Nazareno Borello, Marco Galatola, Francesco Mecca A.A. 2019/2020

Relazione Progetto di Programmazione Mobile

APOLLON

Introduzione

Un numero sempre maggiore di persone si iscrive a servizi di streaming musicale, come Spotify ed Apple Music, per poter ascoltare i propri artisti preferiti in qualunque situazione.
Questi servizi sono in genere supportati da abbonamenti o pubblicità e, nonostante abbiano raggiunto una grande diffusione, sono molto criticate dagli appassionati, dai critici di musica e dagli artisti stessi.

Le critiche più diffuse sono le seguenti:

- **bassa qualità di riproduzione**: i file audio inviati in streaming da Spotify sono in formato mp3 128kpbs, 192kbps per gli utenti premium. Inoltre Spotify mente sull'encoding dei dati inviati (1);
- **censura**: entrambe le piattaforme praticano la censura, anche a fini politici (2);
- **catalogazione imprecisa**: Spotify ed Apple Music offrono nella maggior parte dei casi una sola versione di ogni disco. Inoltre se l'utente carica un mastering differente di un brano già esistente nel loro database, questo viene scartato;
- **assenza di artisti di nicchia**: i database di queste due grandi piattaforme, oltre ad essere prive di tutta la musica non distribuita da etichette riconosciute, offrono uno scarso catalogo per generi meno noti. Ne sono un'esempio i brani di musica classica, che sono solo l'1% del totale (3, 4).
- **pubblicazioni di dati falsi per motivi di PR**: Spotify ha mentito in varie occasioni riguardo al suo catalogo (5).
- **DRM**: il Digital Rights Management è una tecnologia che limita le possibilità di riproduzione del file su dispositivi non autorizzati

La nostra app si propone come un'alternativa per appassionati di musica e collezionisti, che non vogliono rinunciare allo streaming seppur continuando ad usufruire della propria collezione musicale di brani di nicchia ed in alta definizione, scaricati da piattaforme come Bandcamp o "rippati" dai CD audio.
Il nome, Apollon, è un omaggio alla divinità greca del sole e della musica.

Specifiche generali dell'applicazione

L'applicazione avrà come funzionalità principale quella di riprodurre la collezione musicale personale di ogni singolo utente, è possibile eseguire ricerche sulla propria libreria e creare playlist.
Lo streaming è permesso in tre preset di qualità, alto, medio e basso.
All'avvio si verrà connessi ad un server, programmato con le stesse tecnologie e pattern dell'applicazione, gestito da ogni utente. Un approfondimento sull'implementazione di quest'ultimo è offerta in appendice.
Per ora basti notare che è offerta la possibilità di gestire più utenti, autenticati tramite nickname e password.
Per ragioni di usabilità è previsto un layout alternativo per schermi più lunghi (ad esempio quelli provvisti di "notch"), l'app ha inoltre il completo supporto per quattro lingue: Inglese, Francese, Italiano e Swahili.

Alternative su play store

Oltre a piattaforme come Spotify e Apple Music che offrono lo streaming musicale ma con gli svantaggi di cui sopra, vi sono solamente due alternative per chi vuole usufruire dello streaming della propria collezione musicale.
La prima è subsonic, supportato da varie applicazioni (6, 7) ma che non supporta il database di mpd (8) (il più diffuso in ambito audiofilo) e non permette di fare seek della traccia audio.
La seconda alternativa è il già citato mpd, che offre una funzione di streaming originariamente pensato per le webradio. Vi sono app (9, 10, 11)) che supportano un buon numero di features per mpd ma non permettono di creare playlists o di gestire più utenti. La maggiore limitazione di mpd è che permette lo streaming di un solo brano alla volta qualsiasi sia il numero di ascoltatori.

Benefici dell'applicazione

Lo standard più diffuso per i sistemi di riproduzione di musica digitale tra la comunità di audiofili con un particolare interesse per le tecnologie è da anni costituito da sistemi Linux con kernel real time ed MPD come riproduttore.
La nostra applicazione ha come interesse primario quello di attrarre questa comunità permettendo da una parte l'integrazione con le tecnologie già in uso (MPD e Linux), dall'altra ampio margine di controllo sulla riproduzione dei file e la gestione della libreria.
Gli utenti possono condividere la loro vasta libreria musicale, sulla quale hanno pieno controllo con strumenti familiari, e fare streaming dei propri file audio in diversi formati in base alle esigenze e alla banda disponibile.
Il server con il quale l'applicazione comunica supporta la lettura dei metadati dei file e del database di MPD.

Implementazione e funzionalità

Schema di navigazione di base

All'avvio l'applicazione richiede l'inserimento delle credenziali di accesso al server (protocollo, URL, porta, username, password) da parte dell'utente. Tali credenziali vengono memorizzate per poter essere riutilizzate al prossimo avvio.
Una volta effettuato il login si potrà navigare la propria libreria musicale ed effettuare una ricerca tramite keyword sull'intero catalogo. La libreria è automaticamente organizzata in sezioni: artista, album e generi.
È anche possibile creare e gestire playlist personalizzate per ogni utente direttamente dall'applicazione.
La visualizzazione dei brani è arricchita dalle copertine degli album e dalle foto degli artisti, scaricate dinamicamente.
Una volta selezionata una canzone questa verrà messa in riproduzione. Dal player è possibile selezionare la qualità di riproduzione fra i tre preset citati prima, visualizzare le lyrics ed aggiungere o rimuovere il brano alla playlist "Preferiti".

Infine è possibile condividere la canzone in ascolto inviando un messaggio di testo opportunamente formattato con artista e titolo.
È abilitato l'uso di due gesture: il "fling" per navigare fra i brani di una playlist ed il doppio "tap" per aggiungere/rimuovere quello in riproduzione dai preferiti
Se si abbandona la schermata del player la riproduzione non viene interrotta e compare un mini player, provvisto di controlli e informazioni sul brano, nella parte bassa dell'interfaccia; al tocco si viene riportati al player intero.

Scelte tecniche

Riproduzione in streaming

La riproduzione in streaming presenta notevoli difficoltà se abbinata al transcoding on the fly, ovvero la riduzione di qualità di un brano in alta definizione mentre lo si sta ascoltando. La riproduzione di un sorgente tramite protocllo HTTP è fornita dalla classe MediaPlayer del framework Android. Durante la riproduzione la dimensione del file è in continua crescita poiché la dimensione del file sul server è proporzionale alla percentuale di file del quale è già stato compiuto il transcoding. Inoltre il transcoding effettuato tramite Variabile Bit Rate per definizione non permette di conoscere la dimensione finale del file. Per permettere all'utente di fare seek della traccia audio, Apollon interroga la classe MediaPlayer sulla quantità di buffer disponibile. Quando il seek non è disponibile per mancanza di buffer, la barra di riproduzione viene disabilitata ed il seek viene effettuato tramite una nuova richiesta al server (HTTP Accept-Ranges).

Interazione col dispositivo

All'avvio di un brano viene creata una notifica provvista di controlli per la riproduzione e contenente le informazioni della canzone, inclusa la copertina dell'album associato, visibile anche come sfondo della schermata di blocco.
Il volume del brano viene ridotto per qualche istante nel caso sopraggiunga una notifica mentre la riproduzione viene messa in pausa se vengono staccate le cuffie o se arriva una chiamata.

Servizi di terze parti

Le lyrics vengono fornite da ChartLyrics tramite api web in formato XML.
Le immagini sono invece fornite da WikiData e scaricate attraverso la libreria Picasso che si occupa di fare caching talora necessario.

Pattern architetturali

Nel codice risalta un notevole utilizzo di pattern architetturali, alcuni comuni a Java e Kotlin, altri specifici di Kotlin e della programmazione funzionale. Inoltre abbiamo seguito differenti tecniche di programmazione e riuso specifici del framework Android.
All'interno dei pattern tipici della programmazione ad oggetti compaiono:

- **Adapter**: usato per adattare l'interfaccia di una classe ad un'altra. Nel nostro caso è stato usato questo pattern per le classi che forniscono la logica e la visualizzazione di playlist e brani.

- **Builder**: pattern creazionale utilizzato per l'instanziazione step by step di oggetti complessi. Fra le varie classi in cui usiamo questo pattern, la più importante sono quelle per la gestione dei metadati del brano in riproduzione e per l'instanziazione del mediaplayer.

- **Event**: gli eventi sono oggetti instanziati in risposta a degli input esterni (quali rete o utente) e vengono gestiti in maniera asincrona da differenti funzioni. Vengono utilizzati nella nostra applicazione per gestire la UI e gli eventi di rete relativi alla riproduzione in streaming.

Kotlin ci ha permesso da un lato di usare con più facilità e in maniera meno verbosa dei pattern comuni nella programmazione ad oggetti, in particolare:

- **Singleton**: pattern che restringe l'instanziazione di una classe ad un solo oggetto. Utilizzata per la connessione al server e la gestione delle credenziali (che a sua volta è un adapter per SharedPreferences).

- **Companion Object**: un singleton che ha accesso ai membri privati degli oggetti instanziati dalla classe compagno (ovvero della classe all'interno del quale è dichiarato). Ne facciamo uso per la creazione delle instanze dei fragments.

- **Sum Types**: pattern tipicamente funzionale in cui una struttura dati può contenere diversi tipi predefiniti. Abbiamo modellato i sum types in Kotlin attraverso l'uso delle **sealed class** che ci ha permesso di codificare lo stato del programma attraverso i tipi.

- **Pattern Matching**: pattern estremamente correlato all'uso dei sum types. Attraverso il pattern matching siamo sicuri che in ogni branch del nostro codice abbiamo preso in considerazione ogni stato possibile. L'uso di assert(false) ci permette di asserire quali tipi non devono raggiungere quella sezione di codice.

Infine nella nostra codebase abbiamo fatto un uso estensivo delle componenti del framework Android, in particolare:

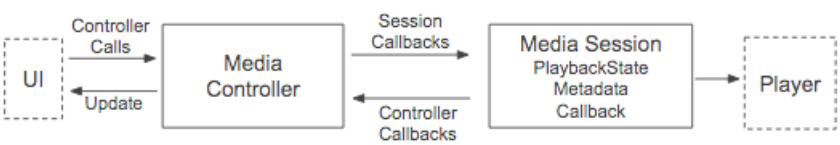
- **Fragments**: componenti che incapsulano il comportamento e l'interfaccia di una parte di applicazione. In questo modo abbiamo potuto utilizzare una sola activity che resta sempre in comunicazione con l'unico service presente.

- **Service**: componente utilizzata per riprodurre la musica anche quando l'applicazione è in background.

- **AsyncTasks**: usati per eseguire operazioni bloccanti, come la comunicazione col server, senza dover interrompere l'interazione dell'utente.

Architettura

L'architettura adottata è quella consigliata da Google per una media application, opportunamente adattata alle nostre esigenze.



Player

Collocato all'interno del service così che possa restare sempre operativo, per evitare uno spreco di risorse, il player viene deallocato se inutilizzato per un lasso di tempo prestabilibito.

Media Session

La media session serve ad esporre le informazioni sullo stato del player (PlaybackState) o sul brano in riproduzione (Metadata), permette inoltre di comunicare con il controller tramite un paradigma ad eventi (Callback).

Media Controller

Il media controller contiene tutte le operazioni necessarie per controllare la riproduzione (play, pausa, ecc.), ogni operazione viene segnalata alla session tramite callback. Viene creato nel service e poi riferito nella main activity.

UI

La UI è interamente gestita da fragment, tutti istanziati dalla medesima activity. Questi usano gli input dell'utente per modificare lo stato del player usando il controller e, alla ricezione degli opportuni callback, modificano l'iterfaccia.

Notification Manager

Il notification manager si occupa della costruzione delle notifiche, che avviene al cambio dei metadata e dello stato del player.
Abbiamo usato uno stile predefinito per la presentazione di media riproducibili che include i metadata (immagini comprese) ed alcuni controlli; la notifica permette il controllo della riproduzione anche dalla schermata di sblocco schermo.
Se si tocca la notifica si viene riportati alla schermata di riproduzione.

Audio Manager

L'audio manager è una componente essenziale per gestire l'audio, evitandone la sovrapposizione con altre applicazioni. Nella fattispecie la riproduzione è messa in pausa alla perdita del "focus", per esempio allo scollegamento di un jack od al sopraggiungere di una chiamata, mentre viene abbassato il volume in caso di arrivo di una notifica [?].

Server Bridge

Il server bridge è la componente che si occupa della comunicazione col server. Contiene un oggetto singleton Server, il cui compito è esporre le funzioni di comunicazione ai fragment e mantenere una cache dei risultati ricevuti.
La comunicazione si basa sull'uso di AsyncTask ai quali viene passato un apposito TaskListener, chiamato asincronicamente ad operazione conclusa. Il listener riceve un TaskResult (specifico per l'operazione) atto a modellare il risultato richiesto.

Backend

Apollon richiede all'utente di installare un backend sul server in cui è presente la libreria musicale, questo utilizza le stesse tecnologie di Apollon (Kotlin) e condivide un'architettura simile.
Il backend richiede all'utente di dichiarare in un file di configurazione (in formato json) gli utenti che utilizzeranno il servizio, identificati da una coppia nome-chiave.
Una volta avviato, si occuperà di leggere e processare il database di MPD, software molto diffuso nelle comunità audiofile per la gestione del proprio catalogo musicale.
La comunicazione con l'app avviene attraverso l'utilizzo di una API REST-like che permette differenti operazioni:

- Streaming di flusso dati:
  - Play: inizia la conversione di un brano e offri lo streaming di un flusso di dati
  - Play-ranged: permette lo streaming di un flusso di dati che inizia dalla percentuale del brano specificata
  - Stop: interrompe la conversione e permette al server di liberare gli handle relativi al flusso audio, se necessario
  - Query: permette di ricevere informazioni sullo stato della conversione. Utilizzato per l'euristica relativa al seek del brano
- Indicizzazione del catalogo:
  - List-songs: permette all'app di richiedere la lista di tutte le canzoni
  - List-artists: permette all'app di richiedere la lista di tutti gli artisti
  - List-albums: permette all'app di richiedere la lista di tutti gli album
  - List-genres: permette all'app di richiedere la lista di tutti i generi
- Query degli oggetti del catalogo:
  - Single-artist: permette all'app di richiedere i brani e gli album di un singolo artista
  - Single-album: permette all'app di richiedere i brani di un singolo album e i metadati dell'album
  - Single-song: permette all'app di richiedere i metadati del brano, che il backend elabora leggendo il file sul disco
- Gestione delle playlists:
  - create: permette all'app di creare una nuova playlist
  - rename: Permette all'app di cambiare titolo ad una playlist
  - get: permette all'app di richiedere una playlist sotto forma di dizionario
  - list-all: permette all'app di richiedere una lista di tutte le playlists presenti sul backend, relative ad un singolo utente
  - add-songs-from: permette di aggiungere dei brani ad una playlist esistente
  - remove-songs-from: permette di rimuovere dei brani da una playlist esistente

Sono inoltre mantenute alcune informazioni relative alle cover art e alle lyric, qualora disponibili.

Kotlin si è rivelato un ottimo linguaggio per sviluppare un backend che espone un api in rete.
Questo perché oltre all'ottima performance della JVM, permette l'utilizzo di uno stile più funzionale rispetto a Java, e quindi tramite tecniche come il pattern matching e l'utilizzo di tipi algebraici (modellati usando le `sealed class`) si può esprimere più facilmente logica di alto livello. Inoltre sono disponibili ottimi framework per la creazione di servizi asincroni/reattivi. Nel nostro caso la scelta è ricaduta su Vertx.
Il codice è disponibile al seguente indirizzo.