JOHANNES KEPLER
UNIVERSITÄT LINZ

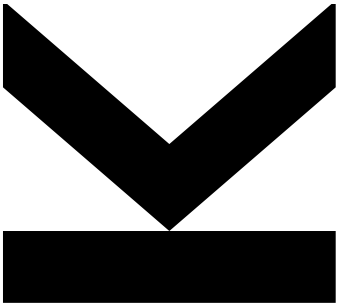
Eingereicht von
Baasanjav Jargal

Angefertigt am
Institut für Systemsoftware

Betreuer
Prof.Dr. Hanspeter Mössenböck

März 2023

MusicLib - Eine Android-App zur Kategorisierung von Musiksammlungen auf SmartPhones



Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Bachelorstudium

Informatik

JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Abstract

A 'Music Player' application for smartphones is the focus of this bachelor's thesis. It's crucial to use the right IDE when developing an app if you want to work efficiently and quickly. And Android Studio is becoming increasingly popular with developers. Kotlin is used as a programming language due to its extensive advantages. Before the app is run, the music collections on the smartphone should be tagged using a music editor. Users can organize albums containing music tracks according to specified categories and search for music tracks or albums using this app.

Kurzfassung

Diese Bachelorarbeit befasst sich mit der Entwicklung einer 'Music Player' Applikation für Smartphones. Bei der Entwicklung einer App ist es wichtig, eine passende IDE zu verwenden, um schnell und effizient arbeiten zu können und Android Studio hat unter Entwicklern an Popularität gewonnen. Aufgrund seiner umfangreichen Vorteile wird Kotlin als Programmiersprache verwendet. Vor dem Ausführen der App sollten die Musiksammlungen auf dem Smartphone mithilfe eines Musikeditors (z.B. MP3Tag) getaggt werden. Die implementierte App ermöglicht es Benutzern ihre Alben, die aus Musikstücken bestehen, nach angegebenen Kategorien (Tags) zu ordnen und Musikstücke bzw. Alben nach Tags zu suchen und anzuzeigen.

1 Einleitung und Motivation

Seitdem die ersten Smartphones der Welt vorgestellt wurden, sind fast dreißig Jahren vergangen. Heutzutage sind Smartphones aus dem Alltagsleben fast nicht mehr wegzudenken. Außer Handys dient Menschen die Kopfhörer als tägliche Begleitung. Über Kopfhörer kann man nicht nur telefonieren, sondern auch Musik überall hören, ohne andere zu stören.

Durch Musik befindet man sich in seiner eigenen kleinen Welt und kann dadurch andere Alltagsgeräusche ausschalten. Einige Leute können Musik nicht besonders leiden, im Vergleich zu anderen hören sie sie nicht gern. Jeder hat einen anderen Geschmack, dies gilt auch für Musik.
Die MusikhörerInnen hören gerne bestimmte Musikgenres oder bestimmte KünstlerInnen. Im Bereich der Musik gibt es umfangreiche verschiedene Musikrichtungen (später Musikgenres oder Genres genannt), die Stilrichtungen der Künstler beschreiben. Die beliebtesten Genres: Pop, Schlager, Klassik, Rock, Elektro, Rap, Hip-Hop usw. Wer gerne öfters Musik hört, hat kleine oder große Musiksammlungen auf seinem Handy und möchte sie ordnen, um bestimmte Musikstücke öfters zu hören. Dafür braucht man geeignete Musik Player auf seinem Smartphone.

Es gibt viele Music Player Applications für Android-Smartphones. Diese haben eigene Vorteile und Funktionen. Aber es gibt keine Music Player, mit denen man Musiksammlungen nach Genre-Tags ordnen und durchsuchen kann. Music Player, die Musikstücke nach Genres ordnen, bieten keine Auswahl von Musikstücken, die ein oder mehrere Genre-Tags aufweisen zu. Dieses Problem soll mit der implementierten App gelöst werden.

Das erste Kapitel widmet sich der Einleitung des Projekts. Kapitel 1 beschreibt die Motivation für das Projekt. Bei der Umsetzung des Projekts kommen verschiedene Hilfsmittel und Technologien zum Einsatz. Daher wird im zweiten Kapitel die verwendete Programmiersprache Kotlin sowie Android Studio beschrieben, auf deren Inhalten die App aufbaut. Die LibVLC API und der Aufbau von MP3-Dateien und WMA-Dateien werden kurz erläutert. Im dritten Kapitel handelt es sich um die Programmablauf. Wie die App funktioniert, wird es mithilfe der umgesetzten Sichten der App detailliert erklärt. Das vierte Kapitel widmet sich der Implementierung des Projekts. Wie die Android Komponenten auf Codeebene verwendet werden, wird in diesem Kapitel zusammengefasst. Die technische Daten werden im fünften Kapitel erläutert, da sich zusätzlich mit der Testumgebung befasst. Zum Schluss wird diskutiert, welche Erweiterungen noch durchgeführt werden könnten. Die Probleme, die bei der Implementierung aufgetreten sind, werden ebenfalls angedeutet.

2 Grundlagen

Dieses Kapital gibt einen Einblick in die in diesem Projekt verwendete Entwicklungsumgebung und Programmiersprache.

2.1 Kotlin

Bei der Entwicklung einer Android-App kommen Java und Kotlin als Programmiersprache in erster Linie infrage. Android-Apps werden traditionell in Java programmiert, aber seit Mai 2019 ist Kotlin die bevorzugte Programmiersprache von Google [**b1**]. Kotlin kommt bei der Entwicklung dieser App zum Einsatz.

Sowohl Java als auch Kotlin werden von der offiziellen Android-Dokumentation und den Schulungsressourcen unterstützt. Standardmäßig werden Kotlin-Codebeispiele bereitgestellt, alternativ stehen Java-Codebeispiele zur Verfügung.
Die Programmiersprache Kotlin basiert teilweise auf Java. Beim Erstellen einer Android-App wird Kotlin-Code in Java-Code transkompiliert und von der Java Virtual Machine (JVM) als Java-Bytecode ausgeführt. Von den Sprachkonstrukten her ähnelt Kotlin Java, erweitert es aber um einige Features aus anderen modernen Programmiersprachen. Eine häufige Exception in Java ist die Null-Pointer-Exception, die in Kotlin selten vorkommt, da Kotlin zwischen nullbaren und nicht nullbaren Typen unterscheidet. Der Entwickler muss in diesem Fall das Verhalten ändern, um eine Variable explizit als null zu deklarieren.
In Kotlin kann man z.B. Mapping-, Filter- und Iterator-Funktionen mit deutlich weniger Zeilen Code schreiben. Laut der Dokumentation [**b18**] behebt Kotlin Probleme, die noch nicht bei Java gelöst wurden: 1) Nullreferenzen werden vom Typsystem gesteuert, 2) Keine raw Typen, 3) Arrays sind unveränderlich, 4) Kotlin hat die richtige Art von Funktionen, im Gegensatz zu Javas SAM-Konvertierungen, 5) Use-Site-Varianz ohne Platzhalter und 6) Kotlin hat keine geprüften Ausnahmen. Im Vergleich zu Java hat Kotlin viele moderne Sprachkonstrukte und nützliche Neuerungen integriert, wodurch die Arbeit einfacher und effizienter wird.

Entwickler können auf folgende Weise von Kotlin für die Android-Entwicklung profitieren [**b2**]:

- Der Code ist weniger komplex und besser lesbar
- Sprach- und Umweltreife: Nicht nur als Sprache, sondern als ganzes Ökosystem mit robusten Toolsets hat sich Kotlin im Laufe der Jahre kontinuierlich weiterentwickelt. Viele Unternehmen nutzen diesen Tool jetzt aktiv für die Entwicklung von Android-Applikationen, da es nahtlos in Android Studio integriert ist.

- Kotlin assistiert Android Jetpack und anderen Bibliotheken
- Kompatibilität mit Java: Ohne ihren gesamten Code auf Kotlin transferieren zu müssen, können Sie Kotlin neben Java in der App verwenden.
- Die Möglichkeit, auf mehreren Plattformen zu entwickeln: nicht nur Android sondern auch IOS, Backend und Web Applikation
- Sicherheit: Das Erkennen von Fehlern macht Kotlin-Code sicher.
- Einfach zu lernen.
- Viele Benutzer: Da die Kotlin-Community auf der ganzen Welt wächst, hat sie breite Unterstützung und viele Beiträge. Kotlin wird von mehr als 60 Prozent der Top-1000-Apps im Google Play Store verwendet.

Da Kotlin ein neuer, offizieller Standard für die Entwicklung von Android-Apps ist und das Schreiben von Quellcode in Kotlin viel schneller und einfacher ist, wird Kotlin Java vorgezogen.

2.2 Gradle [b16][b18]

Gradle ist das Android-Buildsystem. Android Apps müssen viele externe Bibliotheken einbinden, damit die Ziele des Projekts eingehalten werden können. App-Resourcen und Quellcode werden von Gradle kompiliert und in APKs (Application Package Kit) oder Android-App-Bundles verpackt, damit sie auf dem Testgerät ausgeführt werden können. Dies kann mit nur einem Klick auf Android Studio gebaut werden. Um eine APK zu generieren, muss sich eine Manifest-Datei im Root-Verzeichnis des Projektquellsatzes befinden. Die Manifestdatei wird von den Android-Build-Tools, dem Android-Betriebssystem und Google Play verwendet. In Abschnitt 2.4.2 wird erläutert, welche Informationen man für die Entwicklung einer App benötigt. Es ist übrigens möglich, bei der Verwendung vieler Bibliotheken die notwendigen Änderungen am Gradle-Buildskript der App vorzunehmen.

2.3 Android Framework

Als Android App macht der Music Player eingehenden Gebrauch vom Android Java API Framework, das auf die Systemdienste von Android zugreift. Das View-System, die Ressourcen- und Activity-Manager gehören zum Beispiel dazu. Die Funktionen zur Abbildung der Benutzeroberflächen werden von View-System bereitgestellt, während der Ressourcen-Manager die Designs zur Verfügung stellt. Der Activity-Manager gibt Informationen über die Komponenten und ist dafür zuständig, dass Activities aufeinander wirken können. Außer den bereits

genannten Komponenten kommen noch die Shared Preferences zum Einsatz. Sie dienen der Speicherung der Einstellungen der App und der benutzerdefinierten bzw. geänderten Daten.

2.4 Android Studio

Die Music Player App ist speziell auf das Betriebssystem Android ausgerichtet. Die Verwendung einer gut integrierten Entwicklungsumgebung (IDE) kann sich die benötigte Zeit des Entwicklers verkürzen. Dadurch kann sich die Produktivität des Entwicklers verbessern. Die IDE ist ein Softwareprogramm oder eine Kombination von Tools, die Entwickler zum Codieren und Überprüfen des Codes benötigen.

2.4.1 Überblick

Bei der Entwicklung der Music Player App kommt Android-Studio zum Einsatz. Android-Studio ist eine beliebte und freie IDE von Google und wurde am 16.Mai 2013 angekündigt. Es ist für Betriebssysteme wie Windows, macOs und Linux, verfügbar [**b3**].
Beim Erstellen von Android-Apps bietet Android Studio noch mehr Funktionen, um die Arbeit zu erleichtern, darunter [**b4**]:

- Gradle-basiertes Build-System
- Ein schneller und funktionsreicher Emulator
- Eine Android-Entwicklungsumgebung, mit der Code für alle Android-Geräte entwickelt werden kann.
- Die Änderungen können auf laufende Apps angewendet werden, ohne sie neu zu starten
- Enorme Testwerkzeuge und Frameworks
- Leistung, Benutzerfreundlichkeit, Kompatibilität und andere Probleme können mit Lint-Tools angepasst werden

Die IDE unterstützt das Design durch einen grafischen Editor für Benutzeroberflächen und bietet noch mehr Hilfsfunktionen, um die Produktivität der EntwicklerInnen beim Erstellen von Android Apps zu steigern. Dazu gehören ein Editor, einDebugger, ein Compiler, Sprachunterstützung, ein Emulator und vieles mehr. Android-Studio basiert auf leistungsstarken Tools und dem Code-Editor von IntelliJ IDEA.
Der Layout-Editor ermöglicht, dass der Entwickler UI-Elemente in einen visuellen Design-Editor entwickeln kann. Die Größe der Layouts kann vom EntwicklerInnen geändert werden, damit sie absolut an die Bildschirmgröße des Smartphones angepasst werden können. Damit werden Layouts schneller aufgebaut, ohne Code zu schreiben.

Gradle, das Android-Buildsystem, bietet gute Unterstützung für das Abhängigkeitsmanagement. Es ist auch möglich, Abhängigkeiten direkt aus dem Quellcode-Editor hinzuzufügen, ohne das Build-Skript zu bearbeiten. Hier reicht es, die benötigte Klasse der hinzugefügten Bibliothek in den Quellcode zu importieren. Damit wird diese installiert und das Softwareprojekt wird neu aufgebaut. Syntaxhervorhebung und Textvervollständigung werden für Java und Kotlin sowie für importierte Bibliotheken und das Android-Framework beim Schreiben von Quellcode, unterstützt.

Alternativen zu Android Studio sind Eclipse und NetBeans. Beide IDEs bieten Erweiterungen für die Kotlin- und Android-Entwicklung. Beide stellen zwar eine gewisse Unterstützung für die Entwicklung von Android-Apps bereit, jedoch ist sie eingeschränkter als in der Android Studio IDE. Wie bereits erwähnt, bieten IDEs von Android Studio eine bessere Unterstützung für Kotlin-Sprachkonstrukte und -funktionen, als die beiden anderen Entwicklungsumgebungen.

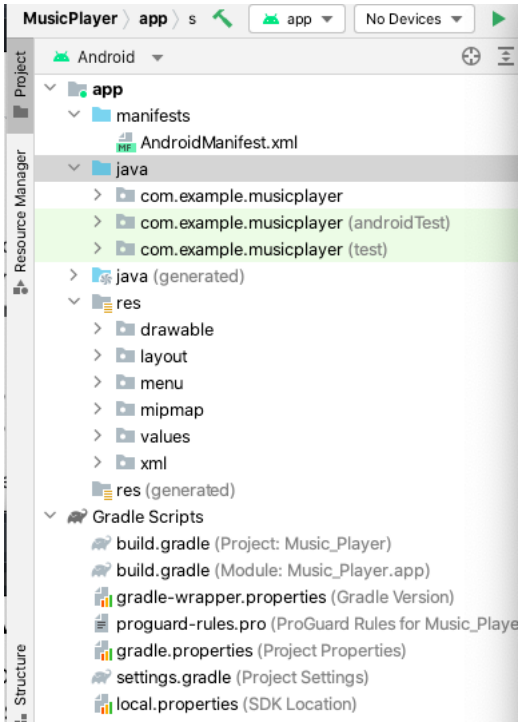


Figure 2.1: Android Projektansicht

Abbildung 2.1 zeigt, wie die Projektdateien in der Android-Projektansicht angezeigt werden. Um einen schnellen Zugriff auf die wichtigsten Quelldateien des Projekts zu ermöglichen, ist diese Projektansicht nach Modulen organisiert. Unter Gradle-Skripts sind alle Build-Dateien zu sehen. Jedes Projekt enthält[**b4**]:

- **manifests:** enthält die AndroidManifest.xml Datei.
- **java:** enthält Java- oder Kotlin-Quellcodedateien.

- **res:** enthält alle XML-Layouts, UI-Strings und Bitmap-Bilder.

2.4.2 Komponenten[b5]

Jede Komponente der Applikation ist ein Einstiegspunkt, über den das System oder ein Benutzer zugreifen kann. Zwischen einigen Komponenten bestehen Abhängigkeiten. Die Kernkomponenten des Frameworks lauten:

Activities

Eine Activity ist der Kontaktpunkt zwischen dem Benutzer und der Applikation. Die Activity stellt ein Fenster bereit, in dem die App eine Benutzeroberfläche anzeigt. Jede Activity in einer App implementiert einen Bildschirm. Es gibt normalerweise mehrere Bildschirme in einer App. Das bedeutet, dass es mehrere Activities darin gibt. Mainactivity einer App ist der erste Bildschirm, den der Benutzer sieht, wenn die App gestartet wird.
Für jede Activity kann eine andere Aktion ausgeführt werden, indem eine andere Activity begonnen wird. Die Activities in einer App arbeiten zusammen, um ein zusammenhängendes Benutzererlebnis zur Verfügung zu stellen, aber sie sind nur lose miteinander verbunden; sie sind oft unabhängig voneinander.
Um Activities in einer App zu verwenden, sind sowohl Lebenszyklen der Activities zu verwalten als auch die Activities im Manifest der App zu registrieren. Mehrere Rückrufe werden von der Activity-Klasse bereitgestellt, damit die Actitivity weiß, wann sich ein Zustand geändert hat: zum Beispiel, wenn das System die Activity startet, stoppt oder fortsetzt oder den Prozess beendet, in dem sie ausgeführt wird.
Die Lebenszyklus-Callback-Methoden definieren, wie sich die Activity verhält, wenn der Benutzer sie verlässt und zu ihr zurückkehrt. Um die Übergänge zwischen den Phasen des Aktivitätslebenszyklus zu navigieren, stellt die Activity-Klasse die folgenden sechs Callbacks bereit: onCreate( ), onStart( ), onResume( ), onPause( ), onStop( ) und onDestroy( ).

Abbildung 2.2 zeigt die Lebenszyklen der Aktivitäten. Die Komplexität der Activities erfordert möglicherweise nicht die Implementierung aller Lebenszyklusmethoden [**b6**].

Aktivitätszustände können durch eine Vielzahl von Ereignissen geändert werden. Falls Konfigurationen geändert werden, werden Aktivitäten beendet und neu aufgebaut. Die folgenden Rückrufe werden auf der ursprünglichen Activity Objekte ausgelöst:

1. onPause() - Diese Methode wird aufgerufen, wenn der Benutzer die Aktivität verlässt, was bedeutet, dass die Aktivität nicht mehr im Vordergrund ist. Wenn der Zustand aus dem ‚Paused' in den ‚Resumed' zurückkehrt, wird onResume() aufgerufen. Das System kann onStop() aufrufen, wenn die Aktivität absolut unsichtbar ist.

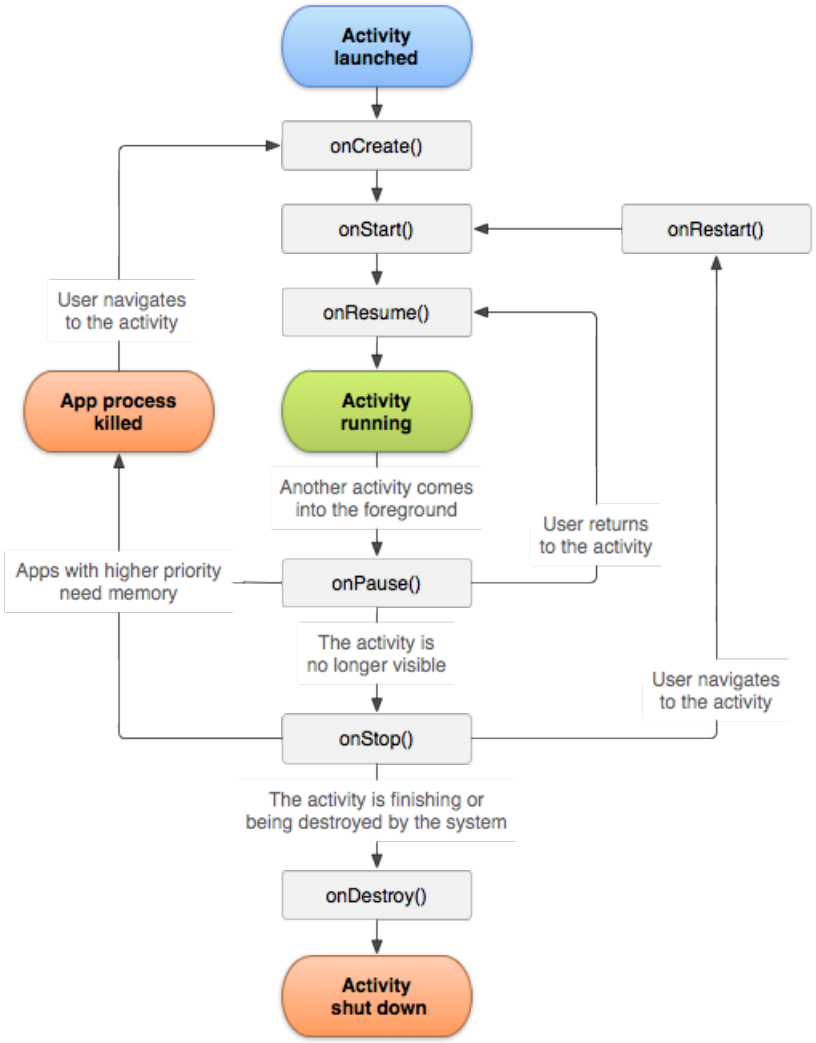


Figure 2.2: Lebenszyklen der Aktivitäten

2. onStop()- Da wird zum Zustand ‚Stopped' gewechselt, und die Ressourcen, die nicht mehr benötigt werden, werden freigegeben oder angepasst. Falls die Aktivität zurückkommt, wird onRestart() aufgerufen. Wenn sie beendet wird, ruft das System onDestroy() auf.

3. onDestroy() - wird aufgerufen, bevor die Aktivität beendet wird. In diesem Fall werden alle Ressourcen freigegeben.

Die folgenden Rückrufe werden ausgelöst, wenn ein neues Activity Objekt erzeugt wird:

1. onCreate() - wenn das System die Aktivität neu aufbaut. Nachdem onCreate() ausgeführt wurde, ruft das System onStart() und OnResume() auf, da die Aktivität zum Zustand ‚Started' gewechselt hat.

2. onStart() - macht die Activity für den Benutzer sichtbar. Der Zustand ist ‚Created'.

Wenn die Activity in den Vordergrund kommt, dann wird onResume() aufgerufen.

3. onResume() - wenn die Activity im Zustand ‚Resumed' ist, bleibt die App in diesem Zustand, bis etwas passiert, z.B: der Benutzer navigiert zu einer anderen Activity. Wenn ein unerwartetes Ereignis eintritt, dann wechselt die Activity zum Zustand ‚Paused' und es wird onPause() aufgerufen [**b7**].

Services

Eine App kann mithilfe von Services im Hintergrund ohne aktive Nutzeroberfläche ausgeführt werden. Bei lang andauernden Vorgängen oder zum Ausführen von Arbeiten ausgelagerter Prozesse werden Services im Hintergrund ausgeführt. z.B: Während sich der Benutzer in einer anderen App befindet, spielt App-Service vom Music Player möglicherweise Musik im Hintergrund ab.
Es gibt zwei Services, die dem System mitteilen, wie die App verwaltet werden soll:

- **Started services** stellen sicher, dass das System weiterläuft, bis seine Arbeit fertig ist. 'Started services' stellen zwei verschiedene Arten dar, wie das System mit ihnen umgeht: Mithilfe einer Benachrichtigung kann die App im Vordergrund imformiert werden, dass ein Hintergrunddienst immer noch läuft. Die Hintergrunddienste laufen im Hintergrund und sind für den Benutzer nicht sichtbar, sodass das System mehr Freiheit bei der Verwaltung seines Prozesses hat.
- **Bounded services** - laufen, solange eine andere App oder das System diese Services nutzen möchte. Dies bietet eine Art von Server-Client-Schnittstelle. Das Service wird ausgeführt, solange es an andere Komponenten gebunden ist.

-

Content Provider

Der Content Provider verwaltet die Daten, die im Dateisystem, in einer SQLite-Datenbank, im Web oder an anderen dauerhaften Speicherorten gespeichert werden können. Die App kann auf den Ort zugreifen. Der ContentProvider bietet dabei auch gewisse Sicherheit für die Daten. Die ContentResolver-Schnittstelle gruppiert die Daten und stellt sie der Anwendung zur Verfügung. Wenn der ContentResolver eine Anfrage über die URl erhält, wird die Berechtigung vom System geprüft und die Anfrage an den Content Provider weitergeleitet. Dies ermöglicht eine indirekte Kommunikation zum Content Provider. Der ist auch geeignet, um Daten zu lesen und zu schreiben, die nicht geteilt werden und privat für die App sind.

Intent

Intents stellen Nachrichtenobjekte dar und binden einzelne Komponenten zur Laufzeit aneinander. Intents gliedern sich in zwei Typen:

- **Explizite Intents** - werden verwendet, um eine Komponente in der App zu starten. Da sind die Informationen über den Klassennamen der Activity oder des Services bekannt.
- **Implizite Intents** - versuchen eine möglichst allgemeine Aktion zu erzeugen, die auf eine Komponente aus einer anderen App zugreift.

Es gibt separate Methoden um jeden Komponententyp zu aktivieren:

- Eine andere Activity kann gestartet werden, um ein Ergebnis bzw. Daten zu erhalten, die in einem Intent von der vorherigen Aktivität gespeichert werden.
- Ein Service kann gestartet werden, indem ein Intent an startService() übergegeben wird. Diese Komponente führt Hintergrundoperationen aus.
- Ein Intent kann dadurch einen Broadcast empfangen werden. Hier werden die Methoden sendBroadcast(), sendOrderedBroadcast() oder sendStickyBroadcast() verwendet.

Manifest

Alle Komponenten der App müssen in der AndroidManifest.xml deklariert werden, damit das System über die Komponenten der App informiert werden kann. Dies beschreibt alle grundsätzliche Informationen über die App für die Android-Build Tools und das Android Betriebssystem. Android kann eine App-Komponente starten, indem das System ihre Manifestdatei liest.
In der Manifestdatei wird folgendes gespeichert:

- Die Komponenten der App, z.B: Activities, Services, Broadcast receivers und Content Provider. Für jede Komponente der App muss ein entsprechendes XML-Element in der Manifestdatei deklariert werden. Ohne eine Deklaration kann das System nicht starten.
- Die erforderliche Berechtigungen, die die App benötigt, um auf geschützte Teile des Systems oder auf sensible Benutzerdaten zuzugreifen.
- Die benötigte API-Level
- Die verwendeten bzw. benötigte Hardware- und Softwarefunktionen
- Die API-Bibliotheken, mit denen die App verknüpft werden muss [9]

2.4.3 Benutzeroberflächen

Für die Darstellung der Benutzeroberflächen stellt Android gewisse Anforderungen. Bevor die Entwickler ihr App-Konzept schreiben, sollten sie das Design der grafischen Benutzeroberfläche zuerst entwerfen. Die Benutzeroberfläche bildet den Kernpunkt des App-Designs; als Hauptsichten dienen z.B die Activities in der App. Für die Benutzeroberflächen müssen verschiedene Aufgaben erledigt werden. Zum Beispiel müssen die hierarchischen Strukturen jedes Bildschirms in XML-Dateien festgelegt werden.

Layout und View

Android bietet ein leistungsstarkes komponentenbasiertes Modell, mit dem Benutzeroberfläche aufgebaut werden können, basierend auf den grundlegenden Layout-Klassen: View und ViewGroup. Layouts bzw. Widgets sind Unterklassen von View und ViewGroup und sind die Grundbausteine einer Benutzeroberfläche. Alle Elemente im Layout werden mithilfe einer Hierarchie von View und ViewGroups gebildet. Eine unvollständige Liste von den Widgets umfasst Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner und die speziellen AutoCompleteTextView, ImageSwitcher und TextSwitcher. Die meisten davon spielen in der Music Player App eine große Rolle.

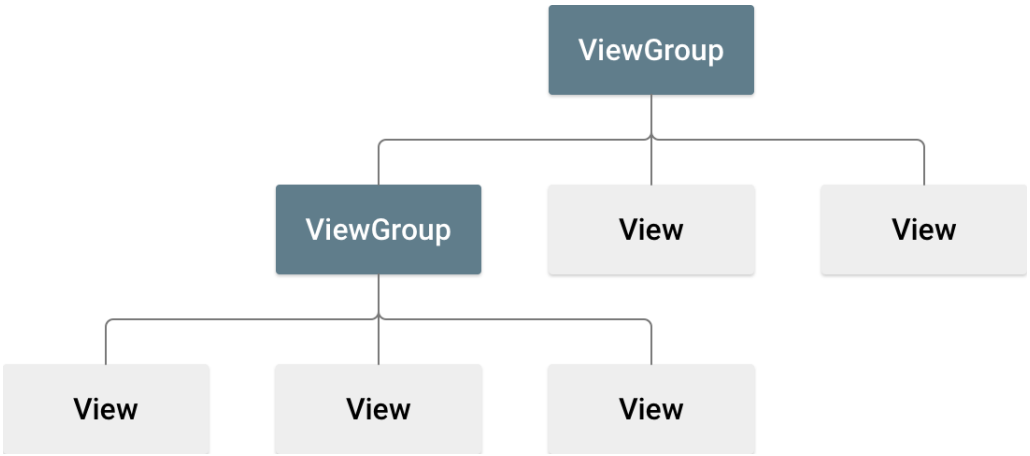


Figure 2.3: Komponenten-Hierarchie

Wie in Abbildung 2.3 dargestellt, definieren EntwicklerInnen die Benutzeroberfläche für jede Activity mithilfe einer Baumstruktur von Viewknoten und Viewgroups.
Die Layouts können auf zwei Arten deklariert werden:

- Die Elemente der Benutzeroberfläche werden in XML deklariert - Jedes Layout muss genau ein Wurzelelement enthalten, und die zusätzlichen Layouts oder Widgets können als untergeordnete Elemente hinzugefügt werden. Dabei handelt es sich um ein View- oder Viewgroup-Objekt. Jede View oder Viewgroup enthält individuelle Attribute.

- Layoutelemente zur Laufzeit erzeugen — View und ViewGroup Objekte können automatisch in der App erstellt werden.

Die am häufigsten verwendeten Layout-Objekte in der App sind [**b8**]:

- LinearLayout — richtet alle untergeordneten Elemente horizontal oder vertikal aus und alle untergeordneten Elemente werden nacheinander gestapelt. Eine vertikale Liste hat nur ein untergeordnetes Element pro Zeile, egal wie breit die Elemente sind, dagegen hat eine horizontale Liste hat nur eine Zeile.
- RelativeLayout — ist anpassbar. Die untergeordneten Elemente können zueinander oder zum übergeordneten Element ausgerichtet werden.
- AbsoluteLayout — die untergeordneten Elemente können genaue x/y Koordinaten festlegen, damit sie auf dem Bildschirm angezeigt werden können. (0,0) ist die obere linke Ecke. Man kann die Werte anpassen, um das Element nach unten oder nach rechts zu bewegen.
- TableLayout —positioniert die untergeordneten Elemente in Zeilen und Spalten.

Die Layouts können auch in einen Adapter eingebettet werden, um das Layout zur Laufzeit mit Views zu füllen, wenn der Inhalt der Layouts nicht vorbestimmt ist. Ein Adapter ruft die Daten ab und bindet sie ans Layout. Die Listview zeigt eine scrollende und einspaltige Liste an. Die Gridview bietet hingegen ein quadratisches Raster mit Spalten und Zeilen. Diese speziellen Layouts werden in der Music Player App benutzt und bei der Implementierung noch detaillierter erklärt.

ViewGroups und Views weisen jeweils ein eigenes Set von XML-Attributen auf. View-Objekt kann bestimmte Attribute haben, und jedes View-Objekt, das diese Klasse erweitert, kann diese Attribute erben. Andere Attribute, die als „Layout-Parameter" bezeichnet werden, beschreiben bestimmte Layout-Ausrichtungen, die durch das übergeordnete ViewGroup-Objekt des View-Objekts definiert sind. z.B: layout_height, layout_width und vieles mehr.

Komponenten

Button

Ein Button besteht aus Text oder einem Symbol oder aus beidem. Er definiert, welche Aktion ausgeführt wird, wenn der Benutzer ihn drückt. Je nachdem, ob ein Button mit einem Text, einem Symbol oder mit beidem deklariert wird, kann im Layout auf drei Arten erzeugt werden:

- die Button-Klasse wird für einen Button mit Text verwendet.

- die ImageButton-Klasse wird verwendet, wenn man einen Button mit einem Symbol benutzt.
- die Button-Klasse mit android:drawableLeft wird verwendet, wenn man einen Button mit sowohl Text als auch einem Symbol braucht.

Wenn der Benutzer auf einen Button klickt, empfängt das Button-Objekt ein On-Click Ereignis. Die Attribute müssen im Button-Element der XML-Datei deklariert werden. Der Wert für diese Attribute muss der Name der Methode sein, die beim Button-Klick aufgerufen wird. Die Activity, die man mit diesem Layout verbindet, muss dann die entsprechende Methode implementieren. Sonst kann man OnClickListener benutzen, indem es nur in der Kotlin- oder Java-Datei aufgerufen werden soll, ohne in der XML-Datei deklariert zu werden.

Checkboxes

Checkboxes ermöglichen dem Benutzer, ein oder mehrere Elemente aus einem Set auszuwählen. Sie wird in ein Layout integriert. Jede Checkbox wird separat verwaltet und muss einen Klick-Listener registrieren. In der Music Player App wird eine Checkbox für die Auswahl der Top-Genres benutzt.

Dialogs

Ein Dialog ist ein kleines Fenster, das den Benutzer bittet, eine Entscheidung zu treffen oder zusätzliche Informationen zur Verfügung zu stellen. Die Dialog-Klasse ist die Basisklasse und definiert den Stil und die Struktur. Die Unterklassen, sowie AlertDialog, DatePickerDialog und TimePickerDiaolog, haben Aufgaben. z.B: Auswahl eines Datums oder einer Uhrzeit usw.
DialogFragment ist ein Container für den Dialog und bietet alle Elemente, die man benötigt, um einen Dialog aufzurichten. Das Dialogdesign kann realisiert werden, indem DialogFragment erweitert und ein AlertDialog in der Rückruf-Methode gebildet wird. Abbildung 2.4 zeigt die Darstellung des Dialogs der Music-Player App. Er besteht aus drei Bereichen:

1. Titel kann man je nach Wunsch verwenden. Wenn der Inhaltsbereich eine detaillierte Nachricht oder eine Liste enthält, dann soll ein Titel verwendet werden. Wenn eine einfache Nachricht oder Frage im Dialog angezeigt wird, dann ist der Titel nicht erforderlich. In einem solchen Dialog lautet der Titel 'Select an option'.
2. Im Inhaltsbereich wird eine Nachricht, eine Frage, ein anderes benutzerdefiniertes Layout oder eine Liste angezeigt. Wenn es sich um eine Liste handelt, sind drei Arten von Listen mit den AlertDialog APIs vorhanden:
    - Eine klassische Single Choice Liste.

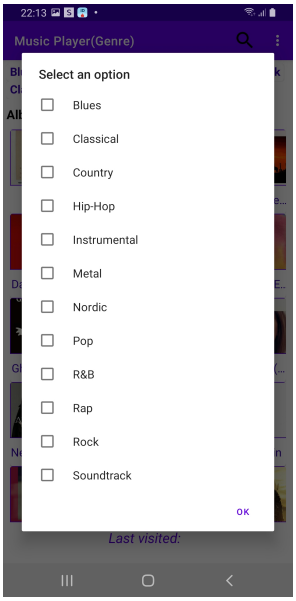


Figure 2.4: Darstellung eines Dialogs

- Eine persistente Single Choice Liste (Radiobuttons).
- Eine persistente Multiple-Choice Liste (CheckBox).

Im Dialog aus Figure 2.4 handelt es sich um eine Multiple-Choice Liste, die aus mehreren CheckBoxes besteht.

3. Ein Action Button kann von folgenden 3 Arten sein:
   - positiv - wird verwendet, um die Action zu akzeptieren und fortzusetzen.
   - negativ - wird für das Abbrechen einer Action verwendet.
   - neutral - wird verwendet, wenn die Action möglicherweise nicht fortgesetzt wird, aber auch nicht unbedingt abgebrochen wird. Die Aktion könnte beispielsweise "Später erinnern" lauten.

   In unserem Fall besteht der Dialog aus einem positiven Action Button.

Ein Dialog kann sangezeigt werden, indem DialogFragments-Objekte erzeugt und die show() Methode aufgerufen wird.

Menüs

Menüs sind eine der Standardkomponenten der Benutzeroberfläche in der App. Sie werden für eine Darstellung der Benutzeraktionen und anderer Optionen der Activities verwendet. Menübuttons stellen meist höchstens 6 Menüpunkte zur Verfügung. Ab Android 3.0(API-Level 11) gibt es keine Menübuttons mehr, sondern eine Action Bar. Obwohl sich das Design

und die Benutzererfahrung geändert hat, basiert die Semantik zum Definieren der Aktionen und Optionen auf den Menü-APIs. Die Menüs teilen sich in drei Typen ein:

- Optionsmenü und App Bar—stellt das primäre Menü dar und besteht aus Menüpunkten, die die aktuelle Activity betreffen. Hier sollten die Actions platziert werden, die sich global auf die App auswirken. z.B: Suchen oder Einstellungen.
- Kontextmenü —hier handelt es sich um ein schwebendes Menü, das angezeigt wird, wenn der Benutzer lange auf ein Element klickt. Hier platzierte Actions wirken sich auf den ausgewählten Inhalt oder Kontextrahmen aus.
- Popupmenu — zeigt eine Liste von Elementen an. Es ist gut geeignet, um erweiterte Aktionen in Bezug auf bestimmte Inhalte anzuzeigen.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/actionTags"
        android:orderInCategory="100"
        android:title="All Tags">
    </item>
    <item
        android:id="@+id/actionTopTags"
        android:orderInCategory="100"
        android:title="Top Tags">
    </item>
    <item
        android:id="@+id/actionExit"
        android:orderInCategory="100"
        android:title="Exit">
    </item>
</menu>
```

Figure 2.5: Implementierung eines Menüs

Abbildung 2.5 zeigt die XML-Datei zum Definieren des Menüs der Music Player App an. Android bietet ein Standard-XML-Format zum Definieren von Menüelementen. Anstatt ein Menü im Code einer Activity aufzubauen, wird das Menü und alle seine Elemente in einer XML-Menüressource definiert. Anschließend können die Menüressourcen in der Activity oder dem Fragment aufgerufen werden.

Action Bar

ActionBar ist eine der wichtigsten Designelemente in den Activities der App. Sie zeigt nicht nur Text an, sondern kann auch Funktionen ausführen. Die wichtigsten Funktionen der ActionBar sind:

- Unterstützung für eine Navigation.
- Zugriff auf die Action

- Anzeige der Identität und des Standorts des Benutzers.

Figure 2.6: Die Darstellung einer ActionBar

Abbildung 2.6 zeigt die Darstellung von ActionBar der Music- Player App.
Wie oben bereits erwähnt, wird die ActionBar ab Android 3.0(API-Level 11) oben im Fenster einer Activity angezeigt. Dadurch verhält sich die ActionBar unterschiedlich, je nachdem, welche Version des Android-Systems ein Gerät verwendet.

Search feature

Die Suche ist eine wichtige Benutzerfunktion auf ein Android. Benutzer müssen in der Lage sein, alle bestehenden Daten zu durchsuchen. Mit dem Android-Such-Framework kann die Suche in jeder App implementiert werden. Das Such-Framework bietet zwei Arten der Sucheingabe:

- Ein Suchdialog - erscheint oben auf dem Bildschirm bei Aktivierung durch den Benutzer. Die Suchvorschläge können geliefert werden, während der Benutzer tippt.
- Ein Such-Widget (SearchView), das irgendwo im Activitylayout aufgestellt wird. Beide bieten die gleiche Funktionalität und können die Suchanfrage an eine bestimmte Activity liefern.

Ab Android 3.0 wird es empfohlen, ein Such-Widget als Action in die ActionBar einzufügen, anstatt den Suchdialog zu verwenden. In der Music Player App wird ein Such-Widget verwendet, wie in Abbildung 2.7 dargestellt.

Settings

Mit Einstellungen können Benutzer die Funktionalität und das Verhalten einer App ändern. Einstellungen können sich auf das Hintergrundverhalten auswirken, z.B. wie oft die Anwendung Daten mit der Cloud synchronisiert. Sie können aber auch den Inhalt und die Darstellung der Benutzeroberfläche ändern. Android empfehlt eine AndroidX Preference Library zum Integrieren benutzerkonfigurierbarer Einstellungen in einer App.
In der Music Player App wird SharedPreferences zur Speicherung und Wiederverwendung der Werte und Daten benutzt. Es geht um die Auswahl der Top-Genres. Mit der SharedPreferences-API ist es möglich, einfache Schlüssel-Wert-Paare aus einer Datei, die über die App Session gespeichert wird, zu lesen und zu schreiben.

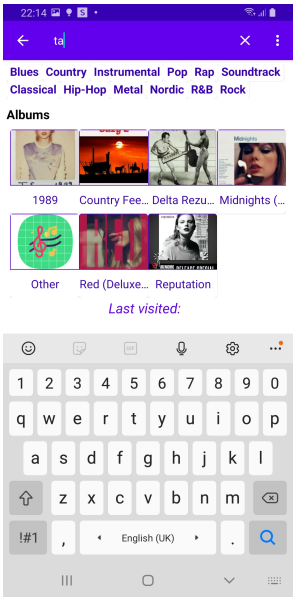


Figure 2.7: Such-Widget zur Implementierung einer Suche

Drawables

Statische Bilder können in der App angezeigt werden, indem die Drawable-Klasse und ihre Unterklassen definiert werden, um die Formen und Bilder zu zeichnen. Es gibt zwei Arten, um Drawables zu bilden:

- Einfügen der Bildressource (Bitmap File), die im Projekt gespeichert wird. Unterstützte Dateitypen sind PNG(bevorzugt), JPG(akzeptabel) und GIF(nicht empfohlen). App-Icons, Logos und andere Grafiken sind Beispiele für diese Art von Bildern. Zur Verwendung werden die Dateien dem Verzeichnis res/drawable/ des Projekts hinzugefügt und aus dem Code oder aus dem XML-Layout referenziert.
- Auffüllen der XML-Ressource mit Eigenschaften. Das Drawable-Objekt ist nicht von Variablen abhängig, die durch den Coder oder die BenutzerInneninteraktion definiert sind. Nachdem die XML-Datei erstellt wurde, wird sie in res/drawable/ hinzugefügt. Das Objekt wird abgerufen, indem Resources #getDrawable() aufgerufen und die Id der XML-Datei übergegeben wird.

Touch und Input

Android bietet mehrere Möglichkeiten, die Ereignisse aus der Interaktion des Benutzers abzufangen. Wenn z.B ein Button gedrückt wird, wird die Methode onTouchEvent() für dieses Objekt aufgerufen. Um dies abzufangen und die Benutzerinteraktion mit der Benutzeroberfläche zu erfassen, ist EventListener vorhanden. Ein EventListener ist eine Schnittstelle in der View-Klasse, die eine einzelne Callback-Methode enthält. Es gibt folgende Methoden

unterschiedlicher Listener:

- onClick() - aus View.OnClickListener. Wenn der Benutzer entweder ein Element berührt oder die Navigationstaste drückt.
- onLongClick() - aus View.OnLongClickListener. Dies wird aufgerufen, wenn der Benutzer das Element entweder berührthält oder die Navigationstaste drückt und hält (eine Sekunde lang)
- onFocusChange() - aus View.OnFocusChangeListener. Dies wird aufgerufen, wenn der Benutzer die Navigationstasten oder den Trackball verwendet, um zu oder von einem Element zu navigieren.
- onKey() - aus View.OnKeyListener. Dies wird aufgerufen, wenn der Benutzer sich auf das Element konzentriert und eine Hardwaretaste auf dem Gerät drückt oder loslässt.
- onTouch() - aus View.OnTouchListener. Dies wird aufgerufen, wenn der Benutzer eine Aktion innerhalb der Grenzen des Elements ausführt, sowie Drücken, Loslassen oder einer beliebigen Bewegungsgeste auf dem Bildschirm.
- onCreateContextMenu() - aus View.OnCreateContextMenuListener. Wenn ein Kontextmenü erzeugt wird, wird das aufgerufen.

Einige dieser Methoden geben einen boolean-Wert zurück, der angibt, ob das Ereignis verbraucht wurde und daher nicht weitergetragen werden soll.

Toast

Es handelt sich um ein kleines Pop-up-Fenster, das dem Benutzer einfaches Feedback geben soll. Ein Toast zeigt einen Text an, der nach einer gewissen Zeit wieder verschwindet. Ab-

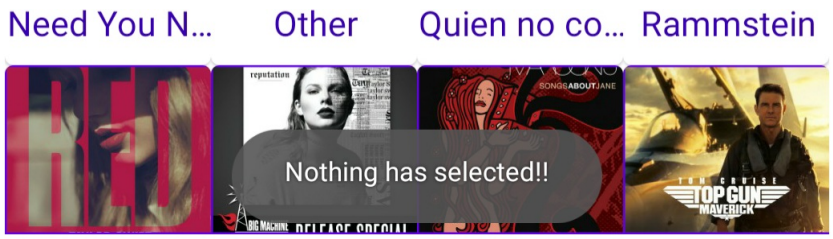


Figure 2.8: Darstellung eines Toasts

bildung 2.8 zeigt ein Beispiel für einen Toast der Music Player App.

2.5 Genre-Tags im MP3 und WMA

Wie in der Abgabenstellung gefordert, sollen Musikstücke als MP3- und WMA-Dateien vorliegen können. Audioformate gliedern sich in unkomprimierte und komprimierte. Unkomprimierte Audioformate enthalten komplette Audiodaten im Vergleich zum komprimierten Audioformaten. Audiodaten werden bei komprimierten Audioformaten nicht entfernt, sondern nur in weniger Platz verpackt, indem der Aufbau der originalen Datei mithilfe von Encoder und Decoder verändert wird. MP3- und WMA-Dateien sind beide komprimierte verlustbehaftete Audioformate, die nicht deutliche oder redundante Messwerte aus dem Daten eliminieren. Obwohl die Dateigröße wesentlich geringer ist, etwa 1/10 der Originalgröße, hört man keinen Unterschied zum Original.

MP3 steht für MPEG-1 Audio Layer 3 und ist eine Audio-Kompressionstechnologie von der Fraunhofer-Gesellschaft. 1987 im Rahmen des EUREKA-Projekts EU147, Digital Audio Broadcasting (DAB), begann das Fraunhofer Institut für Integrierte Schaltungen mit der Erforschung hochwertiger Audiocodierung mit niedriger Bitrate. MP3 wurde von diesem Institut veröffentlicht [**b12**].
WMA steht für Windows Media Audio und wurde von Microsoft entwickelt. Es geht um den Audio-Codec von Windows. Das Prinzip ist ähnlich wie die MP3-Kompression [**b13**].

Unkomprimierte und komprimierte Formate können in einem Container eingebettet werden, was es ermöglicht, alle Informationen über die Datei in der Datei selbst zu speichern. Das WMA-codierte Audio-Format ist in einen ASF-Container integriert. In anderen Worten enthalten ASF-Dateien WMA-Dateien. ASF ist ein Teil des Windows- Media Framework und steht für Advanced Systems Format. Die Byte-Sequenzen werden als dem Ausdruck GUID enkodiert. GUID wird im 8-4-4-4-12 Format dargestellt. Eine ASF-Datei repräsentiert Metadaten und belegt 128 Byte, ähnlich wie ID3-Tags (siehe unten). Das Lesen der ASF-Dateien kann durch das Programm von Microsoft erfolgen [**b15**], wie MPlayer oder VLC Media Player.
MP3 hat keine Methode zum Speichern von Metadaten. ID3 ermöglicht es daher, Metadaten hinzuzufügen. Ein ID3v1-Tag belegt 128 Byte, beginnend mit der Zeichenfolge TAG und schließend mit dem Genre. Abbildung 2.9 zeigt den Aufbau eines ID3-Tags an. Das Genre wird als einzelnes Byte kodiert, wobei das Genre als Zahl abgespeichert wird. Die Verwendung von UTF-8 kodierten Zeichen wird zugelassen. Es wurde am Ende der Datei platziert, damit Media-Player sie erkennen können. Die meisten mobilen MP3-Player und Betriebssysteme lesen ID3-Tags [**b14**].
Man kann die Tags mithilfe eines Tag-Editors bearbeiten und erstellen, was im Kapitel 3 erläutert wird. Ein Musikstück kann mit beliebigen Genres getagt werden, indem sie entweder

Offset	Länge	Bedeutung
0	3	Kennung „TAG“ zur Kennzeichnung eines ID3v1-Blocks
3	30	Titel des Musikstücks
33	30	Künstler/Interpret
63	30	Album
93	4	Erscheinungsjahr
97	30	Beliebiger Kommentar
127	1	Genre

Figure 2.9: Aufbau der 128Bytes eines ID3-Tags

durch ein Schrägstrich einen Beistrich oder einen Strich getrennt werden.

2.6 API zum Abspielen von Musikdateien

Laut Aufgabenstellung sollten MP3- und WMA-Dateien verarbeitet werden können. Fast jedes Abspielgerät akzeptiert MP3-Dateien im Vergleich zu den WMA-Dateien. Mit der API von MediaPlayer für Android war es leider nicht möglich, beide Dateien auf jedem Gerät abzuspielen, da nur die WMA-Dateien angezeigt werden. Windows Media Player oder VLC Media Player haben die Fähigkeit zum Lesen und Abspielen des WMA-Formats. Deswegen wird die VLC-Bibliothek für diese Arbeit verwendet.

Allgemeines über LibVLC
LibVLC ist die Kern-Engine und Schnittstelle für das Multimedia-Framework basierend auf dem VLC Media Player. Viele Entwickler sind mit dem begleitenden SDK, LibVLC, nicht vertraut, obwohl die VLC-App öffentlich und erwünscht ist.
LibVLC ist eine Android-Bibliothek, mit der man auf beliebige Multimedia-Funktionen zugreifen kann, darunter:

- Alle Mediendateiformate, jeder Codec und alle Streaming-Protokolle werden abgespielt
- Läuft auf jeder Plattform, von Desktop bis hin zu Mobilgeräten und Fernsehen.
- Effiziente Decodierung und Hardware auf jeder Plattform bis zu 8K.
- Network-Browsing nach fernen Dateisystemen und Servern.
- Wiedergabe von Audio CD, DVD und Bluray mit Menünavigation.
- Unterstützung für HDR, einschließlich Tonemapping für SDR-Streams.
- Audio-Passthrough mit SP DIF und HDMI, einschließlich für Audio-HD-Codecs wie DD+, TreuHD oder DTS-HD
- Video- und Audiofilter werden unterstützt.

- Unterstützung für 360-Grad Video und 3D-Audiowiedergabe, inklusive Ambisonics.
- Streamen und Übertragen zu entfernten Renderern wie Chromecast und UpnP-Renderern.

Es ist möglich, LibVLC mithilfe einer C-Bibliothek in eine App einzubetten. Sowohl Mobile- als auch Desktop- Plattformen können zum Einsatz kommen. LibVLC ist für Android unter LGPL2.1 lizenziert [**b6**].

Eine allgemeine Beschreibung einiger Methoden der Klassen libVLC_Media_Player und libVLC_Media, die in der libVLC-Medienverarbeitung implementiert sind, werden in Abbildung 2.10 dargestellt. Die Klassen libVLC_Media_Player und libVLC_Media ergänzen

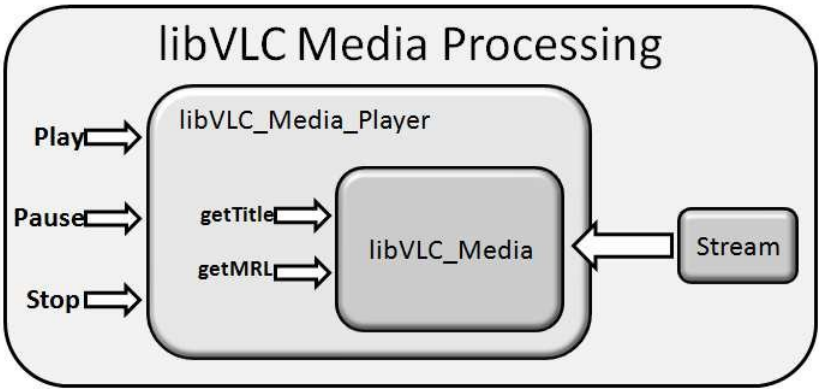


Figure 2.10: Die Beschreibung einiger Methoden der Klassen libVLC

sich auf der höchsten Abstraktionsebene. Hier sind die Hauptfunktionen dieser Klassen [**b7**]:

- LibVLC_Media_Player bietet Methoden zum Steuern der Wiedergabe und zum Zurückgeben von Stream-Flow-Informationen an. Zusätzlich können Untertitel dem Stream hinzugefügt werden.
- Low-Level-Methoden, die von LibVLC_Media bereitgestellt werden, ermöglichen die Steuerung von Medien, wie z. B. das Duplizieren von Deskriptoren, das Berechnen der Länge der Medien usw. LibVLC_video und libVLC_audio, sind die Klassen der niedrigsten Ebene, die Video bzw. Audio regeln.

3 Programmablauf

Die hier entwickelte Music Player App bietet Benutzern die Möglichkeit, ihre Musikstücke auf ihrem Smartphone nach Kategorien zu ordnen und anzuhören. Mit weiteren Verbesserungen wird das Herunterladen der App auf PlayStore in Zukunft zur Verfügung gestellt. In Kapitel 6 wird erwähnt, welche Änderungen bzw. Verbesserungen durchgeführt werden können. Die Implementierung der App wird in Kapitel detailliert erklärt, wo gezeigt wird, wie die Zusammenarbeit von Kotlin-Code für die Komponenten und Manifest funktioniert. Kapitel 3 widmet sich dem detaillierten Programmablauf.

3.1 Tagging

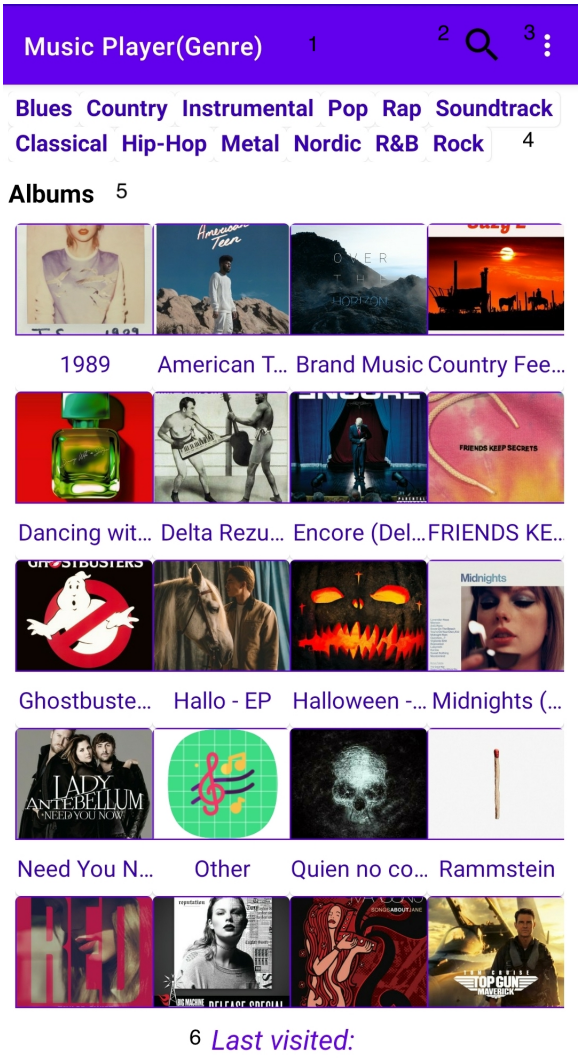
Vor der Benutzung dieser App sollte man die Musikstücke mithilfe eines Musikeditors bearbeiten und die Genres eingeben, damit diese App ihre Hauptaufgabe erledigen kann, die Alben nach Genre zu ordnen. Dafür gibt es zahlreiche Tag-Editoren, wie z.B. MP3Tag, TagScanner, Kid3, TigoTago usw.
MP3Tag [**b10**] gewinnt an Popularität, da es die Daten detailliert bearbeiten kann, einschließlich Interpret, Album, Titel, Album-Cover. Außerdem ist die Benutzung auf unterschiedliche Plattformen möglich und es können die beliebige Audioformate bearbeitet werden. Die Dateien können basierend auf den Tags umbenannt werden, Zeichen oder Wörter können in Tags und Dateinamen ersetzt werden, Playlisten werden erstellt usw.

3.2 Permissions

Nach der Installation und Starten der App werden App-Berechtigungen abgefragt. Dabei kann der Benutzer zwischen ‚Zulassen' und ‚Verweigern' wählen. Sobald die Berechtigung freigegeben wurde, erscheint der Hauptbildschirm (siehe 3.3) , der aus einer Liste der Alben besteht.
Die Berechtigung für das Lesen der Musikstücke wird auf dem Smartphone in der Manifest-Datei angegeben. Viele Laufzeitberechtigungen greifen auf private Benutzerdaten zu, eine spezielle Art von eingeschränkten Daten, die möglicherweise vertrauliche Informationen enthalten. z.B: Kontaktdaten, Fotos usw [**b11**].

3.3 Mainscreen

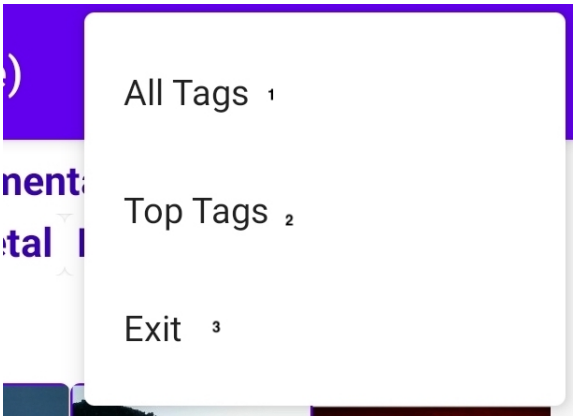
Der Hauptbildschirm besteht aus 6 verschiedenne Teilen:



1. Die ActionBar der App, besteht aus der Identität der App, der Such-Action und dem Optionsmenü.
2. Die Suchfunktion gibt dem Benutzer die Möglichkeit, nach Interpret, Titel der Musik und Albumname zu suchen.
3. Das Menü wird in Kapitel 3.4 erklärt.
4. Die Liste der Genres. Wie es implementiert wird, wird in Kapitel 4.1 erläutert. Wenn der Benutzer ein Genre anklickt, wird der Genre-Bildschirm 3.5 angezeigt.
5. Die Liste der Alben. Dort werden alle Alben in einem Layout dargestellt und pro Reihe befinden sich vier Alben. Jedes Album umfasst mehrere Musikstücke. Die Benutzung der Inhalte eines Albums, wird in Kapitel 3.6 erklärt.
6. Der Titel des Albums, das der Benutzer zuletzt besucht hat. Wenn der Benutzer irgendein Album schon angesehen hat, erscheint hier der Titel des Albums. Mit dem Drücken kann der Benutzer zum Album zurückkehren.

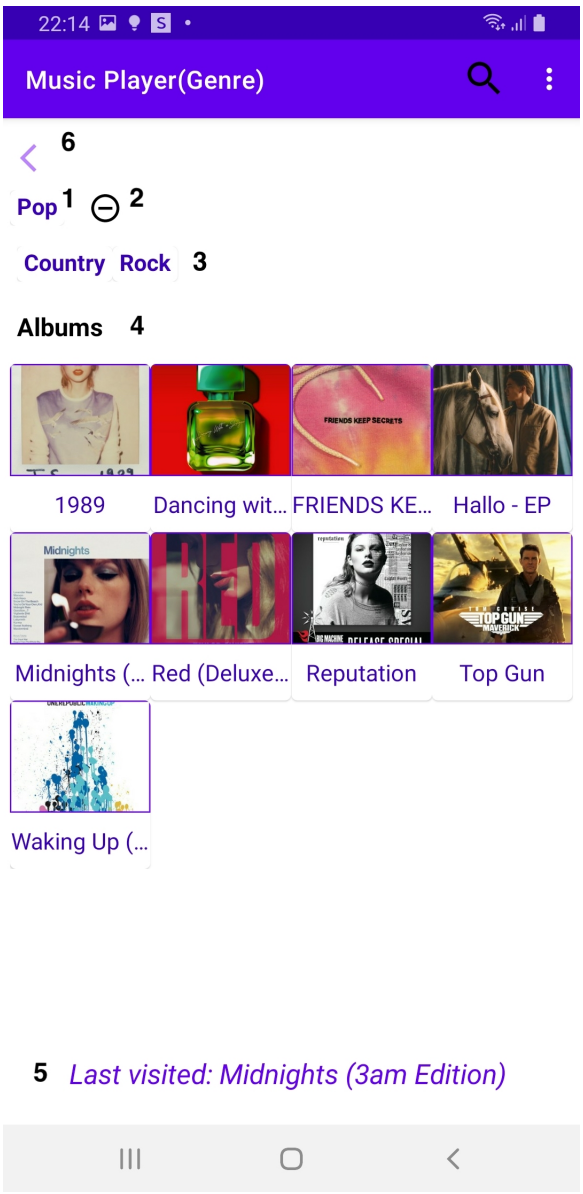
3.4 Menü

Menü setzt sich aus drei Teilen zusammen.



1. Bei Auswahl dieses Menüpunkts, werden alle Genre-Tags zur Auswahl angezeigt. (siehe Kapitel 3.7).
2. Hier wird eine Vorauswahl von Genres benutzt (siehe Kapitel 3.8).
3. Beenden der App.

3.5 Genre-Bildschirm



1. Die Namen der Genres, die der Benutzer ausgewählt hat.
2. Abbrechen-Button. Entfernt das zuletzt ausgewählte Genre-Tag. Da in diesem Beispiel nur ein einziges Tag (Pop) ausgewählt wurde, wird zum Hauptbildschirm zurückkehrt.
3. Weitere mögliche Auswahlen, sind die verbliebenden Genres. Der Benutzer kann hier noch weitere Genres als Verfeinerung auswählen. (Siehe Abbildung 3.1).
4. Es werden alle Alben angezeigt, deren Musikstücke diese ausgewählten Genres aufweisen. Durch den Klick auf eines der Alben kommt man zur Sicht, die in Kapitel 3.6 beschrieben wird.
5. Der Titel des Albums, welches der Benutzer zuletzt besucht hat.
6. Der Zurück-Button geht zum Hauptbildschirm zurück.

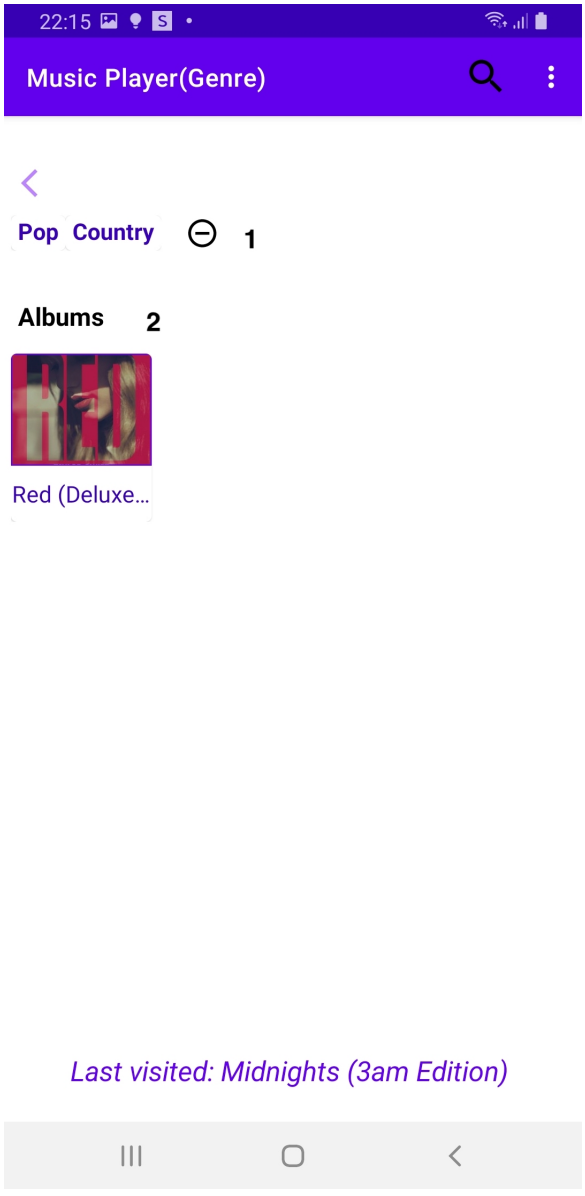
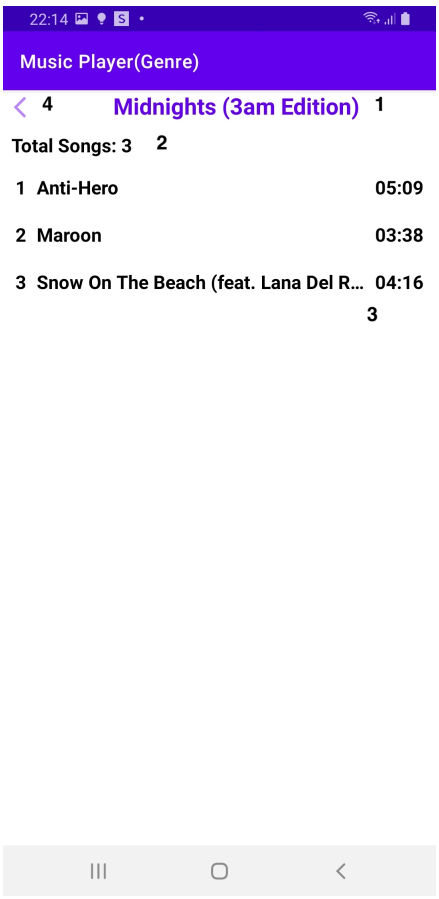


Figure 3.1: Darstellung der verfeinerten Auswahl der Genres

3.6 Auswahl von Alben



1. Der Albumtitel
2. Die Anzahl der Musikstücke dieses Albums
3. Alle Titel dieses Albums mit Name und Dauer. Sie können durch Anklicken abgespielt werden, was in Abschnitt 3.9 beschrieben wird. Wenn ein Musikstück gerade abgespielt wird, ist ein kleines Fenster sichtbar (siehe Abbildung 3.2). Wenn der Benutzer sich im selben Album befindet, kann er zum Player zurückkehren. Sonst kann er nur den spielenden Titel pausieren, wiedergeben und den nächster oder vorherigen Titel im Rahmen dieses Albums abspielen lassen.
4. Der Zurück-Button geht zum letzten Bildschirm. Es handelt sich entweder um den Hauptbildschirm oder einen Genre-Bildschirm.

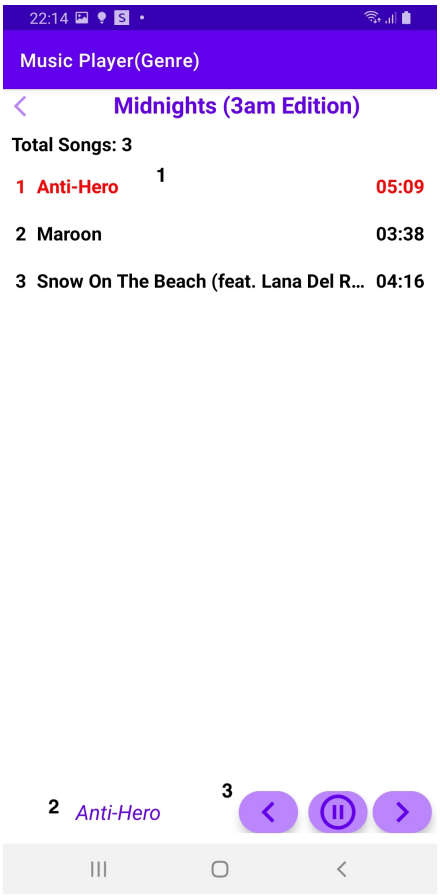
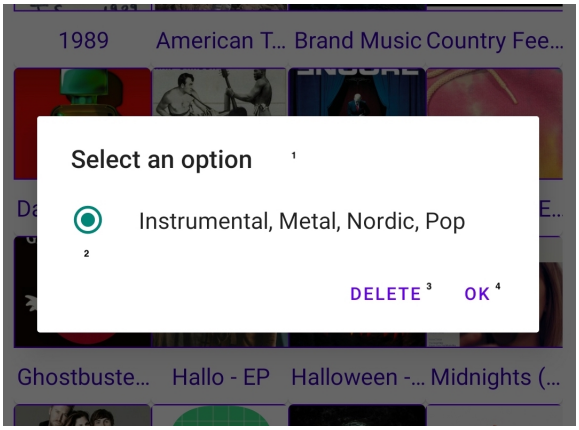


Figure 3.2: Darstellung eines Albums mit dem gerade spielenden Titel

3.7 Auswahl von Genres

Bei diesem Fenster handelt es sich um einen Dialog (siehe Abbildung 2.4). Er besteht aus einem Titel, einer Liste aller Genres und einem Ok-Button. Der Benutzer kann beliebige Genres markieren und den Ok-Button drücken, um die Lieblingsgenres (siehe Kapitel 3.8) zu speichern.

3.8 Top Tags



Hier werden die benutzergespeicherten Listen angezeigt und können einzeln ausgewählt oder gelöscht werden. Diese Listen werden dauerhaft gespeichert, bis der Benutzer sie löscht. Nach dem Löschen dieser Liste zeigt die App ein Toast. Nach der Auswahl einer Liste werden alle Alben angezeigt, die die Genre-Tags dieser Liste aufweisen (siehe Abbildung 3.3). Klickt man auf eines der Genres, kommt der Genre-Bildschirm zum Vorschein (siehe Kapitel 3.5).

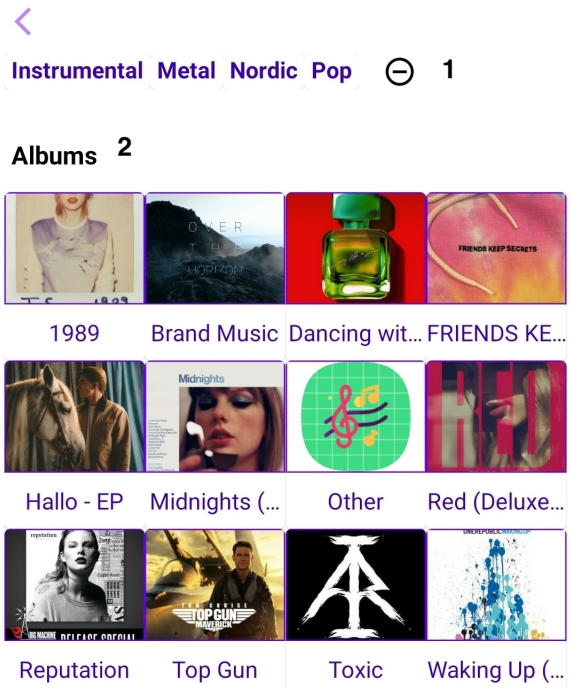
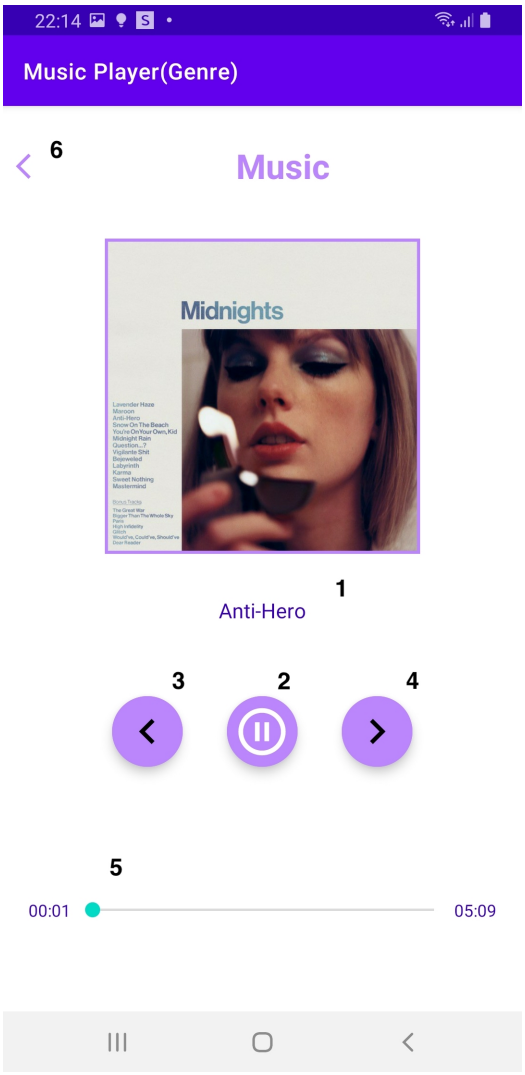


Figure 3.3: Die ausgewählte Top Tags und die Alben, diese Tags aufweisen

3.9 Abspielen von Musikstücken



1. . Der gerade spielende Songtitel und sein Album Cover.
2. Der Wiedergabe- und Pause-Button.
3. Button, der den vorherigen Titel abspielt.
4. Button, der den nächsten Titel abspielt.
5. Anzeige der Position innerhalb des Musikstücks.
6. Zurück-Button.

4 Implementierung

Bei der Implementierungsbeschreibung der Music Player App wird in erster Linie darauf eingegangen, wie Android-Komponenten auf Codeebene verwendet werden. Ein Teil des Manifest wird in Abbildung 4.1 gezeigt. Es beschreibt, wie das Manifest und der Kotlin-Code für die Komponenten mit den XML-Dateien für die Benutzeroberfläche zusammenarbeiten. Wie in Abschnitt 2.4.2 angedeutet, wird mithilfe des Tags <uses-permission> der App-Zugriff

```
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE"
    tools:ignore="ScopedStorage" />
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/music_icon"
    android:label="Music Player(Genre)"
    android:requestLegacyExternalStorage="true"
    android:roundIcon="@mipmap/music_icon_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MusicPlayer"
     tools:targetApi="s">
    <activity
        android:name=".TagActivity"
        android:exported="false">
        <meta-data
            android:name="android.app.lib_name"
            android:value="" />
    </activity>
```

Figure 4.1: Manifest.xml Datei

gewährt. Innerhalb des Tags <application> werden Appname und Design festgelegt. Mit dem Tag <activity> wird eine Activity vermerkt. Hier sollen alle Activities und Service angehängt werden.
Es folgen nun detaillierte Erklärungen der einzelnen Klassen.

4.1 Activities

Activities werden in Android zur Darstellung der Benutzeroberflächen verwendet. Sie können Teilsichten enthalten, z.B. Fragments. In Music Player App wurden vier Activities implementiert. In diesem Abschnitt werden die Activities und ihre Funktionen vorgestellt.

Main-Activity wird beim Start der App angezeigt. Hier werden die Hauptaufgaben der App erledigt, einschließlich Einlesen von Musikstücken (Abbildung 4.2). Es wird ein ContentResolver-

```
//how we get all musics from database
val cursor : Cursor? = this.contentResolver.query(
    MediaStore.Audio.Media.EXTERNAL_CONTENT_URI,
    arrayOf(
        MediaStore.Audio.Media.TITLE,
        MediaStore.Audio.Media.ALBUM,
        MediaStore.Audio.Media.ARTIST,
        MediaStore.Audio.Media.DURATION,
        MediaStore.Audio.Media.GENRE,
        MediaStore.Audio.Media.DATA,
        MediaStore.Audio.Media.ALBUM_ID),
    selection: MediaStore.Audio.Media.IS_MUSIC + "!= 0",
    selectionArgs: null,  sortOrder: null,  cancellationSignal: null)

if(cursor != null) {
    if (cursor.moveToFirst()) {
        do {
            val titleC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.TITLE))?: "Unknown"
            val albumC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM))?: "Unknown"
            val artistC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.ARTIST))?: "Unknown"
            val pathC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.DATA))
            val durationC = cursor.getLong(cursor.getColumnIndex(MediaStore.Audio.Media.DURATION))
            val genreC = cursor.getString(cursor.getColumnIndex(MediaStore.Audio.Media.GENRE))?: "noGenre"
            val albumIdC = cursor.getLong(cursor.getColumnIndex(MediaStore.Audio.Media.ALBUM_ID)).toString()
            val artUriC = Uri.withAppendedPath(Uri.parse( uriString: "content://media/external/audio/albumart"), album

            val music = Music(title =titleC, album =albumC, artist = artistC, duration =durationC, artUri=artUriC,
```

Figure 4.2: Einlesen des einzelnen Musikstückes

Objekt verwendet, um mit der Abstraktion des Medienspeichers zu interagieren. Das System scannt automatisch ein externes Speichermediumund fügt alle darin enthaltenen Musikdateien zur Sammlung hinzu. Um auf Mediendateien zuzugreifen, müssen die Berechtigungen im Manifest deklariert werden [**b17**].
Wie in Abschnitt 4.1 erwähnt, enthält die MainActivity die vererbten Methoden onCreate(), onDestroy() und onResume(). Beim Erzeugen der Activity bzw. der App wird die Methode onCreate() (Abbildung 4.3) ausgeführt, sodass die Daten initialisiert werden und die Ansicht der Genre- und Alben-Liste jeweils mithilfe der Klassen Tag- und Album-Adapter (siehe Kapitel 4.2) abgerufen wird. Die Listen der Genres und der Alben werden mit dem RecyclerView deklariert, das eine erweiterte Version von ListView mit verbesserter Leistung ist. Es hat die Fähigkeit, die Views wiederzuverwenden.
Die Ansicht eines Genres und eines Albums wird in der jeweiligen tag_view.xml Datei und album_view.xml Datei aufgebaut. Um jedes Genre im RecyclerView herzuzeigen, wird ein StaggeredGridLayoutManager mit 2 Zeilen verwendet, da die Namen der Genres unterschiedlich lang sind. Für Alben wird ein GridLayoutManager mit 4 Spalten benötigt, um jeweiliges Album in einem rechteckigen Raster anzuordnen.

```
@RequiresApi(Build.VERSION_CODES.R)
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    TagListMA = ArrayList()
    AlbumListMA = ArrayList()
    MusicListMA = ArrayList()
    if(requestRuntimePerm()) {
        initialize()
    }
    binding.tagsMA.setHasFixedSize(true)
    binding.tagsMA.setItemViewCacheSize(20)
    binding.tagsMA.layoutManager = StaggeredGridLayoutManager( spanCount: 2, StaggeredGridLayoutManager.HORIZONTAL)
    tagAdapter = TagAdapter( context: this, TagListMA)
    binding.tagsMA.adapter = tagAdapter

    binding.albumsMA.setHasFixedSize(true)
    binding.albumsMA.setItemViewCacheSize(20)
    binding.albumsMA.layoutManager = GridLayoutManager( context: this@MainActivity, spanCount: 4)
    albumAdapter = AlbumAdapter( context: this, AlbumListMA)
    binding.albumsMA.adapter = albumAdapter

    supportFragmentManager.beginTransaction().apply { this: FragmentTransaction
        replace(binding.lastVisitedAlbumTA.id, LastAlbum())
        addToBackStack( name: null)
        commit()
    }
}
```

Figure 4.3: Die Methode onCreate() der Main-Activity

Die Methode onResume() wird nach dem Pausieren und Zurückkehren zu dieser Activity ausgeführt. Hier sollten die benutzerdefinierten bzw. geänderten Top-Tags mithilfe von SharedPreferences gespeichert werden.
Wenn die App beendet ist, wird onDestroy() ausgeführt, da wird der Player beendet und nicht mehr im Hintergrund läuft.

Ein Teilsicht der MainActivity ist ein Fragment; es wird im Abschnitt 4.3 detailliert erklärt. Um die Fragmente zu einer Activity hinzuzufügen bzw. zu verwalten, kann die API von FragmentManager benutzt werden .
Das Menü mit Suchfunktion wird mit der Methode onCreateOptionsMenu() erzeugt. Jeder einzelne Menüpunkt wird als DialogFragment() implementiert und führt zum Aufruf der geerbten Methode onOptionsItemSelected().

Album-Activity wird aufgerufen, wenn der Benutzer ein Album aus der Liste ausgewählt hat. Abbildung 4.4 zeigt den Code von Album-Activity.
Die Liste der Musikstücke wird mithilfe der RecyclerView deklariert und besteht aus einem MusicView. Der LinearLayoutManager wird verwendet, um die Musikstücke in einer vertikalen Ordnung anzuzeigen.

```
class AlbumActivity: AppCompatActivity() {

    private lateinit var binding: ActivityAlbumBinding
    private lateinit var musicAdapter: MusicAdapter
    companion object{
        lateinit var musicListAA: ArrayList<Music>
        var albumName: String = ""
    }
    @SuppressLint("SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_album)
        setTheme(R.style.Theme_MusicPlayer)
        binding = ActivityAlbumBinding.inflate(layoutInflater)
        setContentView(binding.root)
        musicListAA = ArrayList()
        initialize()

        binding.backButtonAA.setOnClickListener { finish() }
        binding.songsAA.setHasFixedSize(true)
        binding.songsAA.setItemViewCacheSize(20)
        binding.songsAA.layoutManager = LinearLayoutManager( context: this@AlbumActivity)
        musicAdapter = MusicAdapter( context: this@AlbumActivity, musicListAA)
        binding.songsAA.adapter = musicAdapter
        binding.totalSongs.text = "Total Songs: ${musicAdapter.itemCount}"
        supportFragmentManager.beginTransaction().apply { this: FragmentTransaction
            replace(binding.nowPlayingSong.id, NowPlaying())
            addToBackStack( name: null)
            commit()
        }
    }
```

Figure 4.4: Die Klasse Album-Activity

Der MusicAdapter lädt jede Zeile mit Daten für ein bestimmtes Musikstück. Die Liste wird unterschiedlich initialisiert, da es davon abhängt, von wo die Daten kommen. Dies wird in der jeweiligen Klasse deklariert und mithilfe von Intent übergegeben. Sie enthält übrigens ein Fragment, das die gerade spielende Musik anzeigt.

Tag-Activity wird ähnlich implementiert, wie Album-Activity, weswegen hier keine Erklärung gegeben wird.

Player-Activity wird beim Abspielen des einzelnen Musikstückes aufgerufen. Bei dieser Activity wird der Player mit Fortschrittsbalken anfertigt und angezeigt (siehe Abbildung 4.5). Die anderen Methoden werden zum Abspielen, Pausieren und Abspielen des nächsten bzw. vorherigen Titel verwendet.
Wie in Abschnitt 2.4.2 erwähnt, wird diese Activity Klasse mit einem Service verbunden, damit die Musik im Hintergrund gespielt werden kann, wenn sich der Benutzer nciht in der App befindet.

```
private fun createMP(){
    try{
        if(musicService!!.musicPlayer != null && musicService!!.libVlc != null){
            musicService!!.releasePlayer()
        }
        val options = ArrayList<String>()
        options.add("--aout=opensles")
        options.add("--http-reconnect")
        options.add("--audio-time-stretch")
        options.add("--network-caching=1500")
        options.add("-vvv")
        musicService!!.libVlc = LibVLC( context: this, options)
        musicService!!.musicPlayer = MediaPlayer(musicService!!.libVlc)
        val media = Media(musicService!!.libVlc, musicListPA[musicPosition].path)
        musicService!!.musicPlayer!!.media = media
        media.release()
        musicService!!.musicPlayer!!.play()

        isPlaying = true
        binding.playBtnPA.setIconResource(R.drawable.pause_icon)
        binding.tvSeekBarStart.text = formatDuration(musicService!!.musicPlayer!!.time)
        binding.tvSeekBarEnd.text = formatDuration(musicListPA[musicPosition].duration)
        binding.seekBarPA.progress = 0
        binding.seekBarPA.max = musicListPA[musicPosition].duration.toInt()
        playingTitle = musicListPA[musicPosition].title
        musicService!!.musicPlayer!!.setEventListener(this)
    }catch(e: Exception){return}
}
```

Figure 4.5: Die Methode, die Music Player erstellt.

4.2 Adapters

Der Adapter greift auf die Datenelemente zu und ist dafür verantwortlich, eine Ansicht für ein Element im Datensatz zu erzeugen. Der Adapter verwendet den ViewHolder, um den Ordnungsruf für die View an dieser Position nachzuschlagen. Beim Scrollen bleibt die Position der Liste fest. In der Music Player App wurden drei Adapter implementiert; ihre Funktionen sind ähnlich. Deswegen wird als Beispiel AlbumAdapter definiert (Abbildung 4.6). Adapter verfügen über 3 Methoden, wie in Abbildung 4.6 gezeigt. OnCreateViewHolder() gibt einen Halter zurück. Die Daten werden an der angegebenen Position geladen, indem die Methode OnBindViewHolder() ausgeführt wird. Die Methode getItemCount() gibt die Anzahl der Elemente zurück.

4.3 Fragments

Fragments werden als Module der Benutzeroberfläche implementiert. Um einen Abschnitt wieder in separaten Acitivities verwenden zu können, werden die Fragments benutzt. In der Music Player App werden 2 Fragments als Teilsicht von Activities implementiert. Das Fragment ist für die Ansicht des letztbesuchten Albums zuständig und das andere für das gerade spielende Musikstück. Beide haben die geerbten Methoden onCreateView() und onResume(). Die Methode onCreateView() entwirft eine Ansicht für dieses Fragment und wird von der Ac-

```
class AlbumAdapter(private val context: Context, private var albumList: ArrayList<Album>): RecyclerView.Adapt
    class MyHolder(binding: AlbumViewBinding): RecyclerView.ViewHolder(binding.root){
        val name = binding.albumName
        val image = binding.albumImg
        val root = binding.root
    }
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) : MyHolder {
        return MyHolder(AlbumViewBinding.inflate(LayoutInflater.from(context), parent,  attachToParent: false))
    }
    override fun onBindViewHolder(holder: MyHolder, position: Int) {
        holder.name.text = albumList[position].name
        holder.name.isSelected = true
        holder.root.setOnClickListener{  it: View!
            when{
                MainActivity.search -> sendIntent( ref: "AlbumAdapterSearch", position)
                else -> sendIntent( ref: "AlbumAdapter", position)
            }
        }
        Glide.with(context).load(albumList[position].musicList[0].artUri)
            .apply(RequestOptions().placeholder(R.mipmap.music_icon).centerCrop())
            .skipMemoryCache( skip: true)
            .into(holder.image)
    }
    override fun getItemCount(): Int {
        return albumList.size
    }
```

Figure 4.6: Die Klasse Album-Adapter

```
class LastAlbum : Fragment() {
    companion object{
        @SuppressLint("StaticFieldLeak")
        lateinit var binding: FragmentLastVisitedAlbumBinding
    }
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val view = inflater.inflate(
           R.layout.fragment_last_visited_album, container,  attachToRoot: false)
        binding = FragmentLastVisitedAlbumBinding.bind(view)
        binding.root.visibility = View.INVISIBLE
        binding.root.setOnClickListener {  it: View!
            val bundle = Bundle()
            bundle.putSerializable("musics", AlbumActivity.musicListAA)
            val intent = Intent(requireContext(), AlbumActivity::class.java)
            intent.putExtra( name: "name", AlbumActivity.albumName)
            intent.putExtra( name: "class",  value: "LastAlbum")
            intent.putExtra( name: "bundle", bundle)
            ContextCompat.startActivity(requireContext(), intent,  options: null)
        }
        return view
    }
    @SuppressLint("SetTextI18n")
    override fun onResume() {
        super.onResume()
        binding.albumNameNP.isSelected = true
        binding.root.visibility = View.VISIBLE
        binding.albumNameNP.text = "Last visited: ${AlbumActivity.albumName}"
    }
```

Figure 4.7: Die Klasse Last-Album

tivity aufgerufen, um sie auf dem Bildschirm anzuzeigen. Die Methode onResume() wird vor der Unterhaltung mit dem Benutzer aufgerufen. Als Beispiel dient die Klasse LastAlbum (Abbildung 4.7).

DialogFragment

Hier handelt es sich um ein Fragment, das ein Dialogobjekt anzeigen kann. Im Abschnitt 2.4.3 wurde die Darstellung dieses speziellen Fragments beschrieben. In der Music Player App wurden zwei spezielle DialogFragments aufgebaut; beide enthalten eine geerbte Methode onCreateDialog(). Sie baut ein benutzerdefiniertes Dialogfeld auf, und zwar die Ansicht der Liste von Tags und Top-Tags. Abbildung 4.8 zeigt die Implementierung.

```
class SelectedTagDialog: DialogFragment(), Serializable {
    companion object{
        lateinit var checkedTags : ArrayList<String>
    }
    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return activity?.let {  it: FragmentActivity
            checkedTags = ArrayList()
            val builder = AlertDialog.Builder(it)
            builder.setTitle("Select an option")
            builder.setSingleChoiceItems(SelectedTagsMA.toTypedArray(), checkedItem: 0) { _, index ->
                checkedTags.addAll(SelectedTagsMA[index].split( ...delimiters: ", "))
            }
            builder.setPositiveButton( text: "Ok") { _, _ ->
                if(checkedTags.isEmpty())   Toast.makeText(requireContext(),  text: "Nothing has selected!!", Toast.LENGTH_SHORT).show(
                else {
                    val bundle = Bundle()
                    bundle.putSerializable("tags", checkedTags)
                    val intent = Intent(context, TagActivity::class.java)
                    intent.putExtra( name: "class",  value: "SelectedTagDialog")
                    intent.putExtra( name: "bundle", bundle)
                    ContextCompat.startActivity(requireContext(), intent,  options: null)
                }
            }
            builder.setNegativeButton( text: "Delete") { _, _ ->
                if(checkedTags.isEmpty())   Toast.makeText(requireContext(),  text: "Nothing has selected!!", Toast.LENGTH_SHORT).show(
                else {
                    Toast.makeText(requireContext(),  text: "Tags deleted successfully", Toast.LENGTH_SHORT).show()
                    SelectedTagsMA.remove(checkedTags.joinToString( separator: ", "))
                }
            }
            builder.create()  ^let
```

Figure 4.8: DialogFragment

4.4 Layouts und Views

Wie im Abschnitt 2.4.3 (Layouts) beschrieben, bestehen alle Benutzeroberflächen aus Views oder Layouts. Als Beispiel dienen activity_album.xml und album_view.xml. In Abbildung 4.9 wird die Darstellung einer XML-Datei in einem grafischen Editor für Benutzeroberflächen gezeigt.

In der XML-Datei werden alle Attribute der jeweiligen Layouts deklariert. Das Hauptlayout

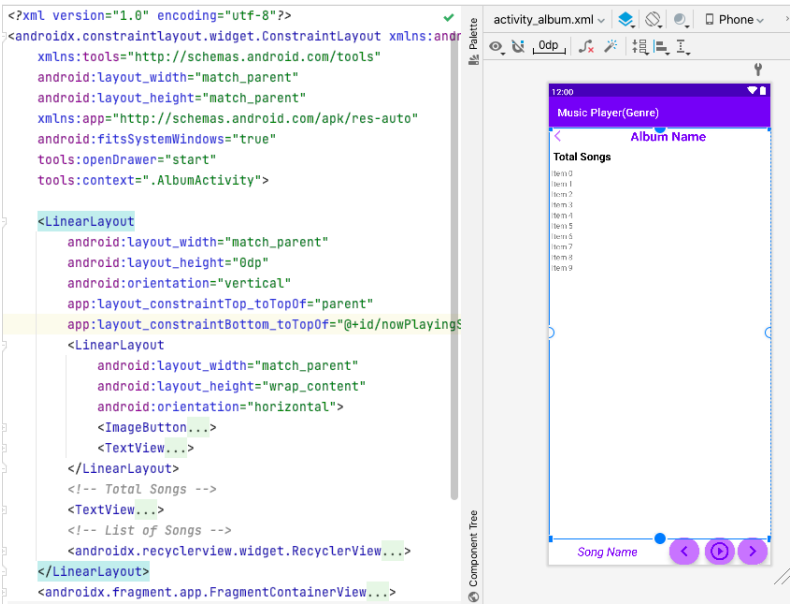


Figure 4.9: Die XML-Datei für die Album-Activity

ist ConstraintLayout und besteht aus einer FragmentContainerView und einem LinearLayout. Das unterliegende LinearLayout enthält eine RecyclerView, eine TextView und ein LinearLayout. Innerhalb dieses LinearLayouts befindet sich eine TextView und ein Button. Buttons können auch in Ausnahmefällen direkt in der XML-Datei zugeordnet werden. Die RecyclerView kann die Elemente in einer Liste oder in einem Raster auflisten und anzeigen. Im vorliegenden Fall werden alle Musikstücke dieser Alben aufgelistet.
Mithilfe von View-Binding wird die Ansicht dieser XML-Datei interagiert. In der Activity-Klasse wird die Methode inflate() aufgerufen, damit ein Objekt der Bindungsklasse für die Activity erzeugt wird. Die Ansicht wird mithilfe von setContentView() auf dem Bildschirm angezeigt.

In Abbildung 4.10 wird die Ansicht eines einzelnen Albums angezeigt. Sie wird mit der MaterialCardView deklariert, die aus einer ShapeableImageView und einer TextView besteht.

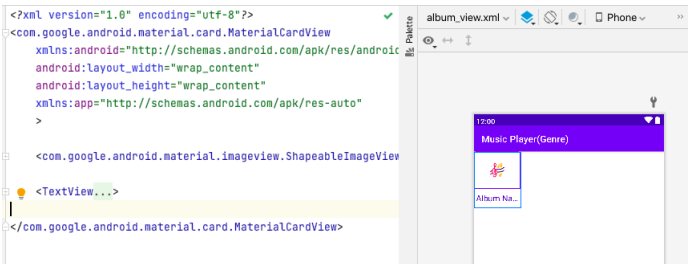


Figure 4.10: Die XML-Datei für die Album-View

5 Technische Daten

Zum Testen der Funktionen der MusicPlayer App wird ein Gerät mit Android als Betriebssystem benötigt. Statt ein echtes Smartphone zu verwenden, bietet Android Studio einen Android-Emulator [**b19**] an, der ein virtuelles Android-Gerät auf dem Computer simuliert. Die App kann auf unterschiedlichen Geräten mit den jeweiligen API-Ebenen getestet werden. Grundsätzlich braucht man für das Testen mit einem Emulator viel Rechenleistung und Speicherplatz, was bei Benutzung einer IDE als Testumgebung problematisch sein könnte. Die Music Player App wurde deswegen auf physischen Geräten getestet.
Zwei Geräte kamen zum Testen zum Einsatz. Ein Samsung Galaxy S9 mit Android 10 dient als Hauptgerät. Es handelt sich hier um ein Handy mit einem 5,8 Zoll großen Bildschirm. Das andere Gerät ist ein Samsung Galaxy Z Flip mit Android 11. Der Bildschirm ist 6.6 Zoll groß.

Es handelt sich nicht um eine aktuelle Version des Android-Betriebssystems (Android 13). Android 14 steht seit Februar 2023 für die ersten Modelle als Upgrade bereit [**b20**]. Für das Haupttestgerät gibt es jedoch noch kein Upgrade auf die nächste Version.
Dank des zweiten Testgeräts konnte jedoch ein Fehler, der auf Android 10 unentdeckt geblieben war, behoben werden. Bei dem Fehler geht darum, dass beim Einlesen der Musikstücke das Genre nicht gleich erkannt wurde. Ab Android 11 können die Genres durch den ContentResolver ganz normal eingelesen werden, während beim Android 10 die Spalte MediaStore.Audio.Media.Genre nicht zugreifbar war. Deswegen werden die Genres auf zwei Arten eingelesen, die jeweils für Android 10 und 11 geeignet sind.
Da Android 10 und Android 11 zum Testen verwendet wurde, wäre es möglich, dass beim Einlesen der Musikstücke auf älteren Android-Versionen noch Probleme auftreten. Deswegen wurden noch zwei weitere virtuelle Geräte zum Testen benutzt. Google Nexus mit Android 8 und Android 8.1 kamen zum Einsatz. Die beiden implementierten Arten funktionierten leider beim Android 8 nicht, d.h. dass die Music Player App erst ab Android-Version 8.1(API-Level 27) funktioniert.

6 Zusammenfassung und Ausblick

Die Implementierung des Projekts ist abgeschlossen. Im Ganzen konnte die Arbeit nach Plan durchgeführt werden. Das Projekt zeigte, dass Android eine mächtige Plattform ist. Durch das Ausführen einer App mit deutlich strukturierten Komponenten, gewinnt Android an Popularität. Der Entwickler kann durch die Benutzung der Programmiersprache Kotlin und der Entwicklungsumgebung Android Studio große Programmiererfahrung sammeln. Dank dieser Erfahrungen sind die Entwicklung individueller Applikationen für Mobilgeräte möglich.

Die Weiterentwicklung dieser App endet nicht mit dem Abschluss dieses Projekts. Momentan enthält die implementierte Music Player App nicht alles, um Erwartungen von Benutzern erfüllen zu können. Viele Verbesserungen und zusätzliche Komponenten könnten hinkommen. Wegen den zahlreichen Erweiterungen ist die Veröffentlichung der App im Play Store unwahrscheinlich. Die Größe der App ist momentan wegen der VLC Bibliothek ziemlich hoch. Dies sollte noch reduziert werden, da es eine Größenbeschränkung für die Apps bei Google Play gibt. Die Beschränkung basiert auf der maximalen komprimierten Größe der APKs und ist 150MB [**b21**]. Bevor die erste solide Version irgendwo veröffentlicht werden kann, sollen noch weitere Verbesserungen durchgeführt und auf verschiedenen Smartphones getestet werden.

Im Bereich der internen Verarbeitung könnte in erster Linie eine Mitteilung implementiert werden, die dem Benutzer die Möglichkeit gibt, den gerade spielenden Titel ändern zu können, während er nicht mit der Music Player App beschäftigt ist. Die Mitteilung kann mithilfe der Klasse Service ermöglicht werden.
Ferner können Playlists zum Einsatz kommen. Hier könnte der Benutzer seine Lieblingstitel aus den unterschiedlichen Alben mithilfe eines Buttons(eventuell ein Herz Button) in eine Liste hinzufügen, während ein Titel abgespielt wird.
Ein Zufallsmodus wäre auch eine gute Idee, da es langweilig sein kann, eine Musiksammlung immer in derselben Reihenfolge zu hören. Mit dieser Funktion kann die Reihenfolge einer Liste verändert werden.
Eine Einstellung sollte unbedingt implementiert werden. Dabei könnte der Benutzer mögliche Änderungen einstellen, z.B. die Auswahl der Themen für die App und vieles mehr.
Eine weitere Möglichkeit für die Erweiterung der App besteht darin, die Musik für andere Plattformen, wie Instagram, freizugeben.

Die Benutzung der App ist derzeit auf die Android-Plattform beschränkt. Daher könnte die Entwicklung der Music Player App für das iOS-Betriebssystem den nächsten logischen Schritt in der Entwicklung der App darstellen.

References

[1] Kotlin Programmiersprache.
URL: https://de.wikipedia.org/wiki/Kotlin_(Programmiersprache). (accessed on 02.02.2023).

[2] Kotlin for Android.
URL: https://kotlinlang.org/docs/android-overview.html (accessed on 02.02.2023).

[3] Android Studio.
URL: https://de.wikipedia.org/wiki/Android_Studio/. (accessed on 03.02.2023).

[4] Meet Android Studio.
URL: https://developer.android.com/studio/intro?hl=en (accessed on 04.02.2023).

[5] Application Fundamentals
URL: https://developer.android.com/guide/components/fundamentals (accessed on 06.02.2023).

[6] libVLC
URL: https://www.videolan.org/vlc/libvlc.html (accessed on 07.02.2023).

[7] Tiago Henrique Trojahn, Luciano Volcan Agostini, Juliano Lucas Concalves. Evaluating two implementations of the component responsible for decoding video and audio in the Brazilian digital TV middleware. März 2012.

[8] Maoqiang Song, Haiyan Song. Methodology of User Interfaces Design Based on Android. Juli 2011.

[9] UI on Android.
URL: https://developer.android.com/develop/ui (accessed on 13.02.2023).

[10] MP3Tag.
URL: https://www.mp3tag.de/index.html (accessed on 14.02.2023).

[11] Permissions.
URL: https://developer.android.com/guide/topics/permissions/overview (accessed on 16.02.2023).

[12] Bellis, Mary. The History of MP3 Technology. Febraur 2021.

[13] Windows Media Audio.
URL: —https://de.wikipedia.org/wiki/Windows_Media_Audio (accessed on 17.02.2023).

[14] ID3.
URL: https://en.wikipedia.org/wiki/ID3 (accessed on 17.02.2023).

[15] Advanced Streaming Format.
URL: https://de.wikipedia.org/wiki/Advanced_Streaming_Format (accessed on 17.02.2023).

[16] Configure your build.
URL: https://developer.android.com/studio/build (accessed on 20.02.2023).

[17] Access media files from shared storage.
URL: https://developer.android.com/training/data-storage/shared/media (accessed on 20.02.2023).

[18] Bruno Gois Matues, Matias Matinez. An empirical study on quality of Android applications written in Kotlin language. 25.June.2019.

[19] Run apps on the Android Emulator.
URL: https://developer.android.com/studio/run/emulator (accessed on 21.02.2023).

[20] Android version history.
URL: https://en.wikipedia.org/wiki/Android_version_history (accessed on 21.02.2023).

[21] Play Store.
URL: https://support.google.com/googleplay/android-developer/answer/9859152?hl=en (accessed on 22.02.2023).