UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA TRIENNALE IN INFORMATICA MUSICALE

PLAYER AUDIO ANDROID PER LO STUDIO DELLA MUSICA

Relatore: Prof. Adriano Baratè
Correlatore: Prof. Luca Andrea Ludovico

Tesi di Laurea di:
Misuraca Francesco
Matr. Nr. 855470

ANNO ACCADEMICO 2016-2017

Dedicato a mio nonno Francesco

Prefazione

Android è la piattaforma mobile più diffusa al mondo, con più di un miliardo di device attivi e un ritmo di crescita rapidissimo. Considerando l'ultimo trimestre 2016 e il primo del 2017, Android conquista il 71,6% di market share italiano, un numero tuttavia nettamente più basso rispetto alle quote globali che si attestano sull'85% *(dati IDC)*. Non stupisce la scesa del mercato dell'est che padroneggia la produzione e la diffusione stando a quanto certificato da *TrendForce*. La casa sudcoreana è riuscita a posizionarsi al primo posto per quanto riguarda la classifica relativa ai principali produttori di smartphone al mondo. Il trimestre 2017 di *Samsung* è iniziato in netta crescita nonostante il problema del *Note 7*, e ha portato al raggiungimento del 26.1% di market share, sottraendo il primo posto ad *Apple* sempre e comunque in crescita; *IDC* stima che, entro il 2021, le spedizioni di smartphone Android continueranno a salire fino a raggiungere 1,5 miliardi di dispositivi in commercio.
Ma perché Android? quali sono i fattori di questo successo e soprattutto quali sono le principali caratteristiche che lo rendono così popolare?. Android è il sistema operativo per dispositivi mobili sviluppato da Google presentato nell'autunno del 2007. La versione presentata questo 21 agosto 2017 è l'ottava realise della sua storia e prende il nome di Android Oreo. Una particolarità da notare fin da subito, è che le versioni Android sono abbastanza semplici da ricordare, in quanto procedono in nome alfabetico e perchè vengono curiosamente identificate con nomi di dolci dalla versione 1.5. La peculiarità di questo sistema sta nella sua politica di licenza di tipo open source basato su kernel Linux. La licenza sotto cui è distribuito consente di modificare e distribuire liberamente il codice sorgente e questo favorisce una vasta comunità di sviluppatori che realizzano applicazioni aumentando sempre di più le funzionalità del sistema. Sicuramente un qualcosa che lo ha contraddistinto e l'ha reso speciale è stato l'adottare dispositivi molto diversi tra loro, non solo per tipologia quindi smartphone piuttosto che tablet, ma soprattutto per fasce di prezzo, da poche decine di euro fino a cifre piuttosto significative.

Organizzazione della tesi

La tesi è organizzata come segue:

- nel Capitolo 1 vengono introdotti i principali strumenti utilizzati e le nozioni di base per la realizzazione dell'applicativo: dalla piattaforma Android Studio alla creazione dell'icona, dal Model-View-Controller al ciclo di vita di un'Activity fino ad arrivare al ruolo dei Fragments e Adapters nelle moderne applicazioni.

- nel Capitolo 2 viene analizzata la struttura fondamentale dell'applicativo utilizzando un'approccio top-down: viene formulata inizialmente una visione generale del sistema descrivendo le finalità principali e i layout più rilevanti per poi scendere nel dettaglio degli elementi impiegati.

- nel Capitolo 3 vengono illustrati tutti i metodi per l'interrogazione dei contenuti multimediali presenti nel dispositivo e le principali classi realizzate per l'audio.

- nel Capitolo 4 viene descritta la classe MediaPlayer utilizzata per la riproduzione e gestione audio dell'applicativo. Nello specifico vengono approfonditi gli stati di controllo dell'oggetto MediaPlayer e i comandi di base che compongono il player audio.

- il Capitolo 5 è dedicato interamente alla manipolazione e controllo audio: registratore, equalizzatore, regolazione del volume, alterazioni di pitch-tempo e la funzione di loop.

Indice

Capitolo 1

Introduzione e concetti di base

1.1 Stato dell'arte

Lo scopo principale di questa tesi è stato quello di acquisire competenze riguardo la programmazione di dispositivi mobile approfondendo in particolare gli aspetti musicali e gli elementi implementativi per lo studio e la manipolazione audio. Ascoltare musica è diventato ormai di uso comune. I servizi di streaming musicali offrono cataloghi che per la maggior parte degli utenti includono la quasi totalità delle canzoni e dei dischi che vogliono ascoltare. In commercio esistono numerosi servizi di streaming musicale come Spotify, Google Play Music o Apple Music che permettono periodi di prova gratuita e offrono la possibilità di effettuare abbonamenti anche per ascoltare musica offline. Spesso i servizi di straming hanno un prezzo contenuto ma a lungo andare possono diventare dispendiosi per gli utenti. I player audio sono un buon compromesso per ascoltare musica senza abbonarsi a nessun servizio. Avere sempre a disposizione le proprie playlist è un fattore fondamentale per la vita di tutti giorni soprattutto quando si ha la necessità di limitare il consumo di internet e ancor di più quando ci si trova in luoghi a bassa connettività. I principali lettori audio Android, come The Music Player, XenoAmp Music Player o Phonograph, spesso sono limitati alle funzioni basilari di creazione di playlist e ascolto dei propri brani. Questo è stato uno dei motivi che mi ha spinto a implementare funzionalità più originali e utili per gli utenti, quali: registrazione audio e voce integrate, equalizzazione personalizzabile e funzioni di alterazione come per esempio il cambio di tonalità e di tempo. Lo scopo è stato quello di arricchire e rendere più pratica la manipolazione e l'usabilità audio agli utenti. L'ultimo capitolo del testo è dedicato interamente alla descrizione di queste funzionalità.

1.2 Strumenti utilizzati

Prima di entrare nello specifico dell'implementazione è bene mostrare i componenti necessari per lo sviluppo dell'applicativo.

1.2.1 Android SDK

Android SDK (Software Development Kit) è un pacchetto di sviluppo per applicazioni che contiene un set completo di API (Application Programming Interface), librerie e strumenti flessibili di compilazione, test e debug. I linguaggi di programmazione alla base della maggior parte delle applicazioni studiate per Android sono Java e C/C++. Con l'SDK, scaricabile dal sito ufficiale https://developer.android.com/sdk, sarà quindi possibile disporre di un'interfaccia in grado creare e modificare le applicazioni Android. Inoltre l'SDK supporta anche le versioni precedenti di Android nel caso in cui gli sviluppatori desiderassero di eseguire le loro applicazioni su dispositivi più vecchi. Gli strumenti di sviluppo e l'ultima versione dell'ambiente della piattaforma sono componenti scaricabili. Per quelli meno recenti si può eseguire il test di compatibilità. È anche possibile ottenere i driver necessari per interfacciare il dispositivo al pc, per poter compilare ed eseguire l'applicazione direttamente su smartphone senza dover usare l'emulatore dell'SDK. Le applicazioni Android hanno estensione .apk e vengono memorizzate nella cartella /data/app del dispositivo.

1.2.2 Android Studio

Android Studio è l'ambiente di sviluppo integrato ufficiale di Google nato appositamente per sviluppo Android, basato su IntelliJ IDEA https://www.jetbrains.com/idea liberamente disponibile per Windows, Mac OS X e Linux, sotto licenza Apache 2.0. Un editor intelligente capace di offrire completamento avanzato del codice, refactoring, e analisi. Grazie a queste caratteristiche rendono lo sviluppatore più produttivo e veloce nel completamento dell'applicazione. Una delle caratteristiche principali è la possibilità di avviare nuovi progetti utilizzando template messi a disposizione da Android Studio. Inoltre è possibile configurare l'ambiente con GitHub https://github.com in modo da importare codice dal canale di Google oppure esportare e rendere pubblico il proprio progetto. Con Android Studio risulta molto semplice sviluppare applicazioni multi-schermo (smartphone Android, tablet, Android Wear, TV Android, Android Auto e Google Glass) poiché è integrato un modulo capace di facilitare la gestione delle risorse e delle risoluzioni. Infatti ad ogni esecuzione/debug dell'applicazione è possibile scegliere il profilo del dispositivo Android da emulare tra quelli più comuni. Uno dei punti di forza di Android Studio è la possibilità di creare APK multipli

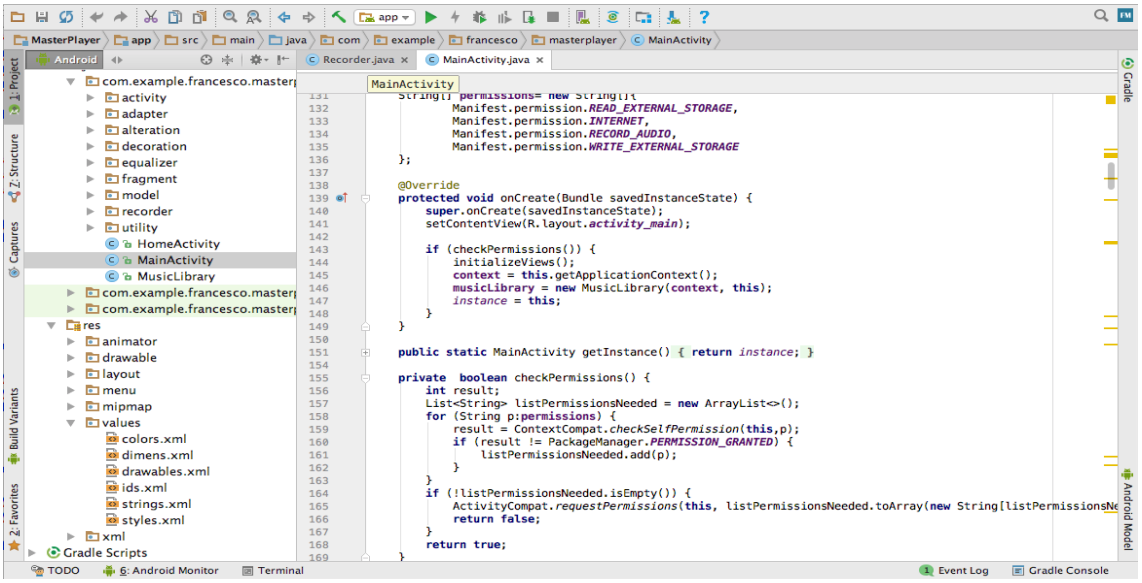


Figura 1: Schermata di sviluppo di Android Studio

dell'applicazione con caratteristiche magari differenti specificate nel file build.gradle del progetto. Il **Gradle** è uno strumento molto flessibile di build automation che permette di integrare librerie di sviluppo prodotte da Android o da sviluppatori di terze parti. Per questo software viene introdotto un Domain-Specific Language (DSL) basato su Groovy che sostituisce la tradizionale modalità XML usata per dichiarare la configurazione del progetto. Gradle https://gradle.org/getting-started-android è stato progettato per sviluppi multi-progetto che possono crescere fino a divenire abbastanza grandi e supporta sviluppi incrementali determinando in modo intelligente quali parti del build tree sono aggiornate (up-to-date), in modo che tutti i processi che dipendono solo da quelle parti non avranno bisogno di essere rieseguiti; In questo modo, il software riduce significativamente il tempo di building del progetto, in quanto, durante il nuovo tentativo di costruzione, verranno eseguite solo le attività il cui codice è effettivamente stato alterato a partire dall'ultima costruzione completata. Gradle supporta anche la costruzione del progetto per processi concorrenti, il che consente di svolgere alcuni compiti durante la costruzione (ad esempio, i test automatizzati attraverso gli unit test), eseguiti in parallelo su più core della medesima CPU, su più CPU o su più computer.

1.2.3 Material Design Icons

Nonostante Android Studio includa uno strumento chiamato Image Asset Studio che aiuta a generare icone di avvio e di notifica posizionandole dopo la creazione nella specifica cartella *res* del progetto, per la creazione dell'icona principale e per tutte le icone presenti all'interno dell'applicativo è stato utilizzato il seguente sito https://materialdesignicons.com. Material Design Icon è un semplice strumento che genera un set di icone alla risoluzione appropriata per ogni densità di schermo supportata dall'applicazione. Come si vede nell'immagine seguente per la creazione dell'icona di avvio si è scelto l'icona *music-circle* presente nel catalogo di Material Design e cliccando su Advanced export, è stato possibile personalizzarla in modo semplice e intuitivo. Vengono di seguito descritti brevemente gli strumenti presenti per la personalizzazione di un'icona in Material Design:

- **Colors:** è possibile cambiare il colore di base dell'icona sia del background che del foreground. Inoltre si ha la possibilità di inserire manualmente il codice colore che si desidera e personalizzare ancora di più la propria icona.
- **Size:** scegliere la dimensione di un'icona è sempre un processo delicato. Questo strumento ti permette di adattare diverse icone in termini di dimensione con layout identico e risolvere quei problemi abituali di risoluzione che possono presentarsi.
- **Padding:** è anche possibile impostare il padding della propria icona.
- **Corner radius:** è possibile dare una forma completamente circolare oppure quadratica, o anche impostare la forma manualmente. Infine è possibile esportare la propria icona appena personalizzata in formato PNG oppure scegliendo tra i diversi formati presenti: XML, SVG e XAML.

Per l'applicazione si sono combinate insieme le funzioni di padding e corner radius per dare un'effetto vinile abbinando una tonalità di grigio al background e un codice colore arancio al foreground.

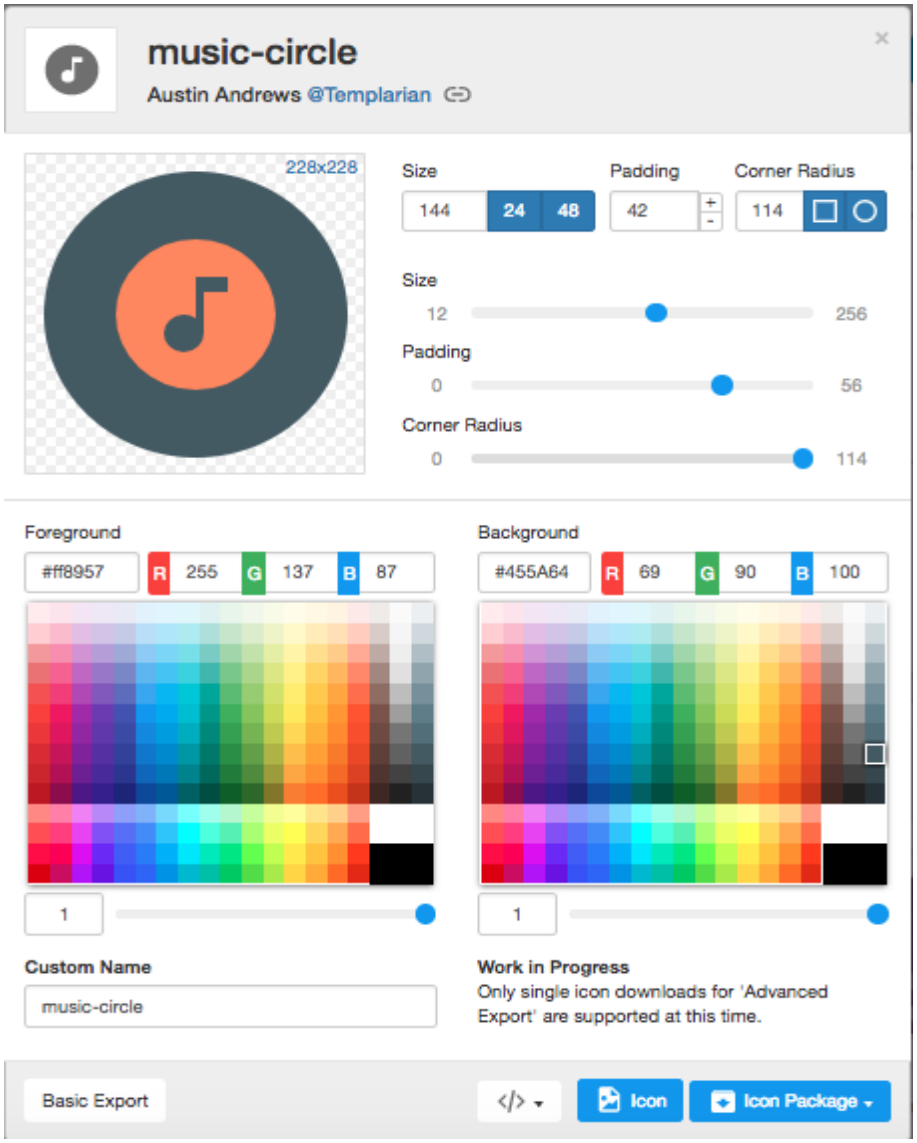


Figura 2: Finestra Advanced Export per la creazione dell'icona di avvio.

1.3 Il Model-View-Controller

Il pattern architetturale usato è il Model View Controller (MVC) uno dei più diffusi nella programmazione object oriented e nello sviluppo di interfacce grafiche in quanto rappresenta uno dei concetti fondamentali della programmazione ad oggetti e permette di strutturare l'applicazione in maniera molto efficiente. Inoltre aumenta la coesione del nostro sistema software, in quanto ogni singolo oggetto può ricoprire solo uno dei seguenti ruoli: modello, vista o controllore. Questi ruoli rappresentano una sorta di classificazione dell'oggetto che stiamo utilizzando e risultano dunque essere elementi separati ai quali però è consentita una stretta comunicazione. Il grafico in figura 3 illustra lo schema di design del pattern utilizzato.

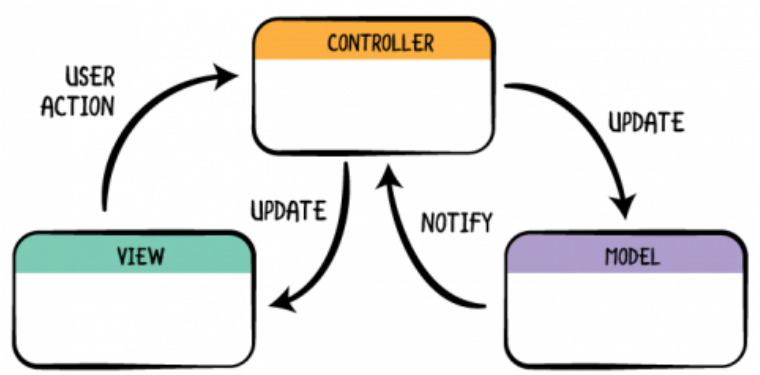


Figura 3: Schema design pattern MVC.

Il pattern non solo stabilisce che ruolo deve avere un determinato oggetto all'interno dell'applicazione, ma anche il modo in cui gli oggetti comunicano tra di loro. Analizziamo ora i tipi di oggetti che il pattern definisce:

- **Model:** memorizza le strutture dati specifiche dell'applicazione e si occupa di definire tutte le varie operazioni che effettuano la manipolazione dei dati stessi, in lettura e scrittura. Si occupa di creare oggetti in memoria persistente, serializzare e deserializzare i dati per l'invio e ricezione via rete. Il modello non può avere connessione diretta con un oggetto di tipo view, in quanto ha il compito di gestire i dati che non devono essere legati ad un particolare tipo di visualizzazione.

- **View:** il ruolo della vista è quello di presentare all'utente i dati contenuti all'interno del Model e di permettergli di interagire con essi. Concettualmente il Model è un oggetto non concreto, mentre la vista è un oggetto concreto con il quale l'utente può interagire. Esempi di classi che compongono la View sono i pulsanti, le liste, le label e tanti altri, e si possono utilizzare così come sono

proposti oppure estenderli. La vista mette a disposizione un'interfaccia per la modifica dei dati contenuti nel modello. L'oggetto di tipo vista non deve avere un riferimento esplicito ad un oggetto di tipo modello e quindi a questo punto viene usato l'oggetto controllore.

- **Controller:** svolge la funzione di intermediario tra oggetti di tipo View ed oggetti di tipo Model. Gestisce il ciclo di vita dell'applicazione, gli eventi generati dalle View e modifica di conseguenza il Model. Un singolo controllore può avere un numero di relazioni arbitrarie tra oggetti di tipo modello e vista, che possono essere relazioni uno ad uno o molti a molti. Il controllore si occupa di inizializzare la vista con i dati contenenti nel modello e informarla sulle modifiche dei dati subite dall'utente.

In teoria sarebbe auspicabile avere una separazione netta tra le componenti MVC. Tuttavia nella pratica accade di avere oggetti che combinino due comportamenti. In breve, il Model gestisce direttamente i dati, la logica e le regole dell'applicazione. Una View può essere una qualsiasi rappresentazione in output di informazioni, come un grafico o un diagramma. Il Controller, accetta l'input e lo converte in comandi per il modello e/o la vista;

1.4 Componenti di base di un'applicazione

Ogni applicazione Android, indipendentemente dalla finalità che si prefigge, affida le sue funzionalità a quattro tipi di componenti. Si tratta di Activity, Service, Content Provider e BroadcastReceiver ed esistono affinchè la nostra applicazione possa integrarsi alla perfezione nell'ecosistema Android. Molto brevemente verrano esaminati questi concetti approfondendo successivamente quelli di Activity Lifecycle e Intent:

- **Activity:** un'Activity e un'interfaccia utente. Ogni volta che si usa un'app generalmente si interagisce con una o più "pagine" mediante le quali si consultano dati o si immettono input.

- **Service:** un Service svolge un ruolo, se vogliamo, opposto all'Activity. Infatti rappresenta un lavoro generalmente lungo e continuato, che viene svolto interamente in background senza bisogno di interazione diretta con l'utente. I Service hanno un'importanza basilare nella programmazione proprio perchè spesso preparano i dati che le Activity devono mostrare all'utente permettendo una reattività maggiore nel momento della visualizzazione. Ogni classe di service deve avere una corrispondente dichiarazione <service> nel suo pacchetto AndroidManifest.xml. I servizi possono essere avviati con Context.startService() e Context.bindService().

- **Content Provider:** nasce con lo scopo della condivisione di dati tra applicazioni. Questi componenti permettono di condividere, nell'ambito del sistema, contenuti custoditi in un database, su file o reperibili mediante accessi in Rete.
- **Broadcast Receiver:** componente che reagisce ad un invio di messaggi a livello di sistema, appunto in broadcast, con cui Android notifica l'avvenimento di un determinato evento, ad esempio l'arrivo di un SMS o di una chiamata o sollecita l'esecuzione di azioni.
- **Intent:** un intent è una descrizione astratta di un'operazione da eseguire. Può essere utilizzato con startActivity per avviare un'attività, broadcastIntent per inviarlo a qualsiasi componente BroadcastReceiver. Il suo uso più significativo è il lancio di attività, dove può essere pensato come la colla tra esse.

1.4.1 Activity

Un'activity è essenzialmente una finestra che contiene l'interfaccia utente di un'applicazione ed il suo scopo è quello di permettere un'interazione con gli utenti. Quindi la classe Activity si occupa di creare una finestra in cui è possibile posizionare l'interfaccia utente con setContentView(View). Un'applicazione può avere zero o più activity, anche se solitamente almeno una è presente. Dal momento in cui un'activity compare sullo schermo al momento in cui scompare essa passa attraverso una serie di stati, il cosiddetto ciclo di vita (Activity Lifecycle). In particolare, nell'illustrazione riportata, gli stati sono rappresentati dalle figure colorate. L'ingresso o l'uscita da uno di questi stati viene notificato con l'invocazione di un metodo di callback da parte del sistema.

Quando un'activity va in esecuzione per interagire direttamente con l'utente vengono obbligatoriamente invocati tre metodi:

- **onCreate:** l'activity viene creata. Il programmatore deve assegnare le configurazioni di base e definire quale sarà il layout dell'interfaccia;
- **onStart:** l'activity diventa visibile. È il momento in cui si possono attivare funzionalità e servizi che devono offrire informazioni all'utente;
- **onResume:** l'activity diventa la destinataria di tutti gli input dell'utente.

Android pone a riposo l'activity nel momento in cui l'utente sposta la sua attenzione su un'altra attività del sistema, ad esempio apre un'applicazione diversa, riceve una telefonata o semplicemente anche nell'ambito della stessa applicazione, viene attivata un'altra activity.

Anche questo percorso, passa per tre metodi di callback:

- **onPause:** (l'inverso di onResume) notifica la cessata interazione dell'utente con l'activity;
- **onStop:** (contraltare di onStart) segna la fine della visibilità dell'activity;
- **onDestroy:** (contrapposto a onCreate) segna la distruzione dell'activity.

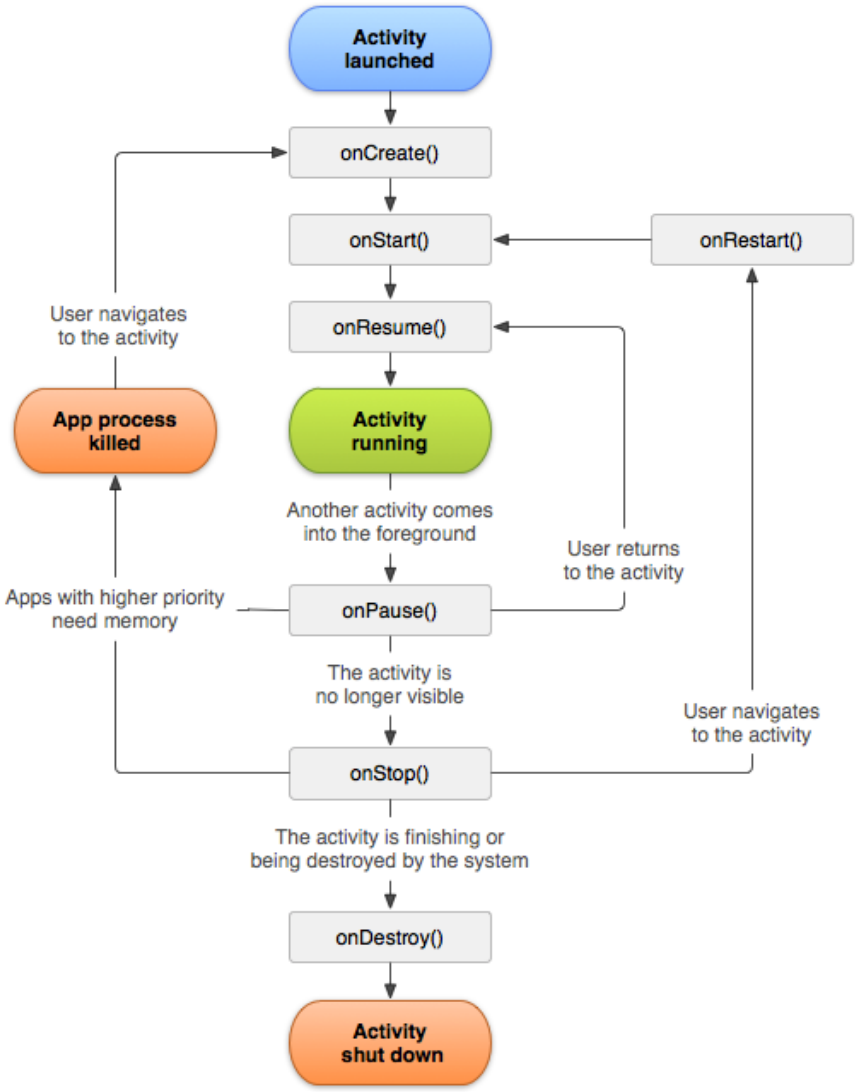


Figura 4: Activity Lifecycle

1.4.2 Fragment

Un fragment rappresenta un comportamento o una parte dell'interfaccia utente in un'activity. È possibile combinare più fragments in un'unica activity per creare un'interfaccia utente a più riquadri e riutilizzare un fragment in più activity. Si può dunque pensare un fragment come a una sezione modulare di un'activity, che ha il suo ciclo di vita e riceve i propri eventi di input che può aggiungere o rimuovere mentre l'activity è in esecuzione. Un fragment deve sempre essere incorporato all'interno di un'activity e il ciclo di vita di esso è direttamente influenzato dal ciclo di vita dell'activity. Ad esempio, quando l'activity è in pausa, lo sono anche tutti i fragments e quando l'activity viene distrutta, lo sono anche tutti i fragments. Tuttavia, mentre un'activity è in esecuzione è possibile manipolare ciascun fragment in modo indipendente, ad esempio aggiungendolo o rimuovendolo. È possibile inserire un fragment dichiarandolo come elemento <fragment> o dal codice dell'applicazione aggiungendolo a un ViewGroup esistente. Tuttavia, non è richiesto che un fragment faccia parte del layout dell'activity; Android ha introdotto i fragmets principalmente per supportare progetti UI più dinamici e flessibili su schermi di grandi dimensioni, come i tablet. Poiché lo schermo di un tablet è molto più grande di quello di un telefono, c'è più spazio per combinare e scambiare i componenti dell'interfaccia utente. È necessario progettare ciascun fragment come un componente di attività modulare e riutilizzabile. Questo è particolarmente importante perchè un fragment modulare consente di modificare le combinazioni di fragments per diverse dimensioni dello schermo.

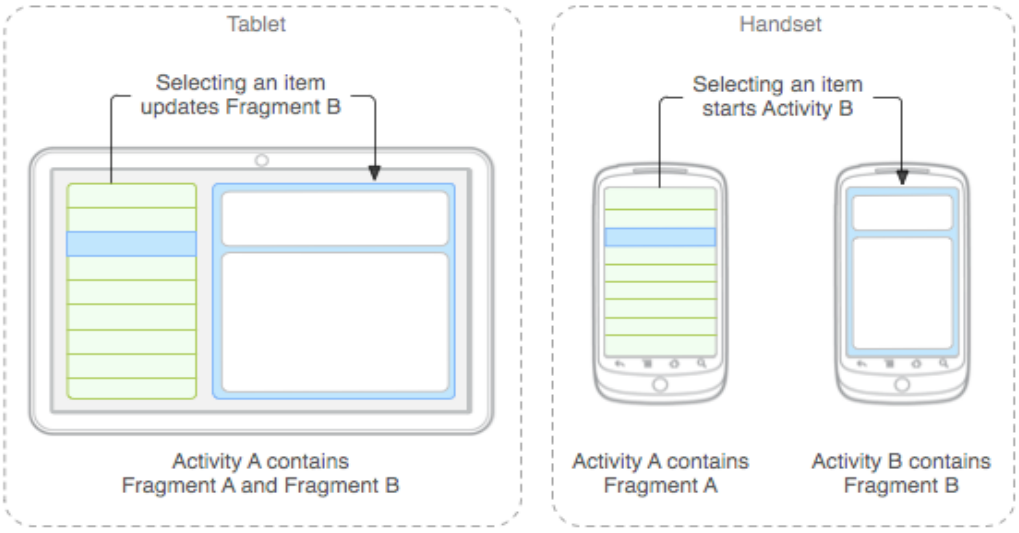


Figura 5: Esempio di fragments per diverse configurazioni dello schermo.

1.4.3 Adapter

Gli adapters sono elementi del controller che permettono di definire view dinamiche rispetto ai dati da mostrare. Essi sono dei componenti che collegano strutture dati di oggetti Java come array, collections o risultati di query, e incapsulano il meccanismo di trasformazione di questi oggetti in altrettante View da mostrare sui layout.
L'AdapterView invece, è un componente visuale che è collegato ad un adapter e raccoglie tutte le View prodotte dall'adapter per mostrarle secondo le sue politiche.
Gli adapter quindi collegano la View con il Model e definiscono quali e come gli elementi devono essere mostrati. L'esempio tipico viene fatto usando il più immediato degli adapters: l'ArrayAdapter; L'ArrayAdapter è un'adapter predefinito per la presentazione dei dati contenuti in un array. Un'altro tipo di adapter predefinito è il SimpleCursorAdapter che permette la presentazione dei dati contenuti mediante un Cursor. Come si può vedere in figura, esempi classici di AdapterView sono le ListView e le GridView. Nel primo caso solitamente abbiamo un array di oggetti String che iterativamente mostra tutte le stringhe disposte in righe. Nel secondo caso invece disponiamo la View per griglia;

Gli adapters sono un buon esempio di MVC: l'adapter è un elemento del Controller che collega un elemento di View (es: ListView) con un elemento di Model (es: un ArrayList). La parte di View è composta da due elementi: l'oggetto contenitore (es: la lista) e gli oggetti contenuti (es: gli elementi della lista); Viene riportato a titolo di esempio il collegamento dinamico tra struttura dati/Adapter/Listview che viene realizzato nell'onCreate dell'activity:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    String[] lettere = new String[]{"A","B","C","D","E"};
    ArrayAdapter<String> adapter =
            new ArrayAdapter<String>(this, R.layout.row,lettere);
    ListView listView = (ListView) findViewById(R.id.listview);
    listView.setAdapter(adapter);
}
```

Capitolo 2

Struttura dell'applicazione

La struttura grafica di un'activity prende il nome di **Layout**. Le interfacce utente in Android possono essere create in modo procedurale o dichiarativo. Nel primo caso si intende l'implementazione dell'interfaccia grafica nel codice: scriviamo il codice Java per creare e manipolare tutti gli oggetti dell'interfaccia utente. Nel secondo caso invece, non richiede la scrittura di codice ma descriviamo cosa vogliamo vedere nella pagina un po' come le pagine web statiche HTML.
Android permette la creazione di interfacce sia procedurali sia dichiarative: possiamo creare un'interfaccia utente completamente in codice Java (metodo procedurale) oppure possiamo creare l'interfaccia utente attraverso un descrittore XML (metodo dichiarativo). Android inoltre permette anche un approccio ibrido, in cui si crea un'interfaccia in modo dichiarativo e la si controlla e specifica in modo procedurale richiamando il descrittore XML. In questo applicativo sono stati utilizzati entrambi i metodi. Nel capitolo vengono illustrati le parti principali dell'interfaccia grafica.

2.1 Splash Screen

La prima pagina che viene mostrata all'utente una volta avviata l'applicazione è SplashActivity. Questa pagina ha due obiettivi principali: in primo luogo permette di non appesantire il caricamento complessivo dell'applicativo eseguendo poche righe di codice e lasciando caricare in background il MainActivity (molto più pesante); In secondo luogo ha lo scopo di presentare l'applicativo grazie ad un semplice layout.
Nel layout è presente una semplice TextView che mostra il nome dell'app e un'ImageView che contiene l'icona. Inoltre all'avvio dell'app sono presenti due animazioni: la prima è un'animazione di tipo dissolvenza su entrambi gli elementi, mentre la seconda è un effetto rotate solo sull'ImageView.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
        <ImageView
            android:id="@+id/spash_app"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@mipmap/icon9"
            android:layout_centerInParent="true"
            android:contentDescription="@string/desc" />

        <TextView
            android:id="@+id/spash_name"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="@string/app_name"
            android:textSize="26sp"
            android:gravity="top"
            android:paddingTop="15dp"
            android:layout_below="@+id/spash_app"
            android:textAlignment="center"
            android:textColor="@color/grey_app"
            android:textStyle="bold"/>
</RelativeLayout>
```

Viene riportata successivamente la classe SplashActivity nella sua interezza. Come si vede vengono identificati gli elementi di layout sopracitati, vengono aggiunte le animazioni e viene istanziato un oggetto Thread che ritarda il passaggio al MainActivity attraverso un intent per circa 1 secondo.

```
public class SplashActivity extends AppCompatActivity {

    TextView tv;
    ImageView iv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        tv = (TextView) findViewById(R.id.spash_name);
        iv = (ImageView) findViewById(R.id.spash_app);

        //animation
        Animation animation = AnimationUtils.loadAnimation
                (getBaseContext(), R.anim.splash);
        tv.startAnimation(animation);
        iv.startAnimation(animation);

        //rotating animation
        Animation rotation = AnimationUtils.loadAnimation
                (getBaseContext(), R.anim.rotate);
        iv.startAnimation(rotation);

        final Intent i = new Intent(this, MainActivity.class);
        Thread timer = new Thread(){
            public void run() {
                try {
                    sleep(800);
                } catch(InterruptedException e){
                    e.printStackTrace();
                } finally {
                    startActivity(i);
                    finish();
                }
            }
        };

        timer.start();
    }
}
```

2.2 System Permissions

Poiché ogni app Android funziona in una sandbox di processo, le app devono richiedere esplicitamente l'accesso a risorse e dati al di fuori della loro sandbox. Il sistema può concedere l'autorizzazione automaticamente o può richiedere all'utente di approvare o rifiutare la richiesta. La seguente finestra appare solo al primo accesso dell'utente dopo aver scaricato l'applicazione. L'utente dovrà consentire l'accesso alle risorse per utilizzare l'applicativo. Qualora non consentisse, l'applicativo viene interrotto fino quando l'utente non autorizza l'accesso.

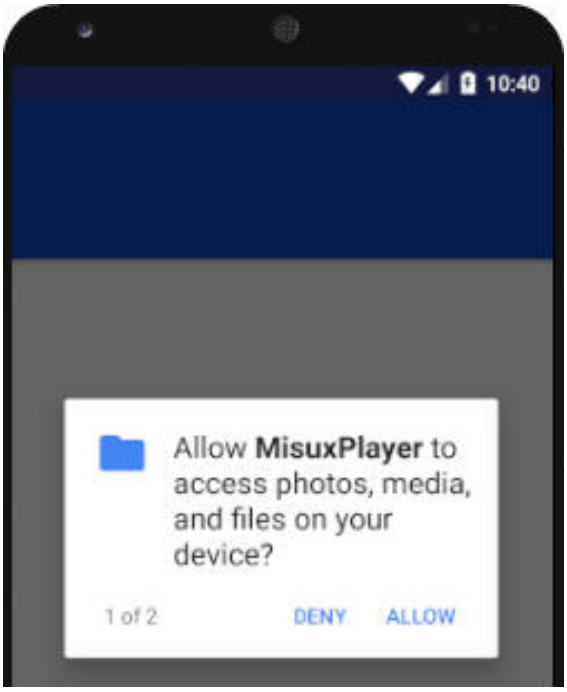


Figura 6: Viene utilizzata una lista di permessi

Le istruzioni seguenti definiscono alcuni dei permessi presenti nel Manifest utilizzati per questo applicativo. Nell'ultima riga viene mostrata la sintassi tipica per definire un permesso.

```
android.permission.INTERNET
android.permission.RECORD_AUDIO
android.permission.READ_EXTERNAL_STORAGE
android.permission.WRITE_EXTERNAL_STORAGE
android.provider.Settings.ACTION_MANAGE_WRITE_SETTINGS
```

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

2.3 ActionBar

L'ActionBar può essere definita come una "cornice" destinata ad ospitare tutte le opzioni di navigazione e di interazione più comuni dell'intera applicazione. In particolari l'ActionBar dell'applicativo contiene: una navigazione a scorrimento laterale, un'icona di ricerca e una navigazione con menu a tendina con diverse opzioni utili. Di seguito verranno analizzati alcuni di questi componenti.

2.3.1 NavigationView

Il Navigation Drawer è un pannello a scorrimento laterale per la navigazione dell'applicazione. La sua presenza si rende utile quando la gerarchia di contenuti che popolano l'app è strutturata su più livelli semplificando il passaggio tra activity. La figura seguente mostra il pannello di navigazione dell'applicazione.

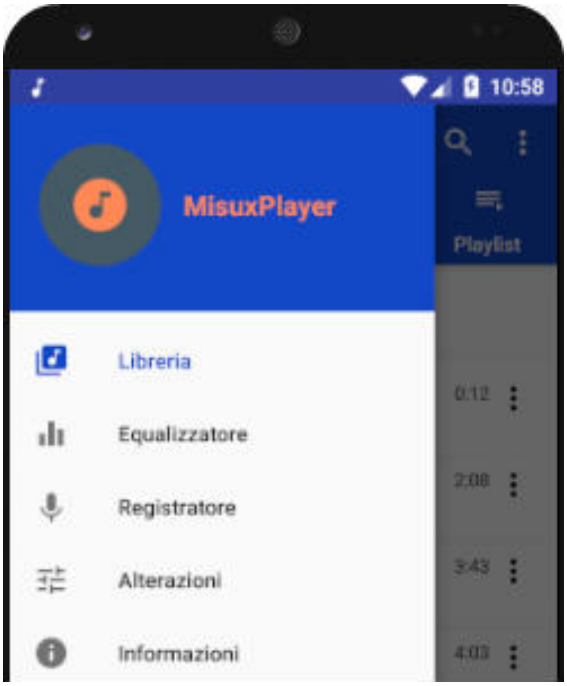


Figura 7: Al semplice click dell'utente si può navigare tra le diverse activity: Libreria, Equalizzatore, Registratore, Alterazioni e Informazioni.

La NavigationView che si vede è l'elenco di voci che apparirà all'interno del Navigation Drawer. Come si intuisce dal suo nome, si tratta di un layout "a cassetto" che costituisce la struttura portante del Navigation Drawer. Include sia i layout dell'activity che quello del menu di navigazione. Entrambi i componenti verranno, come

tutti i widget, individuati tramite il metodo **findViewById**. Il DrawerLayout verrà utilizzato per azionare fisicamente il pannello, mentre il NavigationView permetterà di gestire i comandi inseriti al suo interno. Il NavigationView pertanto verrà popolato con una serie di **item**, definiti come se si trattasse di un Options Menu. Ogni voce del menu interno alla NavigationView possiede un *id* che la identifica. La selezione di ogni singola voce verrà elaborata dal metodo onNavigationItemSelected, appartenente ad un listener di tipo OnNavigationItemSelectedListener.

```
@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nav_library:
            item.setChecked(true);
            return true;
        case R.id.nav_equalizer:
            Equalizer();
            return true;
        case R.id.nav_recorder:
            Recorder();
            return true;
        case R.id.nav_alteration:
            Alteration();
            return true;
        case R.id.nav_info:
            Info();
            return true;
    }

    DrawerLayout drawer = (DrawerLayout)
                findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
private void Equalizer() {
    Intent intent = new Intent(this, Equalizer.class);
    startActivity(intent);
}
```

All'interno di ogni case viene richiamato un metodo che attraverso un intent invoca la rispettiva activity. Viene riportato l'intent dell'equalizzatore come esempio.

2.3.2 Search Widget

Search Widget viene denotato da una comune icona nella Action Bar, tipicamente l'immagine di una lente di ingrandimento. Selezionandola, viene invocato un intent che indirizza ad un'activity dedicata per la ricerca. All'interno di SearchActivity appare un campo di testo che permette l'inserimento dei termini da utilizzare come query di ricerca per ottenere il brano desiderato.

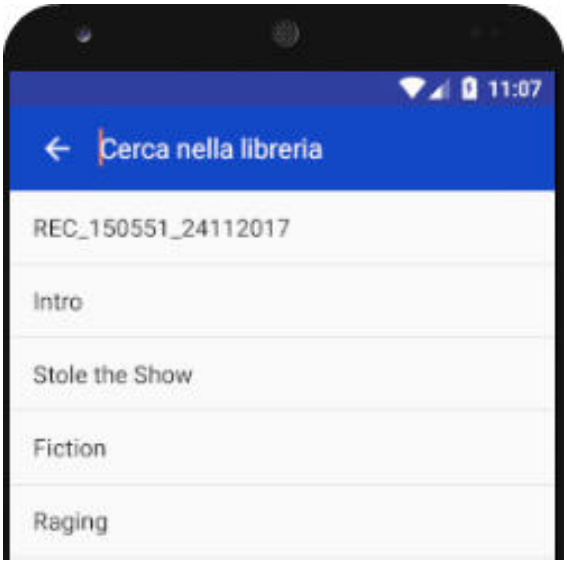


Figura 8: L'utente può cercare all'interno della propria libreria musicale attraverso questa semplice schermata.

Per impostazione predefinita, il widget di ricerca si comporta come un widget EditText standard che si può configurare in modo che il sistema Android gestisca tutti gli eventi di input e fornisca le query all'activity appropriata e i suggerimenti di ricerca. I risultati di ricerca, in questo caso i brani all'interno del dispositivo, sono definiti tramite un'ArrayAdapter che permette di collegare la ListView rispetto ai dati da mostrare. Viene implementata l'interfaccia SearchView.OnQueryTextListener per l'utilizzo dei metodi booleani onQueryTextChange e onQueryTextSubmit. Il primo viene richiamato quando il testo della query viene modificata dall'utente. Il secondo invece, quando l'utente invia la query. Ciò potrebbe essere dovuto alla pressione di un tasto sulla tastiera o alla pressione di un pulsante di invio. Il listener può sovrascrivere il comportamento standard restituendo true per indicare che ha gestito la richiesta di invio. In caso contrario, restituire false per consentire a SearchView di gestire l'invio avviando qualsiasi intent associato.

2.3.3 Dropdown menu

L'ultimo elemento compreso nell'ActionBar è un menu a tendina. Per definire il seguente menu, è stato necessario creare un file XML all'interno della cartella *res/menu* con tag di apertura <menu> contenente le varie opzioni. All'interno di questo nodo radice sono presenti le singole voci definite attraverso il nodo <item>. Le opzioni che si è scelto di utilizzare verranno descritte man mano nel corso del testo.

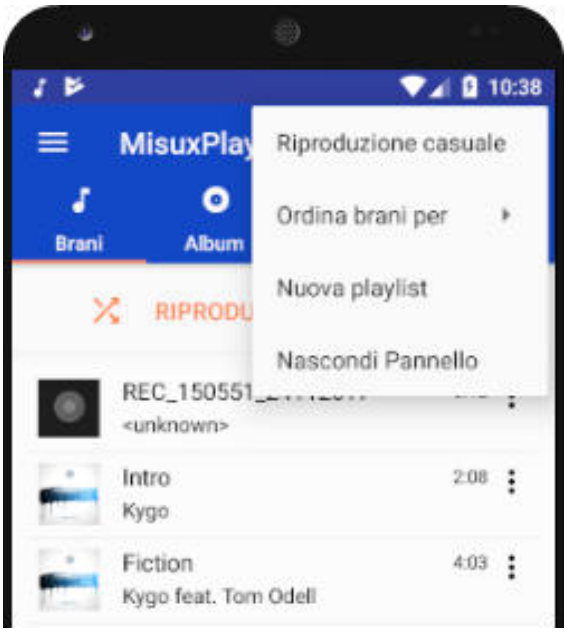


Figura 9: La voce *Ordina brani per* aprirà un submenu per scegliere l'ordinamento.

Viene riportato il codice relativo la creazione del menu.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/ripRand"
          android:title="@string/random"/>
    <item android:id="@+id/sort"
          android:title="@string/sort">
          <menu> ...item for sort... </menu> </item>
    <item android:id="@+id/newPlaylist"
          android:title="@string/new_playlist"/>
    <item android:id="@+id/action_toggle"
          android:title="@string/action_hide"
</menu>
```

2.4 TabLayout e ViewPager

La TabLayout fornisce un layout orizzontale per la visualizzazione delle schede all'interno dell'applicativo. Per scheda intendiamo la pagina scorrevole che mostra i rispettivi brani, album, artisti e playlist. La popolazione delle schede da visualizzare viene eseguita tramite istanze TabLayout.Tab. Nell'applicativo vengono create 4 schede ognuna per ogni fragment. Insieme al TabLayout viene utilizzando ViewPager. Il ViewPager è il contenitore dei fragments e attraverso il metodo setupWithViewPager della TabLayout, collega i rispettivi fragments con la seguente sintassi Tab.setupWithViewPager(mViewPager);
Inoltre attraverso il metodo setupTabIcons() viene aggiunta per ogni scheda un'icona rappresentativa del fragment. Il layout del ViewPager verrà automaticamente popolato con i fragments creati attraverso il PagerAdapter. L'implementazione del PagerAdapter rappresenta ogni pagina come fragment che viene mantenuto in modo persistente nel FragmentManager fino a quando l'utente non torna a quella pagina. La versione utilizzata del PagerAdapter viene usata quando ci sono una serie di fragments da sfogliare come una serie di schede. Il fragment di ogni pagina che l'utente visita, sarà tenuto in memoria sebbene la sua gerarchia delle view possa essere distrutta quando non visibile. Ciò può comportare l'utilizzo di una quantità significativa di memoria poiché le istanze di fragment possono contenere una quantità arbitraria di stati. Quando si utilizza FragmentPagerAdapter, l'host ViewPager deve avere un set ID valido. Le sottoclassi devono solo implementare getItem (int) e getCount () per avere un adattatore funzionante. Viene riportata la classe SectionPagerAdapter e il metodo per la creazione dei fragments nelle pagine successive.

Inizialmente vengono dichiarati due ArrayList ognuno dei quali per contenere la lista e i titoli dei fragments. Il metodo getItem restituisce il fragment associato a una posizione specificata mentre getCount la lunghezza di esso. Il metodo addFragment è una delle parti fondamentali dell'applicativo perchè aggiunge i rispettivi fragment agli ArrayList sopra dichiarati. Infine è necessario un metodo getPageTitle per restituire la posizione del titolo del fragment all'interno di mFragmentTitleList.

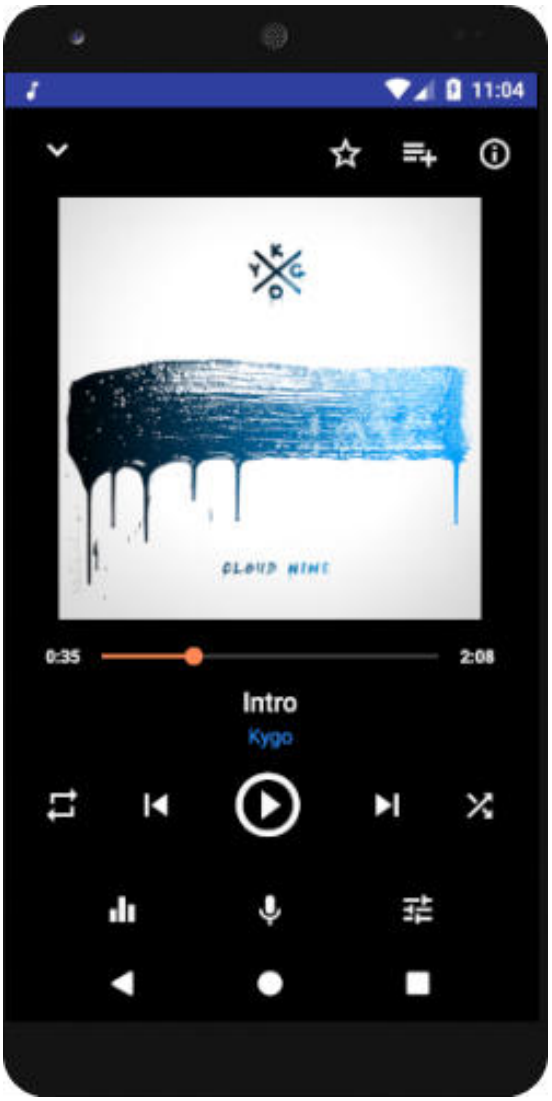
```
public class SectionsPagerAdapter extends FragmentPagerAdapter {

 private final List<Fragment> mFragmentList = new ArrayList<>();
 private final List<String> mFragmentTitleList = new ArrayList<>();

 public SectionsPagerAdapter(FragmentManager manager) {
      super(manager);
 }

 @Override
 public Fragment getItem(int position) {
      return mFragmentList.get(position);
 }

 @Override
 public int getCount() {
     // Mostra 4 pagine
     return mFragmentList.size();
 }

 public void addFragment(Fragment fragment, String title) {
      mFragmentList.add(fragment);
      mFragmentTitleList.add(title);
 }

 @Override
 public CharSequence getPageTitle(int position) {
      return mFragmentTitleList.get(position);
 }
}
```

Nel seguente codice vengono istanziati i fragments e aggiunti al ViewPager tramite il PageAdapter. Inoltre vengono istanziati altri 3 fragments che analizzerò più avanti.

```
private void setupViewPager(ViewPager viewPager) {
    adapter = new SectionsPagerAdapter(getSupportFragmentManager());
    //fragment brani
    bf = new SongFragment();
    bf.parent = this;
    adapter.addFragment(bf, "Brani");

    //fragment album
    alf = new AlbumFragment();
    alf.parent = this;
    adapter.addFragment(alf, "Album");

    //fragment artist
    arf = new ArtistFragment();
    arf.parent = this;
    adapter.addFragment(arf, "Artisti");

    //fragment playlist
    pf = new PlaylistFragment();
    pf.parent = this;
    adapter.addFragment(pf, "Playlist");

    //fragment album song
    alsf = new AlbumSongFragment();
    alsf.parent = this;

    //fragment album artist
    arsf = new ArtistSongFragment();
    arsf.parent = this;
    psf = new PlaylistSongFragment();

    //fragment playlist song
    psf.parent = this;

    //VIEWPAGER
    mViewPager.setOffscreenPageLimit(3);
    viewPager.setAdapter(adapter);
}
```

2.5 SlidingUpPanelLayout

SlidingUpPaneLayout fornisce un layout scorrevole quando l'utente trascina verso l'alto la toolbar che mostra la progressione, il titolo e l'artista del brano in esecuzione. Al primo avvio dell'applicativo questo layout rimane nascosto sotto il ViewPager. Una volta che l'utente avvia il primo brano viene attivata la toolbar con la possibilità di scorrimento. Tra le opzioni presenti nel menu a tendina nell'ActionBar c'è anche la possibilità di attivare e disattivare tale pannello scorrevole. All'interno di questo riquadro scorrevole sono presenti tutti i comandi per la gestione e manipolazione audio. Possiamo dividere il layout del pannello scorrevole in tre parti principali. La prima riguarda le opzioni della toolbar che cambia colore diventando nera una volta che il pannello è aperto. A questo punto diventano visibile le icone *aggiungi a preferiti*, *aggiungi a playlist* e l'icona per i *dettagli* del brano. La seconda sezione di questo layout è relativa all'immagine di copertina del brano e la barra di progressione con la relativa durata, titolo e artista del brano. Se l'immagine di copertina non è presente allora verrà inserita un'immagine di default per il brano. L'ultima sezione riguarda i comandi musicali. Sono previsti i comandi di base di play, stop, next, previous, random e repeat. Infine le ultime tre icone più in basso attivano degli intent che portano alle pagine di equalizzatore, registratore e alterazioni. Viene riportato nella pagina successiva il codice relativo al pannello scorrevole. Inizialmente il pannello viene individuato tramite il metodo findViewById, istanziato e messo in ascolto. Viene utilizzato il metodo onPannelStateChanged per gestire il collpse e drag del pannello. Come si vede nel codice, a seconda dello stato del pannello vengono rese visibili o invisibili le opzioni descritte precedentemente della toolbar. Viene impostato un effetto di *fade* attraverso il metodo setFadeOnClickListener quando lo stato del pannello è collapsed.



```
//SlidingUpPanel
sliding_layout = (SlidingUpPanelLayout)
                            findViewById(R.id.sliding_layout);
sliding_layout.addPanelSlideListener(
      new SlidingUpPanelLayout.SimplePanelSlideListener() {
@Override
public void onPanelSlide(View panel, float slideOffset) {
    Log.i(TAG, "onPanelSlide, offset"+ slideOffset);
}

@Override
public void onPanelStateChanged(View panel,
    SlidingUpPanelLayout.PanelState previousState,
    SlidingUpPanelLayout.PanelState newState) {
    Log.i(TAG, "onPanelStateChanged" + newState);
    //visibility if collapsed or dragging
    if (newState == SlidingUpPanelLayout.PanelState.COLLAPSED) {
        iv_play.setVisibility(View.VISIBLE);
        progressBar.setVisibility(View.VISIBLE);
        optionSong.setVisibility(View.INVISIBLE);
        addPL.setVisibility(View.INVISIBLE);
        iv_favorite.setVisibility(View.INVISIBLE);
     } else
     if (newState == SlidingUpPanelLayout.PanelState.DRAGGING) {
        iv_play.setVisibility(View.INVISIBLE);
        progressBar.setVisibility(View.INVISIBLE);
        optionSong.setVisibility(View.VISIBLE);
        addPL.setVisibility(View.VISIBLE);
        iv_favorite.setVisibility(View.VISIBLE);
     }
   }
});

sliding_layout.setFadeOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
    sliding_layout.setPanelState(SlidingUpPanelLayout.
        PanelState.COLLAPSED);
  }
});
```

Capitolo 3

Music library

In questo capitolo vengono illustrate tutte le classi principali realizzate per il popolamento delle tracce audio all'interno dell'applicativo. Alla fine della descrizione di ogni classe viene mostrato il codice relativo all'estrazione dei contenuti multimediali audio archiviati all'interno del dispositivo.

3.1 Song

All'interno della classe Song vengono dichiarati una serie di attributi che verrano associati al costruttore Song. Il riferimento this all'interno del costruttore, non fa altro che puntare all'oggetto a cui appartiene risolvendo possibili problemi di ambiguità.

```
public class Song {
    private long id, duration, idAlbum;
    private String title, artist, path, track, size, year;

    public Song(long id, String title, String artist,
    long duration,String path, long idAlbum,
    String track, String size, String year) {
        this.id = id;
        this.path = path;
        this.title = title;
        this.artist = artist;
        this.duration = duration;
        this.idAlbum = idAlbum;
        this.track = track;
        this.size = size;
        this.year = year;
    }
```

```
    public long getID() { return id; }
    public long getDuration() { return duration; }
    public long getIdAlbum() { return idAlbum; }
    public String getArtist() { return artist; }
    public String getTitle() { return title; }
    public String getTrack() { return track; }
    public String getPath() { return path; }
    public String getSize() { return size; }
    public String getYear() { return year; }
}
```

3.1.1 SongAdapter

Alla fine di questa classe vengono dichiarati i metodi per i singoli attributi utilizzati nel SongAdapter. La classe SongAdapter ti permette di creare una view dinamica della lista dei brani contenuti all'interno dei singoli fragments. La struttura layout di ogni singolo fragment contiene un NestedScrollView e una RecyclerView. Quest'ultimo widget è una versione più avanzata e flessibile di una ListView. Esso è un contenitore per la visualizzazione di insiemi di dati di grandi dimensioni che possono essere fatti scorrere in modo molto efficiente attraverso un NestedScrollView. Per utilizzare il widget RecyclerView, è necessario specificare a sua volta un elemento adpater e un gestore di layout detto layout manager. Un layout manager posiziona le visualizzazioni delle voci all'interno di una RecyclerView e determina quando riutilizzarle. Per riutilizzare una view, un layout manager può chiedere all'adapter di sostituire il contenuto della view con un elemento diverso dal set di dati. Il riciclo delle view in questo modo migliora le prestazioni evitando la creazione di view non necessarie senza effettuare costose ricerche findViewById(). Nell'applicativo vengono utilizzati i seguenti layout manager: LinearLayoutManager mostra gli oggetti in una lista a scorrimento verticale o orizzontale. Esso viene utilizzato per la lista dei brani e la lista delle playlist. Per gli album e artisti invece, viene usato un GridLayoutManager che mostra gli oggetti in una griglia; Il seguente codice assegna un LinearLayoutManager ai brani.

```
scrollSongs = songView.findViewById(R.id.NestscrollView);
recycler = songView.findViewById(R.id.recycler);
recycler.setLayoutManager(new LinearLayoutManager(
        getActivity().getApplicationContext()));
recycler.setNestedScrollingEnabled(false);
recycler.addItemDecoration(new SimpleDividerItemDecoration(
        getActivity())); //divider
```

La classe SongAdapter estende la classe RecyclerView.Adapter. Il costruttore di questa classe riceve in ingresso il contesto del fragment, l'ArrayList specifico e un RecyclerItemClickListener per catturare l'evento del click sul relativo item in quella posizione della RecyclerView. In questo modo al click dell'utente di uno specifico brano della lista, verrano richiamati tutti i metodi per la creazione e controllo del MediaPlayer che verrà approfonditi nel prossimo capitolo.

```
public SongAdapter(Context songCxt, ArrayList <Song> songList,
                   RecyclerItemClickListener listener){
        this.songCxt = songCxt;
        this.songList = songList;
        this.listener = listener;
}
```

Di seguito viene riportato il metodo che crea la view dalla singola riga di layout.

```
public SongViewHolder onCreateViewHolder(ViewGroup parent,
                int viewType) {
  View view = LayoutInflater.from(parent.getContext()).
                inflate(R.layout.song_row, parent, false);
  return new SongViewHolder(view);
}
```

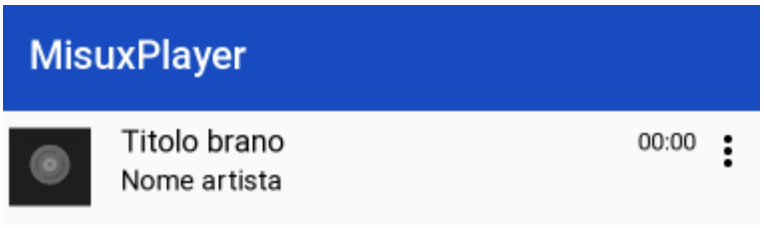


Figura 10: Layout che viene adattato per ogni brano della lista.

Infine il metodo onBindViewHolder riceve in ingresso la SongViewHolder e la relativa posizione e imposta per ogni riga del brano la cover, il titolo, l'artista e la durata.

```
public void onBindViewHolder(final SongViewHolder holder, int pos) {
  final Song song = songList.get(pos);
  holder.cover.setImageBitmap(art);
  holder.title.setText(song.getTitle());
  holder.artist.setText(song.getArtist());
  holder.duration.setText(duration);
  holder.bind(song, listener);
}
```

3.1.2 MediaStore.Audio.Media

```
public static void getSongList() {
 final Uri uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
 final String _id = MediaStore.Audio.Media._ID;
 final String title = MediaStore.Audio.Media.TITLE;
 final String artist = MediaStore.Audio.Media.ARTIST;
 final String duration = MediaStore.Audio.Media.DURATION;
 final String path = android.provider.MediaStore.Audio.Media.DATA;
 final String albumId = MediaStore.Audio.Media.ALBUM_ID;
 final String nTrack = MediaStore.Audio.Media.TRACK;
 final String size = MediaStore.Audio.Media.SIZE;
 final String year = MediaStore.Audio.Media.YEAR;
 String selection = MediaStore.Audio.Media.IS_MUSIC + "!=0";
 final String[] columns = { id, title, artist, duration, path,
 albumId, nTrack, size, year };
 Cursor cursor = context.getContentResolver().
        query(uri, columns, selection, null, title);
 try {
  if (cursor != null && cursor.moveToFirst()) {
   do {
    long id = cursor.getLong(cursor.getColumnIndex(_id));
    String tit = cursor.getString(cursor.getColumnIndex(title));
    String art = cursor.getString(cursor.getColumnIndex(artist));
    long dur = cursor.getLong(cursor.getColumnIndex(duration));
    String path2 = cursor.getString(cursor.getColumnIndex(path));
    long alb = cursor.getLong(cursor.getColumnIndex(albumId));
    String track = cursor.getString(cursor.getColumnIndex(nTrack));
    String dim = cursor.getString(cursor.getColumnIndex(size));
    String yearS = cursor.getString(cursor.getColumnIndex(year));

    songList.add(new Song(id, tit, art, dur,
    path2, alb, track, dim, yearS));

   } while (cursor.moveToNext());
  }
 } catch (Exception e) {
    cursor.close();
 }
}
```

Il codice della pagina precedente permette l'estrazione dei contenuti multimediali audio archiviati all'interno del dispositivo. I metodi di ottenimento dei brani, album, artisti e playlist vengono richiamati nel costruttore della classe MusicLibrary. All'avvio dell'applicativo, dopo la SplashActivity, viene interrogato il dispositivo utilizzando la classe MediaStore.Audio. In questo caso specifico, per i brani viene utilizzata la classe MediaStore.Audio.Media che grazie al riferimento URI (Uniform Resource Identifier) allinea i contenuti musicali interni in forma di stringa. Vengono quindi associate le rispettive stringhe per ogni campo. Successivamente viene dichiarato un cursore che fornisce accesso in lettura e scrittura casuale al set di risultati restituiti da una query dal database del dispositivo. La query riceve in ingresso il riferimento uri, l'array string dei vari campi per l'audio, il tipo di di prelevamento (music) e due ulteriori input per condizioni e ordinamento. A questo punto viene iterato il cursore e aggiunto ogni campo all'interno dell'ArrayList della classe Song.

3.2 Album

Analoga alla classe Song, all'interno della classe Album vengono dichiarati i rispettivi attributi utili per le informazioni dell'album. Un attributo importante riguarda il path dell'album. Esso verrà convertito nell'immagine di cover solo se presente.

```
public class Album {
    private long idAlbum;
    private String albumName, artistName, path, year;
    private int nr_of_songs;

    public Album(long idAlbum, String albumName, String artistName,
    String path, int nr_of_songs, String year) {
        this.idAlbum = idAlbum;
        this.albumName = albumName;
        this.nr_of_songs = nr_of_songs;
        this.artistName = artistName;
        this.path = path;
        this.year = year;
    }

    public long getID(){ return idAlbum; }
    public int getNr_of_songs() { return nr_of_songs; }
    public String getAlbumName(){ return albumName; }
    public String getArtistName() { return artistName; }
    public String getPath() { return path; }
```

```
    public String getYear() { return year; }
}
```

3.2.1 AlbumAdapter

La costruzione delle classi Adapter è identica alla classe SongAdapter. Come abbiamo già visto, questa estende la classe RecyclerView.Adapter e riceve in ingresso il contesto del fragment, l'ArrayList specifico e un RecyclerItemClickListener per il click dell'utente. La grande differenza sta nell'utilizzo del layout manager. Per gli album viene utilizzato un GridLayoutManager che dispone gli oggetti per griglia. Il seguente codice viene utilizzato per assegnare un GridLayoutManager agli album.

```
scrollAlbum = albumView.findViewById(R.id.NestscrollView);
recyclerViewAlbum = albumView.findViewById(R.id.recyclerAlbum);
layoutManager = new GridLayoutManager(
        getActivity().getApplicationContext(), 2);
recyclerViewAlbum.setLayoutManager(layoutManager);
recyclerViewAlbum.setNestedScrollingEnabled(false);
```

Di seguito viene riportato il metodo che crea la view dalla singola riga di layout.

```
public AlbumViewHolder onCreateViewHolder(ViewGroup parent,
                    int viewType) {
  View view = LayoutInflater.from(parent.getContext()).
              inflate(R.layout.album_row, parent, false);
  return new AlbumAdapter.AlbumViewHolder(view);
}
```

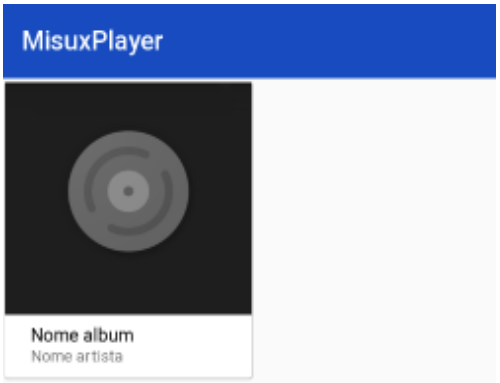


Figura 11: Layout che viene adattato per ogni album della griglia.

3.2.2 MediaStore.Audio.Albums

Viene riportato il codice inerente all'estrazione dei contenuti musicali per gli album. L'unica differenza riguarda l'utilizzo della classe MediaStore.Audio.Albums per il prelevamento delle informazioni.

```
public static void getAlbumsLists(){
 final Uri uri = MediaStore.Audio.Albums.EXTERNAL_CONTENT_URI;
 final String _id = MediaStore.Audio.Albums._ID;
 final String name = MediaStore.Audio.Albums.ALBUM;
 final String artist = MediaStore.Audio.Albums.ARTIST;
 final String path = MediaStore.Audio.Albums.ALBUM_ART;
 final String tracks = MediaStore.Audio.Albums.NUMBER_OF_SONGS;
 final String year = MediaStore.Audio.Albums.FIRST_YEAR;
 final String[] columns = {id, name, artist, art, tracks, year};
 Cursor cursor = context.getContentResolver().
        query(uri, columns, null, null, album_name);
 try {
  if (cursor != null && cursor.moveToFirst()) {
   do {
    long id = cursor.getLong(cursor.getColumnIndex(_id));
    String name = cursor.getString(cursor.getColumnIndex(name));
    String artist = cursor.getString(cursor.getColumnIndex(art));
    String path = cursor.getString(cursor.getColumnIndex(path));
    int nr=Integer.parseInt(cursor.getString(cursor.
                                    getColumnIndex(tracks)));
    String year = cursor.getString(cursor.getColumnIndex(year));

    albumList.add(new Album(id, name, artist, path, nr, year));

   } while (cursor.moveToNext());
  }
 } catch (Exception e) {
    cursor.close();
 }
}
```

3.2.3 AlbumSongFragment

Nel capitolo precedente viene analizzato il metodo setupViewPager per impostare il ViewPager. In questo metodo vengono istanziati i fragments che comporranno il ViewPager. Vengono creati però 3 ulteriori fragments. Questi sono dei fragments di transazione: quando l'utente seleziona un determinato album si aprirà un fragment al di sopra del fragment degli album sempre nel ViewPager, che conterrà tutti brani specifici l'album cliccato. Così per gli artisti e per le playlist.

```
//transaction towards fragment
FragmentTransaction ft = getFragmentManager().beginTransaction()
  .setCustomAnimations(R.anim.fade_in, R.anim.fade_out)
  .add(R.id.fragment_container_album, parent.alsf)
  .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
  .addToBackStack(null);
ft.commit();
```

3.3 Artist

Come si vede la creazione di ogni classe per l'audio è molto simile. All'interno di questa classe vengono dichiarati i rispettivi attributi per l'artista.

```
public class Artist {
    private long idArtist;
    private String artistName;
    private int nr_of_albums, nr_of_songs;

    public Artist(long idArtist, String artistName,
    int nr_of_albums, int nr_of_songs) {
        this.idArtist = idArtist;
        this.artistName = artistName;
        this.nr_of_albums = nr_of_albums;
        this.nr_of_songs = nr_of_songs;
    }

    public long getID(){ return idArtist; }
    public String getNameArtist(){ return artistName; }
    public int getNumAlbum() { return nr_of_albums; }
    public int getNumBrani() { return nr_of_songs; }
}
```

3.3.1 ArtistAdapter

La costruzione delle classe Adapter per gli artisti è identica alla classe AlbumAdapter. Anche in questo caso viene utilizzato un GridLayoutManager già descritto in precedenza.

3.3.2 MediaStore.Audio.Artist

Viene riportato il codice inerente all'estrazione dei contenuti musicali per gli artisti. Anche in questo caso, l'unica differenza riguarda l'utilizzo della classe MediaStore.Audio.Artist per il prelevamento delle informazioni.

```
public static void getArtistsLists() {
 final Uri uri = MediaStore.Audio.Artists.EXTERNAL_CONTENT_URI;
 final String _id = MediaStore.Audio.Artists._ID;
 final String name = MediaStore.Audio.Artists.ARTIST;
 final String numAlbums = MediaStore.Audio.Artists.NUMBER_OF_ALBUMS;
 final String tracks = MediaStore.Audio.Artists.NUMBER_OF_TRACKS;
 final String[] columns = {_id, name, numAlbums, tracks};
 Cursor cursor = context.getContentResolver().
        query(uri, columns, null, null, name);
 try {
  if (cursor != null && cursor.moveToFirst()) {
   do {
     long id = cursor.getLong(cursor.getColumnIndex(_id));
     String artistName=cursor.getString(cursor.getColumnIndex(name));
     int numA = Integer.parseInt(cursor.getString(cursor.
                                 getColumnIndex(numAlbums)));
     int numS= Integer.parseInt(cursor.getString(cursor.
                                 getColumnIndex(tracks)));

     artistList.add(new Artist(id, artistName, numA, numS));
   } while (cursor.moveToNext());
  }
 } catch(Exception e){
     cursor.close();
 }
}
```

3.3.3 ArtistSongFragment

Viene riportato per completezza di codice, le istruzioni riguardo la transazione del fragment che conterrà i brani specifici di un determinato artista.

```
FragmentTransaction ft = getFragmentManager()
   .beginTransaction()
   .setCustomAnimations(R.anim.fade_in, R.anim.fade_out)
   .add(R.id.fragment_container_artist, parent.arsf)
   .setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE)
   .addToBackStack(null);
ft.commit();
```

3.4 Playlist

Di seguito viene riportata la costruzione della classe Playlist.

```
public class Playlist {
    private long playlistID;
    private String namePlaylist;

    public Playlist(long playlistID, String namePlaylist) {
        this.playlistID = playlistID;
        this.namePlaylist = namePlaylist;
    }
    public long getPlaylistID() { return playlistID; }
    public String getNamePlaylist() {return namePlaylist;}
}
```

3.4.1 PlaylistAdapter

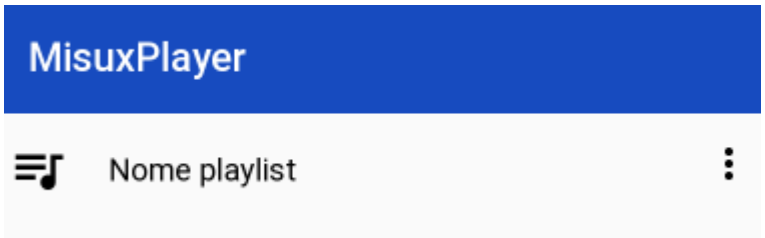


Figura 12: Layout che viene adattato per ogni playlist della lista.

3.4.2 MediaStore.Audio.Playlists

Infine viene utilizzata la classe MediaStore.Audio.Playlists per estrarre le playlists.

```
public static void getPlaylists(){
 final Uri uri = MediaStore.Audio.Playlists.EXTERNAL_CONTENT_URI;
 final String id = MediaStore.Audio.Playlists._ID;
 final String name = MediaStore.Audio.Playlists.NAME;
 final String[] columns = { _id, name};
 Cursor cursor = context.getContentResolver().
        query(uri, columns, null, null, playlist_name);
 try {
  if (cursor != null && cursor.moveToFirst()) {
   do {
    long id = cursor.getLong(cursor.getColumnIndex(_id));
    String plName = cursor.getString(cursor.getColumnIndex(name));

    playList.add(new Playlist(id, plName));
   } while (cursor.moveToNext());
  }
 } catch(Exception e) {
    cursor.close();
 }
}
```

3.4.3 PlaylistTrack

La classe PlaylistTrack è necessaria per la costruzione di un ArrayList di tipo PlaylistTrack che conterrà i relativi brani della playlist.

```
public class PlaylistTrack {
    private long playlistIdTrack;
    private String titleTrack;

    public PlaylistTrack(long playlistIdTrack, String titleTrack) {
        this.playlistIdTrack = playlistIdTrack;
        this.titleTrack = titleTrack;
    }

    public long getAudio_id() { return playlistIdTrack; }
    public String getTitlePL() { return titleTrack; }
}
```

```
//metodo per ottenere i singoli brani di una specifica playlist
public static void getPlaylistTracks(Context ctx, Long playlistId) {
final Uri uri = MediaStore.Audio.Playlists.Members.
                getContentUri("external",playlistId);
final String audio_id = MediaStore.Audio.Playlists.Members.AUDIO_ID;
String pl_title = MediaStore.Audio.Playlists.Members.TITLE;
String[] col = {audio_id, pl_title};
Cursor c = cxt.getContentResolver().query(uri,col,null,null,null);
 try {
  if (cursor != null && cursor.moveToFirst()) {
   do {
     long audioID = c.getLong(c.getColumnIndex(audio_id));
     String title = c.getString(c.getColumnIndex(pl_title));
     playListTrack.add(new PlaylistTrack(audioID,title));
   } while (c.moveToNext());
  }
 } catch (Exception e) {
    c.close();
 }
}
//costruzione oggetto Song se brani appartengo a una playlist
MusicLibrary.getPlaylistTracks(getActivity().
getApplicationContext(), playlist.getPlaylistID());
for (int i = 0; i < MusicLibrary.playListTrack.size(); i++) {
  for(int j = 0; j < MusicLibrary.songList.size(); j++) {
    if (MusicLibrary.playListTrack.get(i).getTitlePL().equals
        (MusicLibrary.songList.get(j).getTitle())) {
        songsPlaylist.add(new Song(
        MusicLibrary.songList.get(j).getID(),
        MusicLibrary.songList.get(j).getTitle(),
        MusicLibrary.songList.get(j).getArtist(),
        MusicLibrary.songList.get(j).getDuration(),
        MusicLibrary.songList.get(j).getPath(),
        MusicLibrary.songList.get(j).getIdAlbum(),
        MusicLibrary.songList.get(i).getTrack(),
        MusicLibrary.songList.get(i).getSize(),
        MusicLibrary.songList.get(i).getYear()));
    }
  }
}
```

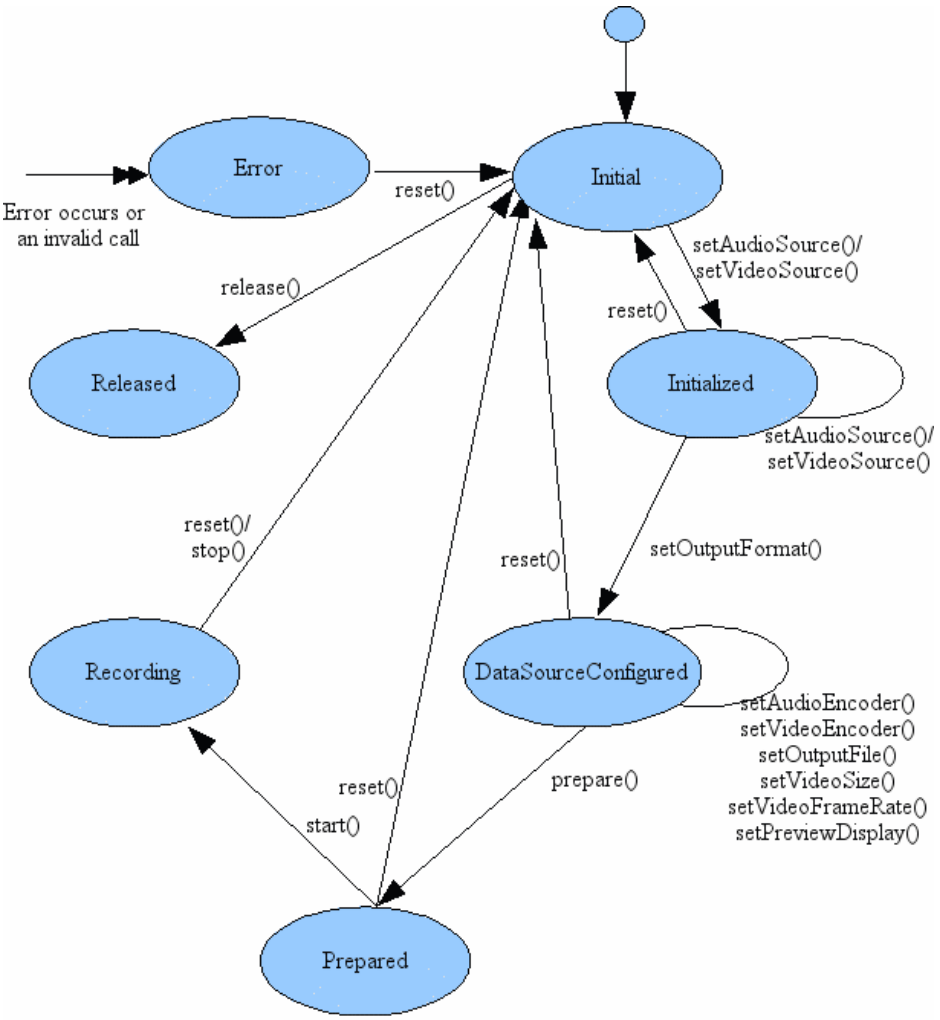
Capitolo 4

Funzionalità di base del player audio

In questo capitolo viene descritta la classe MediaPlayer utilizzata per la riproduzione e gestione audio dell'applicativo soffermandosi su gli stati in cui un oggetto MediaPlayer può risiedere. Successivamente vengono illustrate le parti più rilevanti dei comandi musicali che compongono il player audio.

4.1 Mediaplayer

La classe MediaPlayer può essere utilizzata per controllare la riproduzione di file e flussi audio/video. Essa supporta diverse fonti multimediali come: risorse locali, URI interne ottenuti da un Contenut Resolver, e URI esterne (streaming). Per un approfondimento guardare la classe MediaPlayer su Android Developers. Il controllo della riproduzione di file e flussi audio/video è gestito come macchina di stato. Lo schema seguente mostra il ciclo di vita e gli stati di un oggetto MediaPlayer guidato dalle operazioni di controllo della riproduzione supportate. Nel diagramma le figure ovali rappresentano gli stati in cui un oggetto MediaPlayer può risiedere. Gli archi rappresentano le operazioni di controllo della riproduzione che guidano la transizione dello stato dell'oggetto. Esistono due tipi di archi. Gli archi con una singola freccia rappresentano le chiamate al metodo sincrono, mentre quelli con una doppia freccia rappresentano chiamate di metodo asincrone. Quando un oggetto MediaPlayer viene creato utilizzando new è in uno stato di inattività detto Idle state; In questa fase è un errore richiamare metodi come start(), pause() o stop() in questo stato ancora di attesa perchè essi possono essere richiamati solo dopo la fase di preparazione dell'oggetto MediaPlayer. Una volta che un oggetto MediaPlayer non viene più utilizzato, è necessario chiamare immediatamente la release() in modo che le risorse utilizzate dal motore di lettore interno associato all'oggetto possano essere rilasciate immediatamente. Le risorse possono includere risorse singleton come componenti di



MediaRecorder state diagram

accelerazione hardware o l'errore di chiamata release(). Una volta che l'oggetto MediaPlayer si trova nello stato di fine, non può più essere utilizzato e non è possibile ripristinarlo in nessun altro stato. In generale, alcune operazioni di controllo della riproduzione potrebbero non riuscire a causa di vari motivi, come il formato audio o video non supportato, risoluzione troppo alta, timeout dello streaming e molti altri. Pertanto, la segnalazione degli errori e il recupero rappresentano una preoccupazione importante in queste circostanze. In tutte queste condizioni di errore, il motore del player interno richiama un metodo OnErrorListener.onError(). È importante notare che una volta che si verifica un errore, l'oggetto MediaPlayer entra nello stato di Error. Per riutilizzare un oggetto MediaPlayer che si trova nello stato di errore, è possibile chiamare reset() per ripristinare l'oggetto nel suo stato di inattività. È buona prassi

registrare un OnErrorListener per verificare le notifiche di errore dichiarando una IllegalStateException per impedire errori di programmazione della chiamata prepare(), prepareAsync() o uno dei metodi di overloaddataSource in uno stato non valido. La chiamata setDataSource (FileDescriptor) o setDataSource (Context, Uri) trasferisce un oggetto MediaPlayer nello stato di attesa allo stato inizializzato. Viene generata una IllegalStateException se setDataSource () viene chiamato in qualsiasi altro stato. A questo punto un oggetto MediaPlayer deve prima entrare nello stato Prepare prima di poter avviare la riproduzione. Esistono due modi per la preparazione alla riproduzione: preparazione sincrona o asincrona; Una chiamata a prepare() (sincrona) trasferisce l'oggetto allo stato Prepare. Quando viene restituita la chiamata prepareAsync() (asincrona) invece, viene trasferito l'oggetto nello stato di Prepare solo dopo il ritorno della stessa chiamata mentre il motore del lettore interno continua a lavorare sul resto del lavoro di preparazione fino al termine. Quando la preparazione termina o quando la chiamata prepare() ritorna, il motore del player interno chiama quindi un metodo di callback detto onPrepared() dell'interfaccia OnPreparedListener. È importante notare che lo stato di preparazione è uno stato transitorio. Anche in questo stato è necessario richiamare una UnlegalStateException se prepare () o prepareAsync () chiama un qualsiasi altro stato. Per avviare la riproduzione, deve essere chiamato start(). Dopo l'avvio di start(), l'oggetto MediaPlayer si trova nello stato di avvio. Il metodo isPlaying() può essere chiamato per verificare se l'oggetto MediaPlayer è nello stato di avvio. Mentre è in questo stato, il motore del player interno chiama un metodo OnBufferingUpdateListener.onBufferingUpdate() se un OnBufferingUpdateListener è stato registrato in precedenza tramite setOnBufferingUpdateListener (OnBufferingUpdateListener). Questa funzione di callback consente alle applicazioni di tenere traccia dello stato del buffering durante lo streaming audio. La chiamata start() non ha effetto su un oggetto MediaPlayer che si trova già nello stato di avvio. La riproduzione può essere messa in pausa tramite pause(). Quando viene restituita la chiamata a pause(), l'oggetto MediaPlayer passa allo stato di pausa. Si noti che la transizione dallo stato di avvio allo stato di pausa e viceversa, avviene in modo asincrono nel motore del lettore. È necessario richiamare start() per riprendere la riproduzione di un oggetto MediaPlayer in pausa e la posizione di riproduzione ripresa risulterà la stessa della messa in pausa. Quando viene restituita la chiamata a start(), l'oggetto MediaPlayer in pausa torna allo stato di avvio. La chiamata stop() interrompe la riproduzione passando allo stato di stop. La riproduzione non può essere avviata fino a quando prepare() o prepareAsync() vengono richiamati per impostare nuovamente l'oggetto MediaPlayer sullo stato Prepare. La posizione di riproduzione può essere regolata con una chiamata a seekTo (long, int) e recuperata con una chiamata a getCurrentPosition(). Quando la riproduzione raggiunge la fine del flusso, la riproduzione termina. Se la modalità di loop è stata impostata su true con il metodo setLooping (booleano), l'oggetto MediaPlayer rimanere nello stato di

avvio. Se la modalità di loop è stata impostata su false invece, passa allo stato di completamento della riproduzione. Mentre è in questo stato, la chiamata start() può riavviare una nuova o la stessa riproduzione. Viene riportato una versione più corta del codice utilizzato per la riproduzione dei brani.

```
//creazione e inizializzazione
public static MediaPlayer mp;
mp = new MediaPlayer();
mp.setAudioStreamType(AudioManager.STREAM_MUSIC);

.....

//preparazione
String path = song.getPath(); //ottieni path
try {
      mp.setDataSource(this, Uri.parse(path));
      mp.prepareAsync();
} catch (Exception e) {
      e.printStackTrace();
}

.....

//avvio e stop
if (mp.isPlaying()) {
      mp.stop();
      mp.reset();
} else {
      mp.start();
}

.....

//distruzione
if (mp != null) {
      mp.release();
}
```

4.2 Comandi player audio

Questa sezione descrive i metodi e comandi di base che compongono il player audio. All'interno del MainActivity vengono dichiarati una serie di metodi statici che saranno richiamati per ogni fragment. Nel capitolo precedente sono stati descritti gli adapters per ogni classe costruita. Come si era detto, il costruttore di ogni classe adapter riceve in ingresso il contesto del fragment, l'ArrayList specifico e un oggetto RecyclerItemClickListener per il click dell'utente. All'interno di ogni fragment viene dichiarato il rispettivo adapter e implementato il metodo onClickListener che riceve in ingresso la specifica classe. All'interno di questo metodo verranno riportati i metodi dichiarati nel MainActivity grazie a un variabile chiamata parent. Viene riportata la dichiarazione del SongAdapter all'interno del fragment song.

```
SongAdapter mAdapter;
MainActivity parent;
RecyclerView recycler;

.....

mAdapter = new SongAdapter(getActivity().getApplicationContext(),
MusicLibrary.songList, new SongAdapter.RecyclerItemClickListener(){
 @Override
  public void onClickListener(Song song, int position){
   //get position
   parent.changeSelectedSong(position);
   //prepare song
   parent.prepareSong(song);
   //playing song
   parent.mainSong();
   //detail song
   parent.pushDetail(getActivity(), song);
   //sliding song
   parent.setImageSliding(song);
   //system audio notification
   parent.setNotification(song);
  }
});

recycler.setAdapter(mAdapter);
```

4.2.1 Positioning Song

Quando l'utente clicca il brano desiderato all'interno della RecyclerView, è necessario salvarsi la posizione poiché risulta utile per i comandi di forward e backward.

```
public void changeSelectedSong(int index) {
 mAdapter.notifyItemChanged(mAdapter.getSelectedPosition());
 currentIndex = index;
 mAdapter.setSelectedPosition(currentIndex);
 mAdapter.notifyItemChanged(currentIndex);
}
```

4.2.2 Preparing Song

Viene mostrato per intero il metodo di preparazione alla riproduzione. Ad ogni click dell'utente vengono impostate le TextView presenti nella Toolbar dello SlidingUpPanel, che cambieranno dinamicamente per ogni brano selezionato.

```
public void prepareSong(Song song) {
 firstLaunch = false;
 currentSongLength = song.getDuration();
 title.setVisibility(View.VISIBLE);
 artist.setVisibility(View.VISIBLE);
 title.setText(song.getTitle()); //ottieni titolo
 artist.setText(song.getArtist()); //ottieni artista
 timeEnd.setText(Utility.convertDuration(song.getDuration()));
 play.setImageDrawable(ContextCompat.getDrawable(context,
 R.drawable.ic_play_circle_outline_white));
 pause_hide.setVisibility(View.INVISIBLE);
 String path = song.getPath(); //ottieni path
 timeStart.setText(Utility.convertDuration(song.getDuration()));
 mp.reset();
 try {
   mp.setDataSource(this, Uri.parse(path));
   mp.prepareAsync();
 } catch (Exception e) {
   e.printStackTrace();
 }
}
```

4.2.3 Playing Song

Dopo la fase di preparazione del brano, la sorgente multimediale è pronta per essere riprodotta. Per avviare il brano deve essere chiamato il metodo start(). Viene utilizzata una progressive bar per gestire l'avanzamento del player: per aggiornare il timer della barra è stato implementato un thread che viene eseguito in background, inoltre è stato necessario costruire un ulteriore classe chiamata Utility per convertire la posizione corrente del MediaPlayer da millisecondi a secondi.

```
mp.setOnPreparedListener(new MediaPlayer.OnPreparedListener(){
 @Override
 public void onPrepared(MediaPlayer mp) {
  if (mp.isPlaying()) {
   mp.stop();
   mp.reset();
  } else {
   tb_title.setVisibility(View.VISIBLE);
   tb_artist.setVisibility(View.VISIBLE);
   mp.start();
   iv_play.setImageDrawable(ContextCompat.getDrawable
   (this, R.drawable.ic_pause_circle_outline_white));
   iv_play2.setImageDrawable(ContextCompat.getDrawable
   (this, R.drawable.ic_pause));

   //progressBar
   final Handler mHandler = new Handler();
   this.runOnUiThread(new Runnable() {
   @Override
   public void run() {
   progressBar.setMax((int) currentSongLength / 1000);
   try {
    int mCurrentPosition = mediaPlayer.getCurrentPosition() / 1000;
    progressBar.setProgress(mCurrentPosition);
    tv_timeStart.setText(Utility.convertDuration((long)
    mediaPlayer.getCurrentPosition()));
    mHandler.postDelayed(this, 1000);
   } catch (Exception e) {
    e.printStackTrace();
   }
  }
});
```

Bottoni Audio

I bottoni implementati sono play, pause, next, previous, repeat e random. Essi sono delle ImageView trattate come bottoni che grazie all'oggetto OnClickListener, catturano l'evento al click dell'utente. Non vengono riportati i bottoni di play e pause in quanto il codice risulterà molto simile al Playing Song.

Next / Previous

Viene riportato il codice relativo al bottone previous. Quando l'utente clicca tale bottone vengono richiamati i metodi descritti precedentemente e viene fatta scalare di una posizione la variabile currentIndex. Il bottone next risuterà identico con l'unica differenza che verrà impostato un currentIndex + 1.

```
public void pushPrevious(){
 iv_previous.setOnClickListener(new View.OnClickListener(){
  @Override
  public void onClick(View v){
   firstLaunch = false;
   if (mp != null) {
    if (!flag_queue) {
     if (currentIndex - 1 >= 0) {
      Song previous = MusicLibrary.songList.get(currentIndex - 1);
      changeSelectedSong(currentIndex - 1);
      prepareSong(previous);
     } else {
      changeSelectedSong(MusicLibrary.songList.size() - 1);
      prepareSong(MusicLibrary.songList.get(
      MusicLibrary.songList.size() - 1));
     }
    }
   }
  }
 });
}
```

Repeat / Random

Il bottone repeat ha lo scopo di ripetere il brano una volta terminato. Questo viene gestito all'interno dei metodi previous e next grazie ad una condizione che verifica una variabile booleana che viene impostata nel seguente metodo. Se la variabile booleana è true allora il brano viene ripetuto finchè l'utente non clicca nuovamente il bottone.

```
public void pushQueue() {
 iv_queue.setOnClickListener(new View.OnClickListener() {
 @Override
  public void onClick(View view) {
   if (!flag_queue) {
     iv_queue.setImageDrawable(ContextCompat.getDrawable
     (context, R.drawable.ic_repeat_one));
     flag_queue = true;
    } else {
     iv_queue.setImageDrawable(ContextCompat.getDrawable
     (context, R.drawable.ic_repeat_white));
     flag_queue = false;
    }
   }
 });
}
```

Grazie alla classe Random è possibile riprodurre i brani in ordine causale. Questo metodo genera una posizione causale dalla dimensione della lista dei brani dandola in ingresso al metodo changeSelectedSong.

```
public void pushRandom() {
 iv_random.setOnClickListener(new View.OnClickListener() {
 @Override
  public void onClick(View view) {
  Random r = new Random();
  int pos = r.nextInt(MusicLibrary.songList.size());
  changeSelectedSong(pos);
  .....
  }
 });
}
```

4.2.4 Cover Song

Il codice seguente permette l'inserimento dell'immagine di cover del brano. Se la cover dell'album è presente allora viene impostata all'ImageView della riga del brano.

```
public void setImageSliding(Song song){
 File imgFile;
 Bitmap art;
 //image album song
 cover_sliding.setImageResource(R.drawable.phonograph);
 for (int i = 0; i < MusicLibrary.albumList.size(); i++){
  if (song.getIdAlbum() == MusicLibrary.albumList.get(i).getID()){
   String path = MusicLibrary.albumList.get(i).getPath();
   try {
    imgFile = new File(path);
    if (imgFile.exists()){
     art = BitmapFactory.decodeFile(imgFile.getAbsolutePath());
     cover_sliding.setImageBitmap(art);
    }
   } catch (Exception e){
     e.printStackTrace();
   }
  }
 }
}
```

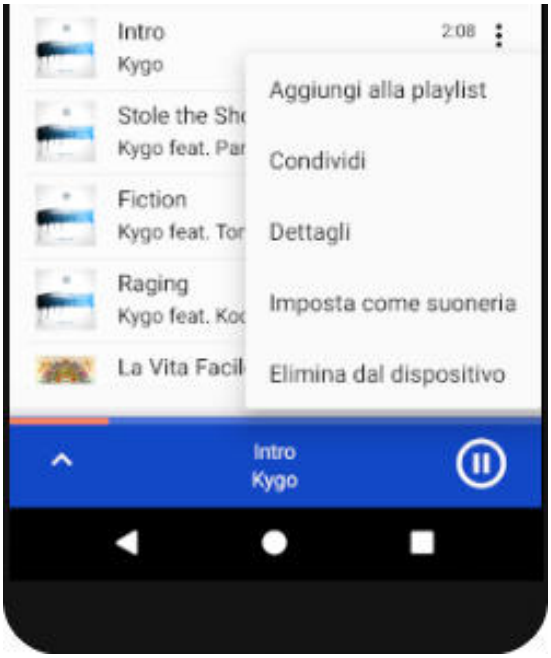
4.2.5 System Audio Notification

È possibile impostare una notifica di sistema che mostra il titolo e l'artista per ogni brano selezionato.

```
public void setNotification(Song song){
 NotificationCompat.Builder mBuilder = (NotificationCompat.Builder)
 new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.ic_music_note)
    .setAutoCancel(true)
    .setContentTitle(song.getTitle())
    .setContentText(song.getArtist());
 NotificationManager notificationManager = (NotificationManager)
 getSystemService(Context.NOTIFICATION_SERVICE);
 notificationManager.notify(0,mBuilder.build());
}
```

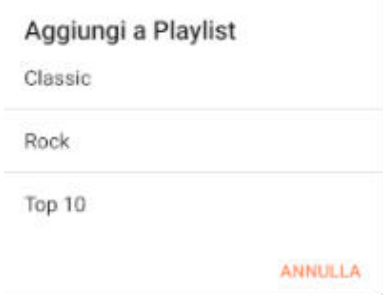
4.3 Opzioni player audio

Le funzionalità scelte per ogni brano sono mostrate nell'immagine seguente. Al click dell'utente sull'icona opzioni presente per ogni brano, si aprirà una finestra con la possibiltà di aggiungerlo ad una specifica playlist, condividerlo o inoltralo, dare una rapida occhiata ai sui dettagli, impostare la suoneria ed eliminarlo dal proprio dispositivo.



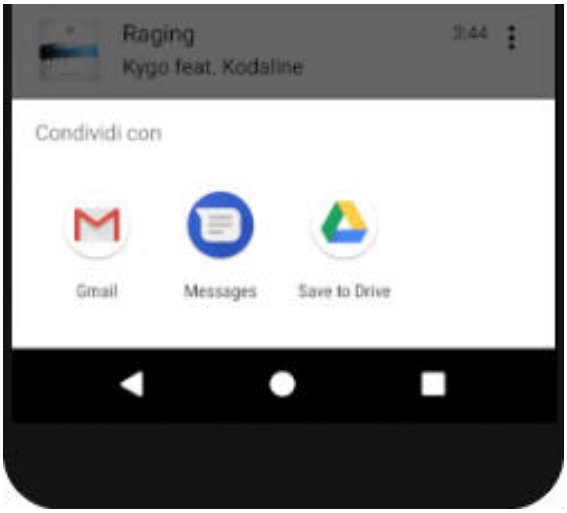
4.3.1 Aggiungi a playlist

Quando l'utente seleziona *Aggiungi alla playlist* si apre un'ulteriore finestra con tutte le playlist presenti. Questa sarà una ListView definita tramite un elemento Adapter.



4.3.2 Condividi

L'implementazione dell'opzione *Condividi* avviene per mezzo di un intent di tipo ACTION_SEND. L'uso più comune dell'azione ACTION_SEND è l'invio di contenuto di testo da un'attività all'altra. Questo è utile per condividere un brano con gli amici tramite e-mail o social network.

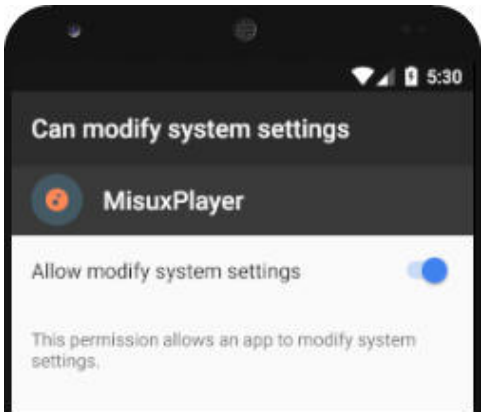


4.3.3 Dettagli

La costruzione di questa finestra contenente le informazioni specifiche per ogni brano è affidata alla classe AlertDialog.Builder. Essa crea un builder per la finestra di avviso permette l'output delle informazioni del brano. Infine viene utilizzata la classe MediaFormat per le informazioni che descrivono il formato dei dati multimediali, il bitrate e la frequenza di campionamento. Vengono utilizzati i metodi definiti nella classe Song per le informazioni standard del brano. Il pulsante Chiudi cancella la finestra di dialogo.

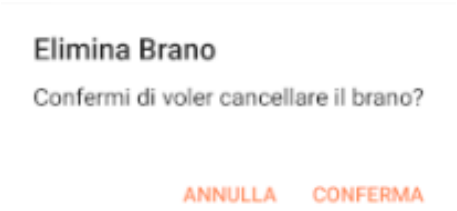
4.3.4 Imposta suoneria

Per impostare la suoneria è necessario concedere i permessi al sistema. Questa schermata appare solo la prima volta quando l'utente seleziona questa opzione. Per impostare la suoneria viene utilizzato l'oggetto RingtoneManager che fornisce l'accesso a suonerie e gestisce l'interrogazione dei diversi fornitori di media.



4.3.5 Elimina dal dispositivo

La costruzione di questa finestra è identica a quella per i dettagli. Se l'utente seleziona il tasto conferma viene richiamato il metodo deleteSong presente nella classe MusicLibrary. Una volta che il brano viene cancellato è necessario utilizzare il metodo clean() per ogni ArrayList, richiamare i metodi di interrogazione audio e fare un refresh dei fragments.




```
public static void deleteSong(Context context, String path) {
 Uri rootUri = MediaStore.Audio.Media.getContentUriForPath(path);
 context.getContentResolver().delete(rootUri,
 MediaStore.MediaColumns.DATA + "=?", new String[]{path});
}
```

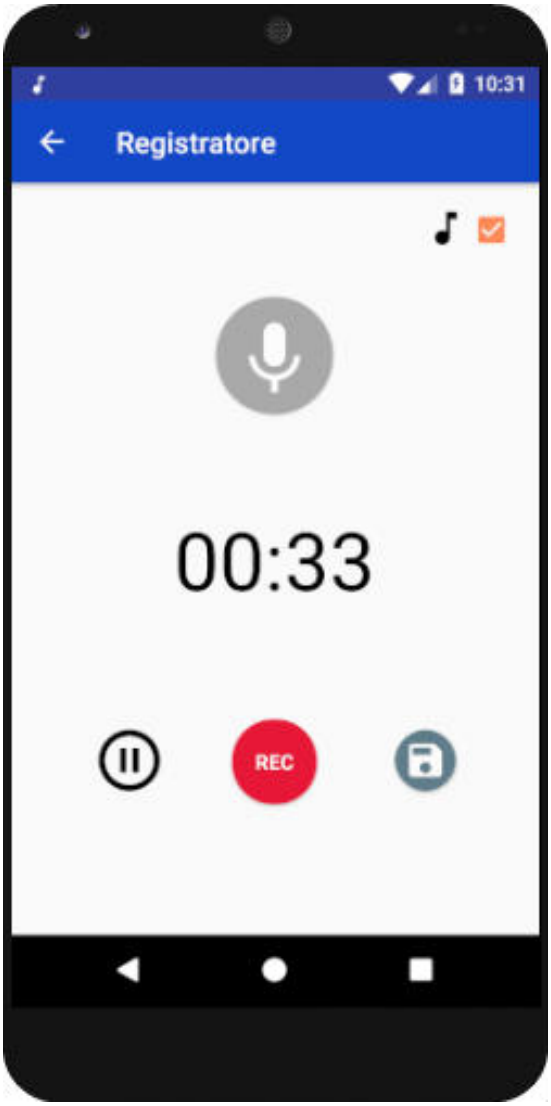
Capitolo 5

Funzioni per la manipolazione audio

5.1 Registratore

Una delle funzionalità più importanti del player audio è la possibilità di registrare in formato MPEG-4 codificato in AMR (Adaptive Multi-Rate) ottimo per il parlato. Si è scelto questo tipo di codec in quanto il registratore di questa applicazione nasce con lo scopo di poter essere utile in quelle occasioni in cui si vuole cantare un brano senza testo o spiegare il brano durante la riproduzione dello stesso. Attivando il bottone di REC è possibile registrare il brano in esecuzione e sopra la propria voce. Questa funzionalità, anche se non di altissima qualità, potrebbe essere utile per mettersi alla prova sul proprio cantato. Il metodo setAudioSource imposta la sorgente audio da utilizzare per la registrazione, in questo caso il microfono del dispositivo. Il metodo setOutputFormat imposta il formato del file di output prodotto e infine il metodo setAudioEncoder imposta l'encoder audio da utilizzare per la registrazione.

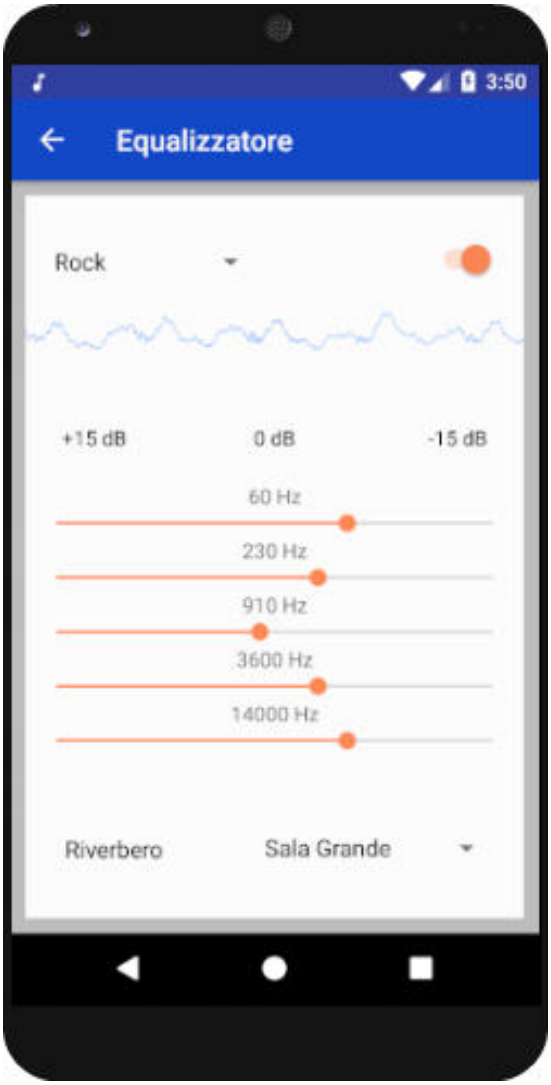
```
public void MediaRecorderReady(){
 mediaRecorder = new MediaRecorder();
 mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
 mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
 mediaRecorder.setAudioEncoder(MediaRecorder.OutputFormat.AMR_NB);
 mOutputFile = getOutputFile();
 mediaRecorder.setOutputFile(mOutputFile.getAbsolutePath());
 recorder.prepare();
 recorder.start();
 recorder.stop();
 recorder.reset();
 recorder.release();
}
```



Il layout presenta tre pulsanti principali: pause, REC e save. I pulsanti pause e save vengono inizialmente disabilitati e attivati una volta che si avvia la registrazione col tasto REC. Il pulsante save memorizza la registrazione all'interno del dispositivo. Selezionando il checkBox è possibile attivare o disattivare il brano in esecuzione durante la registrazione. Infine viene utilizzato un cronometro per la durata della registrazione che lampeggia se viene messo in pausa.

5.2 Equalizzatore

L'equalizzatore viene utilizzato per modificare la risposta in frequenza di una particolare sorgente musicale. Viene creato un oggetto Equalizer nel MainActivity in modo da rendere l'equalizzatore globale per ogni brano. A seconda dello stato dello switch è possibile manipolare l'effetto audio attivando o disattivando l'equalizzazione. L'applicazione può semplicemente utilizzare dei preset predefiniti o avere un controllo più preciso del guadagno in ciascuna banda di frequenza. Per collegare l'equalizzatore a una particolare AudioTrack o MediaPlayer, è necessario specificare l'ID della sessione audio di questi quando si costruisce l'equalizzatore.



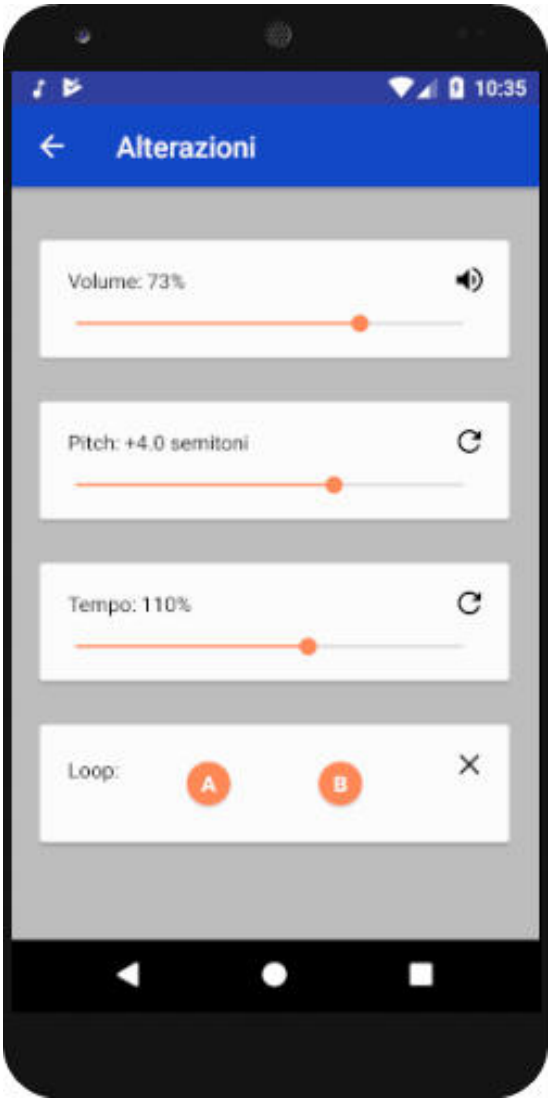
Viene riportata la dichiarazione dell'equalizzatore nel MainActivity. Di seguito, nell'activity Equalizer, troviamo l'inizializzazione dello spinner che conterrà i 9 preset utilizzabili dall'utente. Quando l'utente seleziona un determinato preset, la posiziona di esso viene dato in ingresso al metodo usePreset. Successivamente vengono create 5 seekbar che fanno riferimento alle bande di frequenza del preset scelto. A questo punto è anche possibile regolare dinamicamente la frequenza personalizzando il preset.

```
//MainActivity
public static android.media.audiofx.Equalizer mEqualizer;
public static android.media.audiofx.PresetReverb pReverb;
mEqualizer = new android.media.audiofx.Equalizer(0,
MainActivity.mediaPlayer.getAudioSessionId());

//Equalizer
equalizerPresetSpinner = (Spinner) findViewById(R.id.spinner);
equalizerPresetSpinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
 @Override
 public void onItemSelected(AdapterView<> parent,
  View view, int position, long id) {
  MainActivity.positionEqualizerSpinner = position;
  MainActivity.mEqualizer.usePreset((short) position);
  short numberFrequencyBands = MainActivity.mEqualizer
  .getNumberOfBands();
  short lowerEqualizerBandLevel = MainActivity.mEqualizer
  .getBandLevelRange()[0];
  for (short equalizerBandIndex = 0;
   equalizerBandIndex < numberFrequencyBands;
     equalizerBandIndex++) {

  SeekBar seekBar = (SeekBar) findViewById(equalizerBandIndex);
  seekBar.setProgress(MainActivity.mEqualizer.getBandLevel(
  equalizerBandIndex) - lowerEqualizerBandLevel);
  }
 }
});
```

È possibile inoltre impostare un riverbero al brano grazie al metodo setPreset che riceve in ingresso in modo analogo alla frequenza, un preset predefinito della classe PresetReverb.

5.3 Alterazioni

Le funzioni di alterazioni presenti all'interno dell'applicativo sono visibile nell'immagine seguente. Attraverso delle seekbar è possibile alterare dinamicamente il brano in tempo reale. Si può modificare la velocità dei file audio senza influire sull'intonazione o modificare il tono senza modificare la velocità. In alternativa, è possibile regolare entrambi insieme. Una delle funzioni più importanti è quella di Loop musicale, particolarmente utile per un musicista che ha la necessità di riascoltare lo stesso frammento di brano più volte. Anche le funzioni di Pitch shifting e Time stretching possono essere molto utili per chi ha bisogno di rallentare o aumentare il tempo o esercitarsi in un'intonazione diversa.



5.3.1 Volume

Regolare il volume direttamente dall'applicazione potrebbe risultare più veloce rispetto al controllo con i tasti dal dispositivo soprattutto per silenziare il brano al semplice click. La classe AudioManager fornisce l'accesso al controllo del volume. Viene inizializzata una seekbar che ottiene la posizione corrente di volume dal sistema. Grazie al metodo setStreamVolume è possibile imposta dinamicamente il volume dalla posizione della seekbar. Al click sull'icona silent mode viene impostata la modalità silenziosa dando in ingresso al metodo setStreamVolume RINGER_MODE_SILENT.

```
volume = (SeekBar) findViewById(R.id.seekBar_volume);
value_volume = (TextView) findViewById(R.id.perc_volume);
clear_volume = (ImageView) findViewById(R.id.clear_volume);
.....
aManager=(AudioManager)getSystemService(Context.AUDIO_SERVICE);
int currentV=aManager.getStreamVolume(AudioManager.STREAM_MUSIC);
int maxV=aManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
volume.setMax(maxV);
volume.setProgress(currentV);
.....
volume.setOnSeekBarChangeListener(new OnSeekBarChangeListener(){
 @Override
 public void onProgressChanged(SeekBar arg0, int progress,
 boolean arg2) {
  aManager.setStreamVolume(AudioManager.STREAM_MUSIC,progress,0);
  int perc=(100*progress)/15;
  String x=Integer.toString(perc) + "%";
  value_volume.setText(x);
 }
});

//silent mode
clear_volume.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
  int currentV = aManager.getStreamVolume(AudioManager.STREAM_MUSIC);
  aManager.setStreamVolume(AudioManager.RINGER_MODE_SILENT, 0, 0);
  volume.setProgress(0);
 }
});
```

5.3.2 Pitch shifting

La funzione di Pitch shifting cambia la variazione di frequenza della nota musicale, riducendola o aumentandola di una certa quantità. La variazione implementata segue un andamento lineare del numero di semitoni, dati dalla posizione della seekbar.

```
pitch = (SeekBar) findViewById(R.id.seekBar_pitch);
pitch.setMax(24);
pitch.setOnSeekBarChangeListener(new OnSeekBarChangeListener(){
 @Override
 public void onProgressChanged(SeekBar seekBar, int progress,
 boolean fromUser) {
  float myfloat = (float) progress; //cast in float
  params.setPitch((float) Math.pow(a, myfloat-12));
  mp.setPlaybackParams(params);
  String x = Float.toString(myfloat-12).concat(semitoni);

  if(myfloat-12>0) value_pitch.setText(plus.concat(x));
  else value_pitch.setText(space.concat(x));
  if(myfloat-12==0) value_pitch.setText(space.concat(s));
 }
});
```

5.3.3 Time stretching

Il Time streatching aumenta o diminuisce il tempo di riproduzione audio. È espresso come un fattore moltiplicativo, in cui la velocità normale è 1.0f. È possibile variare la velocità dell'audio dal 50 al 150% della velocità originale.

```
speed = (SeekBar) findViewById(R.id.seekBar_speed);
speed.setMax(10);
speed.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
 @Override
 public void onProgressChanged(SeekBar seekBar, int progress,
 boolean fromUser) {
  double y = (double) progress/10;
  float s = (float) y + 0.5;
  mp.setPlaybackParams(mp.getPlaybackParams().setSpeed(s));
  String k = Integer.toString((progress*100/10) + 50).concat(perc);
  value_speed.setText(space.concat(k));
 }
});
```

5.3.4 Loop

La funzione di loop permette la ripetizione continua del brano. La gestione del loop avviene nel thread runnable della progressive bar. Vengono dichiarati due bottoni: al click dell'utente sul primo bottone (btnA), viene salvata la durata corrente del brano, nella variabile currentPositionA. Una volta che l'utente seleziona il secondo bottone (btnB), viene salvata anche in questo caso, la durata corrente del brano nella variabile currentPositionB e grazie al metodo seekTo viene riportato l'oggetto MediaPlayer alla posizione di A; All'interno del metodo run del thread della progressive bar, viene aggiunta una condizione che verifica se i due bottoni sono selezionati. Se questi sono selezionati, una volta che il MediaPlayer supera la posizione di B, viene riportato ad A così fino alla disattivazione del loop.

```
public static int currentPositionA=-1, currentPositionB=-1;
Button btnA, btnB;
.....
btnB.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View view) {
  if(isLoop) {
   btnB.setEnabled(false);
  } else {
   btnB.setCardBackgroundColor(getResources().getColor(R.color.acc));
   btnB.setEnabled(false);
   currentPositionB = mp.getCurrentPosition();
   mp.seekTo(currentPositionA); //ricomincia da A
   isLoop = true;
  }
 }
});

//condition progress bar thread
if(mediaPlayer.getCurrentPosition()>Alteration.currentPositionB
 && Alteration.currentPositionB!=-1) {
 mediaPlayer.seekTo(Alteration.currentPositionA); //ricomincia da A
 Alteration.isLoop=true;
}
```

Bibliografia

[1] Google, Android Developers: https://developer.android.com/guide/index.html, "Web".

[2] C. De Sio Cesari, Manuale di Java 8. Programmazione orientata agli oggetti con Java standard, 8rd edition, Ulrico Hoepli Milano, 28 mag 2014, "Web".

[3] M. Carli, Android 6. Guida per lo sviluppatore, Apogeo - Idee editoriali Feltrinelli s.r.l., 28 gen 2016, "Web".

[4] C. Haseman, Creare App per Android. Progettazione e sviluppo, Sperling & Kupfer, 4 apr 2017, "Web".

[5] F. Collini, M. Bonifazi, A. Martellucci, S. Sanna, Android: Programmazione avanzata, Edizioni Lswr, mag 2015, "Web".

[6] P. Camagni, R. Nikolassy, E. Falzone, Sviluppare App per Android, Hoepli, 8 ago 2017, "Web".

[7] E. Cisotti, M. Giannino, Android: Guida completa, DigitalLifyStyle, apr 2015, "Web".

[8] D. Piccinelli, "IDC: duopolio Android e iOS negli smartphone, agli altri rimangono le briciole", macitynet.it, Macity Publishing srl, 25 feb 2015, "Web".

[9] G. Congiu, "Samsung è nuovamente il primo produttore di smartphone al mondo nel Q1 2017", HDBLOG.it, "n.p.", 11 apr 2017

[10] G. Maggi, M. Lecce, "L'SDK e l'ambiente di sviluppo.", HTML.it, Roma, "n.p.", 15 nov 2017, "Web".

[11] G. Maggi, "Il ciclo di vita di un'Activity.", HTML.it, Roma, "n.p.", 28 mag 2014, "Web".

[12] G. Maggi, "Fragment in Android.", HTML.it, Roma, "n.p.", 29 set 2016, "Web".

Ringraziamenti

Arrivato a questo punto molto importante della mia carriera sento il desidero di ringraziare tutti coloro che mi hanno aiutato e sostenuto durante questo percorso. Un primo ringraziamento va alla mia famiglia per avermi supportato in questi anni di studio. Ringrazio i Professori Adriano Baratè e Luca Andrea Ludovico, sempre disponibili a dirimere i miei dubbi durante tutto il periodo di stesura di questo lavoro. Un ringraziamento speciale ai miei compagni e amici di università che hanno contribuito alla mia crescita personale per i numerosi consigli informatici e non solo.