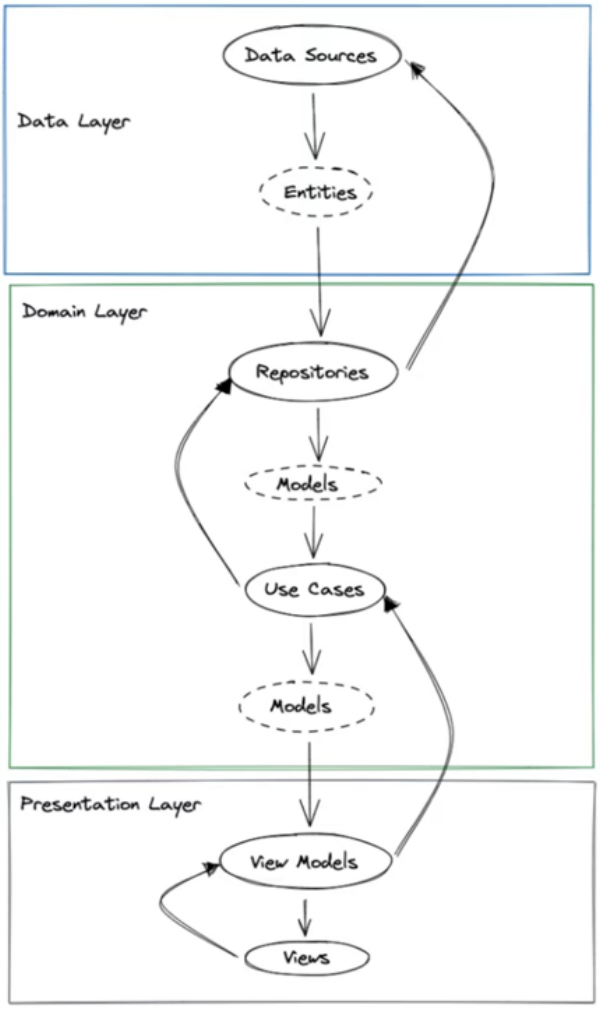


Figure 2.1: Struttura a livelli della clean architecture

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.

2.2 Vantaggi

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.

2.3 Struttura dell'Applicazione

Qui sotto è riportata un'immagine che mostra la struttura delle cartelle dell'applicazione:

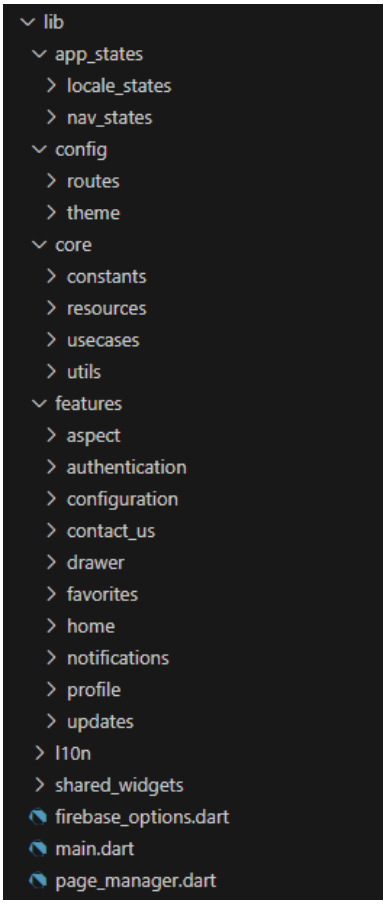


Figure 2.2: Struttura applicazione

Ecco un sommario delle principali cartelle:

- app_states: contiene i file per gestire alcuni stati
  - locale_states: questa cartella contiene i file necessari per localizzare l'applicazione nelle varie lingue
  - nav_states: questa cartella contiene i file per la navigazione tra le varie schermate.
- config
  - routes
  - theme: tutti i file relativi ai temi e alla modalità scura sono contenuti qui dentro.
- core
  - constants: costanti utilizzate nell'applicazione
  - resorurces
  - usecases
  - utils
- features: cartella fondamentale, contiene tutte le cartelle principali divise per schermate

  - aspect: cartella per la schermata di aspetto
  - configuration: cartella per la schermata di configurazione
  - contact_us: cartella per la schermata di contattaci
  - drawer: cartella per la schermata del menu
  - favorites: cartella per la schermata dei preferiti
  - home: cartella per la schermata principale
  - notifications: cartella per la schermata delle notifiche
  - profile: cartella per la schermata del profilo
  - updates: cartella per la schermata degli aggiornamenti
- l10n: cartella contenente i file .arb per le traduzioni
- shared_widgets: widgets che vengono utilizzzati più volte nell'applicazione
- firebase_options.dart
- main.dart
- page_manager.dart

3 Riverpod

Per la gestione dello stato dell'applicazione di AudioGuard ho optato per lo state manager *Riverpod*, degno erede del famoso *Provider*, che si propone come una soluzione più sicura, scalabile e flessibile rispetto alla sua versione precedente.

Figure 3.1: Logo di Riverpod

3.1 Ma cos'è lo state management?

La *gestione dello stato* (state management) è un concetto fondamentale riguardante il processo di gestione e aggiornamento dei dati (stato) dell'applicazione in modo coerente ed efficiente. In un'applicazione Flutter, lo stato può includere dati come l'interfaccia utente, le informazioni dell'utente, i dati recuperati da un'API e molto altro, ed è fondamentale saperlo gestire bene in quanto cruciale per garantire che l'applicazione sia reattiva, scalabile e facile da mantenere.

3.2 Caratteristiche di Riverpod

Riverpod presenta diverse caratteristiche interessanti, tra le quali:

- **Sicurezza del tipo**: Riverpod sfrutta appieno il sistema di tipi di Dart, prevenendo molti errori comuni durante la compilazione anziché a runtime.
- **Modularità**: Ogni provider in Riverpod è indipendente e può essere facilmente testato e riutilizzato.
- **Nessun BuildContext**: Riverpod non richiede l'uso del BuildContext per accedere ai provider, semplificando l'architettura dell'applicazione.
- **Supporto per il codice asincrono**: Riverpod gestisce perfettamente operazioni asincrone, rendendo semplice lavorare con API e altre risorse esterne.

3.3 Tipi di Provider

Riverpod offre diversi tipi di provider per soddisfare diverse esigenze di gestione dello stato. Ecco alcuni dei principali tipi di provider con una breve descrizione:

- **StateNotifierProvider**: Utilizzato per gestire lo stato complesso con la classe StateNotifier.
- **Provider**: Fornisce un valore semplice, ideale per valori costanti o calcoli leggeri.
- **StateProvider**: Permette di creare e gestire uno stato mutabile semplice.
- **FutureProvider**: Gestisce operazioni asincrone e restituisce un Future, utile per chiamate di rete o operazioni di I/O.
- **StreamProvider**: Gestisce uno Stream, utile per aggiornamenti continui di dati come flussi di dati in tempo reale.
- **ChangeNotifierProvider**: Utilizza ChangeNotifier per gestire stati complessi e notificare i listener quando lo stato cambia.

4 Firebase

4.0.1 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 Utilizzati

Di seguito verranno descritti i servizi offerti da Firebase e che sono stati utilizzati nell'applicazione:

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.

Figure 4.2: *Screenshot dalla firebase UI del db realtime di AudioGuard*

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.

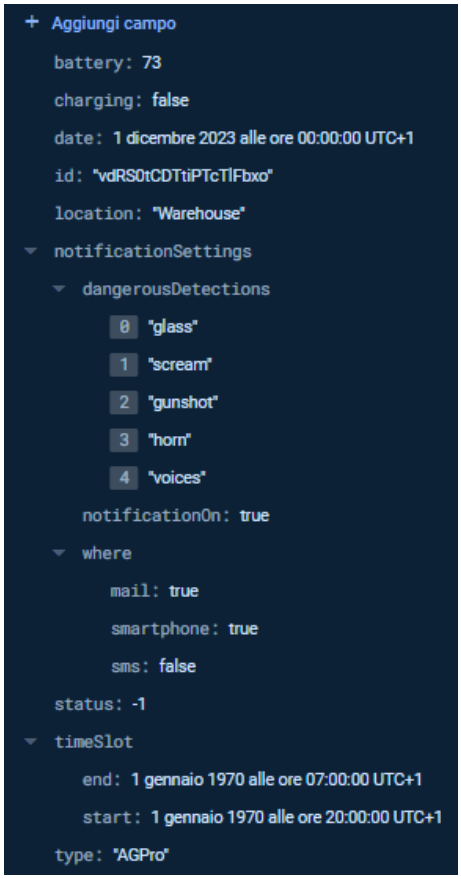


Figure 4.3: *Screenshot dalla firebase UI del db firestore di AudioGuard*

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

Al momento, essendo in fase di test, è possibile accedere solo in modalità anonima nell'applicazione di AudioGuard per facilitare lo sviluppo.