

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Raptor

Autor:
Mateusz Stanek
Dawid Szołdra
Filip Wąchała

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań projektu	5
1.1. Ogólny zarys wymagań	5
1.2. Wykorzystane czujniki	5
1.3. Zarys interfejsu	5
2. Określenie wymagań szczegółowych	8
2.1. Ogólny opis wymagań projektu	8
2.2. Ogólny zarys narzędzi użytych w projekcie	8
2.2.1. Android Studio	8
2.2.2. Kotlin	9
2.3. Wykorzystanie czujników	10
2.4. Zachowanie w niepożądanych sytuacjach	10
2.5. Dalszy rozwój	10
3. Projektowanie	11
3.1. Założenie programu	11
3.2. Przedstawienie menu	11
3.3. Odczyt i przetwarzanie plików	11
3.4. Struktura bazy danych	11
3.4.1. Ogólny opis	11
3.4.2. Opis pól tabel	12
3.4.2.1. Autorzy	12
3.4.2.2. Albumy	12
3.4.2.3. Piosenki	12
3.4.3. Relacje w bazie	12
3.5. Czujnik światła	13
3.6. Uwierzytelnianie biometrią	13
3.7. ViewModel	13
3.8. NavHost	14
3.8.1. Główne właściwości NavHost	14
3.8.2. Rodzaje NavHost	14

3.8.3. Kluczowe komponenty związane z NavHost	14
3.9. Audio player	15
4. Implementacja	16
4.1. Zarządzanie bazą danych	16
4.1.1. Klasa DatabaseManager	16
4.1.2. Klasa LibraryDb	20
4.1.3. Obiekty Dao	20
4.1.4. Tabele	23
4.1.5. Relacje	24
4.1.5.1. AlbumWithSongs	25
4.1.5.3. AlbumWithAuthors i AuthorWithAlbums	25
4.2. Czujnik światła	26
4.3. Autoryzacja odciskiem palca	29
4.4. Odczyt i przetwarzanie plików	33
4.4.1. Tag Extractor	33
4.4.2. MusicFileLoader	38
4.5. NavHost	42
5. Implementacja	47
5.1. Zarządzanie bazą danych	47
5.1.1. Klasa DatabaseManager	47
5.1.2. Klasa LibraryDb	51
5.1.3. Obiekty Dao	51
5.1.4. Tabele	54
5.1.5. Relacje	55
5.1.5.1. AlbumWithSongs	56
5.1.5.3. AlbumWithAuthors i AuthorWithAlbums	56
5.2. Czujnik światła	57
5.3. Autoryzacja odciskiem palca	60
5.4. Odczyt i przetwarzanie plików	64
5.4.1. Tag Extractor	64
5.4.2. MusicFileLoader	69

5.5. Ładowanie obrazów albumów	73
5.6. Audio player	75
6. Testowanie	79
6.1. Włączenie aplikacji	79
6.2. Dodawanie utworów	84
6.3. Navigacja	89
6.4. Działanie odtwarzacza	93
7. Podręcznik użytkownika	99
Literatura	120
Spis rysunków	121
Spis tabel	122
Spis listingów	123

1. Ogólne określenie wymagań projektu

1.1. Ogólny zarys wymagań

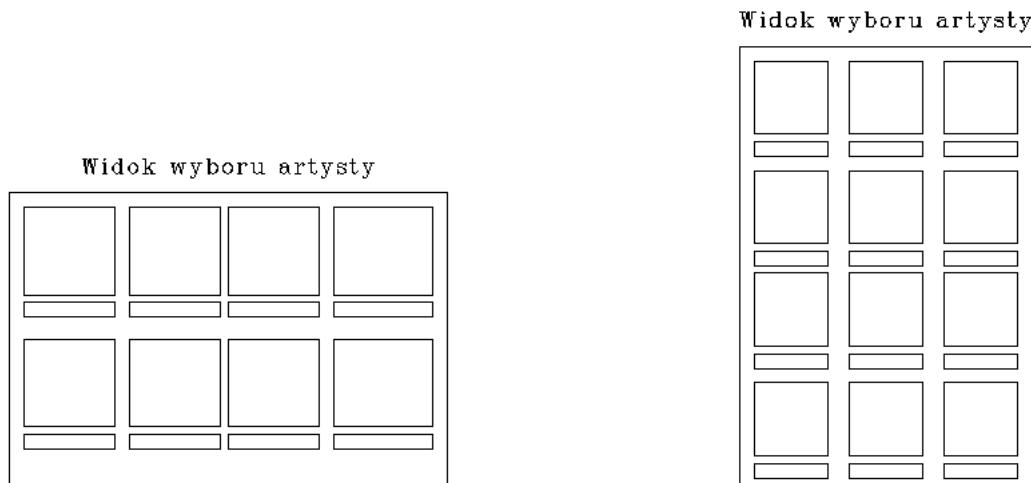
Celem programu jest pełnienie funkcji odtwarzacza muzyki. Program będzie mógł skanować dany folder i jego podfoldery, a w nich zawarty pliki muzyczne i tworzyć na ich podstawie bibliotekę, zapisaną na dysku.

1.2. Wykorzystane czujniki

Program ma na celu wykorzystanie trzech czujników, z którymi użytkownik będzie wchodził w interakcję. Zostaną użyte następujące:

- Żydroskop - Interfejs programu będzie się zmieniał w zależności od orientacji urządzenia.
- Wykrywacz odcisków palca - dostęp do programu powinien być ograniczony dla użytkowników mogących zweryfikować swój odcisk.
- Czujnik światła - Interfejs programu będzie mógł zmieniać swoje kolory w zależności od wykrytego poziomu światła na czujniku

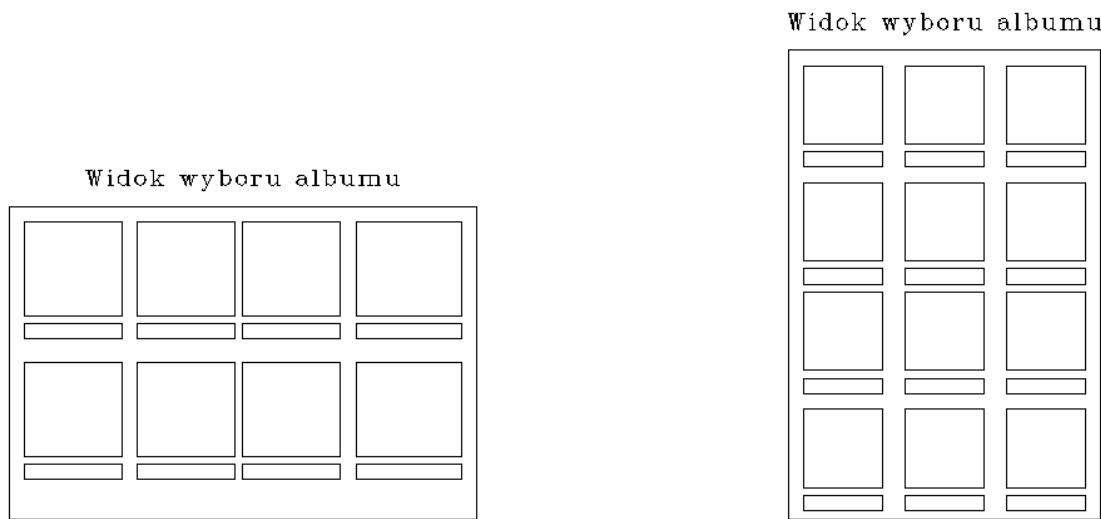
1.3. Zarys interfejsu



Rys. 1.1. Mockup widoku biblioteki - listing wykonawców

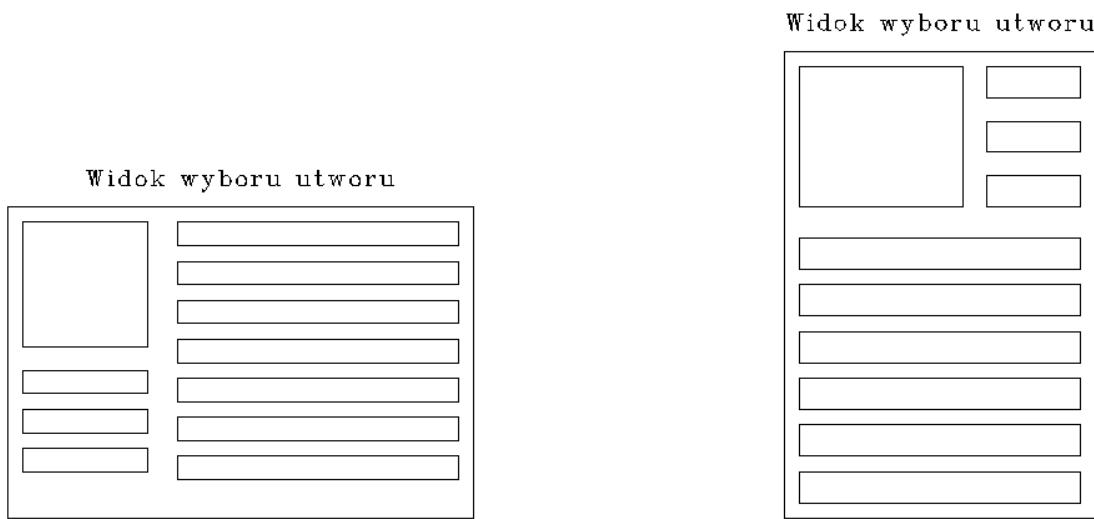
Widok wykonawców, jest przedstawiony na rysunku nr. 1.1. Ten widok będzie ekranem startowym aplikacji. Jako "Kafelki", zwracać będzie się dokument do ułożonych

równomiernie na rysunku kwadratów. Na każdym z nich napisana będzie nazwa danego wykonawcy. Klikanie na jeden z nich przejdzie do widoku albumów danego wykonawcy.



Rys. 1.2. Mockup widoku albumów danego wykonawcy

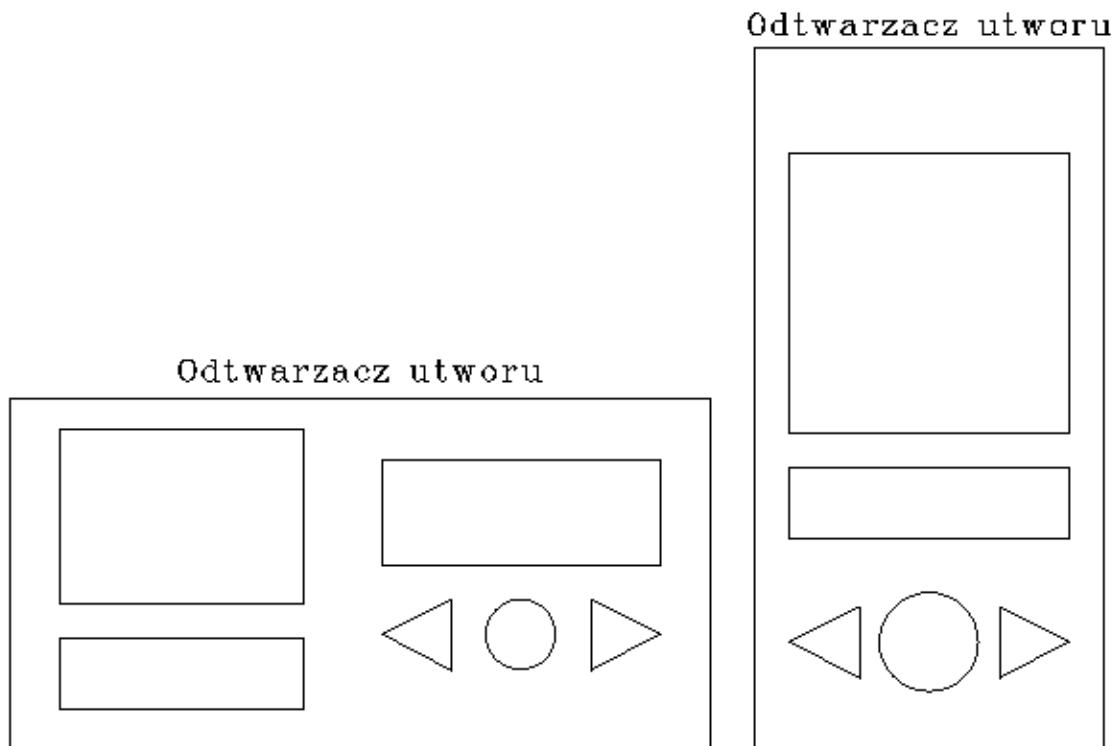
Widok albumów jest przedstawiony na rysunku nr. 1.2. Widok będzie podobny do widoku wykonawców. Różni się on tym, że na "kafelkach", będą pokazane zdjęcia poszczególnych albumów. Pod "kafelkami", znajdują się nazwy danych albumów.



Rys. 1.3. Mockup widoku wyboru utworu

Rysunek nr. 1.3 przedstawia ekran pokazujący się po wybraniu albumu. Po wejściu na jakiś album zaprezentowane zostaną zawarte w nim utwory. W lewym górnym kwadrat to zdjęcie danego albumu, a obok niego jest kilka informacji o albumie jak

wykonawca, data, tytuł, w postaci tekstu. Dłuższe paski zawarte na dole to lista piosenek, w postaci przycisków z napisanymi, tytułami które można kliknąć, aby daną piosenkę włączyć.



Rys. 1.4. Mockup widoku wyboru utworu

Wygląd interfejsu odtwarzacza został zaprezentowany na rysunku nr.1.4. W widoku horyzontalnym po lewej stronie na górze znajduje się obraz albumu a na dole pod obrazem będzie nazwa utworu, po prawej stronie na górze znajduje się pasik przewijania w formie soundwave który jest wyciągany z pliku muzycznego a na dole pod soundwave znajdują się przyciski pozwalające na manipulację utworem jak np. na zatrzymanie go lub przewinięcie. W widoku horyzontalnym na samym dole znajduje się zdjęcie albumu, poniżej nazwa utworu, potem soundwave a na końcu przyciski manipulacyjne.

2. Określenie wymagań szczegółowych

2.1. Ogólny opis wymagań projektu

Aplikacja jest zaprojektowana w Android Studio w języku Kotlin. Całe UI aplikacji będzie zbudowane na podstawie Frameworka Jetpack Compose^[1]. Używając wbudowanych bibliotek w SDK Androida, będzie mogła odczytywać pliki ze wskazanego folderu. Odczytywanie tagów z plików odbędzie się za pomocą biblioteki Media3^[2]. Jest to oficjalna biblioteka Googlea do obsługi plików medialnych na Androidzie. Wszelki processing audio np. na potrzeby wizualizacji może zostać wykonany za pomocą SDK i wbudowanego modułu AudioProcessor^[3]. Odtwarzaniem pliku będzie zajmowała się biblioteka ExoPlayer^[4], posiadająca częściową integrację z Media3. Informacje o utworach powinny być ładowane do bazy danych. Będzie ona lokalnie, na urządzeniu. Ku temu celu, można użyć biblioteki Room^[5]. Biblioteka ta jest wrapperem do wbudowanych funkcjonalności SQL Androida.

2.2. Ogólny zarys narzędzi użytych w projekcie

2.2.1. Android Studio

Android Studio jest IDE stworzonym przez Google, na bazie IntelliJ IDEA od JetBrains. Jest ono przystosowane, jak z nazwy wynika, do tworzenia aplikacji na Androida. Ku temu celu posiada wiele udogodnień, odróżniających program od typowego edytora jak np. wbudowany emulator Androida, integrujący się z całym środowiskiem, czy preview różnych elementów interfejsu - gdzie elementy te generowane są w kodzie, a nie w osobnym języku jak np. xml - bez potrzeby dekompilacji całej aplikacji.

Android Studio został użyty w projekcie, ponieważ:

- Sam program jest crossplatformowy - nasz zespół używa wielu systemów operacyjnych. Platformy takie jak MAUI, są zespolone z Visual Studio, czyli z Windowsem. Android Studio jest dostępny na wszystkie większe systemy operacyjne, co ułatwia nam pracę.
- Jest to program, zbudowany na podstawie IdeaJ, czyli zagłębiony jest w tym ekosystemie. Oznacza to dostęp do większej ilości pluginów niż np. Visual Studio, nie wspominając o ogólnej możliwości dostosowania ustawień.

Wady korzystania z Android Studio to m.in.

- Duże wykorzystanie zasobów - program lubi zżerać duże ilości RAMu. W tym momencie, mając otwarty mały projekt + emulator, program wykorzystuje ponad 9GB RAMu.

2.2.2. Kotlin

Kotlin został stworzony w 2010 roku przez firmę JetBrains oraz jest on przez nią rozwijany. Kotlin jest wieloplatformowym językiem typowanym statystycznie który został zaprojektowany aby współpracować z maszyną wirtualną Javy. Swoją nazwę zawdzięcza wyspie Kotlin która znajduje się w zatoce finlandzkiej.

Kotlin jest wykorzystywany w projekcie ze względów:

- Jest on wspierany przez Android Studio, razem z Javą i C++. Kotlin ponadto, ma dostęp do nowoczesnych frameworków jak Jetpack Compose
- Jest on *defakto* językiem do programowania na Androida - do niedawna Java mogła cieszyć się tym tytułem, ale od 2019 r. Google ogłosiło Kotlinę jako rekomendowany język do tworzenia aplikacji na Android.

Składnia Kotlina wygląda następująco:

```
1 fun main() {  
2     printf("Czesc to ja, kotlin!")  
3 }
```

Listing 1. kotlin001 - Funkcje

Definicja funkcji wykonywana jest za pomocą "fun".

Zmienne w Kotlinie deklarowane są za pomocą **val** i **var**. Różnica polega na tym, że zmienne oznaczone **val** mogą zostać modyfikowane natomiast zmienne oznaczone **val** już nie.

```
1 fun main() {  
2     var nazwa = "Projekt Android"  
3     val liczba = "777"  
4 }
```

Listing 2. kotlin002 - Zmienne

Kompilator Kotlina posiada funkcję autodedukcji typów, więc w wielu wypadkach typu zmiennej nie trzeba adnotować.

2.3. Wykorzystanie czujników

- Żyroskop - Z racji, że każdy element interfejsu w Jetpack jest generowany kodem, można, przynajmniej na początku, ustawić każdą wersję interfejsu jako osobną funkcję. Następnie, w zależności od wykrytej orientacji, przy użyciu API sensorów[6], można wywoływać odpowiednią funkcję.
- Mikrofon - Funkcja dyktafonu najprawdopodobniej będzie całkiem oddzielnym Activity. Funkcjonalność ta, z natury, jest dosyć oddzielna od reszty aplikacji. Nagrania dyktafonem powinny być zapisywane do osobnego folderu. Można by zintegrować nagrania z resztą aplikacji jako osobnego wykonawcę w widoku biblioteki. Mikrofon będzie nagrywany poprzez moduł MediaRecorder[7]
- Czujnik światła - Android Studio oferuje możliwość definiowania własnych klas zajmujących się kolorystyką. Oznacza to że można używać różnych obiektów w zależności od warunków. Wykrywanie światła będzie się odbywało używając API sensorów[6]

2.4. Zachowanie w niepożądanych sytuacjach

Głównym wyjątkiem, na który może napotkać się aplikacja jest błąd odczytu albo plików, albo tagów z pliku. Kotlin, na szczęście, pozwala na łatwe sprawdzanie wartości null danych zmiennych operatorem ?. W odpowiednich fragmentach kodu dotyczących ładowania plików, będzie sprawdzana poprawność danych i najprawdopodobniej pojawi się pop-up po stronie użytkownika, że wystąpił błąd, a po stronie dewelopera błąd zostanie logowany.

2.5. Dalszy rozwój

Jeżeli praca nad aplikacją będzie się odbywała w przyszłości, należy skupić uwagę na lepszym zarządzaniu biblioteką (auto tagowanie, pobieranie miniatur z internetu, itp.). Ponadto, należy szukać błędów, które nadal zostały w aplikacji.

3. Projektowanie

3.1. Założenie programu

W tym rozdziale przedstawiona zostanie ogólna zasada działania programu.

Głównym celem programu jest odtwarzanie muzyki.

3.2. Przedstawienie menu

Program składa się z trzech okien.

- Okno wyboru autora - AuthorsView()
- Okno wyboru albumu - AuthorView()
- Okno wyboru utworów - SongView()

Aplikacja włączając się wyświetla menu wyboru autora. Menu przedstawione jest w postaci kafelkowej.

Po wybraniu autora włączane jest menu wyboru albumu.

Po wybraniu albumu otwirane jest menu wyboru piosenek należących do tego utworu.

3.3. Odczyt i przetwarzanie plików

3.4. Struktura bazy danych

3.4.1. Ogólny opis

Baza danych jest złożona z trzech tabel:

- Autorzy - tabela ta ma zawierać wszystkie informacje o autorach z biblioteki użytkownika. Założeniem jest, że każdy autor ma unikalną nazwę, ponieważ nie ma żadnego dobrego sposobu unikalnej identyfikacji autorów z samych lokalnych plików.
- Albumy - tabela ta, oprócz katalogowania albumów, głównie pełni rolę „pośrednika” między piosenkami a autorami. Ważną informacją jaką zawiera każdy rekord, jest odnośnik do okładki danego albumu. Opisane jest to w sekcji nr. 3.4.3. Warto wspomnieć, że albumy każdego autora muszą mieć unikalne

nazwy - problem identyfikacji jest podobny jak przy autorach - lecz nazwy albumów różnych autorów mogą się powtarzać.

- Piosenki - tabela ta zawiera informacje o wszystkich piosenkach w bibliotece, pozyskane z tagów plików.

Detale dotyczące każdej z tabel można przeczytać w sekcji nr. 3.4.2.

3.4.2. Opis pól tabel

3.4.2.1. Autorzy

- **nazwa** - unikalna nazwa autora, jest zarazem kluczem głównym

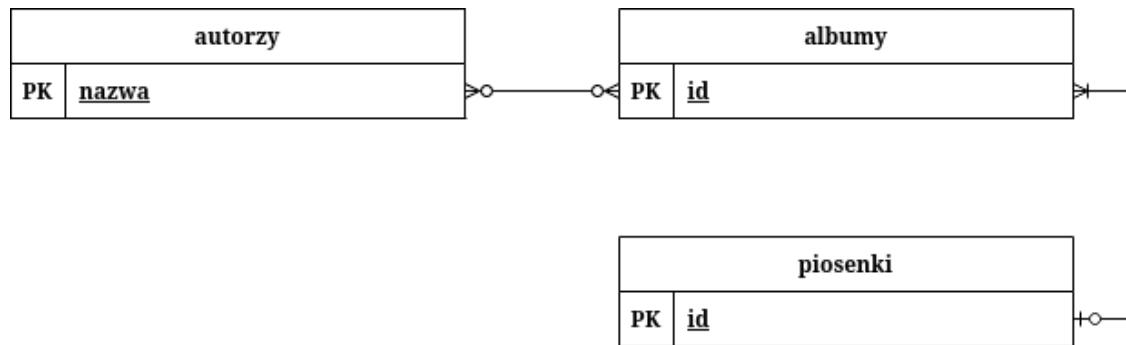
3.4.2.2. Albumy

- **id** - klucz główny, unikatowy identyfikator albumu
- **nazwa** - nazwa albumu, unikalna w obrębie jednego autora
- **tytuł** - tytuł albumu
- **okładka** - ścieżka do pliku z okładką albumu

3.4.2.3. Piosenki

- **id** - klucz główny, unikatowy identyfikator piosenki
- **tytuł** - tytuł piosenki
- **album** - klucz obcy, odniesienie do albumu, do którego należy piosenka
- **ścieżka** - ścieżka do pliku z piosenką

3.4.3. Relacje w bazie



Rys. 3.1. Model relacji bazy

Na rysunku nr. 3.1 ukazany jest uproszczony model bazy danych, biorący tylko pod uwagę komponenty potrzebne do określenia relacji. Autorzy są w relacji M do N z albumami. Każdy autor może mieć wiele albumów, a każdy album wiele autorów. Piosenki z albumami są w relacji 1 do N . Każda piosenka może należeć wyłącznie do jednego albumu, ale każdy album może mieć wiele piosenek.

Warto zauważyć, że piosenki nie bezpośrednio łączą się z autorami. Jeżeli piosenka chce uzyskać swojego autora, musi zrobić to poprzez album.

3.5. Czujnik światła

Zadanie czujnika światła jest dość proste i niezbyt skomplikowane. Celem czujnika jest wykrywanie intensywności światła i na podstawie tej intensywności aktualizacja wyglądu menu poprzez wykorzystanie schematów kolorów oraz motywów. Założenie jest takie, że przy intensywności światła mniejszej niż 50 procent z 10000 lumenów aplikacja będzie przełączać się na tryb ciemny, przy intensywności większej niż 50 procent aplikacja będzie przełączać się na tryb jasny.

3.6. Uwierzytelnianie biometrią

Aplikacja będzie wykorzystywać sensor biometryczny odcisku palca do autoryzacji użytkownika. Przed wejściem do aplikacji, zostanie wywołany ekran uwierzytelniania udostępniony przez system, na tym ekranie będzie okienko na którym będzie znajdowało się miejsce, gdzie będzie można przyłożyć palec. Aplikacja będzie sprawdzać czy odcisk palca znajduje się zapisany na telefonie i będzie wykorzystywać ten odcisk do uwierzytelniania.

3.7. ViewModel

ViewModel jest komponentem architektury, który ma za zadanie zarządzanie danymi i zapewnienie ich przetrwania w przypadku zmian konfiguracji, takich jak rotacja ekranu. Oddziela logikę biznesową i dane od interfejsu użytkownika, co upraszcza zarządzanie cyklem życia aktywności lub fragmentów. Dzięki temu ViewModel przechowuje dane nawet wtedy, gdy zostaną one ponownie utworzone.

Dzięki przeniesieniu logiki biznesowej do ViewModel, aktywność czy fragment nie muszą się martwić o przechowywanie stanu danych. ViewModel jest częścią Android Architecture Components, co umożliwia reakcję na zmiany danych i aktualizację UI

w sposób zautomatyzowany.

3.8. NavHost

NavController jest komponentem Androida używanym do nawigacji w aplikacjach. Działa jako kontener, w którym przechowywane są fragmenty aplikacji, między którymi aplikacja może nawigować. NavController nie tylko przechowuje aktualny stan nawigacji, ale także zarządza przejściami i ich animacjami między fragmentami.

3.8.1. Główne właściwości NavHost

- **Przechowywanie nawigacji:** NavController trzyma aktualny stan nawigacji, który jest określony przez bieżący NavController.
- **Dynamiczna nawigacja:** Pozwala użytkownikom na dynamiczną nawigację między różnymi ekranami aplikacji, a także zarządza animacjami przejść.
- **Integracja z NavController:** NavController pracuje razem z NavController, który kontroluje nawigację między fragmentami i zapobiega ręcznej manipulacji fragmentami bezpośrednio w kodzie.
- **Łatwy dostęp w XML:** Można łatwo definiować NavController w pliku układu XML aplikacji, co ułatwia jego konfigurację i zarządzanie.

3.8.2. Rodzaje NavHost

- **NavControllerFragment:** Jest to specjalna implementacja Fragment, która hostuje inne fragmenty do nawigacji.
- **BottomNavigationView:** Może być zintegrowany z NavController w celu obsługi nawigacji w aplikacjach z dolnym paskiem nawigacyjnym.
- **DrawerLayout:** Może być używany z NavController w aplikacjach, które mają menu nawigacyjne dostępne z boku ekranu.

3.8.3. Kluczowe komponenty związane z NavController

- **NavController:** Centralny obiekt, który zarządza wszystkimi aspektami nawigacji w danym NavController. Używa się go do przeprowadzania przejść między fragmentami oraz zachowywaniu historii nawigacji.

- **NavGraph:** Jest to definicja struktury nawigacji, która określa wszystkie cele nawigacyjne oraz dostępne akcje między nimi.

3.9. Audio player

Audio player korzysta z frameworka **ExoPlayer**. Zadaniem audio player będzie odtwarzanie oraz zarządzanie plikami audio, kontrolę czasu odtwarzania oraz zmianę stanu odtwarzania.

4. Implementacja

4.1. Zarządzanie bazą danych

4.1.1. Klasa DatabaseManager

Za zarządzanie bazą danych odpowiedzialna jest klasa `DatabaseManager`, której kod jest zamieszony na listingu nr. 26. Klasa jest wrapperem do bazy danych Room[5] i do niej akcesorów.

```
1 @Singleton
2 class DatabaseManager @Inject constructor(
3     @ApplicationContext context: Context
4 ) {
5     private val database: LibraryDb = Room.databaseBuilder(
6         context,
7         LibraryDb::class.java, "Library"
8     ).build()
9
10    fun collectAuthorsFlow(): Flow<List<Author>> = database.uiDao()
11        .getAllAuthorsFlow()
12
13    fun collectAlbumsByAuthorFlow(authorName: String): Flow<List<
14        Album>> {
15        return database.uiDao().getAuthorWithAlbums(authorName)
16            .map { it.albums }
17    }
18
19    fun collectSongsByAlbumFlow(albumId: Long): Flow<List<Song>> {
20        return database.uiDao().getAlbumWithSongs(albumId)
21            .map { it.songs }
22    }
23
24    ...
25
26    fun populateDatabase(songs: List<TagExtractor.SongInfo>) {
27        assert(Thread.currentThread().name != "main")
28
29        val dao = database.logicDao()
30
31        fun addAuthors() {
32            songs.fastForEach { song ->
33                //TODO: there should be a distinction between
34                albumartists and regular artists
35                song.albumArtists?.fastForEach { name ->
```

```
33             if(dao.getAuthor(name) == null) {
34                 dao.insertAuthor(Author(name = name))
35             }
36         }
37     }
38 }
39
40
41     fun addAlbumsAndRelations() {
42         // FIXME: xddddddd
43         val distinctAlbumArtistsList = songs
44             .map { Triple(it.album, it.albumArtists, it.
45         coverUri) }
46             .distinct()
47         Log.d(javaClass.simpleName, "Distinct artists set:
48             $distinctAlbumArtistsList")
49
50         distinctAlbumArtistsList.fastForEach {
51             val albumTitle = it.first.toString()
52             val artists = it.second
53             val coverUri = it.third
54
55             val albumId = dao.insertAlbum(Album(
56                 title = albumTitle,
57                 coverUri = coverUri.toString(),
58             ))
59
60             artists?.fastForEach {
61                 dao.insertAlbumAuthorCrossRef(
62                     AlbumAuthorCrossRef(
63                         albumId = albumId,
64                         name = it.toString()
65                     )
66                 }
67             }
68         }
69     }
70
71     fun addSongs() {
72         songs.forEach { song ->
73             Log.d(javaClass.simpleName, "NEW SONG\n")
74             Log.d(javaClass.simpleName, "Album artists: ${song.
75         albumArtists}")
76
77             val albumWithAuthorCandidates = dao
78                 .getAlbumsByTitle(song.album.toString())
79         }
80     }
81
82 }
```

```
74         .map { it.albumId }
75             .map { dao.getAlbumWithAuthors(it) }
76             Log.d(javaClass.simpleName, "
77                 $albumWithAuthorCandidates")
78
79             var correctAlbum: Album? = null
80             albumWithAuthorCandidates.fastForEach {
81                 Log.d(javaClass.simpleName, "${song.
82                     albumArtists}, ${it.authors}")
83                     //FIXME: theese guys shouldn't be ordered, will
84                     have to refactor a bunch of
85                     // stuff with sets instead of lists
86                     if(song.albumArtists?.sorted() == it.authors.
87                         map { it.name }.sorted()) {
88                         correctAlbum = it.album
89                     }
90                 }
91
92             dao.insertSong(Song(
93                 title = song.title,
94                 albumId = correctAlbum?.albumId,
95                 fileUri = song.fileUri.toString(),
96             ))
97         }
98     }
99 }
100 }
```

Listing 3. Strukutura klasy DatabaseManager

Na pierwszej linijce można zauważyć adnotację `@Singleton`. Pochodzi ona z biblioteki `Hilt`[8]. Powiadamia ona bibliotekę o tym że klasa jest singletonem, czyli że ma istnieć tylko jej jedna instancja na cały program. Uczyniono to, dlatego że baza danych powinna być jedna na całą aplikację. Menadżer z nią interfejsujący, dlatego że jest używany w wielu innych klasach, też powinien mieć tylko jedną instancję, aby nie marnować pamięci.

Na linijce nr. 2, widać konstruktor klasy, do którego też przy użyciu `Hilt`, wstrzykiwany jest `context`.

Następnie, na linijce nr. 5, widać inicjalizację samego obiektu bazy `database`.

Baza jest reprezentowana przez klasę `LibraryDb`, definicję której można zobaczyć w sekcji 5.1.2

Dalej, do linijki nr. 22 pokazane są metody zwracające różne elementy bazy. Większość z tych metod zwraca `Flow[TODO:]`. Room natywnie obsługuje Flows, a dlatego że wymusza dostęp do bazy z innych wątków niż główny, większość operacji wykonywanych na bazie odbywa się za pośrednictwem typów Flow

Same metody są wrapperami do obiektów Dao bazy.Więcej o nich w sekcji 5.1.3. Niektóre obrabiają dane jak np. `collectSongsByAlbumFlow()` na linijce nr. 17., która mapuje zwraca piosenki z wyjściowej klasy relacyjnej.

Metod tych jest więcej, lecz wyglądają one bardzo podobnie. Dla zwięzłości, można je pominać.

Metoda `populateDatabase()` zadeklarowana na linijce nr. 24, jest odpowiedzialna za ładowanie wyjętych z plików informacji do bazy. Jako parametr odstaje ona zmienna `songs` typu `List<TagExtractor.SongInfo>`. Zadeklarowane są w niej trzy funkcje pomocnicze: `addAuthors()`, `addAlbumsAndRelations()` i `addSongs()`. Wywoływanie są one po kolej w metodzie głównej.

Funkcja `addAuthors()`, zadeklarowana na linijce nr. 29, jest prosta w swoim działaniu. Lista z `SongInfo` jest iterowana i po kolej wpisywani są wszyscy autorzy, którzy jeszcze w bazie nie istnieją.

Funkcja `addAlbumsAndRelations()`, zadeklarowana na linijce nr. 41, odpowiada za dodawanie albumów do bazy oraz tworzenie relacji między nimi, a autorami. Tworzona zmienna `distinctAlbumArtistsList` mapuje tylko unikalne pary albumów i autorów (zmienna `coverUri` nie ma znaczenia przy określaniu autorstwa, jest przypisywana tutaj dlatego, że trudno było znaleźć dla niej lepsze miejsce). Dzięki temu początkowemu filtrowaniu, wiadomo, że każdy napotkany album będzie unikalny. Następnie, `distinctAlbumArtistsList` jest iterowana - przy każdej iteracji dodawany jest nowy album do bazy. Metoda `insertAlbum()` zwraca `id` nowo dodanego albumu. Wynik jej jest przypisywany do zmiennej `albumId` na linijce nr. 53. Potem, zostaje przypisywana relacja albumu z autorami. Autorów może być kilku, więc są oni reprezentowani przy każdej iteracji przez listę, która jest iterowana, a relacja zostaje dodawana z nazwą autora i `albumId`.

Funkcja `addSongs()`, zadeklarowana na linijce nr. 67, ma na celu dodanie piosenek do bazy. Ciało funkcji jest w pętli iterującej się przez listę piosenek. Na początku pętli, na linijce nr. 72 deklarowana jest zmienna `albumWithAuthorCandidates`. Jest ona listą relacji album - autorzy wszystkich albumów o tej samej nazwie co ten w danym elemencie listy. Następnie, lista ta jest iterowana i przy każdej iteracji spraw-

dzane jest czy lista autorów w danej relacji jest równa z listą autorów danej piosenki. Jeżeli tak, wartość danego albumu z wybranej relacji jest przypisywana do zmiennej zadeklarowanej na linii nr. 78 `correctAlbum`. Na końcu funkcji, piosenka dodana jest do bazy przy użyciu metody `dao`.

4.1.2. Klasa LibraryDb

Klasa `LibraryDb` jest deklaracją faktycznej instancji bazy danych, która jest implementowana i generowana przez bibliotekę `Room`. Z racji tego, że jest to klasa abstrakcyjna, jej zadaniem jest określenie struktury bazy i jakie komponenty ma ona zawierać

```
1 @Database(entities = [Song::class, Author::class, Album::class,  
           AlbumAuthorCrossRef::class], version  
2 = 1)  
3 abstract class LibraryDb : RoomDatabase() {  
4     abstract fun logicDao(): LogicDao  
5     abstract fun uiDao(): UIDao  
6 }
```

Listing 4. Deklaracja bazy LibraryDb

Jak widać na listingu nr. 27, na początku klasy należy zamieścić adnotację `@Database`. Powiadamia ona bibliotekę `Room` o tym, że następująca klasa jest bazą danych. Parametr `entities` określa wszystkie tabele jakie mają się w klasie zawierać. O tabelach więcej w sekcji nr. 5.1.4. Parametr `version` zajmuje się wersjonowaniem bazy. Jest on ważny przy aktualizacjach aplikacji, aby baza mogła zostać odpowiednio zmieniona.

Na linijce nr. 3 umieszczona jest faktyczna deklaracja klasy. Dziedziczy ona z klasy `RoomDatabase`. Jedyne rzeczy jakie są do dziecięcej klasy dodawane, to metody zwracające obiekty `dao`, opisane w sekcji nr. 5.1.3.

4.1.3. Obiekty Dao

Obiekty `dao` (Data Access Object(s)) to obiekty używane do interakcji z zawartością bazy danych. Głównie używa się ich do dodawania elementów do bazy oraz ich odczytywania. Same obiekty definiuje się jako interfejsy z adnotacją `@Dao`. Są one implementowane przez `Room`. Baza danych w projekcie wykorzystuje dwa interfejsy `dao - UIDao`, którego kod zamieszczony jest na listingu nr. 28 oraz `LogicDao`, którego kod zamieszczony jest na listingu nr. 29.

```
1 @Dao
```

```
2 interface UIDao {
3     @Query("SELECT * FROM Song")
4     fun getAllSongs(): Flow<List<Song>>
5
6     @Query("SELECT * FROM Song WHERE songId = :songId")
7     fun collectSongFromId(songId: Long): Flow<Song>
8
9     // Get an album with its songs
10    @Transaction
11    @Query("SELECT * FROM album WHERE albumId = :albumId")
12    fun getAlbumWithSongs(albumId: Long): Flow<AlbumWithSongs>
13
14    @Query("SELECT * FROM album WHERE albumId = :albumId")
15    fun getAlbumById(albumId: Long?): Flow<Album>
16
17    // Get an album with its authors
18    @Transaction
19    @Query("SELECT * FROM album WHERE albumId = :albumId")
20    fun getAlbumWithAuthors(albumId: Long?): Flow<AlbumWithAuthors
?>
21
22    @Query("SELECT * FROM Author")
23    fun getAllAuthorsFlow(): Flow<List<Author>>
24
25    // Get an author with their albums
26    @Transaction
27    @Query("SELECT * FROM author WHERE name = :name")
28    fun getAuthorWithAlbums(name: String): Flow<AuthorWithAlbums>
29 }
```

Listing 5. Deklaracja interfejsu UIDao

```
1 @Dao
2 interface LogicDao {
3     @Insert
4     fun insertSong(song: Song)
5
6     @Insert
7     fun insertAlbum(album: Album): Long
8
9     @Insert(onConflict = OnConflictStrategy.REPLACE)
10    fun insertAuthor(author: Author)
11
12    @Insert(onConflict = OnConflictStrategy.REPLACE)
13    fun insertAlbumAuthorCrossRef(albumAuthorCrossRef:
AlbumAuthorCrossRef)
```

```
14  
15     @Query("SELECT * FROM Author")  
16     fun getAllAuthors(): List<Author>  
17  
18     @Transaction  
19     @Query("SELECT * FROM album WHERE albumId = :albumId")  
20     fun getAlbumWithAuthors(albumId: Long): AlbumWithAuthors  
21  
22     @Transaction  
23     @Query("SELECT * FROM author WHERE name = :name")  
24     fun getAuthorWithAlbums(name: String): AuthorWithAlbums  
25  
26     @Query("SELECT * FROM author WHERE name = :name")  
27     fun getAuthor(name: String): Author?  
28  
29     @Query("SELECT * FROM album WHERE title = :title")  
30     fun getAlbumsByTitle(title: String): List<Album>  
31  
32     @Query("SELECT * FROM album WHERE title = :title LIMIT 1")  
33     fun getAlbumByTitle(title: String): Album?  
34  
35     @Query("SELECT * FROM AlbumAuthorCrossRef WHERE albumId = :  
36         albumId AND name = :authorName LIMIT 1")  
37     fun getCrossRefByAlbumAndAuthor(albumId: Long, authorName:  
38         String): AlbumAuthorCrossRef?  
39  
40 }
```

Listing 6. Deklaracja interfejsu LogicDao

Dlatego, że baza Room wymaga dostępu do elementów z innego wątku niż główny, LogicDao może być tylko używany w kodzie, o którym wiadomo, że nie jest wykonywany na głównym wątku. UI Dao natomiast, służy ekskluzywnie do zwracania Flowów. Większość elementów związanych z interfejsem w reszcie kodu aplikacji już korzysta z Flowów, więc dao to łatwo jest zintegrować.

Typowy sposób w jaki dodaje się element do bazy znajduje się na linijce nr. 4 w kodzie LogicDao, na listingu nr. 29. Funkcja `insertSong()`, zadnotowana jest `@Insert`. Powiadamia to Room, że funkcja ta odpowiedzialna jest za dodawanie elementu. Parametr `song` to piosenka jaka ma być dodana. W następnej funkcji `insertAlbum()` widać, że funkcje `@Insert` mogą zwracać wartości. W tym przypadku funkcja zwraca `id` nowo dodanego albumu. Można też zwrócić uwagę na

metodę `insertAuthor()` na linijce nr. 8, a w szczególności parametr `onConflict` w adnotacji `@Insert`. Wartość parametru `OnConflictStrategy.REPLACE` mówi bibliotece, aby nie pomijała elementów o tych samych wartościach co już są w tabeli, ale zamieniała starsze na te nowe.

Przykład odczytywania elementu jest dobrze zilustrowany na metodzie `getAuthor()` zadeklarowanej na linijce nr. 27. Adnotacja `@Query` przyjmuje parametr `String`, który jest kwerendą SQL jaka ma być wykonana. Kwerenda `SELECT * FROM author WHERE name = :name` wybiera wszystkich autorów, których pole `name` równe jest parametrowi metody `name` (odnoszenie do parametru w kwerendzie poprzedzone jest znakiem „`:`”). Dlatego że w bazie może być tylko jeden autor z daną nazwą, zwracany jest pojedynczy autor, a nie ich lista. Niektóre metody, jak na linijce nr. 20 `getAlbumWithAuthors()`, używają adnotacji `@Transaction`. W przypadku tej metody, zwraca ona relację, czyli czyta z kilku tabel. Adnotacja `@Transaction` zapewnia, że transakcja jest atomiczna, co za tym idzie, inne wątki nie mogą nagle zmienić wartości jakiejs tabeli.

Interfejs `UIdao` działa podobnie jak `LogicDao`, ale zwraca on tylko i wyłącznie Flows, które są natywnie obsługiwane przez Room.

4.1.4. Tabele

W bibliotece Room, każda tabela to `dataclass` określana adnotacją `@Entity`. Kolumny takiej tabeli to po prostu pola klasy. Klucz danej tabeli jest określany adnotacją `@PrimaryKey`

```
1 @Entity
2 data class Author (
3     @PrimaryKey val name: String
4 )
```

Listing 7. Deklaracja tabeli Author

Tabela `Author`, zawarta na listingu nr. 30, określa autorów. Tabela jest prosta, jedynym polem jest `name`, który jest kluczem.

```
1 @Entity
2 data class Album(
3     @PrimaryKey(autoGenerate = true) val albumId: Long = 0,
4     val title: String,
5     val coverUri: String?,
```

6)

Listing 8. Deklaracja tabeli Album

Tabela **Album**, zawarta na listingu nr. 31, określa albumy. Kluczem jest zmienna **albumId**. Klucz jest generowany automatycznie, dzięki parametrowi adnotacji **autoGenerate**. Pole **title** określa tytuł, a pole **coverUri** określa adres URI okładki.

```

1
2 @Entity(
3     foreignKeys = [
4         ForeignKey(
5             entity = Album::class,
6             parentColumns = ["albumId"],
7             childColumns = ["albumId"],
8             onDelete = ForeignKey.CASCADE
9         )
10    ],
11
12    indices = [Index(value = ["albumId"])]
13 )
14 data class Song(
15     @PrimaryKey(autoGenerate = true) val songId: Long = 0,
16     val title: String?,
17     val albumId: Long?,
18     val fileUri: String?,
19 )

```

Listing 9. Deklaracja tabeli Song

Klasa ta, zawarta na listingu nr. 32, określa tabelę piosenek. Pole **foreignKeys** w adnotacji **@Entity** określa obce klucze, którymi posługuje się klasa. W tym przypadku określone jest to, że pole w **Song albumId** wskazuje na pole w **Album albumId**. Pole **indices** każe indeksować pola z **albumId** ku polepszeniu szybkości bazy. W ciele klasy, pole **title** to tytuł piosenki. Pole **albumId** określa ID albumu, do którego należy dana piosenka.

4.1.5. Relacje

Relacje w Room są określane jako osobne **dataclassy**. Są one zadeklarowane adnotacją **@Relation** w danej klasie. Ponadto umieszczenie elementu w adnotacji

@Embedded, pozwala klasie „przyswoić” pola danego elementu. Dzięki temu klasa może odnosić się do pól danego elementu tak jakby były one bezpośrednio w klasie. Konieczne jest umieszczenie elementu głównego, od którego będzie relacja wyodziła, do tej adnotacji.

4.1.5.1. AlbumWithSongs

Klasa `AlbumWithSongs` na listingu nr. 33, określa relację albumów i piosenek.

```
1 data class AlbumWithSongs(
2     @Embedded val album: Album,
3     @Relation(
4         parentColumn = "albumId",
5         entityColumn = "albumId"
6     )
7     val songs: List<Song>
8 )
```

Listing 10. Deklaracja relacji `AlbumWithSongs`

Relacja łączy pole `albumId` albumu z polem `albumId` piosenek. Pole `songs` zawiera wszystkie piosenki z tą samą wartością pola `albumId` co faktyczny klucz danego albumu.

```
1 @Entity(primaryKeys = ["albumId", "name"])
2 data class AlbumAuthorCrossRef(
3     val albumId: Long,
4     val name: String
5 )
```

Listing 11. Deklaracja tabeli relacji `AlbumAuthorCrossRef`

Tabela na listingu nr. 34 określa relację *M* do *N* między albumami a autorami. Jest to tabela z dwoma kluczami głównymi: `albumId` dla tabeli `Album` i `name` dla tabeli `Author`.

4.1.5.3. AlbumWithAuthors i AuthorWithAlbums

Obie klasy są do siebie bardzo podobne więc zostaną omówione razem.

```
1 data class AlbumWithAuthors(
2     @Embedded val album: Album,
3     @Relation(
```

```
4     parentColumn = "albumId",
5     entityColumn = "name",
6     associateBy = Junction(AlbumAuthorCrossRef::class)
7   )
8   val authors: List<Author>
9 )
```

Listing 12. Deklaracja relacji `AlbumWithAuthors`

```
1 data class AuthorWithAlbums(
2   @Embedded val author: Author,
3   @Relation(
4     parentColumn = "name",
5     entityColumn = "albumId",
6     associateBy = Junction(AlbumAuthorCrossRef::class)
7   )
8   val albums: List<Album>
9 )
```

Listing 13. Deklaracja relacji `AuthorWithAlbums`

Na listingu nr. 35 przedstawiona jest klasa `AlbumWithAuthors`. Definiuje ona relację danego albumu z jego autorami. Dlatego, że relacja jest M do N , w anotacji `@Relation` dodane jest odniesienie do tabeli relacji `AlbumAuthorCrossRef`, opisanej w sekcji nr. 5.1.5.2. Klucze, jakie mają być porównywane są zdefiniowane w parametrach `parentColumn`, dla id albumu i `entityColumn` dla nazwy autora. Wynikiem tej relacji jest lista albumów. Sytuacja wygląda podobnie w `AuthorWithAlbums`, na listingu nr. 36. Tym razem to autor jest rodzicem i oczekujemy od relacji listy albumów danego autora.

4.2. Czujnik światła

Czujnik światła został zaimplementowany za pomocą wbudowanej funkcji. Zadaniem czujnika jest dynamiczna zmiana schematu kolorów aplikacji na podstawie danych otrzymanych z czujnika światła wbudowanego w urządzeniu mobilnym z systemem android.

```
1 @AndroidEntryPoint
2 class MainActivity : FragmentActivity(), SensorEventListener {
3   private lateinit var sensorManager: SensorManager
4   private var lightSensor: Sensor? = null
5   private val _isDarkTheme = mutableStateOf(false)
6   private val isDarkTheme: State<Boolean> = _isDarkTheme
7 }
```

```
8     private val _isAuthenticated = mutableStateOf(false)
9     private val isAuthenticated: State<Boolean> = _isAuthenticated
10
11    override fun onCreate(savedInstanceState: Bundle?) {
12        super.onCreate(savedInstanceState)
13        val biometricAuthenticator = BiometricAuthenticator(this)
14
15        sensorManager = getSystemService(Context.SENSOR_SERVICE) as
16        SensorManager
17        lightSensor = sensorManager.getDefaultSensor(Sensor.
18            TYPE_LIGHT)
19
20        setContent {
21            val darkTheme by isDarkTheme
22            val authenticated by isAuthenticated
23            RaptorTheme(darkTheme = darkTheme) {
24                Surface(
25                    modifier = Modifier.fillMaxSize(),
26                    color = MaterialTheme.colorScheme.background
27                ) {
28                    if (authenticated) {
29                        MainScreen()
30                    } else {
31                        AuthenticationScreen(
32                            onAuthenticate = {
33                                promptBiometricAuthentication(
34                                    biometricAuthenticator)
35                            }
36                        )
37                    }
38                }
39            }
40        }
41
42        private fun promptBiometricAuthentication(
43            biometricAuthenticator: BiometricAuthenticator) {
44            biometricAuthenticator.PromptBiometricAuth(
45                title = "Authentication Required",
46                subtitle = "Please authenticate to proceed",
47                negativeButtonText = "Cancel",
48                fragmentActivity = this,
49                onSuccess = {
```

```
49     runOnUiThread {
50         _isAuthenticated.value = true
51     }
52 },
53 onFailed = {
54 },
55 onError = { errorCode, errorString ->
56 }
57 )
58 }
59
60 override fun onResume() {
61     super.onResume()
62     lightSensor?.let { sensor ->
63         sensorManager.registerListener(this, sensor, SensorManager.
64 SENSOR_DELAY_NORMAL)
65     }
66 }
67
68 override fun onPause() {
69     super.onPause()
70     sensorManager.unregisterListener(this)
71 }
72
73 override fun onSensorChanged(event: SensorEvent?) {
74     if (event?.sensor?.type == Sensor.TYPE_LIGHT) {
75         val lightLevel = event.values[0]
76         val maxLightLevel = lightSensor?.maximumRange ?: 10000f
77
78         _isDarkTheme.value = lightLevel < 0.4 * maxLightLevel
79     }
80 }
81
82     override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int)
83 { }
```

Listing 14. Implementacja czujnika światła w `MainActivity.kt`

Na listingu 37 przedstawiona jest funkcja MainScreen. W wierszu 2 mamy `SensorEventListener`, który jest frameworkm w androidzie który pozwala aplikacji na reagowanie na zmiany odczytywane przez czujniki smartfona. Po dodaniu automatycznie tworzone są funkcje `onSensorChanged`, `onResume`, `onPause`, `onAccuracyChanged`. Implementację sensora zaczynamy od utworzenia zmiennych `sensorma-`

nager oraz lightSensor w wierszach 3 i 4. Zmienna sensormanager odpowiedzialna jest za pobranie menadżera czujników z poziomu systemu. Zmienna lightSensor odpowiedzialna jest za uzyskanie referencji do głównego czujnika urządzenia, jeżeli się nie uda to zwraca wartość null. Zmienna _isdarkTheme w wierszu 5 określa czy aplikacja wykorzystuje obecnie tryb ciemny, zmienna isDarkTheme w wierszu 6 pomaga jej w tym za pomocą odczytu w UI.

W SetContent w wierszu 23 do dynamicznej zmiany kolorów wykorzytywane jest "RaptorTheme(darkTheme = darmTheme)" zdefiniowany w Theme.kt w folderze ui.Theme.

Poniżej, w wierszach od 60 do 65 znajduje się funkcja onResume, która rejestruje słuchacza zdarzeń po wznowieniu działania aplikacji. odpowiedzialne jest za częstotliwość aktualizacji(SENSOR_DELAY_NORMAL).

This oznacza implementację SensorEventListener.

Funkcja onpause odpowiedzialna jest zatrzymanie działania słuchacza zdarzeń w przypadku pracy aplikacji w tle.

Funkcja onSensorChanged wywoływaną jest za każdym razem gdy uzyskany zostanie nowy odczyt z czujnika systemowego. Zmienna lightLevel pobiera aktualny poziom oświetlenia(jednostka to luks). Zmienna maxLightLevel pobiera maksymalny zakres pomiarowy czujnika. W przypadku braku przypisana zostanie wartość 10000 luksów. W wierszu 77 znajduje się instrukcja przejścia w tryb ciemny jeżeli obecny wykrywany poziom światła jest mniejszy niż 40 procent. Funkcja onAccuracyChanged nie jest tutaj wykorzystywana. Może być wykorzystana do np. zmiany dokładności wykrywania czujnika.

4.3. Autoryzacja odciskiem palca

Autoryzacja odciskiem palca działa w następujących krokach:

1. Sprawdzenie, czy uwierzytelnianie biometryczne jest dostępne, przedstawione na listingu nr 38
2. Zbudowanie monitu biometrycznego, przedstawione na listingu nr 39
3. Obsługa wyniku monitu biometrycznego, przedstawiona na listingu nr 40

```
1 enum class BiometricAuthenticationStatus(val id: Int) {  
2     READY(1),  
3     NOT_AVAILABLE(-1),  
4     TEMPORARY_NOT_AVAILABLE(-2),
```

```
5     AVAILABLE_BUT_NOT_ENROLLED (-3)
6 }
7
8 fun isBiometricAuthAvailable(): BiometricAuthenticationStatus {
9     return when (biometricmanager.canAuthenticate(BIOMETRIC_STRONG))
10    {
11        BiometricManager.BIOMETRIC_SUCCESS ->
12        BiometricAuthenticationStatus.READY
13        BiometricManager.BIOMETRIC_ERROR_NO_HARDWARE ->
14        BiometricAuthenticationStatus.NOT_AVAILABLE
15        BiometricManager.BIOMETRIC_ERROR_HW_UNAVAILABLE ->
16        BiometricAuthenticationStatus.TEMPORARY_NOT_AVAILABLE
17        BiometricManager.BIOMETRIC_ERROR_NONE_ENROLLED ->
18        BiometricAuthenticationStatus.AVAILABLE_BUT_NOT_ENROLLED
19        else -> BiometricAuthenticationStatus.NOT_AVAILABLE
20    }
21 }
```

Listing 15. Sprawdzenie dostępności uwierzytelnienia biometrycznego

Na zamieszczonym listingu funkcja `BiometricAuthenticationStatus` tworzy stany gotowości uwierzytelniania. Funkcja `isBiometricAuthAvailable` jest odpowiedzialna za sprawdzenie czy telefon obsługuje uwierzytelnianie biometryczne oraz czy w telefonie są zapisane odciski palca. `biometricmanager.canAuthenticate(BIOMETRIC_STRONG)` jest odpowiedzialna za zapytanie systemu, o możliwość użycia uwierzytelniania silnego biometrycznego, w zależności od odpowiedzi systemu zostanie zwrócony odpowiednio zdefiniowany status.

```
1 fun PromptBiometricAuth(
2     title: String,
3     subtitle: String,
4     negativeButtonText: String,
5     fragmentActivity: FragmentActivity,
6     onSuccess: (result: BiometricPrompt.AuthenticationResult) -> Unit
7         ,
8     onFailed: () -> Unit,
9     onError: (errorCode: Int, errorMessage: String) -> Unit,
10    ) {
11        when(isBiometricAuthAvailable()) {
12            BiometricAuthenticationStatus.NOT_AVAILABLE -> {
13                onError(BiometricAuthenticationStatus.NOT_AVAILABLE.id, "Not
14                available on this device")
15            }
16            BiometricAuthenticationStatus.TEMPORARY_NOT_AVAILABLE -> {
```

```
16     onError(BiometricAuthenticationStatus.  
17             TEMPORARY_NOT_AVAILABLE.id, "Not available at this moment")  
18     return  
19 }  
20 BiometricAuthenticationStatus.AVAILABLE_BUT_NOT_ENROLLED -> {  
21     onError(BiometricAuthenticationStatus.  
22             AVAILABLE_BUT_NOT_ENROLLED.id, "Add a fingerprint")  
23     return  
24 }  
25     else -> Unit  
26 }  
27 biometricPrompt = BiometricPrompt(  
28     fragmentActivity,  
29     object: BiometricPrompt.AuthenticationCallback() {  
30         override fun onAuthenticationSucceeded(result:  
31             BiometricPrompt.AuthenticationResult) {  
32             super.onAuthenticationSucceeded(result)  
33             onSuccess(result)  
34         }  
35         override fun onAuthenticationError(errorCode: Int, errString:  
36             CharSequence) {  
37             super.onAuthenticationError(errorCode, errString)  
38             onError(errorCode, errString.toString())  
39         }  
40         override fun onAuthenticationFailed() {  
41             super.onAuthenticationFailed()  
42             onFailed()  
43         }  
44     promptinfo = BiometricPrompt.PromptInfo.Builder()  
45         .setTitle(title)  
46         .setSubtitle(subtitle)  
47         .setNegativeButtonText(negativeButtonText)  
48         .build()  
49     biometricPrompt.authenticate(promptinfo)  
50 }
```

Listing 16. Sprawdzenie monitu biometrycznego

Funkcja `PromptBiometricAuth` odpowiedzialna jest za tworzenie monitu, który zostanie wyświetlony użytkownikowi podczas włączenia aplikacji. Monit zawiera tytuł, podtytuł oraz przycisk negatywny. Od wiersza 10 do 24 znajduje się instruk-

cia when która jest odpowiedzialna za sprawdzenie dostępności uwierzytelniania. Następnie w wierszach od 25 do 43 tworzony jest nowy obiekt który będzie obsługiwał okno dialogowe. `object:BiometricPrompt.AuthenticationCallback()` odpowiedzialne jest za implementację metody obsługi zdarzeń związanych z uwierzytelnianiem. Jeżeli uwierzytelnianie się powiedzie to wywoływana jest metoda `onAuthenticationSucceeded`, która przekazuje wynik jako argument do `PromptBiometricAuth`, co umożliwia odblokowanie aplikacji. `onAuthenticationError` wywoływanie jest w przypadku wystąpienia błędu. `onAuthenticationFailed` jest wywoływanie w przypadku niepowodzenia uwierzytelniania, np. niewłaściwy odcisk palca. Od wiersza 44 do 49 znajduje się prompt builder, odpowiedzialny za skonfigurowanie danych które będą wyświetlane w monicie, na samym końcu w wierszu 49 wywołana jest metoda.

```
1 private fun promptBiometricAuthentication(biometricAuthenticator: BiometricAuthenticator) {
2     biometricAuthenticator.PromptBiometricAuth(
3         title = "Authentication Required",
4         subtitle = "Please authenticate to proceed",
5         negativeButtonText = "Cancel",
6         fragmentActivity = this,
7         onSuccess = {
8             runOnUiThread {
9                 _isAuthenticated.value = true
10            }
11        },
12        onFailed = {
13        },
14        onError = { errorCode, errorMessage ->
15        }
16    )
17 }
```

Listing 17. Obsługa wyniku monitu

Funkcja w MainActivity, przekazuje dane do tworzonego monitu, w przypadku pomyslnego uwierzytelnienia, IsAuthenticated jest ustawiany jako true, dzięki czemu aplikacja wie że można opuścić ekran logowania oraz przejść do wczytywania reszty aplikacji. Fragment kodu odpowiedzialny za załadowanie ekranu uwierzytelniania przed przejściem do ekranu głównego znajduje się w `onCreate`, przedstawionym na listingu nr 41.

```
1 if (authenticated) {
2     MainScreen()
3 } else {
```

```
4     AuthenticationScreen(            
5         onAuthenticate = {            
6             promptBiometricAuthentication(biometricAuthenticator)            
7         }            
8     )            
9 }            
10
```

Listing 18. Zawartość onCreate

4.4. Odczyt i przetwarzanie plików

4.4.1. Tag Extractor

Klasa `TagExtractor` jest odpowiedzialna za wyciąganie tagów z piosenek. Każda piosenka zawiera tagi, na które składają się: nazwa artysty, nazwa artystów z albumu, tytuł, data wydania, nazwa albumu, URI piosenki, URI obrazka cover. Deklaracje tych tagów pokazane są na listingu nr 42.

```
1  data class SongInfo(            
2      val artists: List<String>?,            
3      val albumArtists: List<String>?,            
4      val title: String?,            
5      val releaseDate: String?,            
6      val album: String?,            
7      val fileUri: Uri?,            
8      val coverUri: Uri?,            
9  )            
10
```

Listing 19. Deklaracja tagów

Metoda `buildSongInfo()` przedstawiona na listingu nr 43 jest odpowiedzialna za parsowanie metadanych. Otrzymuje

```
1  @OptIn(UnstableApi::class)            
2  private fun buildSongInfo(metadata: Metadata, uri: Uri?):            
3      SongInfo {            
4      val metadataList = mutableListOf<Metadata.Entry>()            
5      for(i in 0 until metadata.length()) {            
6          metadataList.add(metadata.get(i))            
7      }            
8      Log.d("${javaClass.simpleName}", "Metadata list: $metadataList")            
9      when(metadataList[0]) {
```

```
10     is VorbisComment -> {
11         fun handleMissingTags(map: MutableMap<String?, Any?>) {
12             if(map["ALBUMARTIST"] == null) map["ALBUMARTIST"] = List<
13 String>(1, {"Unknown"} )
14         }
15
16         val entryMap: MutableMap<String?, Any?> = mutableMapOf()
17
18         // the last element 'picture' screws up the logic and it
19         // only has a mimetype value
20         // which i think is useless
21         metadataList.take(metadataList.size - 1).fastForEach {
22             val entry = it as VorbisComment
23
24             val key = entry.key
25             val value = entry.value
26             when(key) {
27                 "ALBUMARTIST", "ARTIST" -> {
28                     if(!entryMap.containsKey(key)) {
29                         entryMap[key] = mutableListOf<String?>(value)
30                     } else {
31                         (entryMap[key] as? MutableList<String?>)?.add(value
32                     )
33                     }
34                 }
35             }
36             else -> {
37                 // assert(!entryMap.containsKey(key))
38                 if(entryMap.containsKey(key)) {
39                     Log.w(javaClass.simpleName, "Unhandled duplicate
key: $key")
40                     return@fastForEach
41                 }
42                 entryMap[key] = value
43             }
44         }
45
46         handleMissingTags(entryMap)
47
48         val coverUri = imageManager.extractAlbumimage(
49             uri,
50             entryMap["ALBUMARTIST"] as List<String>,
51             entryMap["ALBUM"] as String
52         )
53     }
54 }
```

```
51
52     return SongInfo(
53         artists = entryMap["ARTIST"] as? List<String>?,
54         albumArtists = entryMap["ALBUMARTIST"] as List<String>,
55         title = entryMap["TITLE"] as? String?,
56         album = entryMap["ALBUM"] as String?,
57         releaseDate = entryMap["DATE"] as? String?,
58         fileUri = uri,
59         trackNumber = (entryMap["TRACKNUMBER"] as? String?)?.toInt()
60     ),
61     coverUri = coverUri,
62     ).also {
63         Log.d(javaClass.simpleName, "Vorbis Song: $it")
64     }
65
66
67     is Id3Frame -> {
68         fun handleMissingTags(map: MutableMap<String, List<String>>)
69             >> {
70             if (map["TPE2"] == null) map["TPE2"] = List<String>(1, {"Unknown"})
71         }
72
73         val entryMap: MutableMap<String, List<String>> =
74             mutableMapOf()
75             metadataList.fastForEach {
76                 val entry = it as Id3Frame
77                 Log.d(javaClass.simpleName, "Id3 metadata: $entry")
78
79                 when (entry) {
80                     is TextInformationFrame -> {
81                         entryMap[entry.id] = entry.values
82                     }
83
84                     else -> {
85                         Log.w(javaClass.simpleName, "Unimplemented id3 frame: ${entry}")
86                     }
87                 }
88             }
89
90             handleMissingTags(entryMap)
91
92             val coverUri = imageManager.extractAlbumImage()
```

```
91     uri ,
92     entryMap["TPE2"]?: emptyList() ,
93     entryMap["TALB"]?.get(0).toString()
94
95   )
96
97   return SongInfo(
98     artists = entryMap["TPE1"] ,
99     albumArtists = entryMap["TPE2"] as List<String> ,
100    title = entryMap["TIT2"]?.get(0) ,
101    releaseDate = entryMap["TDA"]?.get(0) ,
102    album = entryMap["TALB"]?.get(0) ,
103    fileUri = uri ,
104    trackNumber = entryMap["TRCK"]?.get(0)?.let {
105      return@let it.takeWhile { it != '/' }.toInt()
106    },
107    coverUri = coverUri
108  ).also {
109    Log.d(javaClass.simpleName, "id3 Song: $it")
110  }
111 }
112
113 else -> {
114   metadataList.fastForEach {
115     Log.w(
116       javaClass.simpleName,
117       "Unhanded tag format: ${it::class.simpleName}, metadata:
$it"
118     )
119   }
120
121   return SongInfo(
122     null, mutableListOf("Unknown"), null, null, null, null,
123     null, null
124   )
125 }
126 }
```

Listing 20. Metoda `buildSongInfo()`

Instrukcje w wierszach od 3 do 6 odpowiedzialne są za tworzenie listy metadanych. Następujeinicjalizacja pustej listy `metadataList`. Następnie pętla przeszukuje po `metadata.length`, następnie dodaje każdy wpis do listy. W wierszach od 9 do 120 znajduje się pętla while, odpowiedzialna za warunkowe sprawdzanie formatu danych.

Instrukcja składa się z tzech części: `VorbisComment`, `Id3Frame` oraz `else`.

1. `VorbisComment` znajdujące się w wierszach od 10 do 65 odpowiedzialne jest za parsowanie gdy mamy do czynienia z formatem metadanych typu Vorbis. Funkcja `handleMissingTags()` w wierszach od 11 do 13 jest odpowiedzialna za wypełnienie brakujących pól domyślnymi wartościami. Następnie w wierszach od 15 do 42 tworzona jest mapa wejścia oraz parsowanie. W wierszach od 44 do 50 następuje wywołanie funkcji uzupełniającej brakujące tagi oraz ekstrakcja coveru piosenki. Wyodrębnia wbudowany w plik audio cover oraz zapisuje go do pliku w pamięci aplikacji. W wierszach od 52 do 63 tworzony jest obiekt `SongInfo`.
2. `Id3Frame` znajduje się w wierszach od 67 do 111. Odpowiedzialne za parsowanie gdy mamy do czynienia z formatem Id3Frame. Działa podobnie do poprzednika. Mamy funkcję wewnętrzną która tworzy brakujące tagi, tworzona jest mapa oraz następuje parsowanie, wywołanie funkcji tworzącej brakujące tagi oraz ekstrakcja coveru i na koniec tworzony jest obiekt `SongInfo` dla tego formatu.
3. `else` znajduje się w wierszach od 113 do 124. Jest wykonywana gdy natrafimy na nieobsługiwany przez aplikację format. Ostrzeżenie jest zapisywane do logów oraz zwracane jest `SongInfo` zawierające minimalne informacje.

Metoda `TagExtractor` przedstawiona na listingu 44. Metoda jest odpowiedzialna za skanowanie plików audio przekazywanych jako lista `SongFile` oraz wyodrębnienia ich z metadanych.

```
1  @OptIn(UnstableApi::class)
2  fun extractTags(fileList: List<MusicFileLoader.SongFile>): List<
3      SongInfo> {
4      val tagsList = mutableListOf<SongInfo>()
5
6      for(file in fileList) {
7          val mediaItem = MediaItem.fromUri("${file.uri}")
8
9          val trackGroups = MetadataRetriever.retrieveMetadata(context,
10             mediaItem).get()
11
12          if(trackGroups != null) {
13              // Parse and handle metadata
14              assert(trackGroups.length == 1)
15
16              val tags = trackGroups[0]
```

```
15     .getFormat(0)
16     .metadata
17     .let {
18         buildSongInfo(
19             it!!,
20             file.uri
21         )
22     }
23
24     tagsList.add(tags)
25 }
26 }
27 return tagsList
28 }
```

Listing 21. Metoda TagExtractor()

Instrukcja w wierszu 3 tworzy pustą listę do której będą wkładane obiekty SongInfo.

Pętla for przechodzi przez każdy plik SongFile oraz tworzy obiekt MediaItem w wierszu 6, pobiera metadane w wierszu 8, instrukcja if w wierszach od 10 do 25 jest odpowiedzialna za walidację i parsowanie, sprawdza, czy trackGroups nie jest null. assert w wierszu 12 zakłada, że metadane będą w jednym tracku. Następnie w tags uzysujemy dostęp do metadanych oraz parsujemy w buildSongInfo. W wierszu 24 dodajemy wynik do tagList. W wierszu 27 zwracamy tagList.

4.4.2. MusicFileLoader

Zadanie MusicFileLoader znalezienie wszystkich plików muzycznych z wybranego przez użytkownika folderu. Na listingu nr 45.

```
1 package com.example.raptor
2
3 import android.content.Context
4 import android.content.Intent
5 import android.net.Uri
6 import android.provider.DocumentsContract
7 import android.util.Log
8 import androidx.activity.compose.ManagedActivityResultLauncher
9 import androidx.activity.compose.
10    rememberLauncherForActivityResult
11 import androidx.activity.result.contract.ActivityResultContracts
12 import androidx.compose.runtime.Composable
13 import androidx.compose.ui.util.forEach
```

```
14 import dagger.hilt.android.qualifiers.ApplicationContext
15 import kotlinx.coroutines.flow.MutableStateFlow
16 import javax.inject.Inject
17
18 /**
19 * Handles the loading of music files in a given directory and its
20     subdirectories as well as
21 * launching a file picker instance
22 */
23 class MusicFileLoader
24 @Inject constructor( @ApplicationContext private val context:
25     Context) {
26
27     /**
28     * Dataclass representing a song file
29     *
30     * @param filename Name of the file
31     * @param uri 'Uri' corresponding to the file
32     * @param mimeType The type of the file
33     */
34     data class SongFile(val filename: String, val uri: Uri, val
35     mimeType: String)
36
37     /**
38     * Observable list of 'SongFile' currently loaded
39     */
40     var songFileList = MutableStateFlow<List<SongFile>>(emptyList())
41
42     private set
43
44     private lateinit var launcher : ManagedActivityResultLauncher<
45         Uri?, Uri?>
46
47     private fun traverseDirs(treeUri: Uri): List<SongFile> {
48         val _songFiles = mutableListOf<SongFile>()
49
50         fun visit(uri: Uri) {
51             val root = DocumentFile.fromTreeUri(context, uri)
52             Log.d(javaClass.simpleName, "Visiting dir: ${root?.name}")
53             val childDirs = root?.listFiles()?.filter { it.isDirectory }
54
55             childDirs?.fastForEach { visit(it.uri) }
56
57             val songFiles = root?.listFiles()
58             ?.filter {
```

```
53         it.type?.slice(0..4) == "audio"
54     }
55     ?.map {
56         SongFile(
57             filename = it.name.toString(),
58             uri = it.uri,
59             mimeType = it.type.toString()
60         )
61     }
62
63     Log.d(javaClass.simpleName, "Visited dir ${root?.name},
64 songs: ${songFiles?.map { it
65             .filename
66         }}")
67
68     _songFiles.addAll(songFiles?: emptyList())
69
70     visit(treeUri)
71
72     return _songFiles
73 }
74
75 /**
76 * \@Composable function that prepares the file picker launcher
77 *
78 * Must be called before 'launch()'
79 */
80
81 @Composable
82 fun PrepareFilePicker() {
83     val contentResolver = context.contentResolver
84
85     launcher = rememberLauncherForActivityResult(
86         contract = ActivityResultContracts.OpenDocumentTree()
87     ) { treeUri: Uri? ->
88         treeUri?.let {
89             val permissions = Intent.FLAG_GRANT_READ_URI_PERMISSION
90             or
91                 Intent.FLAG_GRANT_WRITE_URI_PERMISSION
92             contentResolver.takePersistableUriPermission(treeUri,
93             permissions)
94
95             Log.d(javaClass.simpleName, "Selected: $it")
96             songFileList.value = traverseDirs(treeUri)
97         }
98     }
99 }
```

```
95      }
96    }
97  }
98
99  /**
100 * Launches the file picker
101 */
102 fun launch() {
103   launcher.launch(null)
104 }
105 }
```

Listing 22. Kod MusicFileLoader

Na początku znajduje się klasa danych `SongFile` w wierszu 31. Służy do przechowywania informacji o pojedynczym pliku audio. W wierszach 36 i 37 jest zmienna `songFileList`, służącą do przechowywania listy obiektów `SongFile`. W wierszu 39 znajduje się zmienna `launcher`, który zwraca uri folderu wybranego przez użytkownika. Funkcja `TraverseDirs` w wierszach od 41 do 74 jest odpowiedzialna za rekurencyjne przeszukiwanie folderu określonego przez `treeUri`. Zbiera wszystkie pliki audio znajdujące się w tym folderze. Najpierw tworzona jest lista `_songFiles` która będzie przechowywać wyniki. Następnie tworzona jest funkcja wewnętrzna `visit` która wywołuje obiekt reprezentujący wybrany folder, następnie pliki są listowane oraz oddzielane od folderu. Następnie rekurencyjnie wywołujemy dla każdego katalogu `visit(it.uri)`. Następnie w bieżącym katalogu filtrowane są pliki w wierszach od 52 do 54. Następnie z każdego pliku tworzony jest obiekt `SongFile` a następnie dołączany do `_SongFiles`. Na koniec zwracane jest `_SongFiles`. Następnie znajduje się funkcja `PrepareFilePicker` w wierszach od 82 do 97. Funkcja jest odpowiedzialna za przygotowanie launchera. `rememberLauncherForActivityResult` pozwala na stworzenie obiektu launchera który pozwoli na otwarcie `OpenDocumentTree`. Kontrakt `ActivityResultContracts.OpenDocumentTree()` pozwala na wybranie całego folderu. Następnie w wierszach od 88 do 96 znajdują się instrukcje definiujące uprawnienia, aby zachować uprawnienia potrzebne jest

```
takePersistableUriPermission().
```

`songFileList.value = traverseDirs(treeUri)` służy do aktualizacji flow, URI folderu przekazywane jest do funkcji `traverseDirs`.

Na koniec w wierszach od 102 do 104 znajduje się funkcja `launch`, odpowiedzialna za uruchomienie selektora folderów.

4.5. NavHost

NavHost to element Androida, jest kontenerem aplikacji, przechowuje aktualny stan nawigacji i jest odpowiedzialny za zarządzanie przejściami i animacjami między fragmentami.

Używany jest głównie w połączeniu z NavController, aby umożliwić użytkownikowi poruszanie się między różnymi ekranami aplikacji.

W aplikacji znajduje się 6 różnych ekranów:

- MainScreen
- MainScreenContent
- AlbumsScreen
- SongsScreen
- AuthenticationScreen
- AudioPlayer

A samych elementów jest 12, nie wliczając że niektóre ekrany posiadają dwie wersje układów w zależności od ustawienia telefonu, np. *AlbumsScreen* pokazany na listingu nr.23.

```
1 @Composable
2 fun AlbumsScreen(
3     navController: NavHostController,
4     author: String,
5 ) {
6     val viewModel: AlbumsScreenViewModel = hiltViewModel<
7         AlbumsScreenViewModel, AlbumsScreenViewModel.Factory>(
8         creationCallback = { it.create(author) }
9     )
10
11     val albumsAndCovers by viewModel.albumsAndCovers.collectAsState(
12         emptyList()
13     )
14     var selectedAlbum by rememberSaveable { mutableStateOf<Pair<
15         Album, ImageBitmap>?>(null) }
16
17     if (selectedAlbum == null) {
```

```
18         painter = painterResource(id = R.drawable.tans3
19             ),
20             contentScale = ContentScale.Crop,
21             alignment = Alignment.Center
22         ),
23     ) {
24
25     val columns = if (maxWidth < maxHeight) 3 else 5
26
27     LazyVerticalGrid(
28         columns = GridCells.Fixed(columns),
29         modifier = Modifier.fillMaxSize(),
30         contentPadding = PaddingValues(16.dp),
31         verticalArrangement = Arrangement.spacedBy(16.dp),
32         horizontalArrangement = Arrangement.spacedBy(16.dp)
33     ) {
34         items(albumsAndCovers, key = { it.first.albumId })
35         { pair ->
36             val album = pair.first
37             val cover = pair.second
38
39             AlbumTile(
40                 albumName = album.title,
41                 cover = cover,
42                 onClick = {
43                     Log.d("MainActivity", "Album id passed
44                         to navhost: ${album.albumId}")
45                     assert(album.albumId != 0L)
46                     selectedAlbum = pair
47                 },
48                 modifier = Modifier,
49             )
50         }
51     }
52 } else {
53     if (isPortrait()) {
54         PortraitView(selectedAlbum = selectedAlbum!!,
55             navController = navController)
56     } else {
57         LandscapeView(selectedAlbum = selectedAlbum!!,
58             navController = navController)
59     }
60 }
```

56 }

Listing 23. Kod Navhost AlbumsScreen

Do zmiany pozycji elementów na ekranie używane są dwie funkcje *PortraitView* pokazany na listingu nr.24 oraz *LandscapeView* przedstawiony na listingu nr.25.

```
1 @Composable
2 fun PortraitView(selectedAlbum: Pair<Album, ImageBitmap>,
3     navController: NavHostController) {
4     Column(
5         horizontalAlignment = Alignment.CenterHorizontally,
6         modifier = Modifier.fillMaxSize().paint(
7             painter = painterResource(id = R.drawable.tans3),
8             contentScale = ContentScale.Crop,
9             alignment = Alignment.Center
10        ),
11    ) {
12        Row(
13            verticalAlignment = Alignment.CenterVertically,
14            modifier = Modifier
15                .fillMaxWidth()
16                .padding(16.dp)
17        ) {
18            Image(
19                bitmap = selectedAlbum.second,
20                contentDescription = "${selectedAlbum.first.title}
21                cover",
22                modifier = Modifier
23                    .size(200.dp)
24                    .weight(1f)
25            )
26            Text(
27                text = selectedAlbum.first.title,
28                style = MaterialTheme.typography.headlineMedium,
29                textAlign = TextAlign.Start,
30                color = MaterialTheme.colorScheme.onSurface,
31                modifier = Modifier
32                    .padding(start = 16.dp)
33                    .weight(1f)
34            )
35        SongsScreen(
36            navController = navController,
37            libraryViewModel = hiltViewModel(),
```

```
38         albumId = selectedAlbum.first.albumId
39     )
40 }
41 }
```

Listing 24. Kod Navhost PortraitView

```
1 @Composable
2 fun LandscapeView(selectedAlbum: Pair<Album, ImageBitmap>,
3                     navController: NavHostController) {
4     Row(
5         modifier = Modifier.fillMaxSize().paint(
6             painter = painterResource(id = R.drawable.tans3),
7             contentScale = ContentScale.Crop,
8             alignment = Alignment.Center
9         ),
10    ) {
11        Column(
12            modifier = Modifier
13                .fillMaxHeight()
14                .padding(16.dp)
15                .weight(1f)
16        ) {
17            Image(
18                bitmap = selectedAlbum.second,
19                contentDescription = "${selectedAlbum.first.title}
20                cover",
21                modifier = Modifier.size(200.dp)
22            )
23            Text(
24                text = selectedAlbum.first.title,
25                style = MaterialTheme.typography.headlineMedium,
26                textAlign = TextAlign.Center,
27                color = MaterialTheme.colorScheme.onSurface,
28                modifier = Modifier.padding(vertical = 16.dp)
29            )
30        }
31        SongsScreen(
32            navController = navController,
33            libraryViewModel = hiltViewModel(),
34            albumId = selectedAlbum.first.albumId,
35            modifier = Modifier.weight(2f)
36        )
37    }
38 }
```

Listing 25. Kod Navhost LandscapeView

W tych dwóch funkcjach znajdują się elementy ogólnie używane przez wszystkie ekrany pozwalając na ich wielofunkcyjność, pochodzącą z potrzeby wywoływanie tylko ich części. =====

5. Implementacja

5.1. Zarządzanie bazą danych

5.1.1. Klasa DatabaseManager

Za zarządzanie bazą danych odpowiedzialna jest klasa `DatabaseManager`, której kod jest zamieszany na listingu nr. 26. Klasa jest wrapperem do bazy danych Room^[5] i do niej akcesorów.

```
1 @Singleton
2 class DatabaseManager @Inject constructor(
3     @ApplicationContext context: Context
4 ) {
5     private val database: LibraryDb = Room.databaseBuilder(
6         context,
7         LibraryDb::class.java, "Library"
8     ).build()
9
10    fun collectAuthorsFlow(): Flow<List<Author>> = database.uiDao()
11        .getAllAuthorsFlow()
12
13    fun collectAlbumsByAuthorFlow(authorName: String): Flow<List<
14        Album>> {
15        return database.uiDao().getAuthorWithAlbums(authorName)
16            .map { it.albums }
17    }
18
19    fun collectSongsByAlbumFlow(albumId: Long): Flow<List<Song>> {
20        return database.uiDao().getAlbumWithSongs(albumId)
21            .map { it.songs }
22    }
23
24    ...
25
26    fun populateDatabase(songs: List<TagExtractor.SongInfo>) {
27        assert(Thread.currentThread().name != "main")
28
29        val dao = database.logicDao()
30
31        fun addAuthors() {
32            songs.fastForEach { song ->
33                //TODO: there should be a distinction between
34                albumartists and regular artists
35                song.albumArtists?.fastForEach { name ->
```

```
33             if(dao.getAuthor(name) == null) {
34                 dao.insertAuthor(Author(name = name))
35             }
36         }
37     }
38 }
39
40
41     fun addAlbumsAndRelations() {
42         // FIXME: xddddddd
43         val distinctAlbumArtistsList = songs
44             .map { Triple(it.album, it.albumArtists, it.
45 coverUri) }
46             .distinct()
47         Log.d(javaClass.simpleName, "Distinct artists set:
48 $distinctAlbumArtistsList")
49
50         distinctAlbumArtistsList.fastForEach {
51             val albumTitle = it.first.toString()
52             val artists = it.second
53             val coverUri = it.third
54
55             val albumId = dao.insertAlbum(Album(
56                 title = albumTitle,
57                 coverUri = coverUri.toString(),
58             ))
59
60             artists?.fastForEach {
61                 dao.insertAlbumAuthorCrossRef(
62                     AlbumAuthorCrossRef(
63                         albumId = albumId,
64                         name = it.toString()
65                     )
66                 }
67             }
68         }
69
70         fun addSongs() {
71             songs.fastForEach { song ->
72                 Log.d(javaClass.simpleName, "NEW SONG\n")
73                 Log.d(javaClass.simpleName, "Album artists: ${song.
74 albumArtists}")
75
76                 val albumWithAuthorCandidates = dao
77                     .getAlbumsByTitle(song.album.toString())
78             }
79         }
80     }
81 }
```

```
74         .map { it.albumId }
75             .map { dao.getAlbumWithAuthors(it) }
76             Log.d(javaClass.simpleName, ""
77                 $albumWithAuthorCandidates")
78
79             var correctAlbum: Album? = null
80             albumWithAuthorCandidates.fastForEach {
81                 Log.d(javaClass.simpleName, "${song.
82                     albumArtists}, ${it.authors}")
83                     //FIXME: theese guys shouldn't be ordered, will
84                     have to refactor a bunch of
85                     // stuff with sets instead of lists
86                     if(song.albumArtists?.sorted() == it.authors.
87                         map { it.name }.sorted()) {
88                         correctAlbum = it.album
89                     }
90
91             }
92
93         }
94     }
95
96     addAuthors()
97     addAlbumsAndRelations()
98     addSongs()
99 }
100 }
```

Listing 26. Strukutura klasy DatabaseManager

Na pierwszej linijce można zauważyc adnotację `@Singleton`. Pochodzi ona z biblioteki `Hilt`[8]. Powiadamia ona bibliotekę o tym że klasa jest singletonem, czyli że ma istnieć tylko jej jedna instancja na cały program. Uczyniono to, dlatego że baza danych powinna być jedna na całą aplikację. Menadżer z nią interfejsujący, dlatego że jest używany w wielu innych klasach, też powinien mieć tylko jedną instancję, aby nie marnować pamięci.

Na linijce nr. 2, widać konstruktor klasy, do którego też przy użyciu `Hilt`, wstrzykiwany jest `context`.

Następnie, na linijce nr. 5, widać inicjalizację samego obiektu bazy `database`.

Baza jest reprezentowana przez klasę `LibraryDb`, definicję której można zobaczyć w sekcji 5.1.2

Dalej, do linijki nr. 22 pokazane są metody zwracające różne elementy bazy. Większość z tych metod zwraca `Flow[TODO:]`. Room natywnie obsługuje Flowy, a dlatego że wymusza dostęp do bazy z innych wątków niż główny, większość operacji wykonywanych na bazie odbywa się za pośrednictwem typów `Flow`

Same metody są wrapperami do obiektów `Dao` bazy.Więcej o nich w sekcji 5.1.3. Niektóre obrabiają dane jak np. `collectSongsByAlbumFlow()` na linijce nr. 17., która mapuje zwraca piosenki z wyjściowej klasy relacyjnej.

Metod tych jest więcej, lecz wyglądają one bardzo podobnie. Dla zwięzłości, można je pominąć.

Metoda `populateDatabase()` zadeklarowana na linijce nr. 24, jest odpowiedzialna za ładowanie wyjątkowych plików informacji do bazy. Jako parametr odstaje ona zmienną `songs` typu `List<TagExtractor.SongInfo>` Zadeklarowane są w niej trzy funkcje pomocnicze: `addAuthors()`, `addAlbumsAndRelations()` i `addSongs()`. Wywoływanie są one po kolejno w metodzie głównej.

Funkcja `addAuthors()`, zadeklarowana na linijce nr. 29, jest prosta w swoim działaniu. Lista z `SongInfo` jest iterowana i po kolejno wpisywani są wszyscy autorzy, którzy jeszcze w bazie nie istnieją.

Funkcja `addAlbumsAndRelations()`, zadeklarowana na linijce nr. 41, odpowiada za dodawanie albumów do bazy oraz tworzenie relacji między nimi, a autorami. Dworzona zmienna `distinctAlbumArtistsList` mapuje tylko unikalne pary albumów i autorów (zmienna `coverUri` nie ma znaczenia przy określaniu autorstwa, jest przypisywana tutaj dlatego, że trudno było znaleźć dla niej lepsze miejsce). Dzięki temu początkowemu filtrowaniu, wiadomo, że każdy napotkany album będzie unikalny. Następnie, `distinctAlbumArtistsList` jest iterowana - przy każdej iteracji dodawany jest nowy album do bazy. Metoda `insertAlbum()` zwraca `id` nowo dodanego albumu. Wynik jej jest przypisywany do zmiennej `albumId` na linijce nr. 53. Potem, zostaje przypisywana relacja albumu z autorami. Autorów może być kilku, więc są oni reprezentowani przy każdej iteracji przez listę, która jest iterowana, a relacja zostaje dodawana z nazwą autora i `albumId`.

Funkcja `addSongs()`, zadeklarowana na linijce nr. 67, ma na celu dodanie piosenek do bazy. Ciało funkcji jest w pętli iterującej się przez listę piosenek. Na początku pętli, na linijce nr. 72 deklarowana jest zmienna `albumWithAuthorCandidates`. Jest ona listą relacji album - autorzy wszystkich albumów o tej samej nazwie co ten w danym elemencie listy. Następnie, lista ta jest iterowana i przy każdej iteracji spraw-

dzane jest czy lista autorów w danej relacji jest równa z listą autorów danej piosenki. Jeżeli tak, wartość danego albumu z wybranej relacji jest przypisywana do zmiennej zadeklarowanej na linii nr. 78 `correctAlbum`. Na końcu funkcji, piosenka dodana jest do bazy przy użyciu metody `dao`.

5.1.2. Klasa LibraryDb

Klasa `LibraryDb` jest deklaracją faktycznej instancji bazy danych, która jest implementowana i generowana przez bibliotekę `Room`. Z racji tego, że jest to klasa abstrakcyjna, jej zadaniem jest określenie struktury bazy i jakie komponenty ma ona zawierać

```
1 @Database(entities = [Song::class, Author::class, Album::class,
2                     AlbumAuthorCrossRef::class], version
3 = 1)
4 abstract class LibraryDb : RoomDatabase() {
5     abstract fun logicDao(): LogicDao
6     abstract fun uiDao(): UIDao
7 }
```

Listing 27. Deklaracja bazy LibraryDb

Jak widać na listingu nr. 27, na początku klasy należy zamieścić adnotację `@Database`. Powiadamia ona bibliotece `Room` o tym, że następująca klasa jest bazą danych. Parametr `entities` określa wszystkie tabele jakie mają się w klasie zawierać. O tabelach więcej w sekcji nr. 5.1.4. Parametr `version` zajmuje się wersjonowaniem bazy. Jest on ważny przy aktualizacjach aplikacji, aby baza mogła zostać odpowiednio zmieniona.

Na linijce nr. 3 umieszczona jest faktyczna deklaracja klasy. Dziedziczy ona z klasy `RoomDatabase`. Jedyne rzeczy jakie są do dziecięcej klasy dodawane, to metody zwracające obiekty `dao`, opisane w sekcji nr. 5.1.3.

5.1.3. Obiekty Dao

Obiekty `dao` (Data Access Object(s)) to obiekty używane do interakcji z zawartością bazy danych. Głównie używa się ich do dodawania elementów do bazy oraz ich odczytywania. Same obiekty definiuje się jako interfejsy z adnotacją `@Dao`. Są one implementowane przez `Room`. Baza danych w projekcie wykorzystuje dwa interfejsy `dao` - `UIDao`, którego kod zamieszczony jest na listingu nr. 28 oraz `LogicDao`, którego kod zamieszczony jest na listingu nr. 29.

```
1 @Dao
```

```

2 interface UIDao {
3     @Query("SELECT * FROM Song")
4     fun getAllSongs(): Flow<List<Song>>
5
6     @Query("SELECT * FROM Song WHERE songId = :songId")
7     fun collectSongFromId(songId: Long): Flow<Song>
8
9     // Get an album with its songs
10    @Transaction
11    @Query("SELECT * FROM album WHERE albumId = :albumId")
12    fun getAlbumWithSongs(albumId: Long): Flow<AlbumWithSongs>
13
14    @Query("SELECT * FROM album WHERE albumId = :albumId")
15    fun getAlbumById(albumId: Long?): Flow<Album>
16
17    // Get an album with its authors
18    @Transaction
19    @Query("SELECT * FROM album WHERE albumId = :albumId")
20    fun getAlbumWithAuthors(albumId: Long?): Flow<AlbumWithAuthors>
?>
21
22    @Query("SELECT * FROM Author")
23    fun getAllAuthorsFlow(): Flow<List<Author>>
24
25    // Get an author with their albums
26    @Transaction
27    @Query("SELECT * FROM author WHERE name = :name")
28    fun getAuthorWithAlbums(name: String): Flow<AuthorWithAlbums>
29 }

```

Listing 28. Deklaracja interfejsu UIDao

```

1 @Dao
2 interface LogicDao {
3     @Insert
4     fun insertSong(song: Song)
5
6     @Insert
7     fun insertAlbum(album: Album): Long
8
9     @Insert(onConflict = OnConflictStrategy.REPLACE)
10    fun insertAuthor(author: Author)
11
12    @Insert(onConflict = OnConflictStrategy.REPLACE)
13    fun insertAlbumAuthorCrossRef(albumAuthorCrossRef:
    AlbumAuthorCrossRef)

```

```

14
15     @Query("SELECT * FROM Author")
16     fun getAllAuthors(): List<Author>
17
18     @Transaction
19     @Query("SELECT * FROM album WHERE albumId = :albumId")
20     fun getAlbumWithAuthors(albumId: Long): AlbumWithAuthors
21
22     @Transaction
23     @Query("SELECT * FROM author WHERE name = :name")
24     fun getAuthorWithAlbums(name: String): AuthorWithAlbums
25
26     @Query("SELECT * FROM author WHERE name = :name")
27     fun getAuthor(name: String): Author?
28
29     @Query("SELECT * FROM album WHERE title = :title")
30     fun getAlbumsByTitle(title: String): List<Album>
31
32     @Query("SELECT * FROM album WHERE title = :title LIMIT 1")
33     fun getAlbumByTitle(title: String): Album?
34
35     @Query("SELECT * FROM AlbumAuthorCrossRef WHERE albumId = :albumId AND name = :authorName LIMIT 1")
36     fun getCrossRefByAlbumAndAuthor(albumId: Long, authorName: String): AlbumAuthorCrossRef?
37
38     @Query("SELECT * FROM Song WHERE songId = :songId")
39     fun getSongfromId(songId: Long): Song
40 }

```

Listing 29. Deklaracja interfejsu LogicDao

Dlatego, że baza Room wymaga dostępu do elementów z innego wątku niż główny, LogicDao może być tylko używany w kodzie, o którym wiadomo, że nie jest wykonywany na głównym wątku. UI dao natomiast, służy ekskluzywnie do zwracania Flowów. Większość elementów związanych z interfejsem w reszcie kodu aplikacji już korzysta z Flowów, więc dao to łatwo jest zintegrować.

Typowy sposób w jaki dodaje się element do bazy znajduje się na linijce nr. 4 w kodzie LogicDao, na listingu nr. 29. Funkcja `insertSong()`, zadnotowana jest `@Insert`. Powiadamia to Room, że funkcja ta odpowiedzialna jest za dodawanie elementu. Parametr `song` to piosenka jaka ma być dodana. W następnej funkcji `insertAlbum()` widać, że funkcje `@Insert` mogą zwracać wartości. W tym przypadku funkcja zwraca `id` nowo dodanego albumu. Można też zwrócić uwagę na

metodę `insertAuthor()` na linijce nr. 8, a w szczególności parametr `onConflict` w adnotacji `@Insert`. Wartość parametru `OnConflictStrategy.REPLACE` mówi bibliotece, aby nie pomijała elementów o tych samych wartościach co już są w tabeli, ale zamieniała starsze na te nowe.

Przykład odczytywania elementu jest dobrze zilustrowany na metodzie `getAuthor()` zadeklarowanej na linijce nr. 27. Adnotacja `@Query` przyjmuje parametr `String`, który jest kwerendą SQL jaka ma być wykonana. Kwerenda `SELECT * FROM author WHERE name = :name` wybiera wszystkich autorów, których pole `name` równe jest parametrowi metody `name` (odnoszenie do parametru w kwerendzie poprzedzone jest znakiem „:”). Dlatego że w bazie może być tylko jeden autor z daną nazwą, zwracany jest pojedynczy autor, a nie ich lista. Niektóre metody, jak na linijce nr. 20 `getAlbumWithAuthors()`, używają adnotacji `@Transaction`. W przypadku tej metody, zwraca ona relację, czyli czyta z kilku tabel. Adnotacja `@Transaction` zapewnia, że transakcja jest atomiczna, co za tym idzie, inne wątki nie mogą nagle zmienić wartości jakieś tabeli.

Interfejs `UIdao` działa podobnie jak `LogicDao`, ale zwraca on tylko i wyłącznie `Flowy`, które są natywnie obsługiwane przez `Room`.

5.1.4. Tabele

W bibliotece `Room`, każda tabela to `dataclass` określana adnotacją `@Entity`. Kolumny takiej tabeli to po prostu pola klasy. Klucz danej tabeli jest określany adnotacją `@PrimaryKey`

```
1 @Entity
2 data class Author (
3     @PrimaryKey val name: String
4 )
```

Listing 30. Deklaracja tabeli `Author`

Tabela `Author`, zawarta na listingu nr. 30, określa autorów. Tabela jest prosta, jedynym polem jest `name`, który jest kluczem.

```
1 @Entity
2 data class Album(
3     @PrimaryKey(autoGenerate = true) val albumId: Long = 0,
4     val title: String,
5     val coverUri: String?,
```

6)

Listing 31. Deklaracja tabeli Album

Tabela `Album`, zawarta na listingu nr. 31, określa albumy. Kluczem jest zmienna `albumId`. Klucz jest generowany automatycznie, dzięki parametrowi adnotacji `autoGenerate`. Pole `title` określa tytuł, a pole `coverUri` określa adres URI okładki.

```

1
2 @Entity(
3     foreignKeys = [
4         ForeignKey(
5             entity = Album::class,
6             parentColumns = ["albumId"],
7             childColumns = ["albumId"],
8             onDelete = ForeignKey.CASCADE
9         )
10    ],
11
12    indices = [Index(value = ["albumId"])]
13 )
14 data class Song(
15     @PrimaryKey(autoGenerate = true) val songId: Long = 0,
16     val title: String?,
17     val albumId: Long?,
18     val fileUri: String?,
19 )

```

Listing 32. Deklaracja tabeli Song

Klasa ta, zawarta na listingu nr. 32, określa tabelę piosenek. Pole `foreignKeys` w adnotacji `@Entity` określa obce klucze, którymi posługuje się klasa. W tym przypadku określone jest to, że pole w `Song albumId` wskazuje na pole w `Album albumId`. Pole `indices` każe indeksować pola z `albumId` ku polepszeniu szybkości bazy. W ciele klasy, pole `title` to tytuł piosenki. Pole `albumId` określa ID albumu, do którego należy dana piosenka.

5.1.5. Relacje

Relacje w Room są określane jako osobne `dataclassy`. Są one zadeklarowane adnotacją `@Relation` w danej klasie. Ponadto umieszczenie elementu w adnotacji

@Embedded, pozwala klasie „przyswoić” pola danego elementu. Dzięki temu klasa może odnosić się do pól danego elementu tak jakby były one bezpośrednio w klasie. Konieczne jest umieszczenie elementu głównego, od którego będzie relacja wyodziła, do tej adnotacji.

5.1.5.1. AlbumWithSongs

Klasa AlbumWithSongs na listingu nr. 33, określa relację albumów i piosenek.

```
1 data class AlbumWithSongs(
2     @Embedded val album: Album,
3     @Relation(
4         parentColumn = "albumId",
5         entityColumn = "albumId"
6     )
7     val songs: List<Song>
8 )
```

Listing 33. Deklaracja relacji AlbumWithSongs

Relacja łączy pole `albumId` albumu z polem `albumId` piosenek. Pole `songs` zawiera wszystkie piosenki z tą samą wartością pola `albumId` co faktyczny klucz danego albumu.

```
1 @Entity(primaryKeys = ["albumId", "name"])
2 data class AlbumAuthorCrossRef(
3     val albumId: Long,
4     val name: String
5 )
```

Listing 34. Deklaracja tabeli relacji AlbumAuthorCrossRef

Tabela na listingu nr. 34 określa relację M do N między albumami a autorami. Jest to tabela z dwoma kluczami głównymi: `albumId` dla tabeli `Album` i `name` dla tabeli `Author`.

5.1.5.3. AlbumWithAuthors i AuthorWithAlbums

Obie klasy są do siebie bardzo podobne więc zostaną omówione razem.

```
1 data class AlbumWithAuthors(
2     @Embedded val album: Album,
3     @Relation(
```

```

4     parentColumn = "albumId",
5     entityColumn = "name",
6     associateBy = Junction(AlbumAuthorCrossRef::class)
7   )
8   val authors: List<Author>
9 )

```

Listing 35. Deklaracja relacji `AlbumWithAuthors`

```

1 data class AuthorWithAlbums(
2   @Embedded val author: Author,
3   @Relation(
4     parentColumn = "name",
5     entityColumn = "albumId",
6     associateBy = Junction(AlbumAuthorCrossRef::class)
7   )
8   val albums: List<Album>
9 )

```

Listing 36. Deklaracja relacji `AuthorWithAlbums`

Na listingu nr. 35 przedstawiona jest klasa `AlbumWithAuthors`. Definiuje ona relację danego albumu z jego autorami. Dlatego, że relacja jest M do N , w adnotacji `@Relation` dodane jest odniesienie do tabeli relacji `AlbumAuthorCrossRef`, opisanej w sekcji nr. 5.1.5.2. Klucze, jakie mają być porównywane są zdefiniowane w parametrach `parentColumn`, dla id albumu i `entityColumn` dla nazwy autora. Wynikiem tej relacji jest lista albumów. Sytuacja wygląda podobnie w `AuthorWithAlbums`, na listingu nr. 36. Tym razem to autor jest rodzicem i oczekujemy od relacji listy albumów danego autora.

5.2. Czujnik światła

Czujnik światła został zaimplementowany za pomocą wbudowanej funkcji. Zadaniem czujnika jest dynamiczna zmiana schematu kolorów aplikacji na podstawie danych otrzymanych z czujnika światła wbudowanego w urządzeniu mobilnym z systemem android.

```

1  @AndroidEntryPoint
2  class MainActivity : FragmentActivity(), SensorEventListener {
3    private lateinit var sensorManager: SensorManager
4    private var lightSensor: Sensor? = null
5    private val _isDarkTheme = mutableStateOf(false)
6    private val isDarkTheme: State<Boolean> = _isDarkTheme
7

```

```
8  private val _isAuthenticated = mutableStateOf(false)
9  private val isAuthenticated: State<Boolean> = _isAuthenticated
10
11 override fun onCreate(savedInstanceState: Bundle?) {
12     super.onCreate(savedInstanceState)
13     val biometricAuthenticator = BiometricAuthenticator(this)
14
15     sensorManager = getSystemService(Context.SENSOR_SERVICE) as
16 SensorManager
17     lightSensor = sensorManager.getDefaultSensor(Sensor.
18 TYPE_LIGHT)
19
20     setContent {
21         val darkTheme by isDarkTheme
22         val authenticated by isAuthenticated
23         RaptorTheme(darkTheme = darkTheme) {
24             Surface(
25                 modifier = Modifier.fillMaxSize(),
26                 color = MaterialTheme.colorScheme.background
27             ) {
28                 if (authenticated) {
29                     MainScreen()
30                 } else {
31                     AuthenticationScreen(
32                         onAuthenticate = {
33                             promptBiometricAuthentication(
34                             biometricAuthenticator
35                         )
36                     }
37                 }
38             }
39         }
40     }
41
42     private fun promptBiometricAuthentication(
43         biometricAuthenticator: BiometricAuthenticator) {
44         biometricAuthenticator.PromptBiometricAuth(
45             title = "Authentication Required",
46             subtitle = "Please authenticate to proceed",
47             negativeButtonText = "Cancel",
48             fragmentActivity = this,
49             onSuccess = {
```

```
49         runOnUiThread {
50             _isAuthenticated.value = true
51         }
52     },
53     onFailure = {
54     },
55     onError = { errorCode, errorMessage ->
56     }
57 )
58 }

59

60     override fun onResume() {
61         super.onResume()
62         lightSensor?.let { sensor ->
63             sensorManager.registerListener(this, sensor, SensorManager.
64             SENSOR_DELAY_NORMAL)
65         }
66     }

67     override fun onPause() {
68         super.onPause()
69         sensorManager.unregisterListener(this)
70     }

71

72     override fun onSensorChanged(event: SensorEvent?) {
73         if (event?.sensor?.type == Sensor.TYPE_LIGHT) {
74             val lightLevel = event.values[0]
75             val maxLightLevel = lightSensor?.maximumRange ?: 10000f
76
77             _isDarkTheme.value = lightLevel < 0.4 * maxLightLevel
78         }
79     }

80

81     override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int)
82     {
83     }
84 }
```

Listing 37. Implementacja czujnika światła w `MainActivity.kt`

Na listingu 37 przedstawiona jest funkcja MainScreen. W wierszu 2 mamy `SensorEventListener`, który jest frameworkm w androidzie który pozwala aplikacji na reagowanie na zmiany odczytywane przez czujniki smartfona. Po dodaniu automatycznie tworzone są funkcje `onSensorChanged`, `onResume`, `onPause`, `onAccuracyChanged`. Implementację sensora zaczynamy od utworzenia zmiennych sensorma-

nager oraz lightSensor w wierszach 3 i 4. Zmienna sensormanager odpowiedzialna jest za pobranie menadżera czujników z poziomu systemu. Zmienna `lightSensor` odpowiedzialna jest za uzyskanie referencji do głównego czujnika urządzenia, jeżeli się nie uda to zwraca wartość null. Zmienna `_isDarkTheme` w wierszu 5 określa czy aplikacja wykorzystuje obecnie tryb ciemny, zmienna `isDarkTheme` w wierszu 6 pomaga jej w tym za pomocą odczytu w UI.

W `SetContent` w wierszu 23 do dynamicznej zmiany kolorów wykorzytywane jest "RaptorTheme(darkTheme = darmTheme)" zdefiniowany w `Theme.kt` w folderze `ui.Theme`.

Poniżej, w wierszach od 60 do 65 znajduje się funkcja `onResume`, która rejestruje słuchacza zdarzeń po wznowieniu działania aplikacji. odpowiedzialne jest za częstotliwość aktualizacji(`SENSOR_DELAY_NORMAL`).

This oznacza implementację `SensorEventListener`.

Funkcja `onpause` odpowiedzialna jest zatrzymanie działania słuchacza zdarzeń w przypadku pracy aplikacji w tle.

Funkcja `onSensorChanged` wywoływaną jest za każdym razem gdy uzyskany zostanie nowy odczyt z czujnika systemowego. Zmienna `lightLevel` pobiera aktualny poziom oświetlenia(jednostka to luks). Zmienna `maxLightLevel` pobiera maksymalny zakres pomiarowy czujnika. W przypadku braku przypisana zostanie wartość 10000 luksów. W wierszu 77 znajduje się instrukcja przejścia w tryb ciemny jeżeli obecny wykrywany poziom światła jest mniejszy niż 40 procent. Funkcja `onAccuracyChanged` nie jest tutaj wykorzystywana. Może być wykorzystana do np. zmiany dokładności wykrywania czujnika.

5.3. Autoryzacja odciskiem palca

Autoryzacja odciskiem palca działa w następujących krokach:

1. Sprawdzenie, czy uwierzytelnianie biometryczne jest dostępne, przedstawione na listingu nr 38
2. Zbudowanie monitu biometrycznego, przedstawione na listingu nr 39
3. Obsługa wyniku monitu biometrycznego, przedstawiona na listingu nr 40

```
1 enum class BiometricAuthenticationStatus(val id: Int) {  
2     READY(1),  
3     NOT_AVAILABLE(-1),  
4     TEMPORARY_NOT_AVAILABLE(-2),
```

```

5     AVAILABLE_BUT_NOT_ENROLLED (-3)
6 }
7
8 fun isBiometricAuthAvailable(): BiometricAuthenticationStatus {
9     return when (biometricmanager.canAuthenticate(BIOMETRIC_STRONG)) {
10         BiometricManager.BIOMETRIC_SUCCESS ->
11             BiometricAuthenticationStatus.READY
12         BiometricManager.BIOMETRIC_ERROR_NO_HARDWARE ->
13             BiometricAuthenticationStatus.NOT_AVAILABLE
14         BiometricManager.BIOMETRIC_ERROR_HW_UNAVAILABLE ->
15             BiometricAuthenticationStatus.TEMPORARY_NOT_AVAILABLE
16         BiometricManager.BIOMETRIC_ERROR_NONE_ENROLLED ->
17             BiometricAuthenticationStatus.AVAILABLE_BUT_NOT_ENROLLED
18         else -> BiometricAuthenticationStatus.NOT_AVAILABLE
19     }
20 }
```

Listing 38. Sprawdzenie dostępności uwierzytelnienia biometrycznego

Na zamieszczonym listingu funkcja `BiometricAuthenticationStatus` tworzy stany gotowości uwierzytelniania. Funkcja `isBiometricAuthAvailable` jest odpowiedzialna za sprawdzenie czy telefon obsługuje uwierzytelnianie biometryczne oraz czy w telefonie są zapisane odciski palca. `biometricmanager.canAuthenticate(BIOMETRIC_STRONG)` jest odpowiedzialna za zapytanie systemu, o możliwość użycia uwierzytelniania silnego biometrycznego, w zależności od odpowiedzi systemu zostanie zwrocony odpowiednio zdefiniowany status.

```

1 fun PromptBiometricAuth(
2     title: String,
3     subtitle: String,
4     negativeButtonText: String,
5     fragmentActivity: FragmentActivity,
6     onSuccess: (result: BiometricPrompt.AuthenticationResult) -> Unit
7         ,
8     onFailed: () -> Unit,
9     onError: (errorCode: Int, errorMessage: String) -> Unit,
10 ) {
11     when(isBiometricAuthAvailable()) {
12         BiometricAuthenticationStatus.NOT_AVAILABLE -> {
13             onError(BiometricAuthenticationStatus.NOT_AVAILABLE.id, "Not
14             available on this device")
15         }
16         BiometricAuthenticationStatus.TEMPORARY_NOT_AVAILABLE -> {
```

```
16     onError(BiometricAuthenticationStatus.  
17             TEMPORARY_NOT_AVAILABLE.id, "Not available at this moment")  
18     return  
19 }  
20 BiometricAuthenticationStatus.AVAILABLE_BUT_NOT_ENROLLED -> {  
21     onError(BiometricAuthenticationStatus.  
22             AVAILABLE_BUT_NOT_ENROLLED.id, "Add a fingerprint")  
23     return  
24 }  
25     else -> Unit  
26 }  
27 biometricPrompt = BiometricPrompt(  
28     fragmentActivity,  
29     object: BiometricPrompt.AuthenticationCallback() {  
30         override fun onAuthenticationSucceeded(result:  
31             BiometricPrompt.AuthenticationResult) {  
32             super.onAuthenticationSucceeded(result)  
33             onSuccess(result)  
34         }  
35         override fun onAuthenticationError(errorCode: Int, errString:  
36             CharSequence) {  
37             super.onAuthenticationError(errorCode, errString)  
38             onError(errorCode, errString.toString())  
39         }  
40         override fun onAuthenticationFailed() {  
41             super.onAuthenticationFailed()  
42             onFailed()  
43         }  
44     promptinfo = BiometricPrompt.PromptInfo.Builder()  
45     .setTitle(title)  
46     .setSubtitle(subtitle)  
47     .setNegativeButton(negativeButtonText)  
48     .build()  
49     biometricPrompt.authenticate(promptinfo)  
50 }
```

Listing 39. Sprawdzenie monitu biometrycznego

Funkcja `PromptBiometricAuth` odpowiedzialna jest za tworzenie monitu, który zostanie wyświetlony użytkownikowi podczas włączenia aplikacji. Monit zawiera tytuł, podtytuł oraz przycisk negatywny. Od wiersza 10 do 24 znajduje się instruk-

cja when która jest odpowiedzialna za sprawdzenie dostępności uwierzytelniania. Następnie w wierszach od 25 do 43 tworzony jest nowy obiekt który będzie obsługiwał okno dialogowe. `object:BiometricPrompt.AuthenticationCallback()` odpowiedzialne jest za implementację metody obsługi zdarzeń związanych z uwierzytelnianiem. Jeżeli uwierzytelnianie się powiedzie to wywoływana jest metoda `onAuthenticationSucceeded`, która przekazuje wynik jako argument do `PromptBiometricAuth`, co umożliwia odblokowanie aplikacji. `onAuthenticationError` wywoływanie jest w przypadku wystąpienia błędu. `onAuthenticationFailed` jest wywoływanie w przypadku niepowodzenia uwierzytelniania, np. niewłaściwy odcisk palca. Od wiersza 44 do 49 znajduje się prompt builder, odpowiedzialny za skonfigurowanie danych które będą wyświetlane w monicie, na samym końcu w wierszu 49 wywołana jest metoda.

```
1  private fun promptBiometricAuthentication(biometricAuthenticator: BiometricAuthenticator) {
2      biometricAuthenticator.PromptBiometricAuth(
3          title = "Authentication Required",
4          subtitle = "Please authenticate to proceed",
5          negativeButtonText = "Cancel",
6          fragmentActivity = this,
7          onSuccess = {
8              runOnUiThread {
9                  _isAuthenticated.value = true
10             }
11         },
12         onFailed = {
13     },
14         onError = { errorCode, errorString ->
15     }
16     )
17 }
```

Listing 40. Obsługa wyniku monitu

Funkcja w `MainActivity`, przekazuje dane do tworzonego monitu, w przypadku pomyslnego uwierzytelnienia, `IsAuthenticated` jest ustawiany jako `true`, dzięki czemu aplikacja wie że można opuścić ekran logowania oraz przejść do wczytywania reszty aplikacji. Fragment kodu odpowiedzialny za załadowanie ekranu uwierzytelniania przed przejściem do ekranu głównego znajduje się w `onCreate`, przedstawionym na listingu nr 41.

```
1  if (authenticated) {
2      MainScreen()
3  } else {
```

```
4     AuthenticationScreen(            
5         onAuthenticate = {            
6             promptBiometricAuthentication(biometricAuthenticator)            
7         }            
8     )            
9 }            
10
```

Listing 41. Zawartość onCreate

5.4. Odczyt i przetwarzanie plików

5.4.1. Tag Extractor

Klasa `TagExtractor` jest odpowiedzialna za wyciąganie tagów z piosenek. Każda piosenka zawiera tagi, na które składają się: nazwa artysty, nazwa artystów z albumu, tytuł, data wydania, nazwa albumu, URI piosenki, URI obrazka cover. Deklaracje tych tagów pokazane są na listingu nr 42.

```
1 data class SongInfo(            
2     val artists: List<String>?,            
3     val albumArtists: List<String>?,            
4     val title: String?,            
5     val releaseDate: String?,            
6     val album: String?,            
7     val fileUri: Uri?,            
8     val coverUri: Uri?,            
9   )            
10
```

Listing 42. Deklaracja tagów

Metoda `buildSongInfo()` przedstawiona na listingu nr 43 jest odpowiedzialna za parsowanie metadanych. Otrzymuje

```
1 @OptIn(UnstableApi::class)            
2 private fun buildSongInfo(metadata: Metadata, uri: Uri?):            
3     SongInfo {            
4         val metadataList = mutableListOf<Metadata.Entry>()            
5         for(i in 0 until metadata.length()) {            
6             metadataList.add(metadata.get(i))            
7         }            
8         Log.d("${javaClass.simpleName}", "Metadata list: $metadataList")            
9     when(metadataList[0]) {
```

```
10     is VorbisComment -> {
11         fun handleMissingTags(map: MutableMap<String?, Any?>) {
12             if(map["ALBUMARTIST"] == null) map["ALBUMARTIST"] = List<
13                 String>(1, {"Unknown"})
14         }
15
16         val entryMap: MutableMap<String?, Any?> = mutableMapOf()
17
18         // the last element 'picture' screws up the logic and it
19         // only has a mimetype value
20         // which i think is useless
21         metadataList.take(metadataList.size - 1).fastForEach {
22             val entry = it as VorbisComment
23
24             val key = entry.key
25             val value = entry.value
26             when(key) {
27                 "ALBUMARTIST", "ARTIST" -> {
28                     if(!entryMap.containsKey(key)) {
29                         entryMap[key] = mutableListOf<String?>(value)
30                     } else {
31                         (entryMap[key] as? MutableList<String?>)?.add(value)
32                     }
33                 }
34             else -> {
35                 // assert(!entryMap.containsKey(key))
36                 if(entryMap.containsKey(key)) {
37                     Log.w(javaClass.simpleName, "Unhandled duplicate
38                         key: $key")
39                     return@fastForEach
40                 }
41             }
42         }
43
44         handleMissingTags(entryMap)
45
46         val coverUri = imageManager.extractAlbumimage(
47             uri,
48             entryMap["ALBUMARTIST"] as List<String>,
49             entryMap["ALBUM"] as String
50         )
51     }
52 }
```

```
51
52     return SongInfo(
53         artists = entryMap["ARTIST"] as? List<String>?,
54         albumArtists = entryMap["ALBUMARTIST"] as List<String>,
55         title = entryMap["TITLE"] as? String?,
56         album = entryMap["ALBUM"] as String?,
57         releaseDate = entryMap["DATE"] as? String?,
58         fileUri = uri,
59         trackNumber = (entryMap["TRACKNUMBER"] as? String?)?.toInt()
60     ),
61     coverUri = coverUri,
62 ).also {
63     Log.d(javaClass.simpleName, "Vorbis Song: $it")
64 }
65
66
67     is Id3Frame -> {
68         fun handleMissingTags(map: MutableMap<String, List<String>>> {
69             if(map["TPE2"] == null) map["TPE2"] = List<String>(1, {"Unknown"})
70         }
71
72         val entryMap: MutableMap<String, List<String>> =
73             mutableMapOf()
74             metadataList.fastForEach {
75                 val entry = it as Id3Frame
76                 Log.d(javaClass.simpleName, "Id3 metadata: $entry")
77
78                 when(entry) {
79                     is TextInformationFrame -> {
80                         entryMap[entry.id] = entry.values
81                     }
82
83                     else -> {
84                         Log.w(javaClass.simpleName, "Unimplemented id3 frame: $entry")
85                     }
86                 }
87
88             handleMissingTags(entryMap)
89
90             val coverUri = imageManager.extractAlbumImage(
```

```
91     uri,
92     entryMap["TPE2"]?: emptyList(),
93     entryMap["TALB"]?.get(0).toString()
94   )
95
96
97   return SongInfo(
98     artists = entryMap["TPE1"],
99     albumArtists = entryMap["TPE2"] as List<String>,
100    title = entryMap["TIT2"]?.get(0),
101    releaseDate = entryMap["TDA"]?.get(0),
102    album = entryMap["TALB"]?.get(0),
103    fileUri = uri,
104    trackNumber = entryMap["TRCK"]?.get(0)?.let {
105      return@let it.takeWhile { it != '/' }.toInt()
106    },
107    coverUri = coverUri
108  ).also {
109    Log.d(javaClass.simpleName, "id3 Song: $it")
110  }
111}
112
113 else -> {
114   metadataList.fastForEach {
115     Log.w(
116       javaClass.simpleName,
117       "Unhanded tag format: ${it::class.simpleName}, metadata: $it"
118     )
119   }
120
121   return SongInfo(
122     null, mutableListOf("Unknown"), null, null, null, null,
123     null, null
124   )
125 }
126 }
```

Listing 43. Metoda buildSongInfo()

Instrukcje w wierszach od 3 do 6 odpowiedzialne są za tworzenie listy metadanych. Następuje inicjalizacja pustej listy `metadataList`. Następnie pętla przeszukuje po `metadata.length`, następnie dodaje każdy wpis do listy. W wierszach od 9 do 120 znajduje się pętla while, odpowiedzialna za warunkowe sprawdzanie formatu danych.

Instrukcja składa się z tzech części: **VorbisComment**, **Id3Frame** oraz **else**.

1. **VorbisComment** znajdujące się w wierszach od 10 do 65 odpowiedzialne jest za parsowanie gdy mamy do czynienia z formatem metadanych typu Vorbis. Funkcja **handleMissingTags()** w wierszach od 11 do 13 jest odpowiedzialna za wypełnienie brakujących pól domyślnymi wartościami. Następnie w wierszach od 15 do 42 tworzona jest mapa wejścia oraz parsowanie. W wierszach od 44 do 50 następuje wywołanie funkcji uzupełniającej brakujące tagi oraz ekstrakcja coveru piosenki. Wyodrębnia wbudowany w plik audio cover oraz zapisuje go do pliku w pamięci aplikacji. W wierszach od 52 do 63 tworzony jest obiekt **SongInfo**.
2. **Id3Frame** znajduje się w wierszach od 67 do 111. Odpowiedzialne za parsowanie gdy mamy do czynienia z formatem Id3Frame. Działa podobnie do poprzednika. Mamy funkcję wewnętrzną która tworzy brakujące tagi, tworzona jest mapa oraz następuje parsowanie, wywołanie funkcji tworzącej brakujące tagi oraz ekstrakcja coveru i na koniec tworzony jest obiekt **SongInfo** dla tego formatu.
3. **else** znajduje się w wierszach od 113 do 124. Jest wykonywana gdy natrafimy na nieobsługiwany przez aplikację format. Ostrzeżenie jest zapisywane do logów oraz zwracane jest **SongInfo** zawierające minimalne informacje.

Metoda **TagExtractor** przedstawiona na listingu 44. Metoda jest odpowiedzialna za skanowanie plików audio przekazywanych jako lista **SongFile** oraz wyodrębnienia ich z metadanych.

```
1  @OptIn(UnstableApi::class)
2  fun extractTags(fileList: List<MusicFileLoader.SongFile>): List<
3      SongInfo> {
4      val tagsList = mutableListOf<SongInfo>()
5
6      for(file in fileList) {
7          val mediaItem = MediaItem.fromUri("${file.uri}")
8
9          val trackGroups = MetadataRetriever.retrieveMetadata(context,
10             mediaItem).get()
11
12          if(trackGroups != null) {
13              // Parse and handle metadata
14              assert(trackGroups.length == 1)
15
16              val tags = trackGroups[0]
```

```
15     .getFormat(0)
16     .metadata
17     .let {
18         buildSongInfo(
19             it!!,
20             file.uri
21         )
22     }
23
24     tagsList.add(tags)
25 }
26 }
27 return tagsList
28 }
```

Listing 44. Metoda TagExtractor()

Instrukcja w wierszu 3 tworzy pustą listę do której będą wkładane obiekty SongInfo.

Pętla for przechodzi przez każdy plik **SongFile** oraz tworzy obiekt **MediaItem** w wierszu 6, pobiera metadane w wierszu 8, instrukcja if w wierszach od 10 do 25 jest odpowiedzialna za walidację i parsowanie, sprawdza, czy **trackGroups** nie jest null. **assert** w wierszu 12 zakłada, że metadane będą w jednym tracku. Następnie w **tags** uzysujemy dostęp do metadanych oraz parsujemy w **buildSongInfo**. W wierszu 24 dodajemy wynik do **tagList**. W wierszu 27 zwracamy **tagList**.

5.4.2. MusicFileLoader

Zadanie **MusicFileLoader** znalezienie wszystkich plików muzycznych z wybranego przez użytkownika folderu. Na listingu nr 45.

```
1 package com.example.raptor
2
3 import android.content.Context
4 import android.content.Intent
5 import android.net.Uri
6 import android.provider.DocumentsContract
7 import android.util.Log
8 import androidx.activity.compose.ManagedActivityResultLauncher
9 import androidx.activity.compose.
10    rememberLauncherForActivityResult
11 import androidx.activity.result.contract.ActivityResultContracts
12 import androidx.compose.runtime.Composable
13 import androidx.compose.ui.util.fastForEach
14 import androidx.documentfile.provider.DocumentFile
```

```
14 import dagger.hilt.android.qualifiers.ApplicationContext
15 import kotlinx.coroutines.flow.MutableStateFlow
16 import javax.inject.Inject
17
18 /**
19 * Handles the loading of music files in a given directory and its
20 * subdirectories as well as
21 * launching a file picker instance
22 */
23 class MusicFileLoader
24 @Inject constructor( @ApplicationContext private val context:
25 Context) {
26
27     /**
28     * Dataclass representing a song file
29     *
30     * @param filename Name of the file
31     * @param uri 'Uri' corresponding to the file
32     * @param mimeType The type of the file
33     */
34     data class SongFile(val filename: String, val uri: Uri, val
35     mimeType: String)
36
37     /**
38     * Observable list of 'SongFile' currently loaded
39     */
40     var songFileList = MutableStateFlow<List<SongFile>>(emptyList()
41 )
42     private set
43
44     private lateinit var launcher : ManagedActivityResultLauncher<
45 Uri?, Uri?>
46
47     private fun traverseDirs(treeUri: Uri): List<SongFile> {
48         val _songFiles = mutableListOf<SongFile>()
49
50         fun visit(uri: Uri) {
51             val root = DocumentFile.fromTreeUri(context, uri)
52             Log.d(javaClass.simpleName, "Visiting dir: ${root?.name}")
53             val childDirs = root?.listFiles()?.filter { it.isDirectory }
54
55             childDirs?.fastForEach { visit(it.uri) }
56
57             val songFiles = root?.listFiles()
58             ?.filter {
```

```
53         it.type?.slice(0..4) == "audio"
54     }
55     ?.map {
56         SongFile(
57             filename = it.name.toString(),
58             uri = it.uri,
59             mimeType = it.type.toString()
60         )
61     }
62
63     Log.d(javaClass.simpleName, "Visited dir ${root?.name},
64 songs: ${songFiles?.map { it
65             .filename
66         }}")
67
68     _songFiles.addAll(songFiles?: emptyList())
69 }
70
71     visit(treeUri)
72
73     return _songFiles
74 }
75
76 /**
77 * \@Composable function that prepares the file picker launcher
78 *
79 * Must be called before 'launch()'
80 */
81 @Composable
82 fun PrepareFilePicker() {
83     val contentResolver = context.contentResolver
84
85     launcher = rememberLauncherForActivityResult(
86         contract = ActivityResultContracts.OpenDocumentTree()
87     ) { treeUri: Uri? ->
88
89         treeUri?.let {
90             val permissions = Intent.FLAG_GRANT_READ_URI_PERMISSION
91             or
92                 Intent.FLAG_GRANT_WRITE_URI_PERMISSION
93             contentResolver.takePersistableUriPermission(treeUri,
94             permissions)
95
96             Log.d(javaClass.simpleName, "Selected: $it")
97             songFileList.value = traverseDirs(treeUri)
98         }
99     }
100 }
```

```
95      }
96    }
97  }
98
99 /**
100 * Launches the file picker
101 */
102 fun launch() {
103   launcher.launch(null)
104 }
105 }
```

Listing 45. Kod MusicFileLoader

Na początku znajduje się klasa danych `SongFile` w wierszu 31. Służy do przechowywania informacji o pojedynczym pliku audio. W wierszach 36 i 37 jest zmienna `songFileList`, służącą do przechowywania listy obiektów `SongFile`. W wierszu 39 znajduje się zmienna `launcher`, który zwraca uri folderu wybranego przez użytkownika. Funkcja `TraverseDirs` w wierszach od 41 do 74 jest odpowiedzialna za rekurencyjne przeszukiwanie folderu określonego przez `treeUri`. Zbiera wszystkie pliki audio znajdujące się w tym folderze. Najpierw tworzona jest lista `_songFiles` która będzie przechowywać wyniki. następnie tworzona jest funkcja wewnętrzna `visit` która wywołuje obiekt reprezentujący wybrany folder, następnie pliki są listowane oraz oddzielane od folderu. Następnie rekurencyjnie wywołujemy dla każdego katalogu `visit(it.uri)`. Następnie w bieżącym katalogu filtrowane są pliki w wierszach od 52 do 54. Następnie z każdego pliku tworzony jest obiekt `SongFile` a następnie dołączany do `_SongFiles`. Na koniec zwracane jest `_SongFiles`. Następnie znajduje się funkcja `PrepareFilePicker` w wierszach od 82 do 97. Funkcja jest odpowiedzialna za przygotowanie launchera. `rememberLauncherForActivityResult` pozwala na stworzenie obiektu launchera który pozwoli na otwarcie `OpenDocumentTree`. Kontrakt `ActivityResultContracts.OpenDocumentTree()` pozwala na wybranie całego folderu. Następnie w wierszach od 88 do 96 znajdują się instrukcje definiujące uprawnienia, aby zachować uprawnienia potrzebne jest

```
takePersistableUriPermission().
```

`songFileList.value = traverseDirs(treeUri)` służy do aktualizacji flow, URI folderu przekazywane jest do funkcji `traverseDirs`.

Na koniec w wierszach od 102 do 104 znajduje się funkcja `launch`, odpowiedzialna za uruchomienie selektora folderów.

5.5. Ładowanie obrazów albumów

Za ładowanie obrazów z plików muzycznych odpowiedzialna jest klasa `ImageManager`, kod której umieszczony jest na listingu nr. 46.

```
1 class ImageManager @Inject constructor(@ApplicationContext private
2     val context: Context) {
3
4     fun extractAlbumimage(
5         uri: Uri?,
6         artistNames: List<String>,
7         albumName: String
8     ): Uri? {
9
10        val retriever = MediaMetadataRetriever()
11
12        retriever.setDataSource(context, uri)
13        val pictureBytes = retriever.embeddedPicture
14
15        val bitmapFile = File(context.filesDir,
16            artistNames.fastJoinToString(" ; ") + ":$albumName"
17        )
18
19
20        pictureBytes?.let {
21            bitmapFile.writeBytes(pictureBytes)
22        }
23
24
25        retriever.release()
26
27        return bitmapFile.toUri()
28    }
29
30
31    fun getBitmapFromAppStorage(uri: Uri?): ImageBitmap {
32        Log.d(javaClass.simpleName, "Collecting bitmap with uri: "
33        + uri)
34
35        if(uri != null) {
36            try {
37
38                context.contentResolver.openInputStream(Uri.parse(
39                    uri.toString())).use {
40
41                    return BitmapFactory.decodeStream(it)
42                        .asImageBitmap()
43                }
44            } catch(e: FileNotFoundException) {
45
46                Log.e(javaClass.simpleName, "Can't parse thumbnail
47                at: " + uri)
48
49                return ImageBitmap(1,1)
50            }
51        }
52    }
53}
```

```
37     } else {
38         return ImageBitmap(1,1,)
39     }
40 }
41 }
```

Listing 46. Struktura klasy `ImageManager`

Na początku klasy, do konstruktora, wstrzykiwana jest zależność `context` kontekstu aplikacji. Klasa potrzebuje go, dlatego, że wchodzi w szerszą interakcję z funkcjami systemowymi.

Metoda `extractAlbumimage()` zadeklarowana na linijce nr. 2, odpowiedzialna jest za wyciąganie miniaturki albumu z pliku i umieszczanie jej w katalogu głównym aplikacji. Parametr `uri` to odnośnik do danego pliku muzycznego. Parametry `artistNames` i `albumName` to respektownie lista nazw autorów danej piosenki i nazwa albumu. Dwa ostatnie parametry będą potrzebne do nazwania danej miniaturki później w metodzie. Na początku metody, czyli linijce nr. 7, tworzony jest obiekt `MediaMetadataRetriever` i przypisany do zmiennej `retriever`. Klasa ta służy do wyciągania informacji z plików muzycznych. W tym przypadku pozwoli ona wyciągnąć miniaturkę w danym pliku zawartą. Na linijce nr. 9, ustawiane jest źródło dla `retirevera` jako parametr `uri`. Następnie, tworzona jest zmienna `pictureBytes`, do której przypisywane są surowe bajty tworzące miniaturkę. Zmienna `bitmapFile` reprezentuje instancję pliku do jakiego ma być zapisany bitmap. Struktura nazwy pliku to `<nazwa_artysty1;nazwa_artysty2...>;<nazwa_albumu>`. Następnie, `pictureBytes` jest zapisywane do pliku `bitmapFile`. Na końcu funkcji, na linijce nr. 20, zwalniane jest miejsce potrzebne na `retrievera` i na linijce nr. 22 zwracane jest `uri bitmapFile`.

Druga metoda w klasie to `getBitmapFromAppStorage()`, zadeklarowana na linijce nr. 25. Ma ona na celu wczytanie miniaturki z pamięci aplikacji, nie z pliku muzycznego. Parametr `uri` to odnośnik do pliku z miniaturką. Na linijce nr. 27 sprawdzane jest czy parametru `uri` istnieje. Jeżeli tak to otwierany jest plik przy pomocy `uri`, używając API `contentResolver`. Używając zwróconego przez niego strumienia, tworzony jest bitmap, który jest konwertowany do klasy `ImageBitmap`, używanej bezpośrednio przez `Compose`. Jeżeli `uri` nie istnieje, zostaje zwracany placeholderowy `ImageBitmap`, jak widać na linijce nr. 37.

5.6. Audio player

Logika audio player znajduje się w pliku AudioPlayer.kt. Działanie rozpoczyna od inicjalizacji ExoPlayer przedstawionej na listingu nr 47.

```
1  class AudioPlayer @Inject constructor(@ApplicationContext context
2      : Context) {
3
4      private val player = ExoPlayer.Builder(context).build().apply {
5          setAudioAttributes(
6              AudioAttributes.Builder()
7                  .setContentType(C.AUDIO_CONTENT_TYPE_MUSIC)
8                  .setUsage(C.USAGE_MEDIA)
9                  .build(),
10             true
11         )
12     }
13 }
```

Listing 47. Inicjalizacja exoplayer

setAudioAttributes ustawia atrybuty odtwarzania audio a ExoPlayer.Builder tworzy nową instancję odtwarzacza. Następnie w kodzie znajduje się funkcja updateListeners, przedstawiona na listingu nr 48. Funkcja jest odpowiedzialna za zarządzanie stanem odtwarzania, używa obserwowanego stanu odtwarzacza playbackState i słuchacza, aby reagować na zmiany w stanie odtwarzania.

```
1  private fun updateListeners(song: Song) {
2      if(currentSong == song) {
3          return
4      }
5
6      currentSong = song
7
8      player.addListener(
9          object: Player.Listener {
10          override fun onIsPlayingChanged(isPlaying: Boolean) {
11              Log.d(AudioPlayer::class.simpleName, if (isPlaying) "
12                  player is playing $song"
13                  else "player is not playing")
14
15              // TODO: this feels hacky... maybe this callback should
16              // be remove altogether
17              if(
18                  player.playbackState == ExoPlayer.STATE_ENDED ||
19                  player.playbackState == ExoPlayer.STATE_BUFFERING
20              ) {
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
13
```

```
19         return
20     }
21
22     playbackState.value = if(isPlaying)
23         PlaybackStates.STATE_PLAYING else PlaybackStates.
24     STATE_PAUSED
25     Log.d(AudioPlayer::class.simpleName, "Playback: ${playbackState
26         .value.name}")
27 }
28
29     override fun onPlaybackStateChanged(playbackState: Int) {
30         when(playbackState) {
31             Player.STATE_BUFFERING -> {
32                 Log.d(AudioPlayer::class.simpleName, "Playback: ${playbackStates
33                     .STATE_BUFFERING}")
34                 this@AudioPlayer.playbackState.value = PlaybackStates
35             .STATE_BUFFERING
36         }
37
38         Player.STATE_ENDED -> {
39             Log.d(AudioPlayer::class.simpleName, "Playback: ${playbackStates
40                 .STATE_ENDED.name}")
41             this@AudioPlayer.playbackState.value = PlaybackStates
42             .STATE_ENDED
43         }
44
45         Player.STATE_IDLE -> {
46             Log.d(AudioPlayer::class.simpleName, "Playback: ${playbackStates
47                 .STATE_IDLE.name}")
48             this@AudioPlayer.playbackState.value = PlaybackStates
49             .STATE_IDLE
50         }
51
52         Player.STATE_READY -> {
53             Log.d(AudioPlayer::class.simpleName, "Playback: ${playbackStates
54                 .STATE_READY.name}")
55             this@AudioPlayer.playbackState.value = PlaybackStates
56             .STATE_READY
57         }
58 }
```

```
58     }
59 }
60 )
61 }
62 }
```

Listing 48. Zarządzanie stanem

Funkcja wewnętrzna `onIsPlayingChanged` jest odpowiedzialna za wykrywanie czy odtwarzacz gra oraz aktualizowanie stanu. Funkcja `onPlaybackStateChanged` jest odpowiedzialna za reagowanie na zmiany stanu exoplayera. `playbackState` przechowuje stan. Następnie jest funkcja `playSong` odpowiedzialna za ustawienie pliku audio do odtwarzania oraz rozpoczęcie odtwarzania, przedstawiona na listingu nr 49.

```
1  fun playSong(song: Song) {
2      assert(isPlayingInternal == false)
3      assert(!player.isPlaying)
4      assert(playbackState.value != PlaybackStates.STATE_PLAYING)
5      Log.d(javaClass.simpleName, "calling playSong")
6
7      Log.d(AudioPlayer::class.simpleName, "Song uri: ${song.
fileUri}")
8
9      updateListeners(song)
10
11     if(player.currentMediaItem.hashCode() != MediaItem.fromUri(
12         song.fileUri!!.toUri())
13         .hashCode()) { //FIXME: we ball
14         player.apply {
15             setMediaItem(MediaItem.fromUri(song.fileUri.toUri()))
16             prepare()
17         }
18
19         Log.d(javaClass.simpleName, "playSong(), songs don't match"
20     )
21     }
22
23     player.play()
24 }
```

Listing 49. Funkcja playsong

`MediaItem.fromUr` tworzy element multimedialny z URI pliku audio. `setMediaItem`

ustawia element multimedialny do odtwarzania. `prepare` przygotowuje odtwarzacz do odtwarzania. `play` rozpoczyna odtwarzanie. Funkcje `pause` oraz `restartCurrentPlayback` przedstawione na listingu nr 50 odpowiedzialne są za pauzowanie oraz ponownym odtwarzaniem.

```
1 fun restartCurrentPlayback() {
2     player.seekTo(0)
3     player.play()
4 }
5 fun pause() {
6     assert(playbackState.value == PlaybackStates.STATE_PLAYING)
7     assert(isPlayingInternal)
8
9     Log.d(javaClass.simpleName, "calling pause")
10    player.pause()
11
12 }
13
```

Listing 50. Funkcje pausei restartCurrentPlayuback

Funkcja `changeCurrentPosition` przedstawiona na listingu nr 51 odpowiedzialna jest za zmianę aktualnej pozycji odtwarzania.

```
1 fun changeCurrentPosition(time: Long) {
2     player.seekTo(time)
3 }
4
```

Listing 51. Funkcja zmainy pozycji

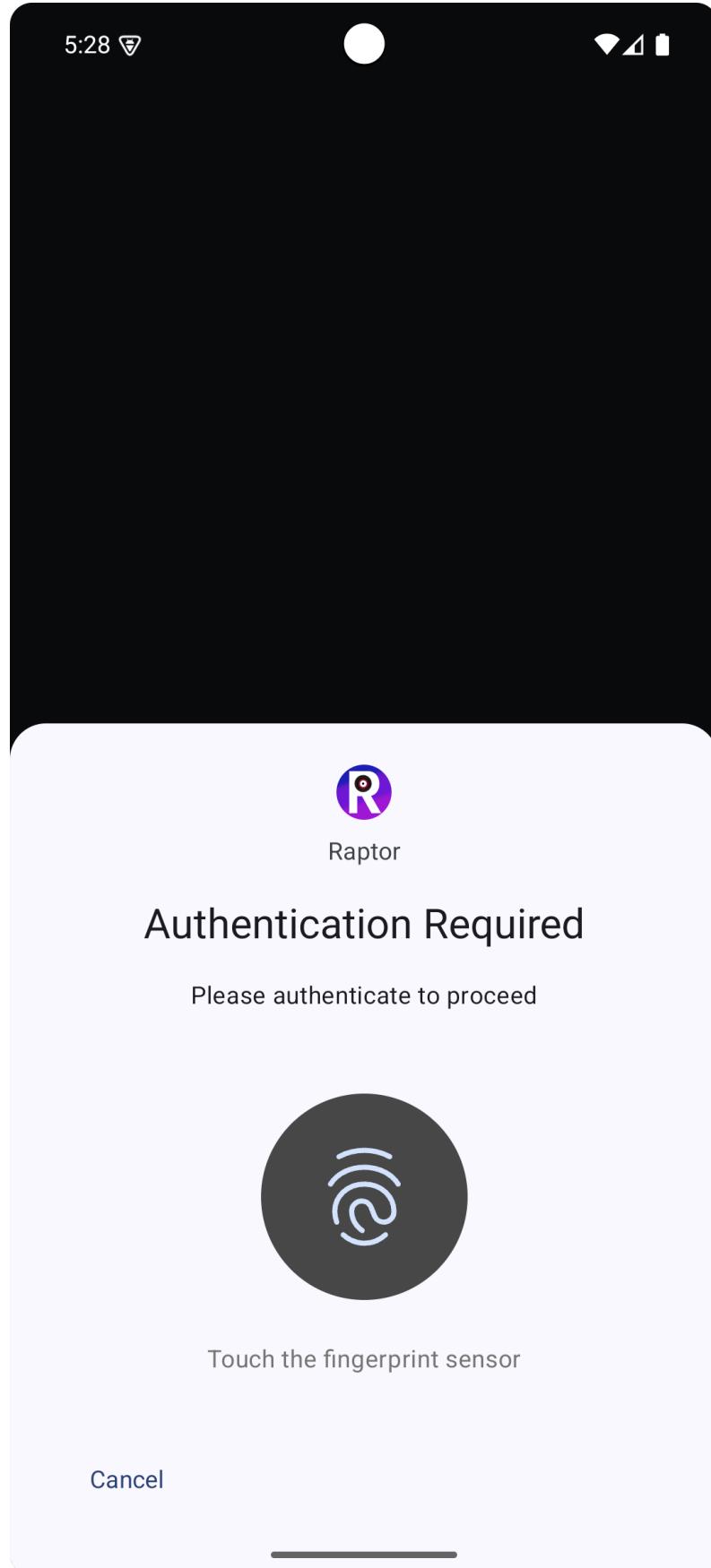
`seekTo` przenosi odtwarzacz do konkretnego miejsca. Na samym końcu znajduje się funkcja `releasePlayer` która jest odpowiedzialna za czyszczenie pamięci, przedstawiona jest na listingu nr 52

```
1 fun releasePlayer() {
2     player.release()
3 }
4
```

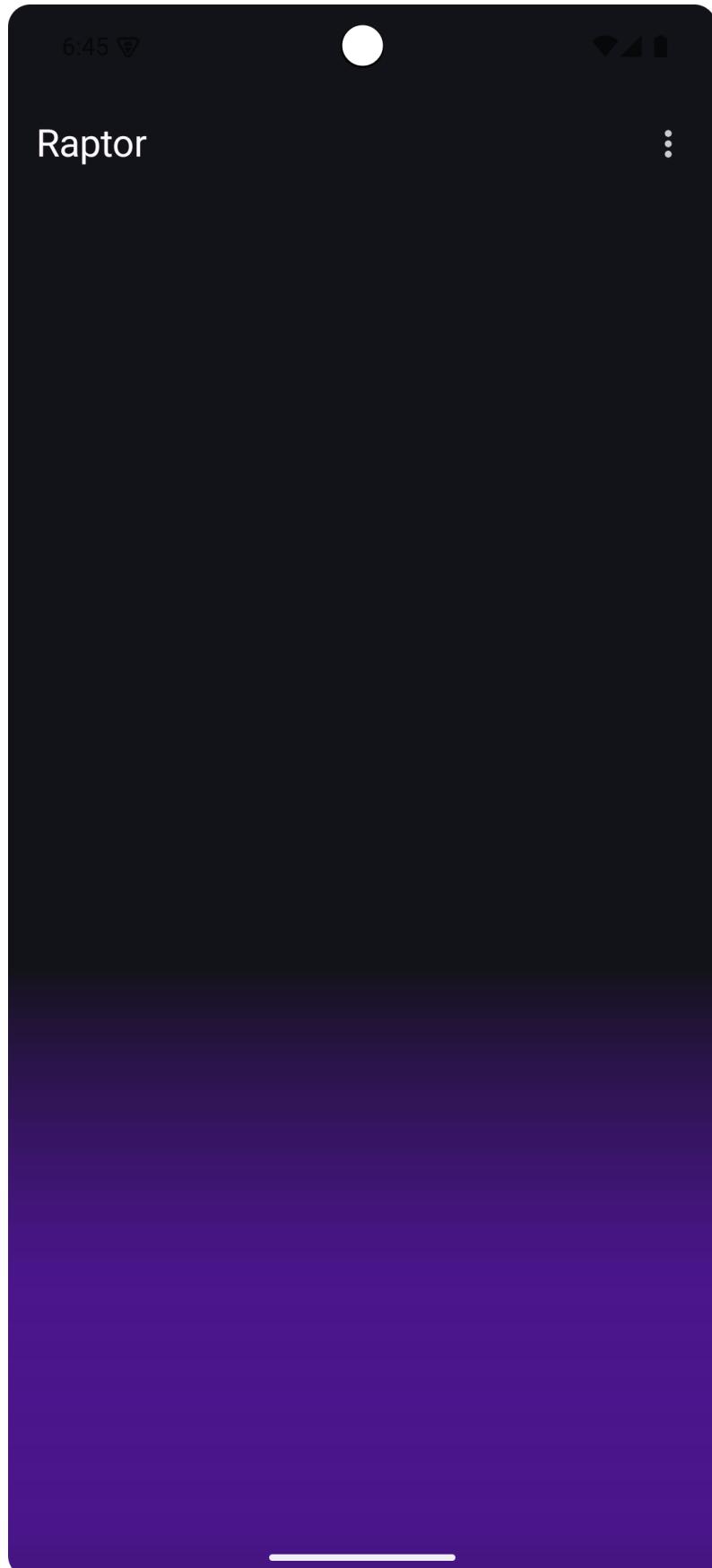
Listing 52. Funkcja czyszczenia pamięci

6. Testowanie

6.1. Włączenie aplikacji



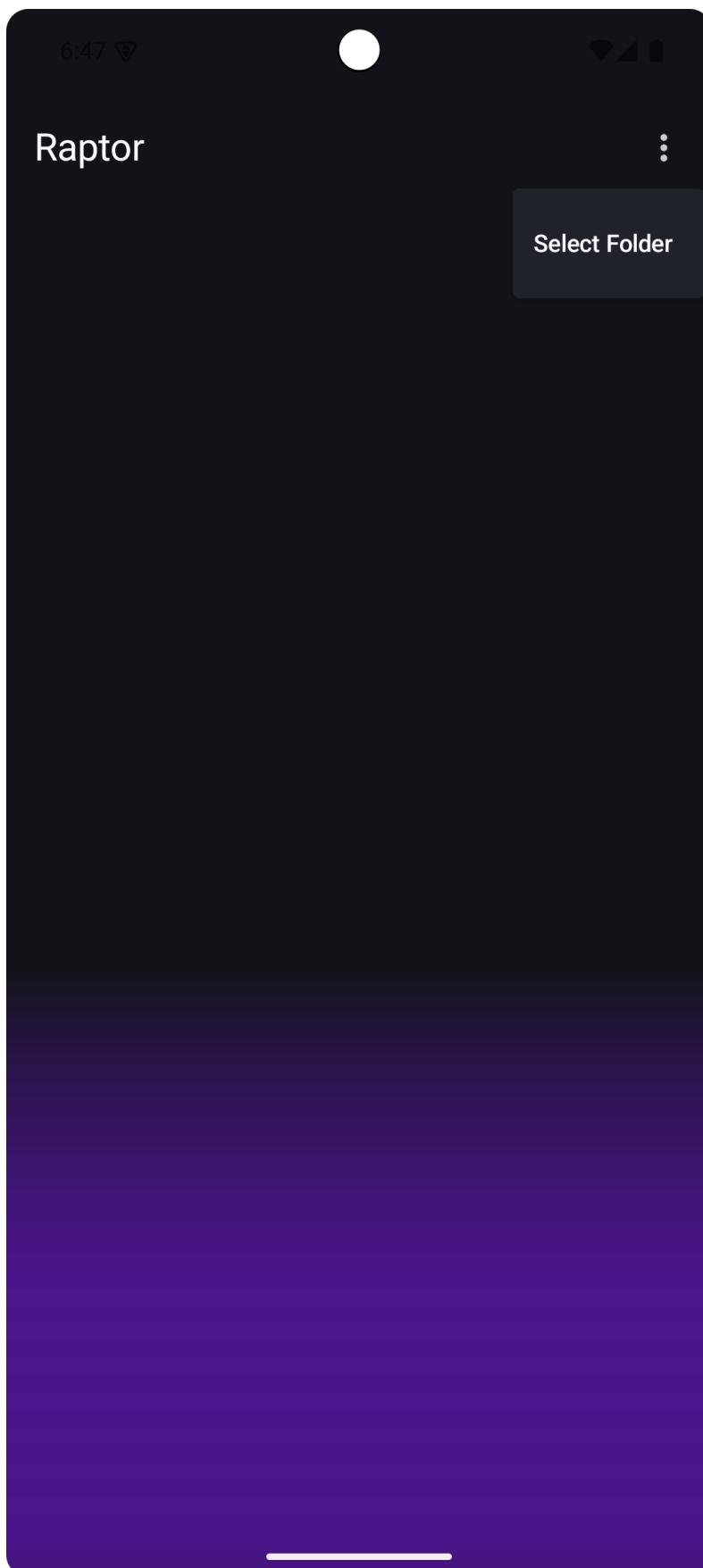
Rys. 6.1. Prośba o podanie odcisku palca.



Rys. 6.2. Widok po zweryfikowaniu odcisku.

Zgodnie z założeniami, po otwarciu aplikacji pojawia się prośba o podanie odcisku palca (rysunek nr. 6.1). Po zweryfikowaniu, aplikacja pomyślnie przechodzi do głównego ekranu, jak pokazano na rysunku nr. 6.2.

6.2. Dodawanie utworów



Rys. 6.3. Włączenie wyboru folderu.

6:53



☰ Music



sdk_gphone64_x86_64 > [Music](#)

Files in Music



Andrew Prahlow

chilldive

Jacek Kaczmarski

Victoria II OST R...

What Do I Wish F...

USE THIS FOLDER

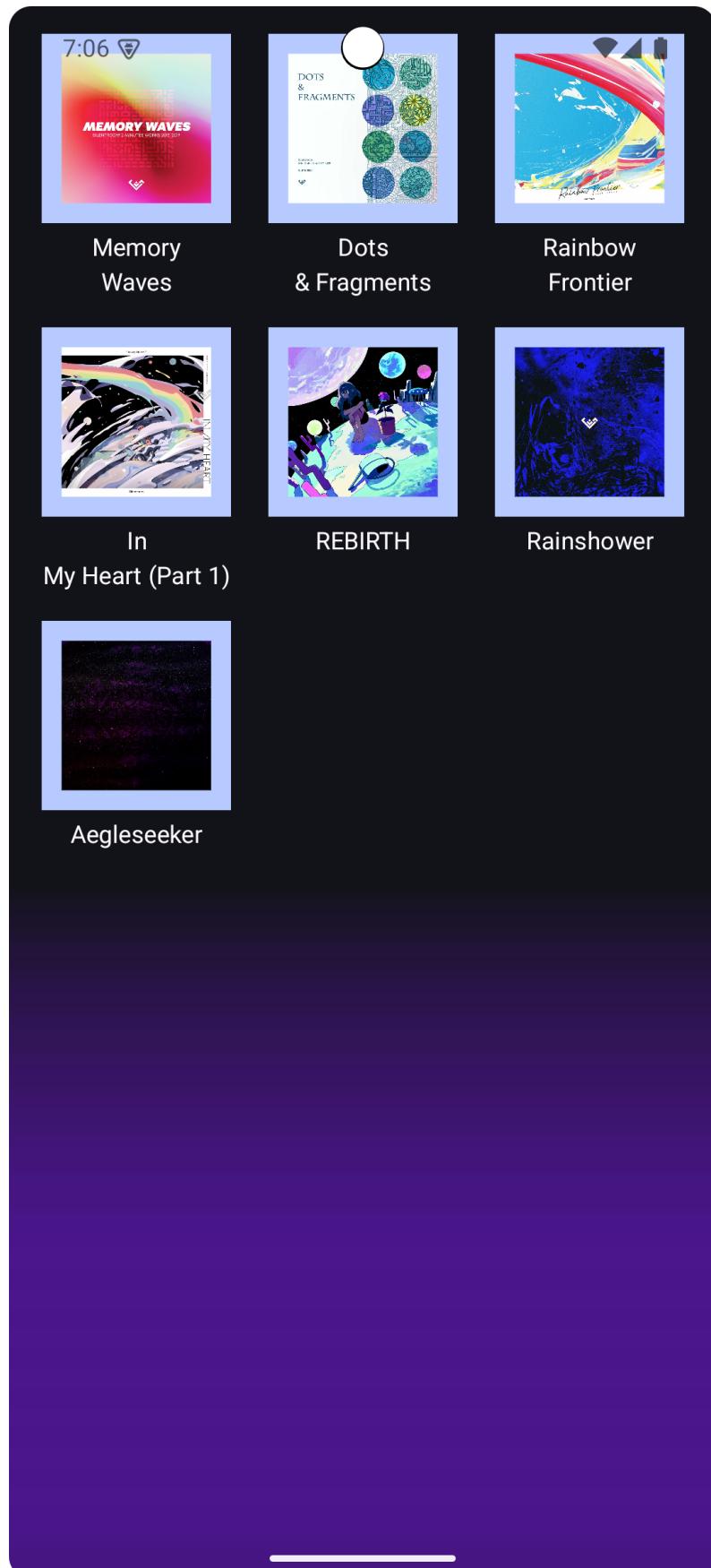
Rys. 6.4. Wybieranie folderu z muzyką.



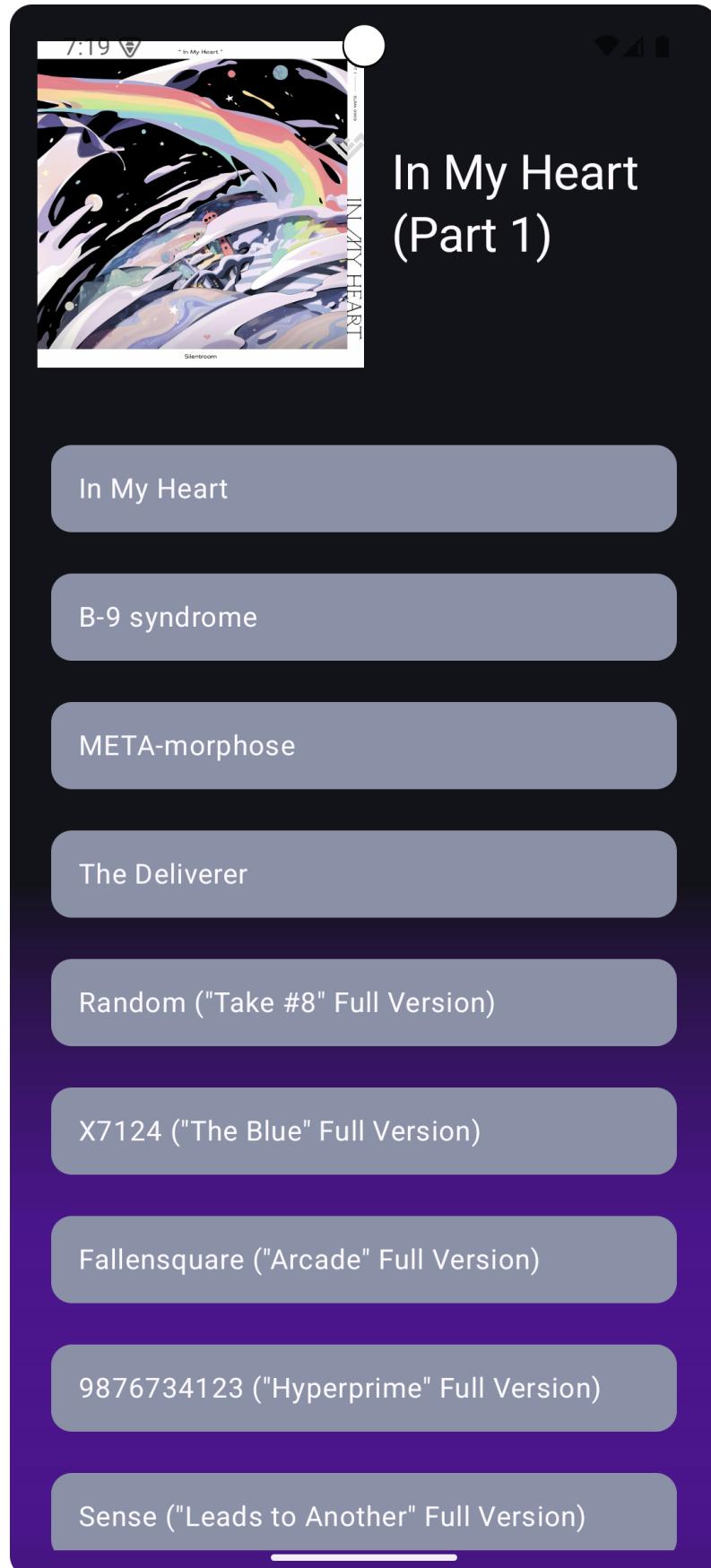
Rys. 6.5. Widok po załadowaniu plików.

Na rysunkach nr. 6.3, 6.4 i 6.5 ukazana jest procedura ładowania plików piosenek. Jak widać piosenki załadowały się pomyślnie i interfejs odpowiednio reaguje na te zmiany. W kafelku o nawie „Unknown”, zawarte są piosenki bez otagowanego wykonawcy.

6.3. Nawigacja



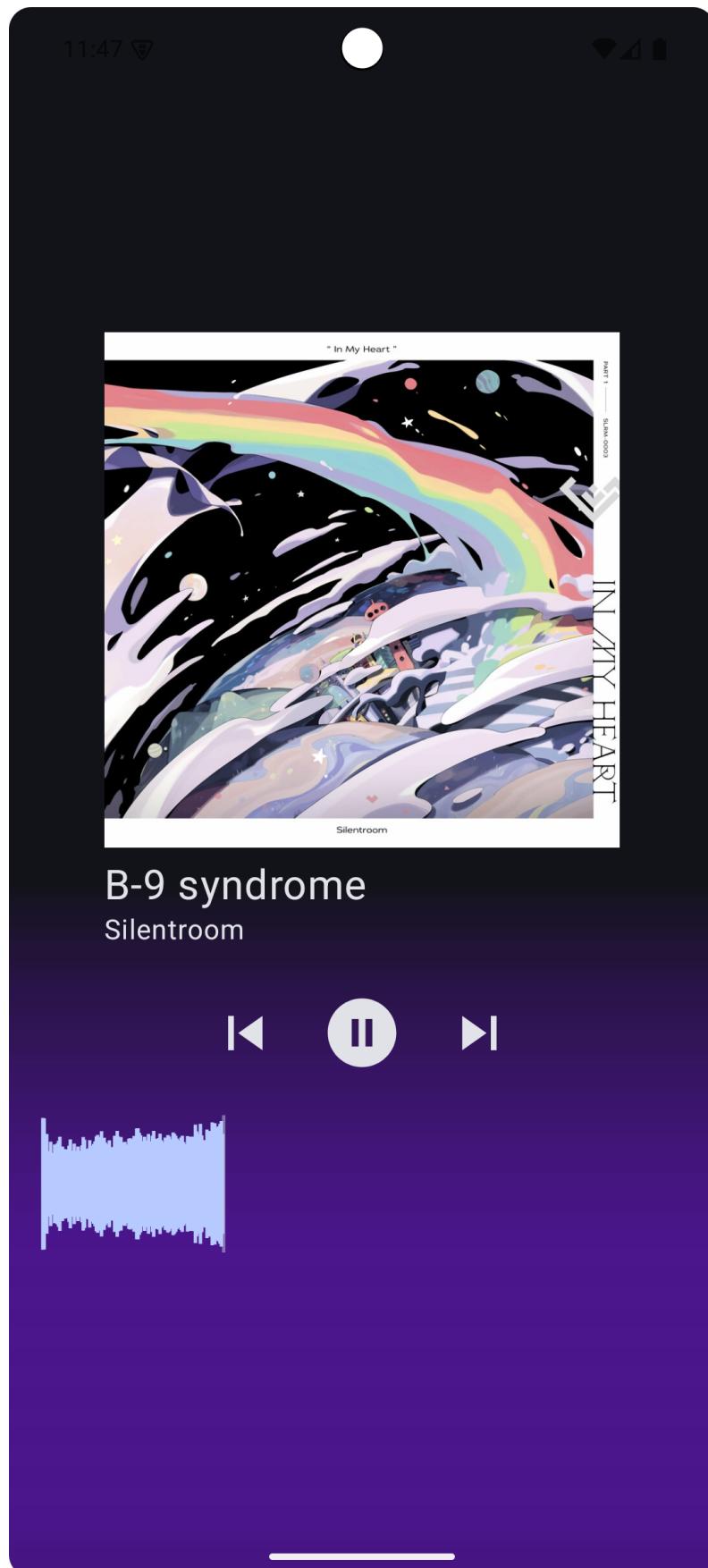
Rys. 6.6. Widok po wejściu w autora.



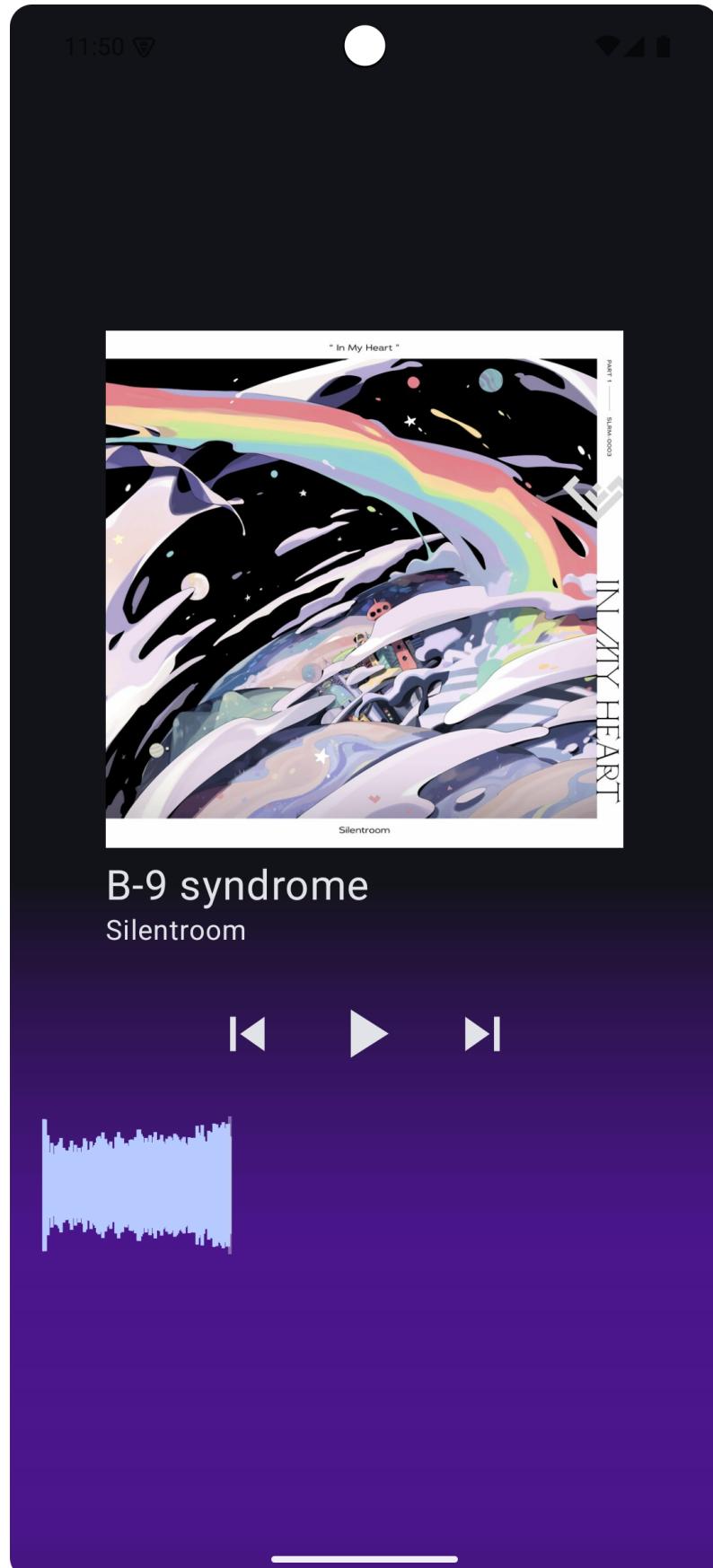
Rys. 6.7. Widok wyboru piosenki.

Na rysunku nr. 6.6 pokazano, że pomyślnie można przejść do ekranu widoku albumów. Pomyślnie wyświetlają się wszystkie miniaturki. Analogicznie, z rys. nr. 6.7 wynika, że można przejść do widoku piosenek. Podobnie, informacje o albumie, w którym są piosenki wyświetlają się pomyślnie.

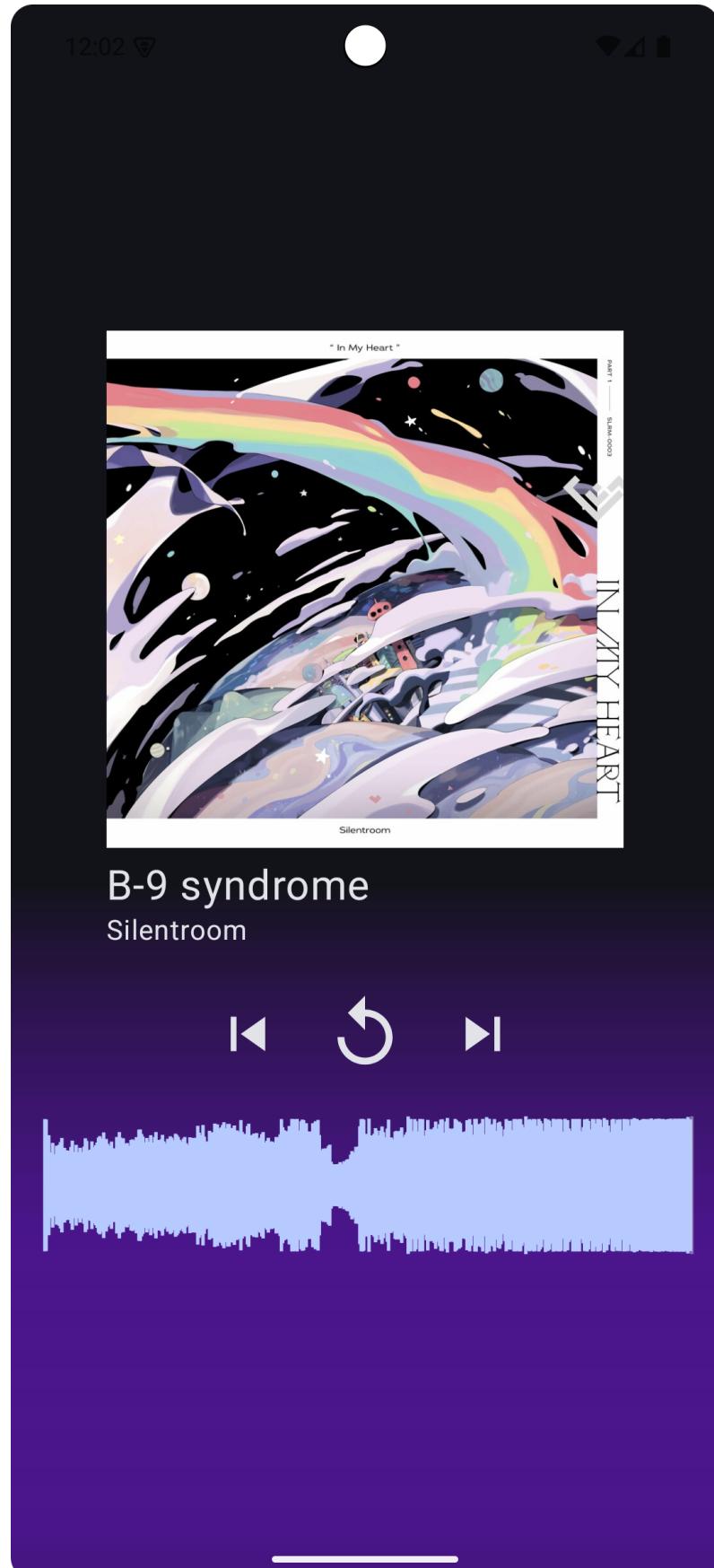
6.4. Działanie odtwarzacza



Rys. 6.8. Odtwarzacz odgrywa piosenkę.

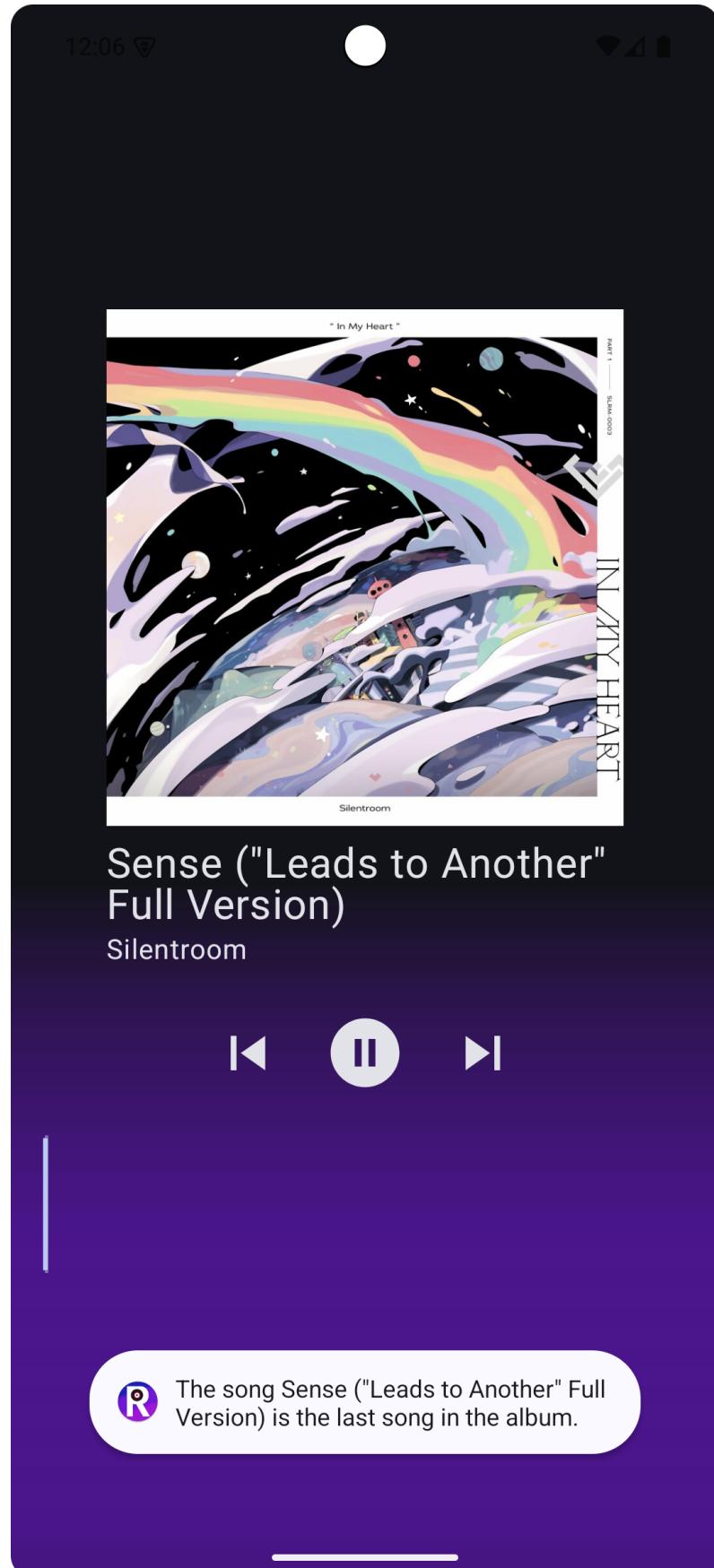


Rys. 6.9. Piosenka w odtwarzaczu jest zapauzowana.



Rys. 6.10. Zatrzymany odtwarzacz

Jak widać na rysunku nr. 6.8, odtwarzacz pomyślnie odgrywa piosenkę. Z wiadomością przyczyn, trudno jest pokazać to, że słyszać dźwięk. Ponadto pokazana jest miniaturka, generowany jest waveform oraz wyświetlane są informacje o piosence, mianowicie tytuł oraz wykonawcy. Na rysunku nr. 6.9, pokazano, że odtwarzacz może być zapauzowywany. Przeciągając progress bar do końca, można zauważyc, że pojawia się możliwość zrestartowania utworu.

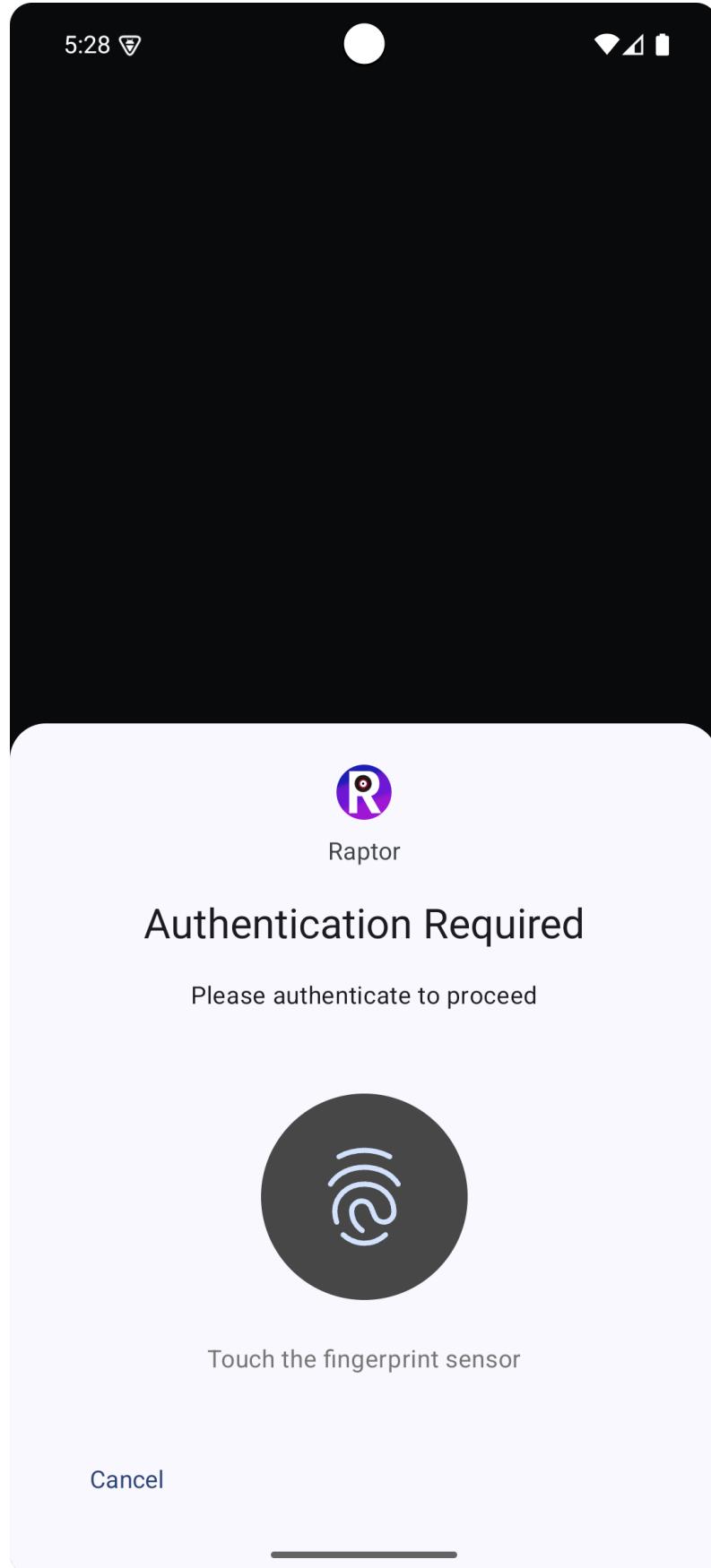


Rys. 6.11. Przechodzenie przez album do końca

Na rysunku nr. 6.11 pokazano możliwość przechodzenia przez piosenki w albumie, przyciskami obok głównego, na środku. Jak widać na tym samym rysunku program wykrywa, gdy użytkownik dostanie się do ostatniego utworu - zostaje wtedy powiadomiony, że nie może przejść dalej.

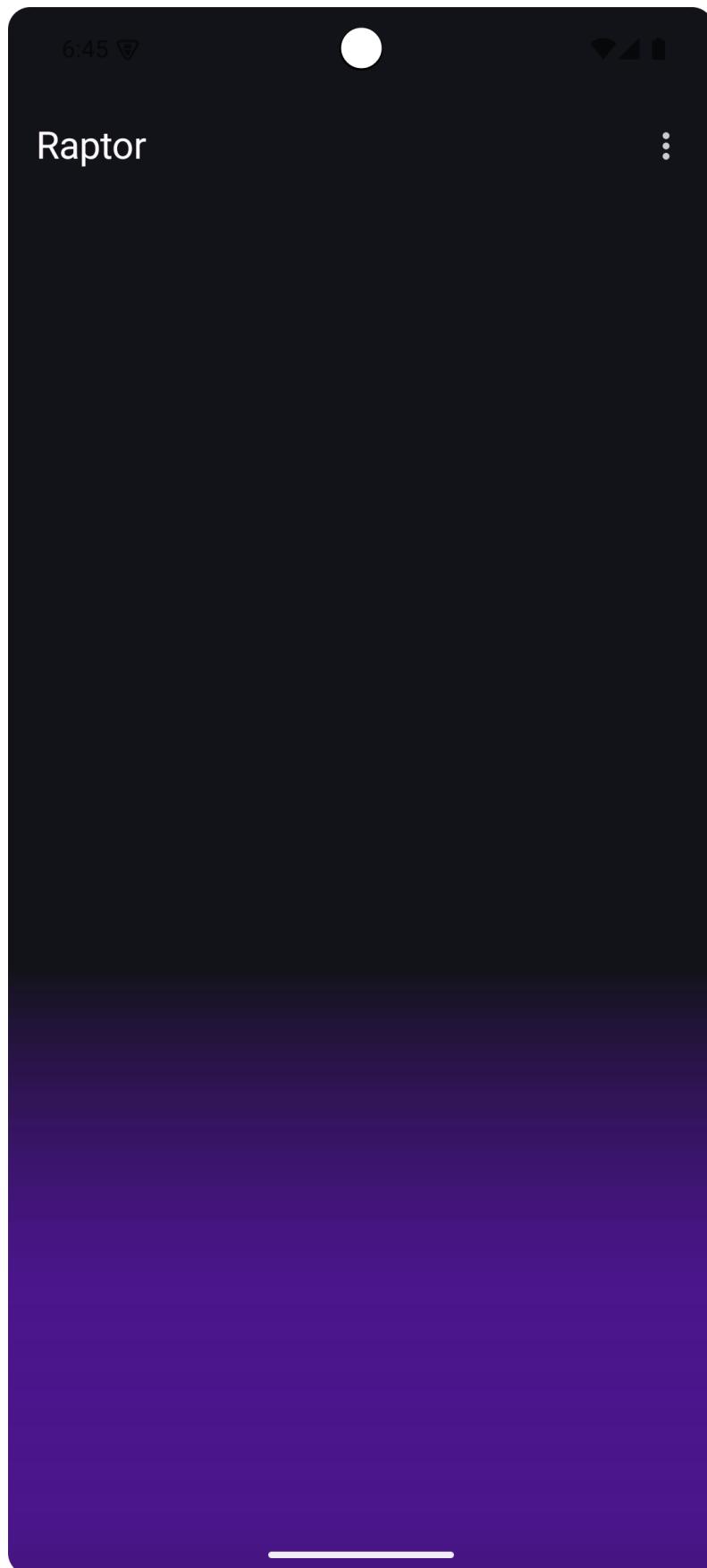
7. Podręcznik użytkownika

Przed włączeniem aplikacji należy upewnić się, że w telefonie użytkownika zapisany jest odcisk palca. Jest on potrzebny do dostania się do aplikacji.



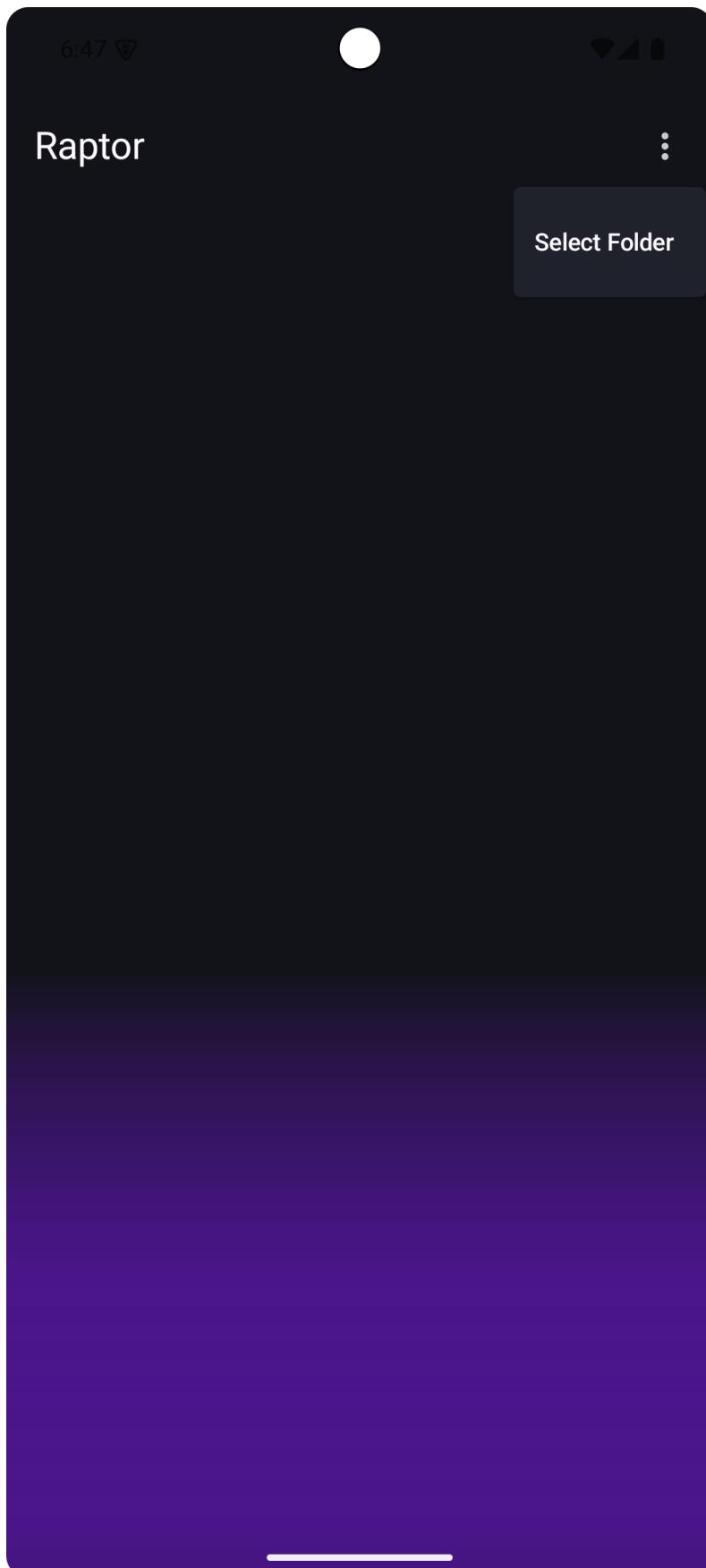
Rys. 7.1. Prośba o podanie odcisku palca.

Po włączeniu aplikacji użytkownik zostanie poproszony o podanie odcisku palca jak widać na rysunku nr. 7.1. Po zweryfikowaniu, aplikacja przechodzi na główny ekran. Domyślnie, wygląda on jak na rysunku nr. 7.2



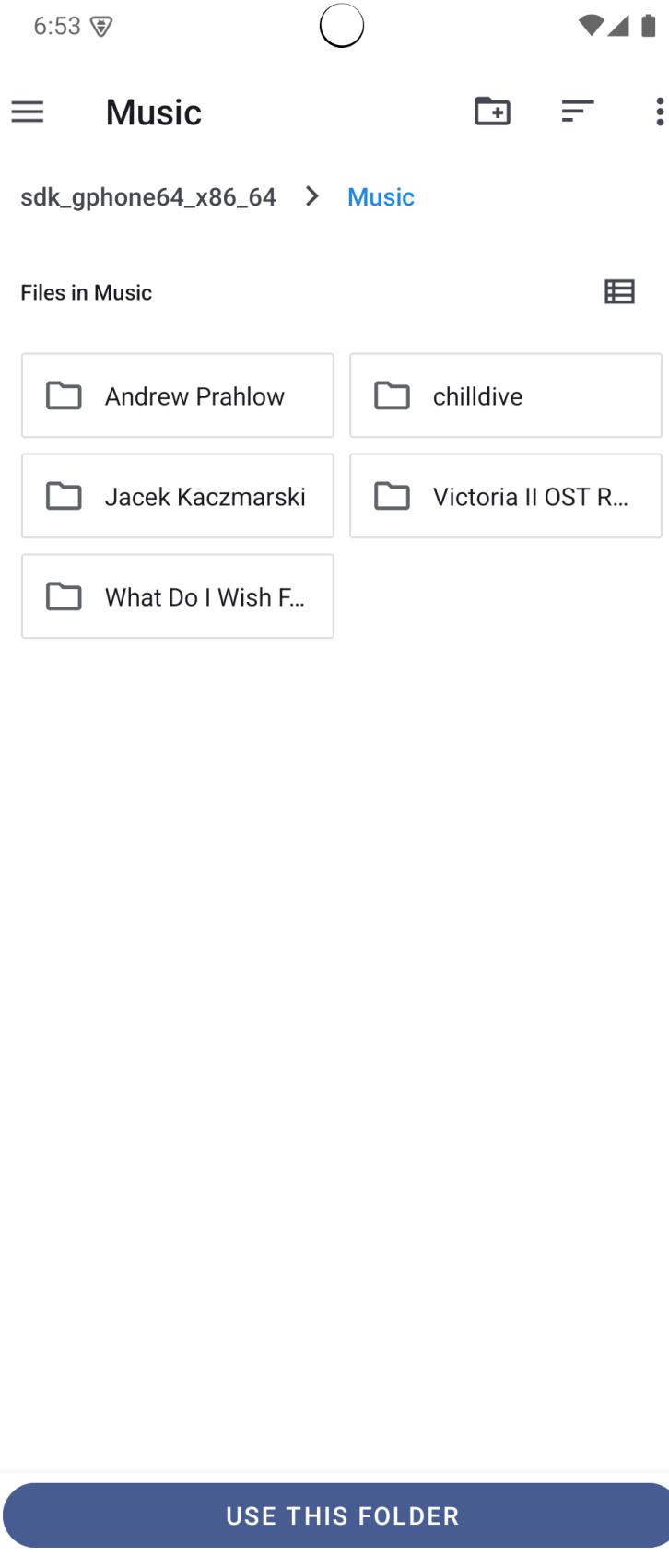
Rys. 7.2. Widok po zweryfikowaniu odcisku.

Jeżeli użytkownik chce dodać swoje piosenki, powinien nacisnąć przycisk w postaci trzech kropek w prawym górnym rogu i wybrać opcję select folder, jak widać na rysunku nr. 7.3.



Rys. 7.3. Włączenie wyboru folderu.

Otworzy się następnie systemowy dialog wyboru folderu. Użytkownik powinien wybrać jakiś, jak na rys. nr. 7.4. Nie musi mieć on bezpośrednio w sobie plików z muzyką, może zawierać podfoldery.



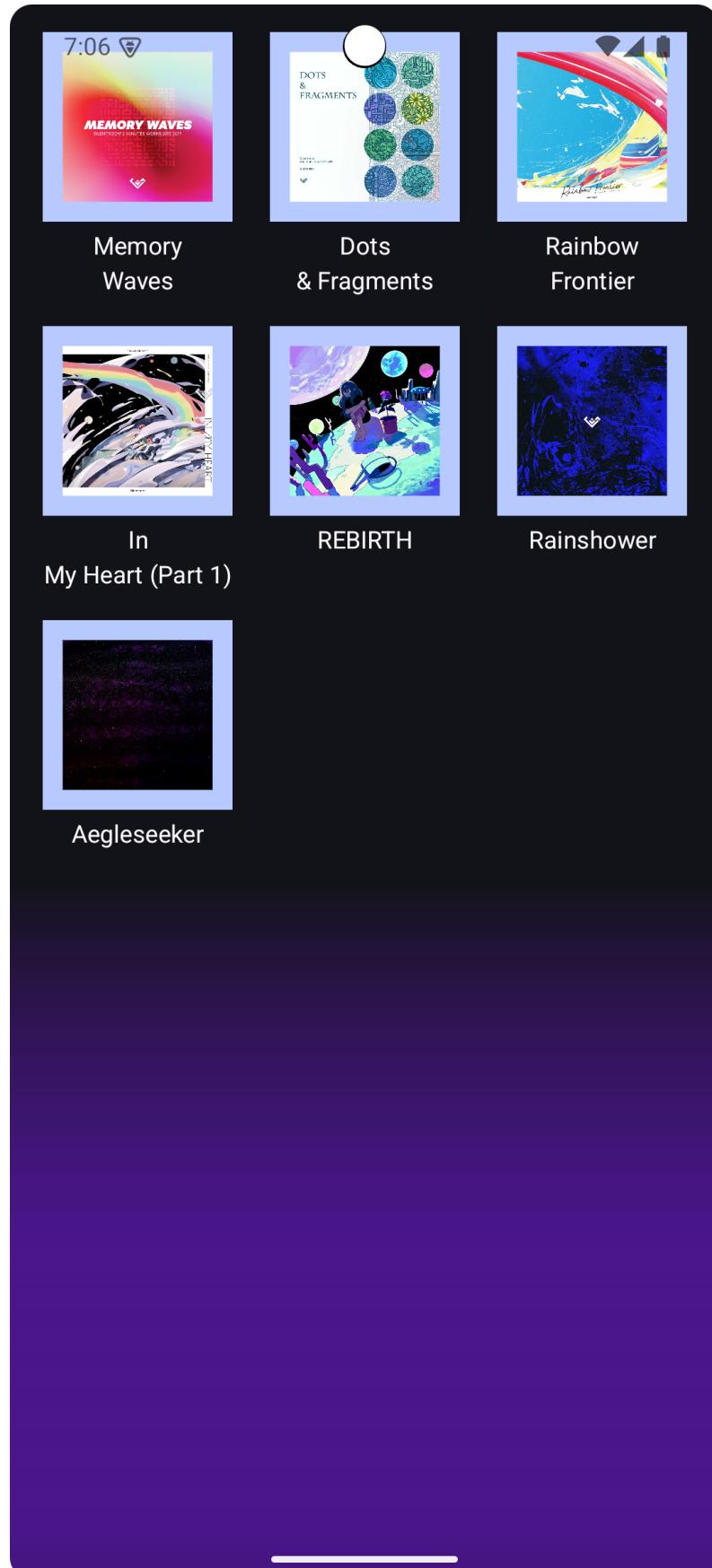
Rys. 7.4. Wybieranie folderu z muzyką.

Po wybraniu folderu, aplikacja wraca użytkownika do ekranu głównego. Po poczekaniu chwili, pokaże się lista autorów, jakie aplikacja wykryła. Jeżeli jakieś pliki nie mają w swoich tagach autora, zostają one przypisane do sekcji **Unknown**. Nowy widok jest przedstawiony na rysunku nr. 7.5.



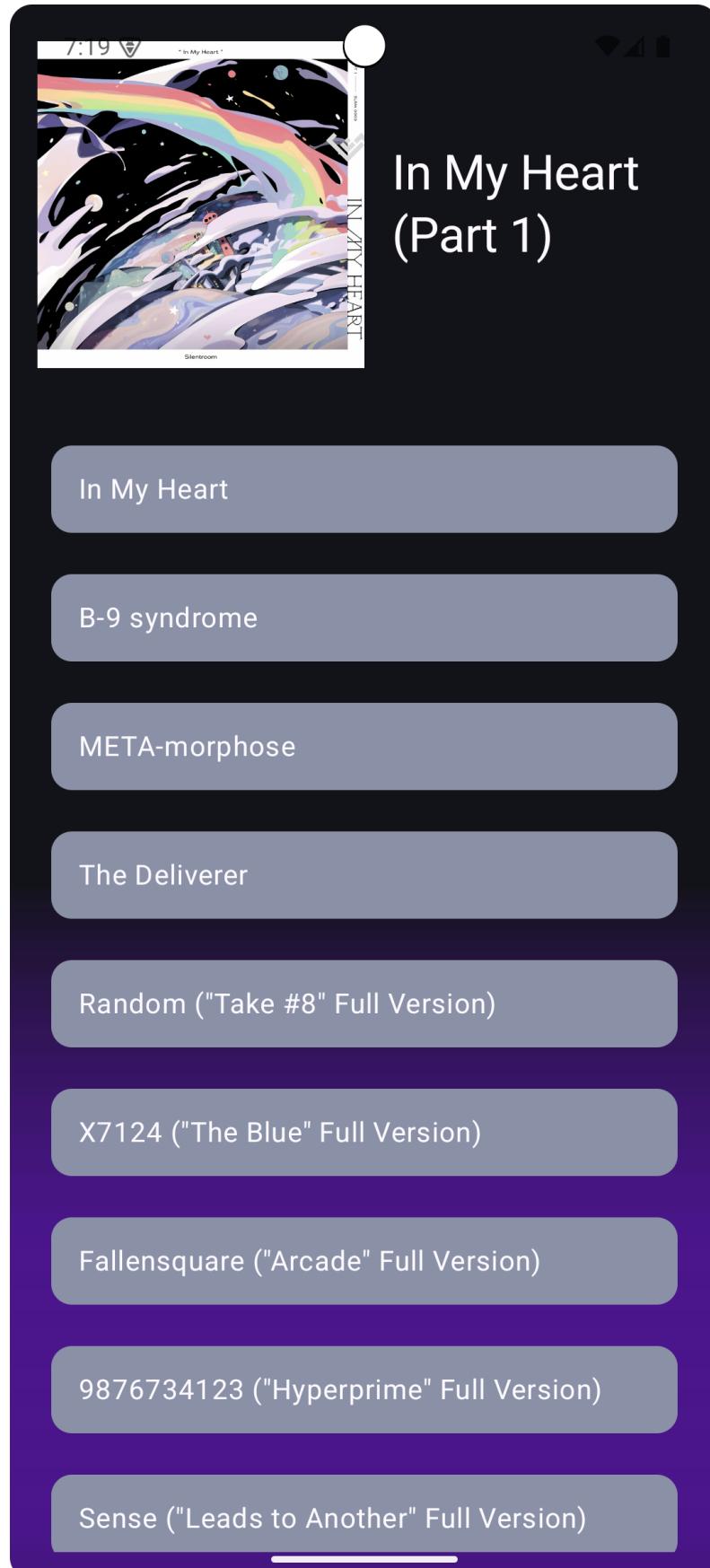
Rys. 7.5. Widok po załadowaniu plików.

Użytkownik może teraz wejść w katalog albumów dowolnego autora, klikając na kafelek z jego nazwą. Przeniesie to użytkownika do widoku albumów, wyglądający jak na rysunku nr. 7.6.



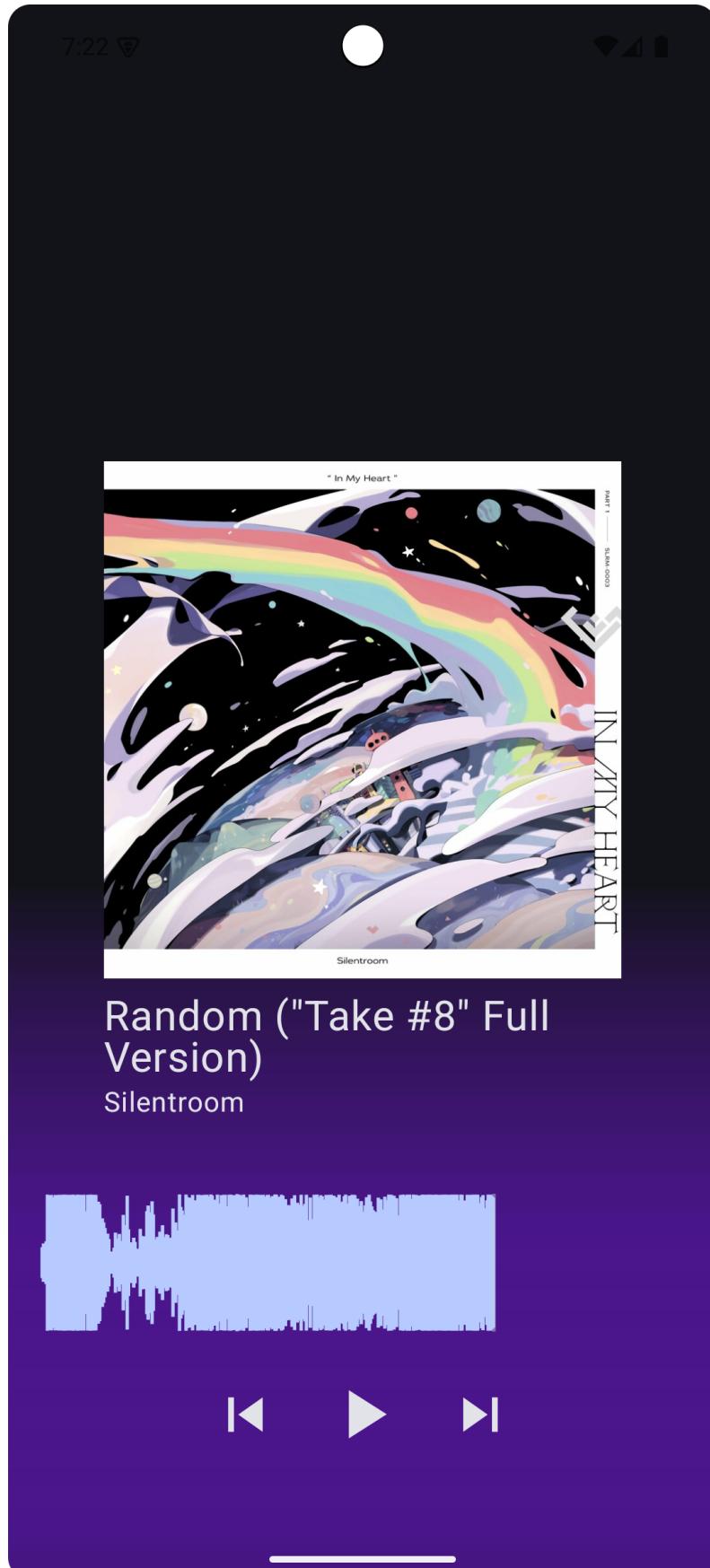
Rys. 7.6. Widok po wejściu w autora.

Analogicznie, należy kliknąć na kafelek korespondujący do odpowiedniego albumu. Aplikacja wtedy przejdzie do widoku piosenek, gdzie można faktycznie wybrać piosenkę do odtworzenia, co pokazano rys. nr. 7.7.



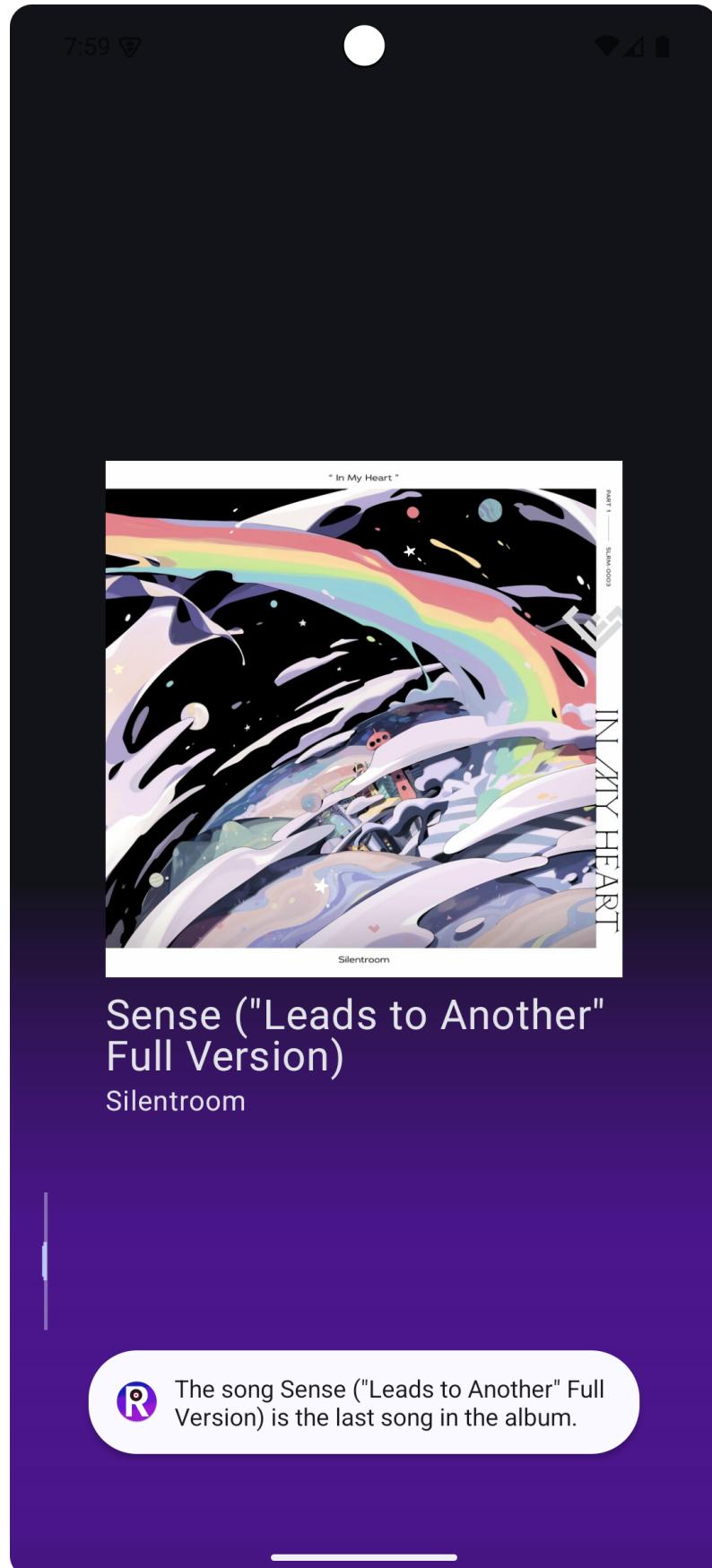
Rys. 7.7. Widok wyboru piosenki.

Po wybraniu piosenki, aplikacja otwiera ekran odtwarzacza, pokazany na rys. nr. 7.8



Rys. 7.8. Widok odtwarzacza.

Waveform na dole odtwarzacza pokazuje postęp odtwarzania. Przycisk na środku, pod nim pełni funkcje odtwarzania, pauzowania i restartowania piosenki, zależnie od tego w jakim stanie jest odtwarzanie. Przyciski po lewej i prawej jego stronie odtwarzają poprzednie i następne utwory. Kolejność odtwarzania determinowana jest poprzez numer w albumie piosenki, jeżeli jest on w tagach. Czyli, dla piosenki o numerze 5, klikając w lewy przycisk, odtwarzanie przejdzie do piosenki o numerze 4 w albumie. Przy prawej strzałce, odtwarzanie przejdzie do piosenki nr. 6. Jeżeli użytkownik spróbuje przejść do piosenki, której numer nie istnieje w albumie, to aplikacja powiadomi go o tym, co widać na rys. nr. 7.9.



Rys. 7.9. Próba zagrania następnej piosenki, grając ostatnią.

Można też zwrócić uwagę na inne funkcje aplikacji. Aplikacja obsługuje standar-dową nawigację w Androidzie, można wyjść z odtwarzacza wbudowanym przyciskiem w tył / przesunięciem palca, łącznie z innymi ekranami. Ponadto zademonstrować można inne czujniki obsługiwane przez aplikację. Jeżeli zwiększy się ilość światła, to aplikacja przejdzie w jasny tryb (bądź ciemny jeżeli światło się zmniejszy), jak pokazano na rys. nr. 7.10.



Rys. 7.10. Aplikacja zmieniła kolorystykę na jasną.

Ekran można też obrócić o 90°, co zmieni nieco układ aplikacji. Pokazano to na rys. nr. 7.11



Rys. 7.11. Obrócenie ekranu.

Bibliografia

- [1] *Dokumentacja Jetpack Compose.* URL: <https://developer.android.com/compose>.
- [2] *Dokumentacja Biblioteki Media3.* URL: <https://developer.android.com/media/media3>.
- [3] *Dokumentacja modułu AudioProcessor.* URL: <https://developer.android.com/reference/androidx/media3/common/audio/AudioProcessor>.
- [4] *Dokumentacja Modułu Exoplayer w Media3.* URL: <https://developer.android.com/media/media3/exoplayer>.
- [5] *Dokumentacja Biblioteki Room.* URL: <https://developer.android.com/jetpack/androidx/releases/room>.
- [6] *Dokumentacja Sensor API.* URL: https://developer.android.com/develop/sensors-and-location/sensors/sensors%5C_overview.
- [7] *Dokumentacja modułu MediaRecorder.* URL: <https://developer.android.com/media/platform/mediarecorder>.
- [8] *Dokumentacja Biblioteki Hilt.* URL: <https://developer.android.com/training/dependency-injection/hilt-android>.

Spis rysunków

1.1.	Mockup widoku biblioteki - listing wykonawców	5
1.2.	Mockup widoku albumów danego wykonawcy	6
1.3.	Mockup widoku wyboru utworu	6
1.4.	Mockup widoku wyboru utworu	7
3.1.	Model relacji bazy	12
6.1.	Prośba o podanie odcisku palca.	80
6.2.	Widok po zweryfikowaniu odcisku.	81
6.3.	Włączenie wyboru folderu.	84
6.4.	Wybieranie folderu z muzyką.	85
6.5.	Widok po załadowaniu plików.	86
6.6.	Widok po wejściu w autora.	89
6.7.	Widok wyboru piosenki.	90
6.8.	Odtwarzacz odgrywa piosenkę.	93
6.9.	Piosenka w odtwarzaczu jest zapauzowana.	94
6.10.	Zatrzymany odtwarzacz	95
6.11.	Przechodzenie przez album do końca	97
7.1.	Prośba o podanie odcisku palca.	100
7.2.	Widok po zweryfikowaniu odcisku.	102
7.3.	Włączenie wyboru folderu.	104
7.4.	Wybieranie folderu z muzyką.	106
7.5.	Widok po załadowaniu plików.	108
7.6.	Widok po wejściu w autora.	110
7.7.	Widok wyboru piosenki.	112
7.8.	Widok odtwarzacza.	114
7.9.	Próba zagrania następnej piosenki, grając ostatnią.	116
7.10.	Aplikacja zmieniła kolorystykę na jasną.	118
7.11.	Obrócenie ekranu.	119

Spis tabel

Spis listingów

1.	kotlin001 - Funkcje	9
2.	kotlin002 - Zmienne	9
3.	Strukutura klasy DatabaseManager	16
4.	Deklaracja bazy LibraryDb	20
5.	Deklaracja interfejsu UIdao	20
6.	Deklaracja interfejsu LogicDao	21
7.	Deklaracja tabeli Author	23
8.	Deklaracja tabeli Album	23
9.	Deklaracja tabeli Song	24
10.	Deklaracja relacji AlbumWithSongs	25
11.	Deklaracja tabeli relacji AlbumAuthorCrossRef	25
12.	Deklaracja relacji AlbumWithAuthors	25
13.	Deklaracja relacji AuthorWithAlbums	26
14.	Implementacja czujnika światłą w MainActivity.kt	26
15.	Sprawdzenie dostępności uwierzytelnienia biometrycznego	29
16.	Sprawdzenie monitu biometrycznego	30
17.	Obsługa wyniku monitu	32
18.	Zawartosć onCreate	32
19.	Deklaracja tagów	33
20.	Metoda buildSongInfo()	33
21.	Metoda TagExtractor()	37
22.	Kod MusicFileLoader	38
23.	Kod Navhost AlbumsScreen	42
24.	Kod Navhost PortraitView	44
25.	Kod Navhost LandscapeView	45
26.	Strukutura klasy DatabaseManager	47
27.	Deklaracja bazy LibraryDb	51
28.	Deklaracja interfejsu UIdao	51
29.	Deklaracja interfejsu LogicDao	52
30.	Deklaracja tabeli Author	54
31.	Deklaracja tabeli Album	54

32. Deklaracja tabeli Song	55
33. Deklaracja relacji <code>AlbumWithSongs</code>	56
34. Deklaracja tabeli relacji <code>AlbumAuthorCrossRef</code>	56
35. Deklaracja relacji <code>AlbumWithAuthors</code>	56
36. Deklaracja relacji <code>AuthorWithAlbums</code>	57
37. Implementacja czujnika światłą w <code>MainActivity.kt</code>	57
38. Sprawdzenie dostępności uwierzytelnienia biometrycznego	60
39. Sprawdzenie monitu biometrycznego	61
40. Obsługa wyniku monitu	63
41. Zawartość <code>onCreate</code>	63
42. Deklaracja tagów	64
43. Metoda <code>buildSongInfo()</code>	64
44. Metoda <code>TagExtractor()</code>	68
45. Kod <code>MusicFileLoader</code>	69
46. Struktura klasy <code>ImageManager</code>	73
47. Inicjalizacja exoplayer	75
48. Zarządzanie stanem	75
49. Funkcja <code>playsong</code>	77
50. Funkcje <code>pause</code> i <code>restartCurrentPlayuback</code>	78
51. Funkcja zmiany pozycji	78
52. Funkcja czyszczenia pamięci	78