Report Progetto LAM 2024

Your City Is Listening To (**YCLIT**)

Francesco Licchelli #0001041426 francesco.licchelli@studio.unibo.it

Novembre 2024

Introduzione

YCLIT è un’applicazione Android sviluppata per il corso di Laboratorio di Applicazioni Mobili. L'app consente agli utenti di visualizzare su una mappa interattiva informazioni sulla musica dal vivo nei loro dintorni e di caricare nuove registrazioni, fornendo dettagli come BPM, "danceability" e altre metriche estratte dal backend. Se una registrazione caricata da un utente è impostata come pubblica, sarà visibile a tutti gli altri utenti sulla mappa interattiva integrata con Google Maps.
Le principali funzionalità sono:

- Visualizzazione delle registrazioni pubbliche
- Caricamento di nuove registrazioni
- Gestione delle registrazioni personali

Scelte progettuali

L’applicazione supporta le versioni di SDK 33+ (Android 13+) per garantire la compatibilità con il pacchetto Geocoder, utilizzato per convertire le coordinate in indirizzi comprensibili all’utente. Il token di autenticazione viene memorizzato nelle SharedPreferences, consentendo agli utenti di evitare il login ad ogni avvio dell’app, a meno che il backend non richieda una nuova autenticazione per la scadenza del token.

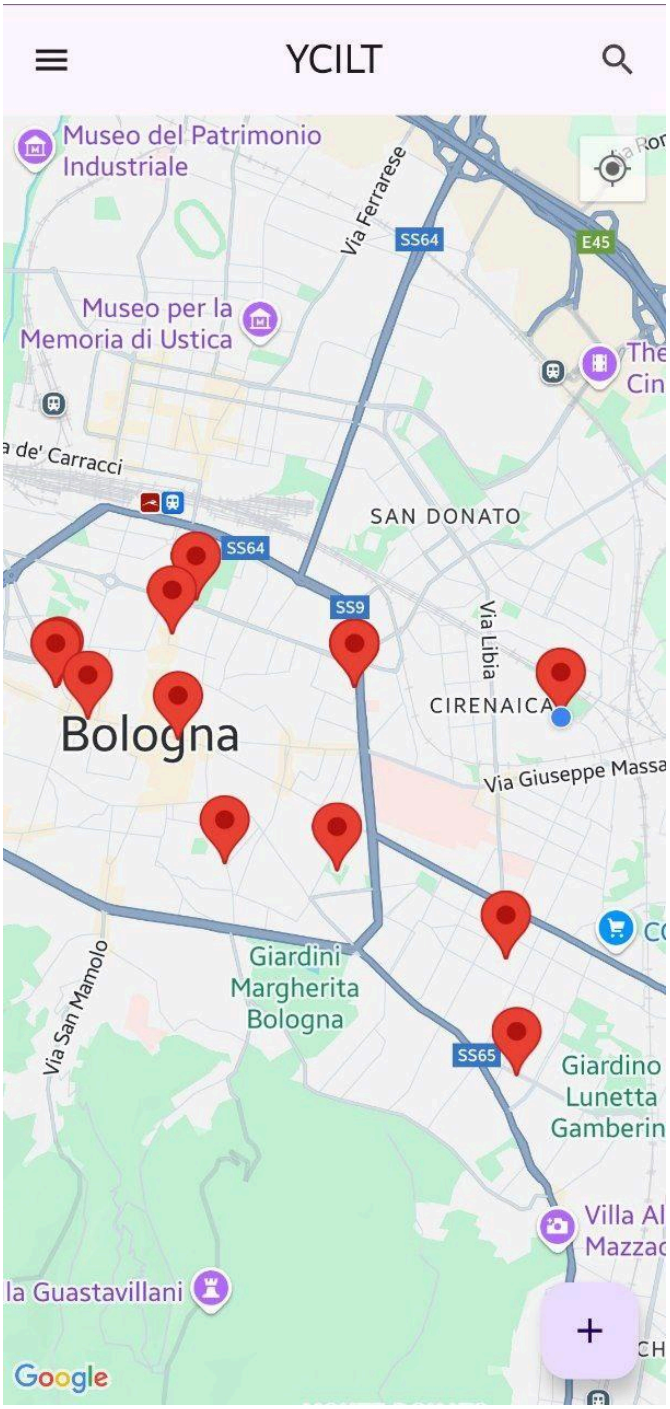
Ogni layout è stato realizzato utilizzando Jetpack Compose, che, grazie alla sua API dichiarativa, semplifica la scrittura dei componenti e ne velocizza la creazione. Tutti i file vengono salvati nella memoria interna dell’applicazione, eliminando la necessità dei permessi di accesso a memoria esterna (`READ_EXTERNAL_STORAGE` e `WRITE_EXTERNAL_STORAGE`).

Activities

MainActivity

La `MainActivity` è il punto d'ingresso dell'applicazione. Se non viene rilevato un token valido oppure se il token è scaduto, l'utente viene reindirizzato alla schermata di login per effettuare nuovamente l'autenticazione.

L'interfaccia, definita nella `MainScreen`, comprende i seguenti elementi:



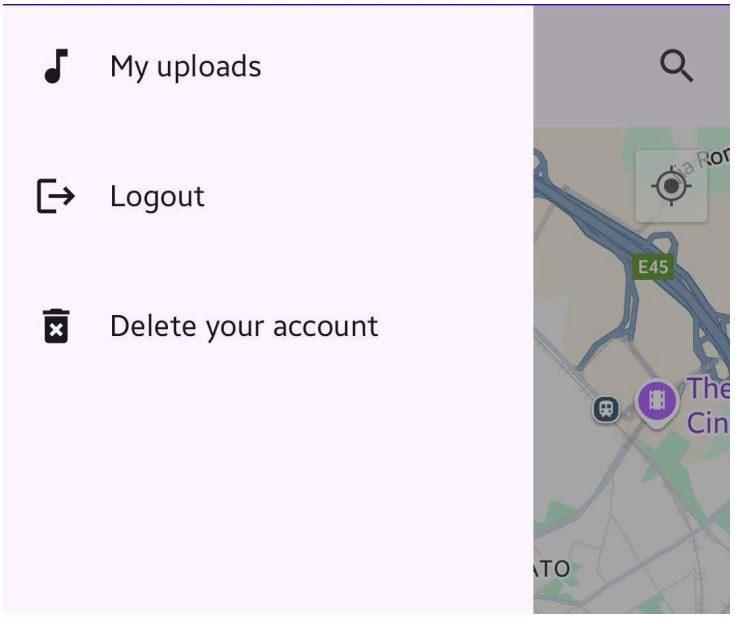
- **GoogleMap**: una mappa in cui vengono aggiunti dei marker che rappresentano ogni audio attualmente pubblico sul server. I marker vengono caricati all'avvio della mappa e aggiornati ogni 5 secondi per mantenere le informazioni aggiornate. Cliccando su un marker, l'utente viene reindirizzato alla `AudioInfoActivity`, che mostra informazioni estratte dal backend sull'audio selezionato.
- **Pulsante per la registrazione audio**: un pulsante che permette di accedere alla `AudioRecorderActivity`, in cui è possibile registrare nuovi audio e salvarli nell'app.
- **Drawer**: un menu laterale che consente di navigare verso `MyAudioActivity` per visualizzare e gestire le registrazioni personali. Il drawer include anche l'opzione per effettuare il logout, che svuota le SharedPreferences riportando l'utente alla `LoginActivity`, e l'opzione per eliminare l'account. Quest'ultima esegue una richiesta DELETE all'endpoint `/auth/unsubscribe`; se la richiesta ha esito positivo, vengono eliminati i file

caricati dall'utente (file audio e relativi metadati).

- **SearchBar**: una barra di ricerca che consente all'utente di trovare località o indirizzi. La ricerca utilizza le API di Geocoder tramite una richiesta GET a `https://maps.googleapis.com/maps/api/geocode/json?key=<APIKEY>&address=<ADDRESS>`. Se la richiesta ha successo e l'indirizzo è trovato, viene restituito un JSONArray. Se l'oggetto è vuoto, significa che l'indirizzo non è stato identificato, altrimenti i risultati sono ordinati per probabilità decrescente, e viene usato il primo elemento per estrarre i campi `.geometry.location.lat` e `.geometry.location.lng` per aggiornare la posizione della mappa.

Metodi principali

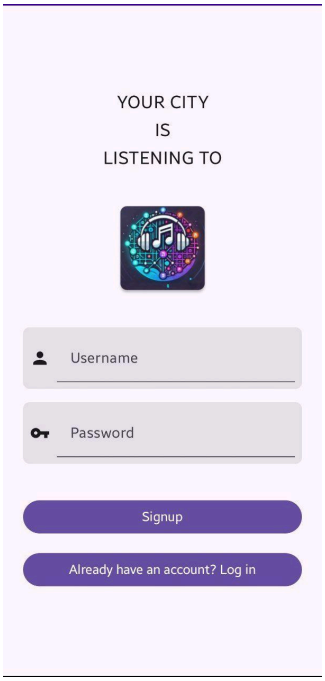
- **onCreate**: è il metodo principale che inizializza l'interfaccia `MainScreen`, configurando la mappa, la barra di ricerca e il drawer. All'interno di questo metodo vengono impostati anche i listener per le navigazioni verso altre activity, come la `AudioRecorderActivity` e `MyAudioActivity`. Inoltre, `onCreate` gestisce lo stato dei permessi di geolocalizzazione, chiedendo il permesso se non è stato ancora concesso e registrando un listener per reagire al cambiamento del permesso.



- **loadAudio**: è un metodo dedicato a scaricare le informazioni sugli audio pubblici presenti sul backend. Invia una richiesta GET all'endpoint `/audio/all`, ricevendo i dati relativi agli audio pubblici come coordinate, ID, e visibilità. In caso di successo, `loadAudio` aggiorna i marker sulla mappa, posizionando un marker per ciascun audio pubblico e associando a ogni marker un listener che, al clic, reindirizza alla `AudioInfoActivity` per visualizzare i dettagli dell'audio selezionato.

Auther, LoginActivity, SignupActivity

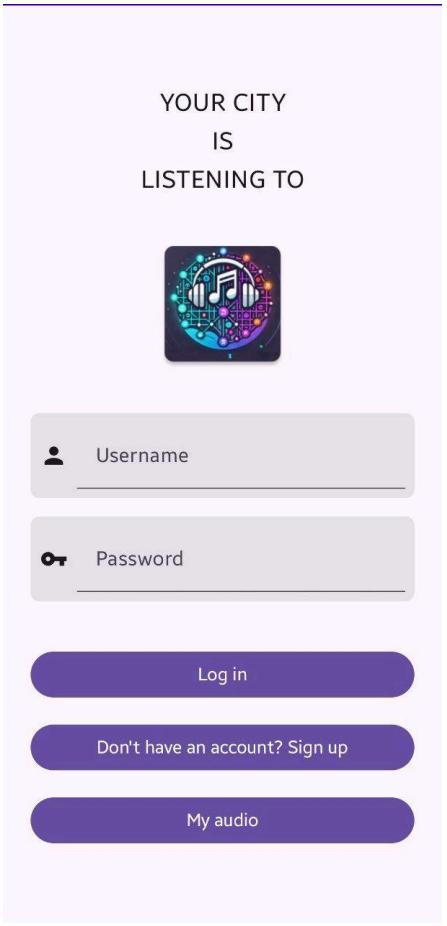
`Auther` è una classe astratta che fattorizza le operazioni comuni tra `LoginActivity` e `SignupActivity`. Questa classe definisce due metodi principali:



- **auth**: metodo che invia una richiesta POST verso il proprio endpoint dedicato, che può essere `/auth` per la registrazione o `/auth/token` per il login. Ogni chiamata al metodo `auth` gestisce l'invio delle credenziali e la ricezione della risposta dal backend.
- **perform**: metodo che esegue `auth` e gestisce il risultato della richiesta secondo la logica specifica delle implementazioni concrete di `Auther`. Nel caso del login, ad esempio, `perform` salva il token di autenticazione nelle `SharedPreferences` per consentire all'utente di accedere all'app senza dover effettuare nuovamente il login. Inoltre, riavvia tutti i worker che erano stati messi in pausa a causa dell'invalidità del token, ora rinnovato dopo il login.

Nel caso di `SignupActivity`, `perform` mostra un messaggio di conferma della corretta registrazione e reindirizza l'utente alla `LoginActivity`.

Nel caso di `LoginActivity`, `perform` salva nelle Shared Preferences il nuovo token, riavvia tutti i Worker messi precedentemente in pausa quando il precedente token e' scaduto e reindirizza verso la `MainActivity`.



Entrambe le interfacce contengono campi per l'inserimento di username e password, un pulsante per il login o la registrazione e un pulsante per passare all'altra schermata (da Login a Signup e viceversa).

Nella `LoginActivity` è presente un terzo pulsante che reindirizza a `MyAudioInfoActivity`, consentendo la riproduzione dei propri audio e la visualizzazione delle relative informazioni anche senza connessione.

MyAudioActivity

`MyAudioActivity` visualizza una lista degli audio salvati dall'utente. Ogni elemento della lista rappresenta un audio e mostra le seguenti informazioni:

- **Nome del file MP3**: il nome del file audio salvato.
- **Data e ora di registrazione**: la data e l'ora in cui è stata effettuata la registrazione.
- **Durata della registrazione**: la durata totale dell'audio registrato.

Ogni volta che l'activity viene ripresa, la lista degli audio viene aggiornata tramite la funzione `updateMetadataFromBackend`. Se l'utente è loggato, questa funzione invia una richiesta GET all'endpoint `/audio/my`, che restituisce un JSONArray contenente i dettagli degli audio caricati dall'utente. Per ciascun audio, vengono recuperati l'ID, lo stato di privacy (pubblico o privato) e le coordinate geografiche associate.

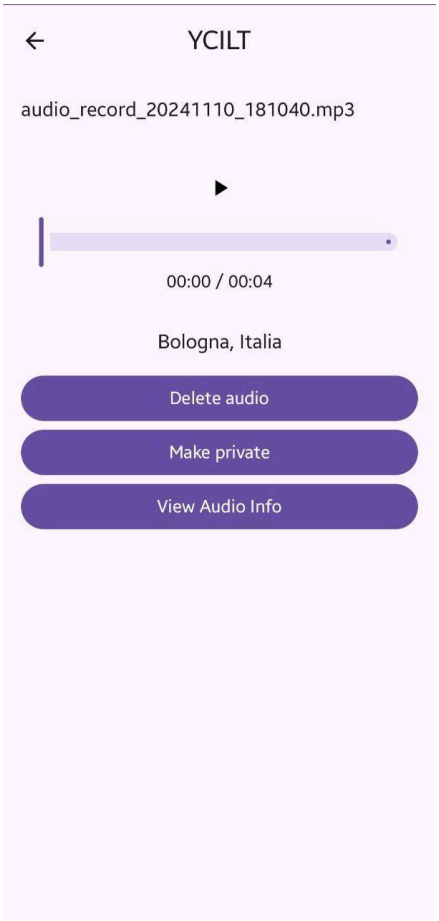
Se viene effettuata la richiesta, durante il periodo della stessa verra' visualizzata un'animazione di caricamento.

Le informazioni ricevute vengono utilizzate per aggiornare i metadati locali. Inoltre, per ogni "nuova" canzone — ovvero quelle per le quali non è ancora stato salvato un ID — viene inviata una richiesta GET all'endpoint `/audio/<AUDIO_ID>` per ottenere ulteriori dettagli sull'audio, come le caratteristiche audio (ad esempio, BPM, danceability, ecc.), e aggiornare così le informazioni locali.

MyAudioInfoActivity

`MyAudioActivity` permette agli utenti di riprodurre gli audio salvati e, se l'utente è loggato, offre ulteriori funzionalità di gestione degli audio:

- **Eliminazione dell'audio**:
  - Se l'audio è stato caricato sul server, viene attivato un `DeleteAudioWorker` per rimuoverlo dal backend.
  - Se l'audio non è stato ancora caricato, viene eliminato il relativo `UploadAudioWorker`, e viene aggiornato di conseguenza anche il `NotificationWorker` per riflettere il cambiamento.
- **Modifica della privacy dell'audio**: Un pulsante dedicato permette di modificare la privacy dell'audio (rendendolo pubblico o privato). Questo pulsante è attivo solo se l'utente è loggato, l'audio è stato caricato sul server e sono stati successivamente recuperati i dati relativi all'audio.
- **Visualizzazione delle informazioni dell'audio**: L'utente può visualizzare informazioni dettagliate sull'audio selezionato navigando verso la `AudioInfoView`. Questo pulsante è attivato solo se l'audio è stato caricato sul server.



La riproduzione dell'audio avviene tramite il componente `AudioPlayer`.

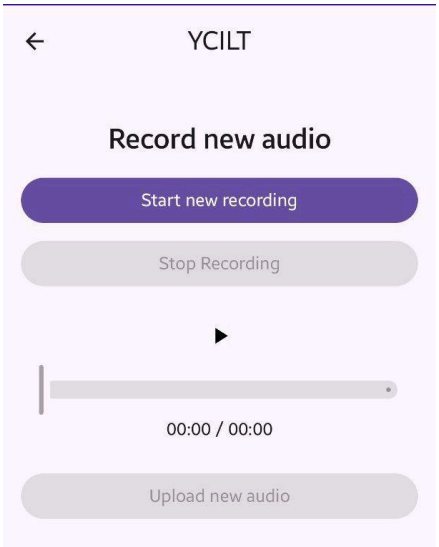
`AudioPlayer` utilizza MediaPlayer per riprodurre il file audio specificato, consentendo agli utenti di avviare/interrompere la riproduzione tramite un pulsante e di regolare la posizione tramite una seekbar. Mostra anche la durata totale e la posizione corrente. Il componente è abilitato solo se il file audio è disponibile

Viene inoltre visualizzata la posizione salvata per l'audio in questione, attraverso `LocationInfoView`. La funzione `coordToAddr` si occupa di convertire le coordinate in un indirizzo, utilizzando `Geocoder()`, che mette a disposizione il metodo `.getFromLocation()`, il quale ritorna l'indirizzo corrispondente alle coordinate. Nel caso in cui `coordToAddr` fallisse, verrebbero mostrate semplicemente le coordinate corrispondenti all'audio.

AudioInfoActivity

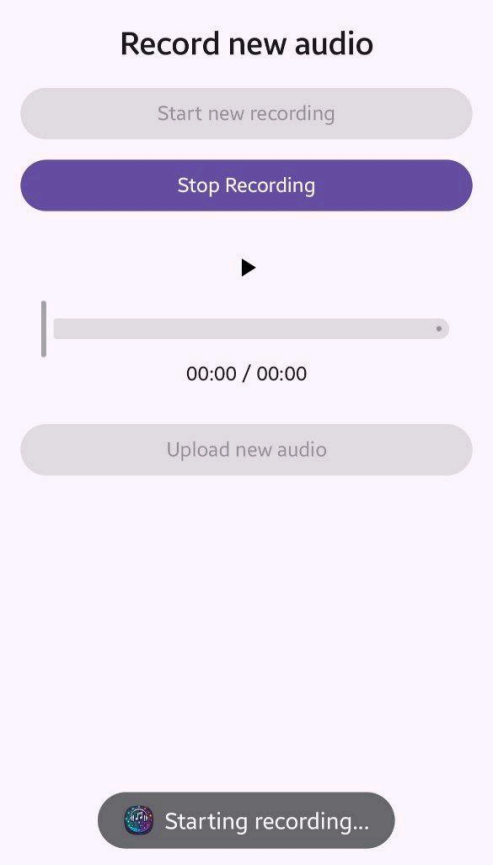
`AudioInfoActivity` visualizza tutte le informazioni dettagliate relative a uno specifico audio selezionato dall'utente. Tra i dati mostrati troviamo l'autore, il BPM (Battiti per Minuto), la danceability, la rumorosità (loudness), e per le categorie **mood**, **genere** e **strumenti**, vengono mostrate solo le 5 opzioni più probabili. Questo approccio permette di mantenere l'interfaccia pulita e facilmente leggibile, evitando di sovraccaricare l'utente con troppe informazioni. In questo modo, l'utente può visualizzare le informazioni più rilevanti senza distrazioni.

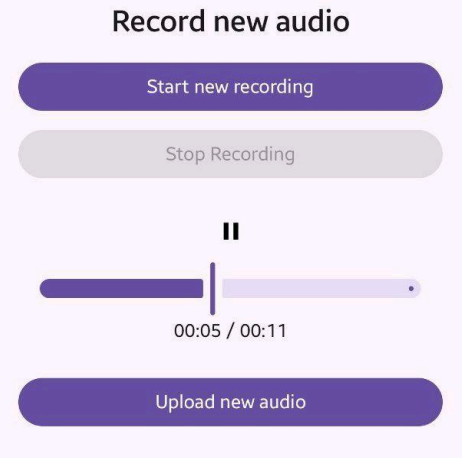
AudioRecorderActivity



`AudioRecorderActivity` permette di registrare un audio e di riascoltarlo prima di salvarlo e inviarlo al backend. In particolare, quando si salva un audio, verrà schedulato un `UploadAudioWorker` che caricherà l'audio una volta connessi a una rete Wi-Fi. Inoltre, se al momento del salvataggio non si è connessi a una rete Wi-Fi, viene schedulato anche un `NotificationWorker`. La durata minima della registrazione audio è pari a 2 secondi, in quanto, da prove, è emerso che il backend blocca audio più brevi.

Per l'ascolto dell'audio viene utilizzato il composable `AudioPlayer`.





Workers

WorkerManager

`WorkerManager` è un singleton che gestisce l'esecuzione e la supervisione di lavori in background tramite `WorkManager`. Consente di configurare e accodare lavori con dati di input, vincoli e tag, osservando il loro completamento. In caso di fallimento, i lavori possono essere messi in sospeso per essere ripresi successivamente. Il manager memorizza i lavori in attesa e permette di riprendere tutti i lavori interrotti tramite il metodo `resumeAll()`. È progettato per integrarsi con il ciclo di vita dell'app, gestendo i risultati tramite callback personalizzati per il successo o il fallimento dei lavori.

AuthedWorker

`AuthedWorker` è una classe astratta che estende `Worker` e si occupa di gestire lavori in background che richiedono un'autenticazione tramite token. La classe definisce un metodo astratto `doAuthedWork()` che deve essere implementato dalle classi concrete per eseguire operazioni specifiche. Il metodo `doWork()` verifica la validità del token tramite la funzione `verifyToken`, e se il token è valido, esegue il lavoro autenticato. In caso contrario, restituisce un risultato di fallimento con un dato che segnala la necessità di un nuovo tentativo. Il metodo `parseResult` interpreta il risultato della risposta HTTP, restituendo un `Result` di successo contente la risposta del server o di fallimento a seconda del codice di risposta.

UploadAudioWorker

`UploadAudioWorker` è una classe che estende `AuthedWorker` e gestisce l'upload di un file audio in background. Nel metodo `doAuthedWork()`, prima controlla se il file audio esiste nel percorso specificato. Se il file non esiste, restituisce un risultato di fallimento. Se il file è valido, esegue una richiesta HTTP POST all'endpoint "upload", inviando i parametri di latitudine, longitudine e il file audio stesso. La risposta della richiesta viene poi interpretata tramite il metodo `parseResult`. In base al codice di risposta, il risultato del lavoro è successivo o fallito.

Quando il lavoro viene schedulato in `AudioRecorderActivity`, viene impostato in modo che, qualora il processo termini con successo, venga eseguita la funzione `updateMetadataFromBackend`, così da rendere disponibili immediatamente i dati relativi all'audio appena caricato.

Questo worker viene eseguito solo quando ci si connette ad una rete Wi-Fi

DeleteAudioWorker

La classe `DeleteAudioWorker` estende `AuthedWorker` e gestisce l'eliminazione di un audio dal server. Riceve un ID audio tramite `inputData` e, in modalità sospesa, invia una richiesta `DELETE` all'endpoint `audio/<audioID>`. Il risultato della richiesta viene elaborato tramite `parseResult`. Questo worker ha come unico constraint l'essere connessi ad una qualsiasi rete.

EditPrivacyWorker

La classe `EditPrivacyWorker` estende `AuthedWorker` e gestisce la modifica della privacy di un audio. Riceve l'ID audio e la nuova privacy tramite `inputData`, quindi invia una richiesta GET all'endpoint `audio/my/<audioID>/<NewPrivacy>` per aggiornare la privacy. Il risultato della richiesta viene elaborato tramite `parseResult`. Come `DeleteAudioWorker` non richiede di essere connessi ad un tipo particolare di rete.

NotificationWorker

La classe `NotificationWorker` estende `Worker` e si occupa dell'invio di una notifica personalizzata all'utente, indicando il numero di audio che devono ancora essere inviati al backend. Il worker viene schedulato solo quando l'utente non è connesso a una rete Wi-Fi, per notificare l'aggiunta di nuovi audio alla coda. Prima di iniziare, elimina eventuali altri `NotificationWorker` in coda. Al termine con successo, imposta il numero di audio da caricare a 0 nelle SharedPreferences.

NetworkUtils

In `NetworkUtils.kt` viene definita l'interfaccia `ApiService`, dove sono specificati i metodi per eseguire le operazioni HTTP necessarie per comunicare con il backend. Tutti i metodi supportano l'autenticazione tramite token Bearer.
Nell'oggetto `NetworkUtils` vengono definiti i metodi `getRequest`, `postRequest`, `deleteRequest` che inviano le richieste HTTP usando `ApiService` e gestiscono le risposte. Se il login è richiesto, viene controllata la validità del token. Vengono anche definite le funzioni `jsonEncoding` e `wwwEncoding` per creare il corpo della richiesta rispettivamente in formato JSON o `application/x-www-form-urlencoded`. Tutte queste richieste vengono svolte in modo sincrono utilizzando `execute()`, che blocca il thread finché la risposta non è valida.