

ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management



YOUR CITY IS LISTENING TO

De Rosa Davide
Matricola 1054948
davide.derosa@studio.unibo.it

Laboratorio di Applicazioni Mobili
Anno Accademico 2023/2024

Scelte di base

Per il versionamento dell'app è stato utilizzato **Git**, con *repository* accessibile su **GitHub**.

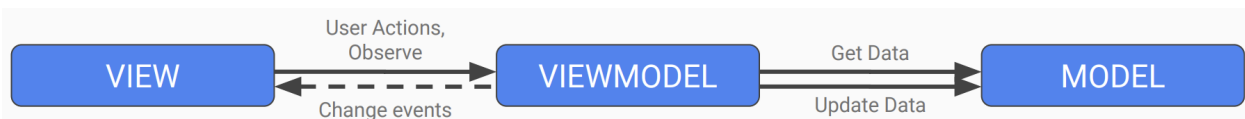
Per la realizzazione dell'applicativo è stato scelto il linguaggio **Kotlin**, realizzando un'applicazione in **Android Nativo**, seguendo il pattern architetturale **MVVM (Model View View-Model)**. In questo modo si ottiene una separazione netta tra la *UI* – ovvero le diverse *View* visibili agli utenti finali – ed i dati gestiti dal *Model*.

Il *ViewModel* permette alle *View* ed al *Model* di comunicare e di scambiarsi dati. Alcuni dei vantaggi dell'utilizzo di *ViewModel* sono:

- Se l'activity viene distrutta e ricreata non c'è bisogno di salvare l'*instance state* ogni volta
- Permette di avere una separazione netta tra il proprietario dei dati (*ViewModel*) e la logica dell'interfaccia grafica (*View*)

Tutta l'implementazione di questo pattern architetturale gira attorno all'utilizzo di **LiveData**. I *LiveData* sono basati sul concetto di **Observables**, classi che notificano un cambiamento su dati osservati. Un *Observable* ha una lista nascosta. Ogni volta che un oggetto si *iscrive ai cambiamenti* viene aggiunto a questa lista. Dato ogni cambiamento al dato osservato, tutti i membri della lista vengono notificati richiamando una funzione **callback**.

Inoltre, i *LiveData* sono componenti *Observable lifecycle-aware* che notificano solamente gli iscritti in uno stato attivo (*Resumed* o *Started*). Questo è molto utile per activity e fragments, permettendo di osservare dati senza preoccuparsi del loro stato.



Passando alla gestione del *Model*, sono state utilizzate le librerie **Room** e **Retrofit**.

Room è stata utilizzata per la persistenza dei dati all'interno di un Database Relazionale. Ha permesso di ottenere accesso al database in maniera semplice e veloce.

L'utilizzo di *Room* risulta essere abbastanza intuitivo. Una volta dichiarata una *INSTANCE* del database, basterà creare le diverse tabelle (tramite la creazione di **Entity**) e il suo corrispettivo **Dao**, dove verranno dichiarate tutte le funzioni desiderate.

Retrofit è stata utilizzata per la gestione delle chiamate API verso il backend. Ha permesso di effettuare chiamate al server in maniera asincrona e di gestire le risposte in maniera efficiente.

L'utilizzo di *Retrofit* è, come per *Room*, molto intuitivo. Anche in questo caso viene dichiarata una *INSTANCE* di *Retrofit*, nella quale viene dichiarato l'URL del server dove effettuare le chiamate. Successivamente viene creata un'interfaccia contenente tutte le chiamate desiderate.

Tutte le funzioni inerenti al Database e alle chiamate API vengono gestite all'interno di *ViewModel*, permettendo un approccio moderno e sicuro.

Si osservino ora alcuni esempi, inerenti all'utilizzo di *Room* e *Retrofit* nel contesto dell'applicativo.

```

private var fileCorrectlyUploadedLiveData = MutableLiveData<UploadData>()
private var fileCorrectlyUploadedErrorLiveData = MutableLiveData<String>()

fun uploadAudio(token: String, longitude: Double, latitude: Double, audio:
MultipartBody.Part) {
    RetrofitInstance.api.uploadAudio(token, longitude, latitude, audio)
        .enqueue(object : Callback<ResponseBody> {
            override fun onResponse(
                call: Call<ResponseBody>,
                response: Response<ResponseBody>
            ) {
                if (response.isSuccessful) {
                    val uploadData =
                        gson.fromJson(
                            response.body()!!.string(),
                            UploadData::class.java
                        )

                    Log.d(
                        "UploadAudio 200",
                        "UploadAudio 200: " + uploadData.toString()
                    )

                    fileCorrectlyUploadedLiveData.value = uploadData
                } else {
                    when (response.code()) {
                        401 -> {
                            val userNotAuthorized = gson.fromJson(
                                response.errorBody()!!.string(),
                                UserNotAuthorized::class.java
                            )

                            Log.d(
                                "UploadAudio 401",
                                "UploadAudio 401 User Not Authorized: " +
                                    userNotAuthorized.detail
                            )

                            fileCorrectlyUploadedErrorLiveData.value =
                                userNotAuthorized.detail
                        }

                        413 -> {
                            val fileTooBig = gson.fromJson(
                                response.errorBody()!!.string(),
                                FileTooBig::class.java
                            )

                            Log.d(
                                "UploadAudio 413",
                                "UploadAudio 413 File too big: " +
                                    fileTooBig.detail
                            )

                            fileCorrectlyUploadedErrorLiveData.value =
                                fileTooBig.detail
                        }
                    }
                }
            }
        })
}

```

```

        415 -> {
            val fileNotAudio = gson.fromJson(
                response.errorBody()!!.string(),
                FileNotAudio::class.java
            )

            Log.d(
                "UploadAudio 415",
                "UploadAudio 415 File not audio: " +
                    fileNotAudio.detail
            )

            fileCorrectlyUploadedErrorLiveData.value =
                fileNotAudio.detail
        }
    }
}

override fun onFailure(call: Call<ResponseBody>, t: Throwable) {
    Log.d("Fail Upload Audio", t.message.toString())

    fileCorrectlyUploadedErrorLiveData.value = t.message
}

})
}

fun observeFileCorrectlyUploadedLiveData(): LiveData<UploadData> {
    return fileCorrectlyUploadedLiveData
}

fun observeFileCorrectlyUploadedErrorLiveData(): LiveData<String> {
    return fileCorrectlyUploadedErrorLiveData
}
}

```

Il seguente blocco di codice rappresenta un esempio di chiamata API eseguita tramite Retrofit, nello specifico la chiamata inerente all'*upload dell'audio*. Come si può notare, il tutto è gestito tramite l'utilizzo di *LiveData* – in questo caso due: uno per l'esito positivo, l'altro per quello negativo – il quale ci permette di gestire al meglio le chiamate asincrone effettuate da Retrofit.

```

private fun observeUpload() {
    audioViewModel.observeFileCorrectlyUploadedLiveData().observe(this) {
        Toast.makeText(this, "Caricamento avvenuto con successo!",
            Toast.LENGTH_SHORT).show()

        audioViewModel.insertAudioDb(
            AudioDataEntity(
                username = DataSingleton.username,
                longitude = longitude,
                latitude = latitude,
                locationName = getLocationName(longitude, latitude),
                bpm = it.bpm,
                danceability = it.danceability,
                loudness = it.loudness,
                genre = it.genre.getMaxGenre().first,
                mood = it.mood.getMaxMood().first,
                instrument = it.instrument.getMaxInstrument().first
            )
        )
    }
}

```

```

        )
    )

    goToAppActivity()
}

audioViewModel.observeFileCorrectlyUploadedErrorLiveData().observe(this) {
    Toast.makeText(this, it, Toast.LENGTH_SHORT).show()
    deleteRecording()
    goToLogin()
}
}

```

Ovunque si voglia utilizzare la funzione *uploadAudio(...)* sarà necessario **osservare** i due LiveData inerenti, implementando tutta la logica all'interno del blocco di codice eseguito una volta ricevuto il LiveData aggiornato (come si può notare nel blocco di codice).

Per completezza, ecco anche un esempio di utilizzo di *Room*.

```

private lateinit var audioDbLiveData: LiveData<AudioDataEntity>

fun getAudioById(id: Int) {
    audioDbLiveData = audioDatabase.audioDataDao().getAudioById(id)
}

fun observeAudioDbLiveData(): LiveData<AudioDataEntity> {
    return audioDbLiveData
}

```

il quale richiama il seguente metodo presente nel **Dao**:

```

@Query("SELECT * FROM audiodata WHERE id = :id")
fun getAudioById(id: Int): LiveData<AudioDataEntity>

```

Anche in questo caso viene seguita la logica di osservare i LiveData, permettendo una gestione asincrona delle chiamate e di gestione delle risposte.

Per concludere le diverse scelte di base, segnalo anche l'utilizzo delle **Shared Preferences**. Quest'ultime sono state utilizzate per diversi controlli, come quello per la connessione ad Internet e per la gestione dell'account all'interno dell'app.

Per non dover accedere ad ogni avvio dell'app, vengono utilizzate le shared preferences per mantenere persistenti *token* ed *username* ad ogni avvio.

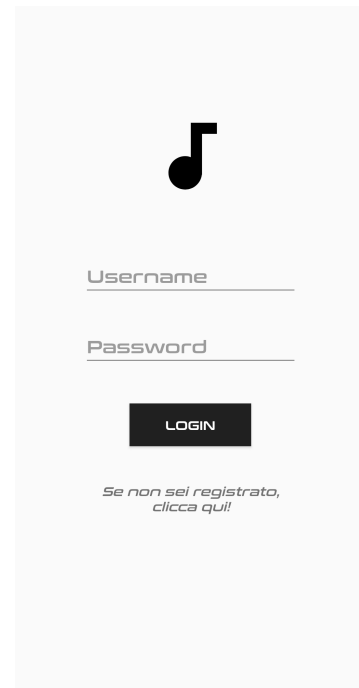
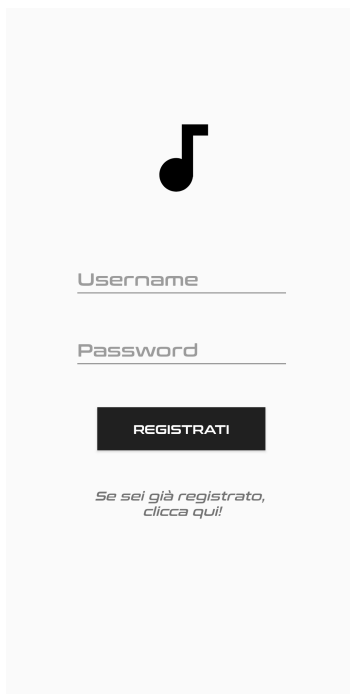
Nel momento in cui avviene un logout, o il token risulta essere non valido, vengono resettate.

Implementazione

Login e Registrazione

All'avvio dell'app, la prima *Activity* mostrata è quella inerente al **Login** dell'account. Prima di mostrare la schermata visibile in foto, viene effettuato un controllo di connessione ad Internet. Se non è presente alcuna connessione, si verrà reindirizzati ad una sezione **Offline**, con funzionalità limitate (se ne parlerà successivamente).

In caso di connessione attiva, si può procedere all'accesso. Un ulteriore controllo viene effettuato in caso di presenza di accesso già effettuato. Se il *Token* risulta essere ancora valido, non sarà necessario effettuare il Login, ma si verrà reindirizzati direttamente alla schermata principale dell'app. Inoltre, all'avvio dell'app viene richiesto il permesso per ricevere notifiche, necessario per la funzionalità di *upload in background* (se ne parlerà successivamente).

A vertical rectangular mockup of a login screen. At the top center is a large black musical note icon. Below it are two text input fields: the first is labeled 'Username' and the second 'Password', both with light gray placeholder text. Below the password field is a black rectangular button with the word 'LOGIN' in white capital letters. At the bottom, centered, is a line of small gray text that reads 'Se non sei registrato, clicca qui!'.A vertical rectangular mockup of a registration screen. At the top center is a large black musical note icon. Below it are two text input fields: the first is labeled 'Username' and the second 'Password', both with light gray placeholder text. Below the password field is a black rectangular button with the word 'REGISTRATI' in white capital letters. At the bottom, centered, is a line of small gray text that reads 'Se sei già registrato, clicca qui!'.

Per quanto riguarda la **Registrazione**, è possibile registrarsi tramite un'Activity molto simile a quella del Login.

In questo caso, una volta registrati con successo si verrà reindirizzati alla schermata di Login, per effettuare l'accesso.

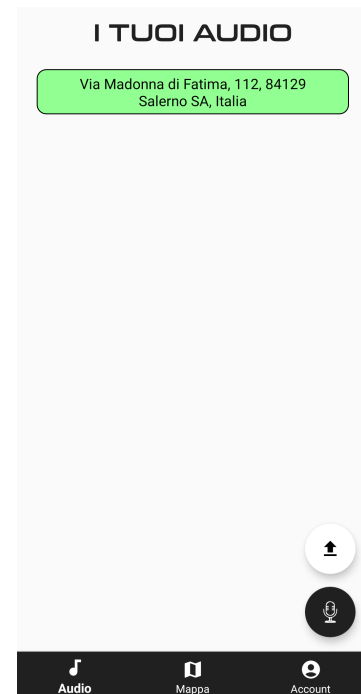
In caso di account già esistente, verrà notificato l'errore.

Audio Personali

La seguente Activity presenta una **Navigation Bottom Bar**, la quale permette di selezionare tre diversi **Fragment**.

Il primo di questi è quello inerente agli **Audio Personali**. In questa schermata sono presenti tutti gli audio caricati dall'utente. Gli audio sono visualizzati tramite una **Recycler View**, caricata dinamicamente attraverso la chiamata API *"audio/my"*.

Per ogni audio caricato sul server, vengono salvati all'interno del database locale tutti i suoi metadati, permettendone una visualizzazione anche Offline (se ne parlerà successivamente).



INFO AUDIO

Username del creatore: prova
Località: Via Madonna di Fatima, 112, 84129 Salerno SA, Italia
Longitudine: 14.7953632
Latitudine: 40.6629017
BPM: 157
Danzabilità: 1.208294153213501
Rumorosità: 16.203601837158203
Mood: energetic
Genere: electronic
Strumento principale: synthesizer



Una volta cliccato uno degli elementi nella lista, si verrà reindirizzati alla seguente Activity, dove vengono mostrati tutti i metadati inerenti all'audio caricato. E' stato inoltre implementato un controllo che verifica la presenza dei metadati in locale una volta cliccato l'elemento. Se non è presente, verrà chiamata l'API *"audio/{id}"* per ottenere i suoi metadati e salvarli in locale.

Sarà quindi possibile **ascoltare la registrazione** caricata in precedenza.

Inoltre, è possibile **nascondere** l'audio dalla piattaforma, o **cancellarlo** definitivamente.

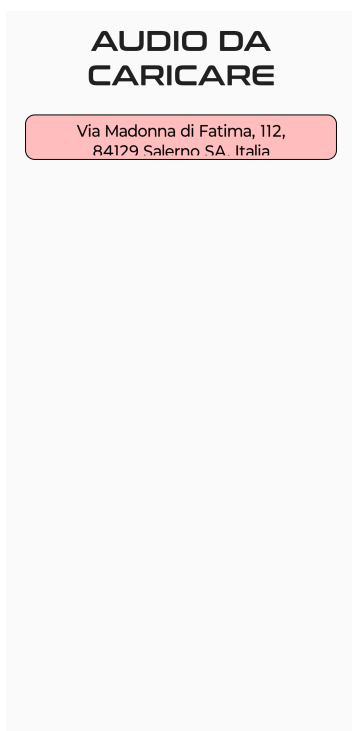
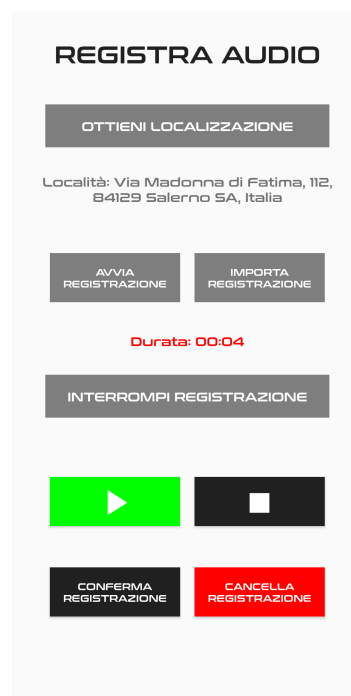
Registrazione Audio e Lista Upload

Premendo sul **Floating Button** nero presente nel Fragment degli Audio Personali si verrà reindirizzati all'Activity per la **registrazione di un nuovo audio**.

Inizialmente, è presente solo il bottone *Ottieni Localizzazione*, il quale richiede i permessi di localizzazione. Una volta geolocalizzato l'audio, diventa visibile la località con i pulsanti *Avvia Registrazione* ed *Importa Registrazione* (se ne parlerà successivamente nella sezione Extra).

Una volta avviata la registrazione, diventano visibili la durata della registrazione e il bottone *Interrompi Registrazione*. Premuto l'ultimo, verranno mostrati tutti gli altri bottoni.

Si potrà **riascoltare** l'audio appena registrato, **cancellarlo**, o **confermarlo** effettuando il caricamento. Prima di procedere al caricamento, viene controllata la presenza di una connessione Wi-Fi. Se non presente, verrà richiesto se si vuole comunque procedere con i dati mobili. In caso contrario, l'audio verrà salvato in locale nel database per poter essere caricato successivamente.



Gli audio salvati per essere caricati successivamente sono presenti in questa Activity, visibili all'interno di una Recycler View.

Qui è possibile avviare manualmente il caricamento, anche con dati mobili. Verrà mostrato lo stesso *Alert Dialog* mostrato nella fase di registrazione. In caso di connessione Wi-Fi, il caricamento avverrà senza mostrare alcun pop-up.

Nota Bene: vedremo successivamente come gli audio verranno caricati automaticamente in background in caso di connessione Wi-Fi.

Un altro dettaglio è inerente all'audio registrato. Il formato audio registrato tramite la classe **MediaRecorder** di Android non è supportato nativamente dal server. Viene quindi utilizzata la libreria **ffmpeg** per gestire la conversione del file audio in un formato supportato.

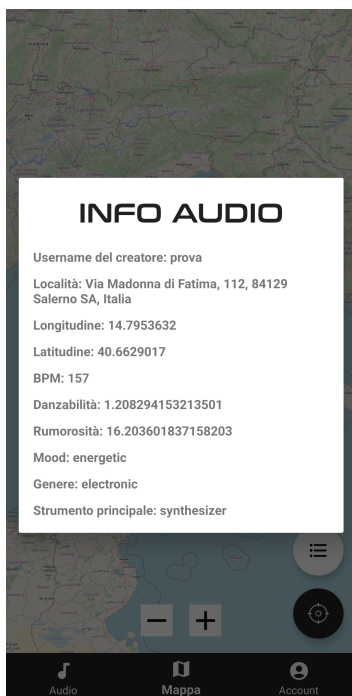
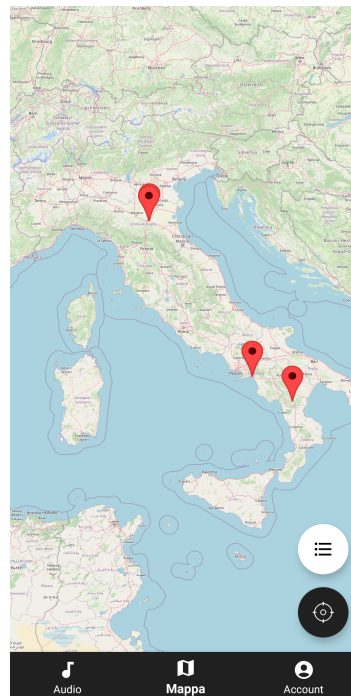
Mappa

Il pulsante **Mappa** nella Navigation Bottom Bar permette di caricare il fragment inerente alla mappa. Sulla mappa sono presenti tutti gli audio della chiamata API `"audio/all"`.

Per la mappa viene utilizzata la libreria **osmdroid**, alternativa gratuita a Google Maps.

Il Fragment presenta inoltre due Floating Button. Uno per ottenere l'intero elenco degli audio in piattaforma sotto forma di lista (se ne parlerà successivamente nella sezione Extra).

L'altro per ottenere la localizzazione e zoommare sulla mappa nel punto in cui ci troviamo. Vengono inoltre richiesti i permessi per la localizzazione la prima volta.



Una volta premuto su uno dei *marker* presenti sulla mappa, verrà aperto un **DialogFragment** custom. All'interno di questo Fragment si possono osservare tutti i metadati inerenti all'audio selezionato. Anche in questo caso, viene controllata la presenza dei metadati dell'audio nel database locale. In caso negativo, viene effettuata la chiamata API `"audio/{id}"` per ottenere e salvare i metadati.

Una funzionalità extra, della quale parleremo successivamente, effettua un **fetching automatico** se sullo schermo sono presenti dieci o meno marker. In questo modo non ci sarà alcun delay durante l'apertura del DialogFragment, avendo già i metadati salvati in locale.

Account

L'ultimo Fragment accessibile tramite la Navigation Bottom Bar è quello inerente all'**Account**. In questa schermata è presente una breve descrizione dell'applicativo.

Vengono inoltre mostrati due bottoni.

Uno per effettuare il **Logout**, tornando all'Activity Login.

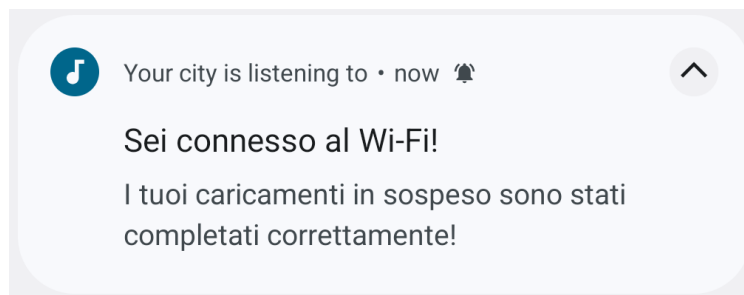
L'altro, invece, per **cancellare il proprio account**, effettuando la chiamata API `"auth/unsubscribe"`.



Servizio in Background

Come richiesto dalla traccia, è stato implementato un **servizio che agisce in background**. Il **Service** in questione controlla ogni cambiamento della connessione Wi-Fi, e grazie ad un **Broadcast Receiver** ogni cambiamento viene ascoltato e gestito. In caso di connessione Wi-Fi stabile e connessa ad Internet, verrà controllato l'elenco degli upload in attesa. Se l'elenco risulta vuoto, non verrà eseguita alcuna funzione. In caso contrario, tutti gli upload presenti verranno caricati sul server, salvando il risultato nel database locale per utilizzi futuri.

Concluso l'upload, verrà avviato un **Worker**, il quale si occupa di **inviare la notifica** all'utente.



Premendo sulla notifica, verrà aperta l'app, permettendo di interagire con gli audio caricati.

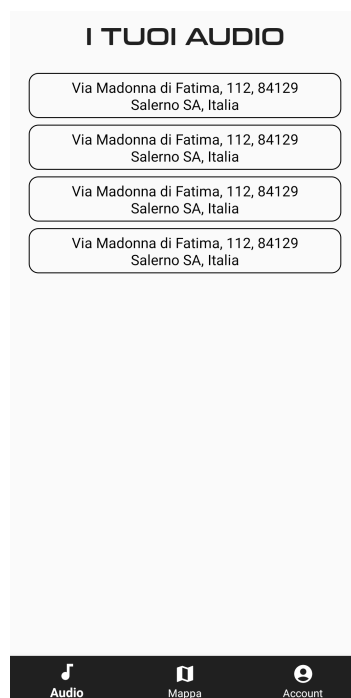
Sezione Offline

Si accede alla **sezione Offline** se si apre l'app senza connessione, o se durante l'utilizzo dell'app non è più disponibile una connessione ad Internet.

In questa sezione l'Activity principale propone funzionalità limitate. Il Fragment della Mappa e dell'Account non offrono alcuna funzionalità, dipendendo da Internet (Mappa) e da un account specifico (Account).

L'implementazione della sezione Offline si basa su una logica indipendente da un singolo account.

Nella sezione Audio Personali sarà possibile consultare tutti gli audio salvati all'interno del database locale. Anche in questo caso, la visualizzazione dei singoli audio avviene tramite Recycler View.



INFO AUDIO

Username del creatore: prova
Località: Via Madonna di Fatima, 112, 84129 Salerno SA, Italia
Longitudine: 14.7953632
Latitudine: 40.6629017
BPM: 157
Danzabilità: 1.208294153213501
Rumorosità: 16.203601837158203
Mood: energetic
Genere: electronic
Strumento principale: synthesizer



Una volta premuto su l'elemento scelto, verrà aperta un'Activity molto simile alla sua controparte "*Online*", privata di alcune funzionalità dipendenti dal server.

Sarà ovviamente ancora possibile ascoltare l'audio, essendo salvato in memoria e non sul server.

Per tornare alla versione completa dell'applicativo, bisognerà chiudere a riaprire l'app una volta stabilita una connessione ad Internet.

Funzionalità Extra

Come anticipato, sono state aggiunte **funzionalità extra** all'interno dell'app. Alcune di queste sono richieste opzionali presenti nella traccia, altre sono funzionalità aggiunte in autonomia.

Importa audio

Una funzionalità che può risultare comoda in molti casi è l'**importare audio** dall'esterno. Come visto in precedenza nell'Activity inerente alla *registrazione audio*, è presente un bottone che permette di importare audio dalla propria memoria.

Questa funzione è realizzata tramite un *Intent* di tipo *"audio/*"*, il quale permette di selezionare soltanto file audio. Una volta selezionato, la libreria *ffmpeg* procederà a convertire e salvare l'audio in memoria, rendendolo fruibile all'interno dell'app.

Salvataggio in locale

Come consigliato dalla traccia, i metadati inerenti agli audio vengono salvati in locale, all'interno di un database, per essere riutilizzati successivamente.

Questo procedimento avviene sia per gli audio caricati dall'utente – denominati *MyAudio* – sia per gli audio presenti in piattaforma – denominati *AllAudio*.

Il salvataggio in locale dei metadati permette all'applicativo di non dover interrogare il server per ogni richiesta effettuata dall'utente. Prima di interpellare il server, verrà controllata la presenza dei metadati in locale. Se non presenti, verrà effettuata la chiamata API al server, con conseguente salvataggio in locale della risposta ottenuta.

Upload in background

Anche in questo caso, viene aggiunta una funzionalità extra alla richiesta della traccia.

L'app non invia una notifica quando una connessione Wi-Fi stabile è presente, ma prosegue in automatico con l'upload in background, inviando una notifica al termine dell'upload.

Come spiegato in precedenza, il tutto avviene grazie ad un **Service**, il quale controlla in background ogni cambiamento della connessione Wi-Fi grazie ad un **Broadcast Receiver**. Una volta stabilita la connessione Wi-Fi stabile, verrà controllata la presenza di upload in attesa di essere caricati. Se presenti, il Service procederà all'upload in background, inviando la notifica al termine dell'operazione.

Fetching automatico AllAudio

Per rendere l'esperienza più fluida, è stato implementato un **fetching automatico** per gli AllAudio visibili sulla mappa.

Ogni volta che avviene un movimento sulla mappa, un controllo sul numero dei marker visibili viene effettuato (*Nota Bene: è presente un debounce di 7 secondi tra i controlli*). Se il numero è minore o uguale a 10, avviene un *fetching automatico* dei metadati dei marker visibili a schermo, permettendo una visualizzazione più rapida del DialogFragment.

Il fetching salva in automatico all'interno del database locale tutti i metadati ottenuti, utilizzandoli per tutte le richieste successive.

Lista di AllAudio

Oltre alla visualizzazione sulla mappa dei diversi *marker* inerenti ad un singolo audio, è stata aggiunta una nuova visualizzazione sotto forma di lista. Anche in questo caso, come per le liste presenti in tutta l'app, è stata usata una Recycler View.

Premendo su un elemento verrà aperto il DialogFragment custom utilizzato anche per il click sui marker.

TUTTI GLI AUDIO
Luogo sconosciuto! Longitude: 1.34 Latitudine: 2.33
Luogo sconosciuto! Longitude: 1.34 Latitudine: 2.33
Luogo sconosciuto! Longitude: 1.34 Latitudine: 2.33
Corso Giuseppe Garibaldi, 57, 85030 San Severino Lucano PZ, Italia
Unnamed Road, Mountain View, CA 94043, Stati Uniti
Via Madonna di Fatima, 112, 84129 Salerno SA, Italia
Via Madonna di Fatima, 112, 84129 Salerno SA, Italia
Via Madonna di Fatima, 112, 84129 Salerno SA, Italia
Via di Corticella, 238, 40128 Bologna BO, Italia
Via Guglielmo Marconi, 18, 40122 Bologna BO, Italia
Via delle Lame, 29, 40122 Bologna BO, Italia
Via Madonna di Fatima, 112, 84129 Salerno SA, Italia