

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Raptor

Autor:
Mateusz Stanek
Dawid Szołdra
Filip Wachała

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań projektu	4
1.1. Ogólny zarys wymagań	4
1.2. Wykorzystane czujniki	4
1.3. Zarys interfejsu	4
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
4. Implementacja	13
4.1. Zarządzanie bazą danych	13
4.1.1. Klasa DatabaseManager	13
4.1.2. Klasa LibraryDb	16
4.1.3. Obiekty Dao	16
5. Testowanie	17
6. Podręcznik użytkownika	18
Literatura	19
Spis rysunków	20
Spis tabel	21

Spis listingów

22

1. Ogólne określenie wymagań projektu

1.1. Ogólny zarys wymagań

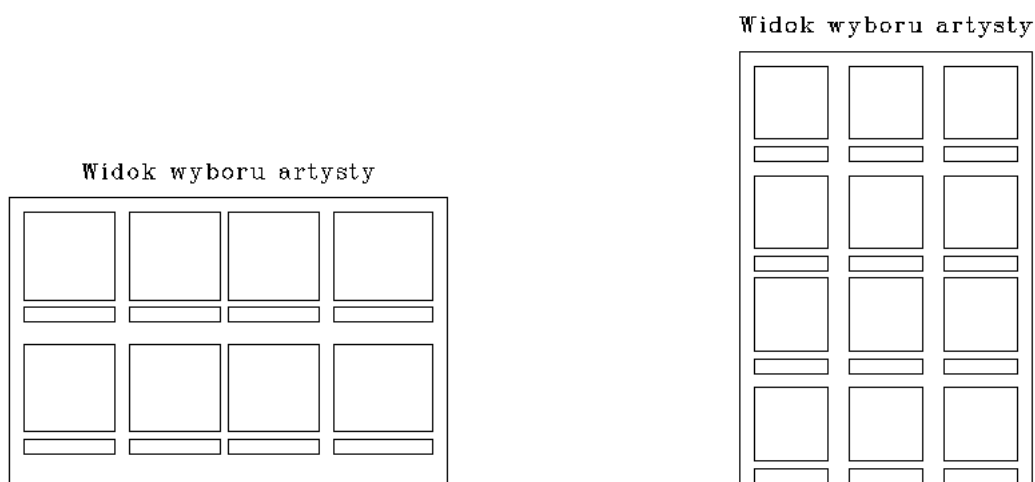
Celem programu jest pełnienie funkcji odtwarzacza muzyki oraz dodatkowo ma on pełnić rolę dyktafonu. Program będzie mógł skanować dany folder, a w nim tagi zawartych plików muzycznych i tworzyć na jego podstawie graficzną reprezentację biblioteki.

1.2. Wykorzystane czujniki

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

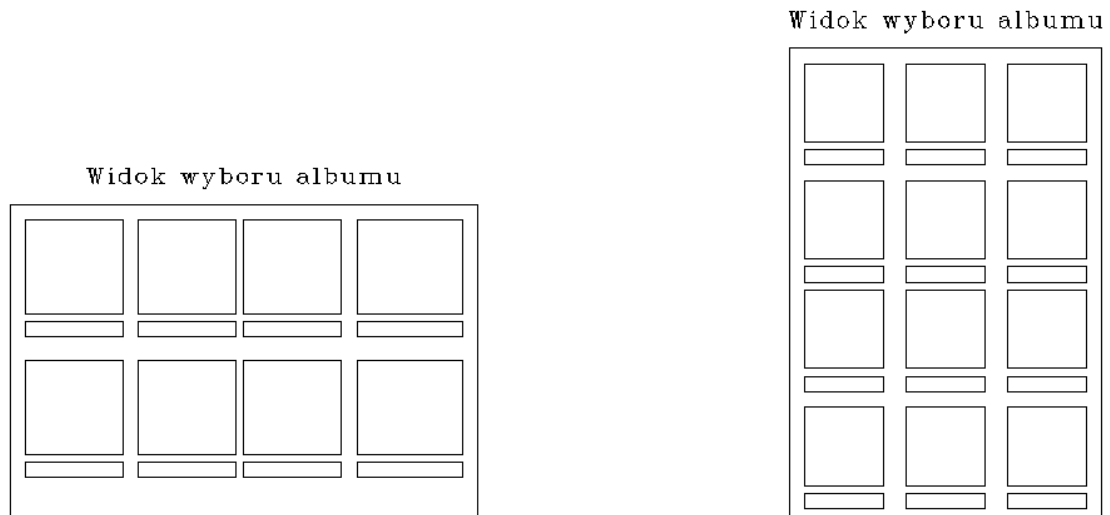
- Żyroskop - Interfejs programu będzie się zmieniał w zależności od orientacji urządzenia.
- Mikrofon - Program będzie posiadał funkcję nagrywania dźwięku. Nagrane pliki będzie można odtwarzać w odtwarzaczu
- Czujnik światła - Interfejs programu będzie mógł zmieniać swoje kolory w zależności od wykrytego poziomu światła na czujniku

1.3. Zarys interfejsu



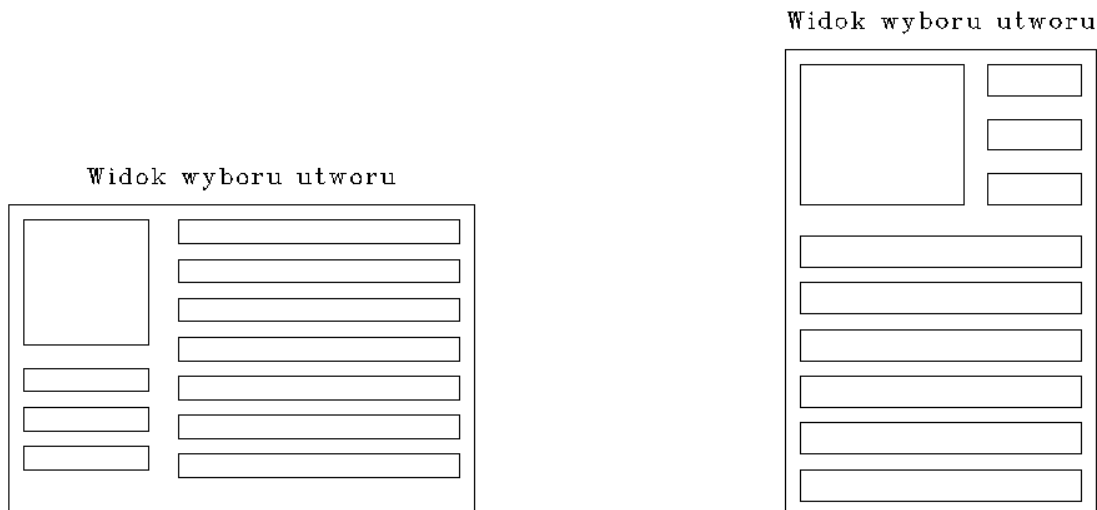
Rys. 1.1. Mockup widoku biblioteki - listing wykonawców

Widok wykonawców, jest przedstawiony na rysunku 1.1. Ten widok będzie ekranem startowym aplikacji. "Kafelki" będą zdjęciami wykonawców. Klikanie na jeden z nich przejdzie do widoku albumów danego wykonawcy



Rys. 1.2. Mockup widoku albumów danego wykonawcy

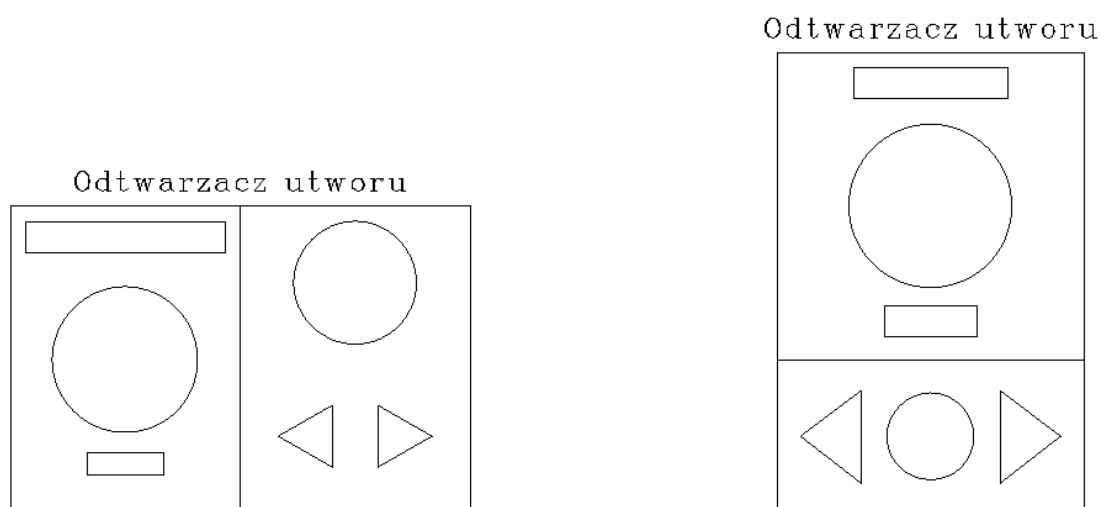
Widok albumów jest przedstawiony na rysunku 1.2. Widok będzie identyczny jak widok wykonawców. Jedyna różnica polega tym, że zdjęcia na kafelkach będą zdjęciami albumów.



Rys. 1.3. Mockup widoku wyboru utworu

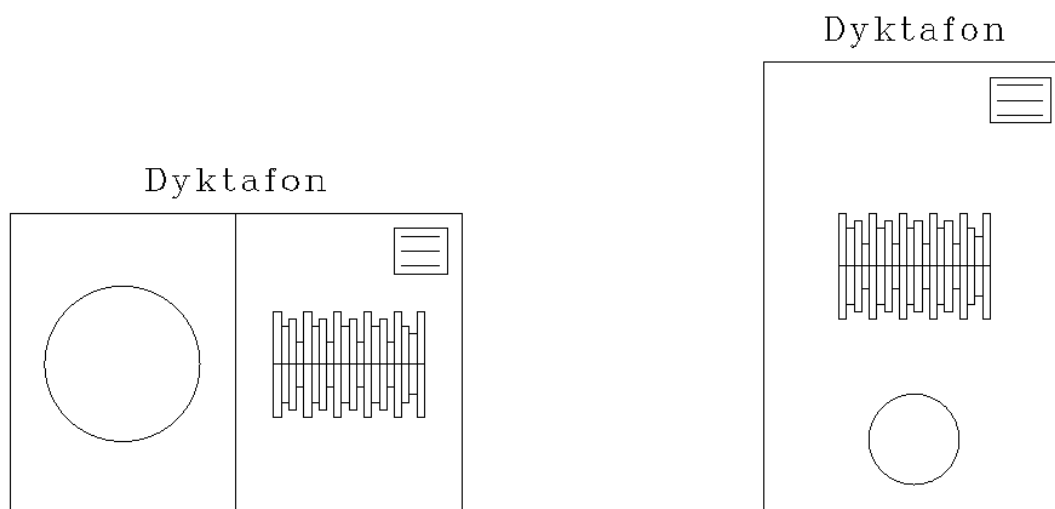
Rysunek 1.3 przedstawia okno pokazujące się po wybraniu albumu. Po wejściu na jakiś album zaprezentowane zostaną zawarte w nim utwory. W lewym górnym jest zdjęcie danego albumu, a obok niego jest kilka informacji o albumie jak wykonawca,

data, tytuł. Dłuższe paski to lista tytułów piosenek, które można kliknąć, aby daną piosenkę włączyć.



Rys. 1.4. Mockup odtwarzacza

Wygląd interfejsu odtwarzacza został zaprezentowany na rysunku 1.4. Odtwarzacz będzie działał następująco: duże koło będzie stylizowane na płytę, gdzie wypełniona ona będzie obrazem albumu. Płyta ta będzie się kręcić w czasie gdy gra piosenka. Kąt płyty (od 0° , do 360°) będzie określał jak duża część piosenki została odtworzona. Kąt ten będzie określony jeszcze niezdefiniowanym efektem graficznym. Prostokąty wokół płyty to tytuł piosenki, a na dole czas grania. Kółko i wokół niego trójkąty to przyciski odtwarzania - graj/pauza, następny, poprzedni.



Rys. 1.5. Mockup dyktafonu

Na rysunku 1.5 zaprezentowany został interfejs dyktafonu. Do dyktafonu będzie można się dostać przesuwając palcem w prawo na ekranie wykonawców. Dyktafon jest aktywowany wielkim okrągłym przyciskiem. Te kreski obok niego to wizualizacja dźwięku z mikrofonu. Menu w rogu będzie pozwalało na m.in. skonfigurowanie folderu zapisu nagrań.

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 Google'a 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 fińskiej.

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 Kotlin jako rekomendowany język do tworzenia aplikacji na Android.

Składnia Kotliny 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 **var** już nie.

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

Listing 2. kotlin002 - Zmienne

Kompilator Kotliny 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 dość 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 i nagrywanie dźwięku.

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 otwierane jest menu wyboru piosenek należących do tego utworu.

3.3. Odczyt i przetwarzanie plików

Aplikacja będzie posiadać bazę danych. Baza danych będzie się składać z trzech tabel.

- Tabela Autorów
- Tabela Albumów
- Tabela Utworów

W bazie danych będą istniały powiązania. W tabelce autorów będą trzy kolumny:

- id_aut
- name
- id_album

Id_aut zawiera id autora, name to nazwa autora, id_album zawiera id albumu. Id będą potrzebne do utworzenia powiązań.

W tabelce Albumów będą cztery kolumny:

- id_album
- name
- id_song,
- id_aut

Jak powyżej, id_album to id albumu, name to nazwa, id_sog to id piosenki, id_aut to id autora.

W tabelce utworów będą cztery kolumny

- id_song
- name
- id_album
- id_aut

Jak powyżej, id_song to id piosenki, name to nazwa, id_album to id albumu, id_aut to id autora.

Id są potrzebne do utworzenia powiązań między tabelkami.

Tabelka autorów będzie odwoływała się do tabelki albumów za pomocą id albumu. Tabelka albumów będzie odwoływała się do tabelki piosenek przy pomocy id song. Tabelka piosenek będzie posiadała odwołania do tabelki autorów i albumów za pomocą id aut i id album.

4. Implementacja

4.1. Zarządzanie bazą danych

4.1.1. Klasa DatabaseManager

Za zarządzanie bazą danych odpowiedzialna jest klasa `DatabaseManager`, której kod jest zamieszczony na listingu nr. 3. 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: xddddddddd
43     val distinctAlbumArtistsList = songs
44         .map { Triple(it.album, it.albumArtists, it.
coverUri) }
45         .distinct()
46     Log.d(javaClass.simpleName, "Distinct artists set:
$distinctAlbumArtistsList")
47
48     distinctAlbumArtistsList.fastForEach {
49         val albumTitle = it.first.toString()
50         val artists = it.second
51         val coverUri = it.third
52
53         val albumId = dao.insertAlbum(Album(
54             title = albumTitle,
55             coverUri = coverUri.toString(),
56         ))
57
58         artists?.fastForEach {
59             dao.insertAlbumAuthorCrossRef(
AlbumAuthorCrossRef(
60                 albumId = albumId,
61                 name = it.toString()
62             ))
63         }
64     }
65 }
66
67 fun addSongs() {
68     songs.fastForEach { song ->
69         Log.d(javaClass.simpleName, "NEW SONG\n")
70         Log.d(javaClass.simpleName, "Album artists: ${song.
albumArtists}")
71
72         val albumWithAuthorCandidates = dao
73             .getAlbumsByTitle(song.album.toString())
```

```

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

```

Listing 3. Struktura klasy DatabaseManager

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

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

Baza jest reprezentowana przez klasę `LibraryDb`, definicję której można zobaczyć w sekcji 4.1.2

Dalej, do liniiki nr. 22 pokazane są metody zwracające różne elementy bazy. Większość z tych metod zwraca `Flow[TODO:]`. Room natywnie obsługuje `Flow`, 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 4.1.3. Niektóre obrabiają dane jak np. `collectSongsByAlbumFlow()` na liniice nr. 17., która mapuje zwraca piosenki z wyjściowej klasy relacyjnej.

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

4.1.2. Klasa LibraryDb

4.1.3. Obiekty Dao

5. Testowanie

6. Podręcznik użytkownika

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	4
1.2. Mockup widoku albumów danego wykonawcy	5
1.3. Mockup widoku wyboru utworu	5
1.4. Mockup odtwarzacza	6
1.5. Mockup dyktafonu	6

Spis tabel

Spis listingów

1.	kotlin001 - Funkcje	9
2.	kotlin002 - Zmienne	9
3.	Struktura klasy <code>DatabaseManager</code>	13