Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Maturitní práce

HUDEBNÍ PŘEHRÁVAČ PRO ANDROID

Profilová část maturitní zkoušky

Studijní obor: Informační technologie

Třída: ITB4

Školní rok: 2020/2021 Valentyn Vorobec

SPŠT

Střední průmyslová škola Třebíč
Manželů Curieových 734, 674 01 Třebíč

Zadání
maturitní práce
studijní obor: **18-20-M/01 informační technologie (počítačové systémy)**

Příjmení a jméno žáka: **Vorobec Valentyn**

Třída: **ITB4** Školní rok: **2020/2021**

Téma č.: **11**

Název tématu: **Hudební přehrávač**

Rozsah práce: **10 - 50 stran**

Konkrétní úkoly, které jsou v práci řešeny:

Naprogramujte hudební přehrávač na Android. Hudební přehrávač bude zvládat přehrávat hudbu, pozastavit přehrávání, posunout o určitý čas, přejít na další skladbu, přejít na předchozí skladbu.

Přehrávač bude zobrazovat seznam všech skladeb v mobilu, bude možno přidávat do fronty přehrávání, přejmenovat, odstranit, změnit obrázek, nastavit jako vyzváněcí tóny.

Dále přehrávač bude umožňovat přidávání skladeb do svého seznamu oblíbených, sdílet a zobrazovat detaily (velikost souboru, trvání, jméno autora atd.).

Přehrávač bude mít taky možnost vyhledat skladbu.

Přehrávat hudbu bude možno buď v pořadí určeném, nebo náhodném.

Dále bude přehrávač umožňovat třídění skladeb podle autora.

Použijte programovací jazyk Kotlin ve vývojovém prostředí Android Studio.

Termín odevzdání: **19. 3. 2021**

Vedoucí práce: **Ing. Jaroslav Riedel** Riedel J.

Oponent: **Ing. Jiří Šálek** Šálek

Zadání schváleno dne: 12. 10. 2020

Schválil: Ing. Zdeněk Borůvka, ředitel školy

ABSTRAKT

Cílem této práce je návrh hudebního přehrávače pro mobily s operační systémem Android, vytvořená Valentynem Vorobcem, třída ITB4. Jedná se o ročníkový projekt pro rok 2021. Dokument obsahuje čtyři hlavní části. První dvě jsou teoretické, třetí popisuje praktické vypracování programu a čtvrtou část tvoří závěr shrnující výsledky této práce. Tato práce popisuje, jaké možnosti nabízí a jak aplikace funguje.

KLÍČOVÁ SLOVA

Hudební přehrávač, Kotlin, Android, Hudba

ABSTRACT

The aim of this work is to design a music player for mobile phones with the Android operating system, created by Valentyn Vorobec, class ITB4. This is an annual project for 2021. The document contains four main parts. The first two are theoretical, the third describes the practical elaboration of the program and the fourth part is the conclusion summarizing the results of this work. This work describes what options it offers and how the application works.

KEYWORDS

Music Player, Kotlin, Android, Music

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil.

V Třebíči dne 31. března 2021 .....

podpis autor

PODĚKOVÁNÍ

Chtěl bych poděkovat Panu Riedlovi za výpomoc a rady při tvorbě abstraktu a za obětavost.

V Třebíči dne 31. března 2021 .....

podpis autora

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Má práce se snaží poukázat, jakým způsobem lze naprogramovat primitivní hudební přehrávač, který bude zvládat základní jeho vlastnosti. Tyto vlastnosti jsou pak nadále využívány aplikacemi jako Spotify, Google hudba, Audible atd. Zároveň, tato práce může sloužit jako vzor komukoliv, kdo by si chtěl sám hudební přehrávač udělat. Předem upozorňuji, že aplikace byla naprogramována pro android verze 8.1 a proto nemohu zaručit funkčnost u mobilů s jinou verzí operačního systému. Použitý emulátor je Pixel. Hudba, jenž používám, je též předem vybraná, abych předešel komplikacím se souborovými typy.

Práce je strukturována do dvou hlavních částí. První část je teoretická, druhá popisuje praktické vypracování aplikace a vše shrnující závěr tvoří poslední část. První část (teoretická) se zaměřovat na popis IDE, jazyků, knihoven a gradle. Poté začne část praktická, kde už je popisováno zobrazení layoutu, jeho UI prvky a struktura kódu.

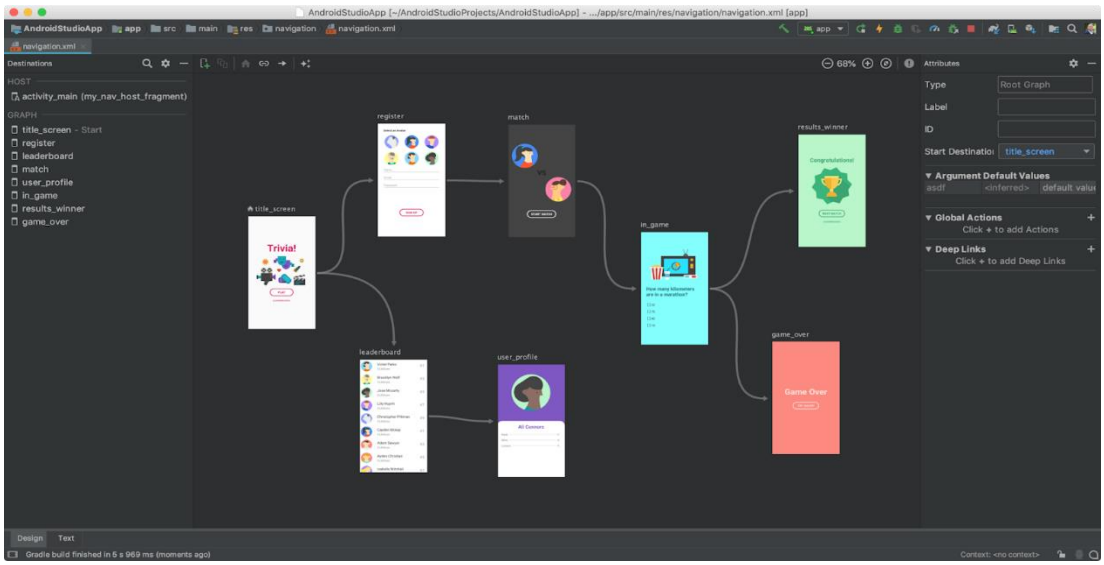
Práce je primárně myšlena jako demonstrace hudebního přehrávače. Aplikace sice ztrácí praktické využití, avšak díky znalostem získaným z této práce by bylo možné navrhnout aplikaci, která by byla mnohem lépe použitelná.

1 TEORETICKÁ ČÁST

V teoretické části popisuji použité IDE Android Studio a programovací jazyk Kotlin.

1.1 Android studio

K vývoji této aplikace jsem použil známé IDE Android Studio od společnosti Google. V tomto prostředí se učíme od začátku čtvrtého ročníku. Může být použito pro vývoj mobilních aplikací pro telefony s operační systémem Android. Prostředí podporuje primárně programovací jazyky Java a Kotlin. Zároveň lze jednoduše propojit s platformami Git a Github.



Obr. 1 Ukázka Android Studia od Google Developers Blog

1.2 Programovací jazyk Kotlin

V jazyce Kotlin se sice ve škole programovat neučíme, ale vzhledem jeho podobnosti k Javě nám bylo dovoleno s ním pracovat. Při vypracovávání této práce jsem se v něm velmi zlepšil a naučil spoustu nových věcí.
Kotlin je staticky typovaný programovací jazyk běžící souběžně se sadou počítačových programů JVM s možností kompilace do JavaScriptu, vyvinutý ruskou firmou JetBrains, jež má hlavní sídlo v Praze. [1]
V květnu roku 2017 na konferenci Google I/O bylo oznámeno, že se Kotlin stává oficiálním programovacím jazykem pro mobilní platformu Android. [2]

```kotlin
fun sayHello(maybe : String?, neverNull : Int) {
    // použití operátoru Elvis
    val name : String = maybe ?: "stranger"
    println("Hello $name")
}

// vrátí null pokud je foo null, nebo bar() nebo
foo ?. bar() ?. baz()
```

Obr. 2 Ukázka jazyka Kotlinu z Wikipedie

1.3 Objektově orientované programování

Objektově orientované programování, zkráceně OOP. Tato metoda se využívá pro zjednodušení kódu, k jeho větší přehlednosti a k možnosti znovupoužití.

1.4 XML

Extensible Markup Language, zkráceně **XML**, je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků (tzv. aplikací) pro různé účely a různé typy dat. Používá se pro serializaci dat, v čemž soupeří např. s JSON či YAML. Zpracování XML je podporováno řadou nástrojů a programovacích jazyků.

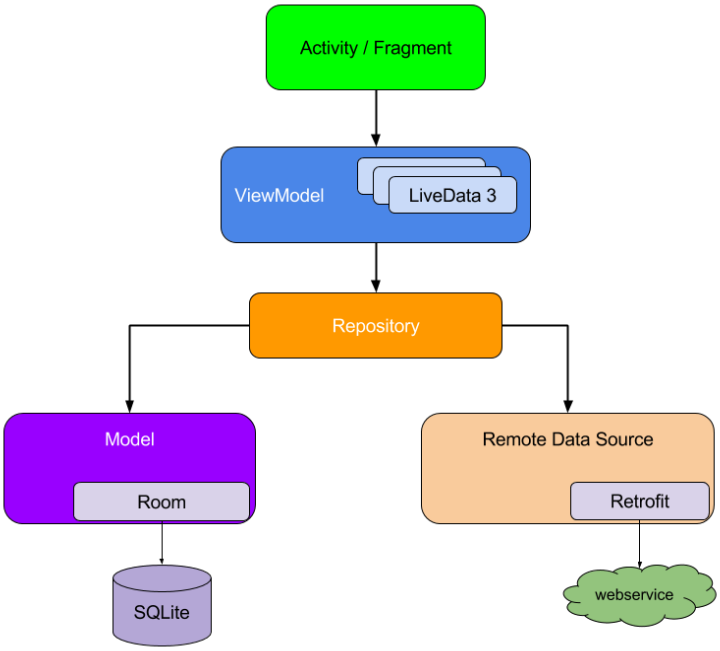
Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých popisuje strukturu z hlediska věcného obsahu jednotlivých částí, nezabývá se vzhledem. Prezentace dokumentu (vzhled) může být definována pomocí kaskádových stylů. Další možností zpracování je transformace do jiného typu dokumentu, nebo do jiné aplikace XML. [3]

1.5 Gradle

Gradle je systém sestavení (otevřený zdroj), který se používá k automatizaci vytváření, testování, nasazení atd. „Build.gradle“ jsou skripty, kde lze automatizovat úkoly. Například jednoduchý úkol zkopírovat některé soubory z jednoho adresáře do druhého lze provést skriptem Gradle build dříve, než dojde ke skutečnému procesu sestavení. [4]

1.6 MVVM

MVVM (Model View ViewModel) je architektura, která usnadňuje práci s databází. Skládá se ze třídy ViewModel, která spolupracuje s daty pomocí Repository, tedy prostředníka ze vzdálené nebo lokální (RoomDatabase) databáze. Moje aplikace pracuje s Room databází. Ta obsahuje dvě tabulky (playlist table a favorite table).



Obr. 3 MVVM architektura

2 PRAKTICKÉ VYPRACOVÁNÍ

V praktickém oddílu je popsán samotný kód, struktura aplikace, jednotlivé funkcionality programu a uživatelské rozhraní.

V následujících podkapitolách je postupně rozebráno uživatelské rozhraní a logika programu tak, jak se s ní uživatel setkává. K vytvoření UI jsem využil XML.

2.1 Gradle knihovny

Nejdřív potřebujeme importovat knihovny, které potřebujeme k realizaci aplikace. Ty se implementují ve složce gradle.build. Některé knihovny jsou automaticky přidány při vytvoření projektu.

```
    // -- Navigation
    def navigation_version = "2.3.1"
    implementation "androidx.navigation:navigation-fragment-ktx:$navigation_version"
    implementation "androidx.navigation:navigation-ui-ktx:$navigation_version"

    // -- Dexter
    def dexter_version = "6.0.2"
    implementation "com.karumi:dexter:$dexter_version"

    // -- Coroutines
    def coroutines_version = "1.3.9"
    implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:$coroutines_version"

    // -- Room
    def room_version = "2.2.5"
    implementation "androidx.room:room-runtime:$room_version"
    kapt "androidx.room:room-compiler:$room_version"
    implementation "androidx.room:room-ktx:$room_version"

    // -- Gson
    def gson_version = "2.8.6"
    implementation "com.google.code.gson:gson:$gson_version"

    // -- Mp3agic
    def mp3agic_version = "0.9.1"
    implementation "com.mpatric:mp3agic:$mp3agic_version"
}
```

Obr. 4 Potřebné knihovny v gradle.build

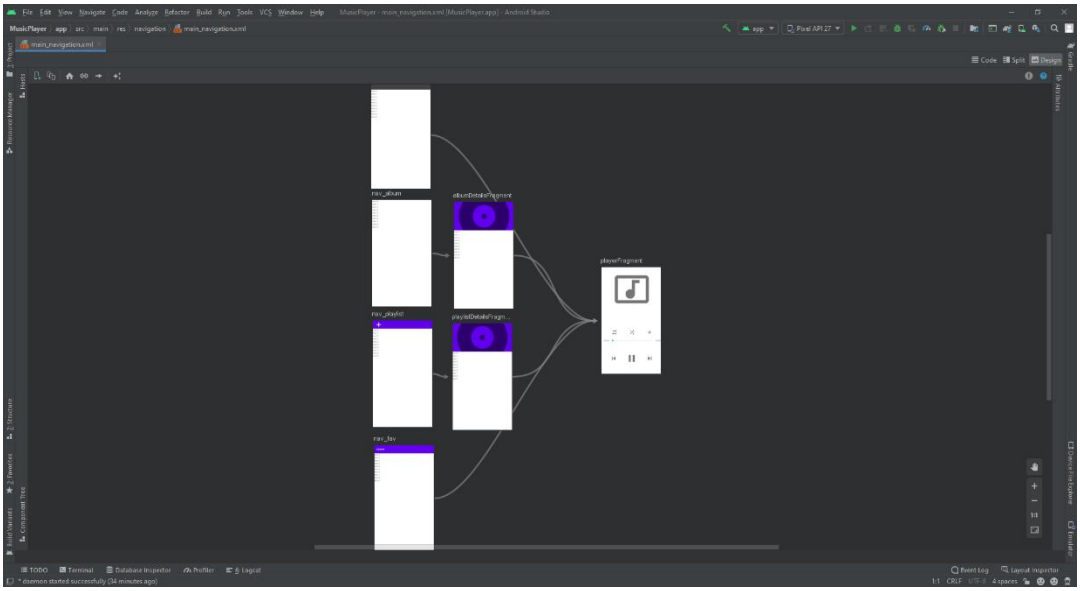
2.1.1 Knihovny

- Jetpack Navigation, je knihovna, která byla přidána roku 2017. Ta má sloužit k zjednodušení tvorby navigace (přemísťování mezi fragmenty) aplikací. Jediné co nám stačí udělat je vytvořit všechny potřebné fragmenty a pomocí drag-and-drop je přidat do xml souboru s názvem main_navigation.xml. [5]

- Dexter je knihovna pro Android, která zjednodušuje proces vyžádání oprávnění za běhu. [6]
- Coroutines jsou komponenty počítačového programu, které generalizují podprogramy pro nepreventivní multitasking tím, že umožňují pozastavení a obnovení provádění. Coroutines jsou vhodné pro implementaci známých součástí programu, jako jsou kooperativní úkoly, výjimky, smyčky událostí, iterátory, nekonečné seznamy a kanály. [7]
- Knihovna Room poskytuje práci s SQLite, která umožňuje robustnější přístup k databázi a zároveň využívá plný výkon SQLite. [8]
- Gson je knihovna Java, kterou lze použít k převodu objektů Java do jejich reprezentace JSON. Lze jej také použít k převodu řetězce JSON na ekvivalentní objekt Java. Gson může pracovat s libovolnými objekty Java, včetně již existujících objektů, jejichž zdrojový kód nemáte. [9]
- Mp3agic je Java knihovna pro čtení souborů mp3 a čtení / manipulaci s tagy ID3 (ID3v1 a ID3v2.2 až ID3v2.4). [10]

2.2 Hlavní layout

Aplikace funguje na principu Single Activity (Jediné Aktivity), což znamená, že je vytvořené třída MainActivity a xml activity_main, se nachází prvek frame, ten přepíná mezi Fragmenty. Hlavní layout obsahuje bottom navigation view (spodní navigovací lišta), pomocí které lze přepínat mezi fargmenty a tím se lze přemísťovat mezi Music, Album, Playlist a Favorite fragment. Bottom navigation view používá k navigaci mezi layouty knihovnu Jetpack.



Obr. 5 Jetpack navigation, návrh, soubor main_navigation.xml

Je zapotřebí propojit main_navigation s jednotlivými fragmenty. Toho docílíme tak, že vytvoříme soubor main_menu.xml ve složce res/menu. Menu obsahuje čtyři itemy s vlastní ikonkou a textem. Je taky důležité, aby id itemu se schodoval s id fragmentu v main_navigation.xml.

Obr. 6 BottomNavigation, aktivity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/nav_music"
        android:icon="@drawable/ic_music"
        android:title="MUSIC" />

    <item
        android:id="@+id/nav_album"
        android:icon="@drawable/ic_album"
        android:title="ALBUM" />

    <item
        android:id="@+id/nav_playlist"
        android:icon="@drawable/ic_playlist"
        android:title="PLAYLIST" />

    <item
        android:id="@+id/nav_fav"
        android:icon="@drawable/ic_fav"
        android:title="FAVORITE" />
</menu>
```

Obr. 7 Ikonky pro navigační menu

Nakonec v MainActivity dokončíme nastavení bottomview navigation pomocí následujícího kódu.

```
28  class MainActivity : AppCompatActivity() {
29      private lateinit var navController: NavController
30      private lateinit var viewModel: MainViewModel
31      var musicList: ArrayList<Music>? = null
32
33      override fun onCreate(savedInstanceState: Bundle?) {
34          super.onCreate(savedInstanceState)
35          setContentView(R.layout.activity_main)
36          viewModel = ViewModelProvider( owner: this).get(MainViewModel::class.java)
37
38          navController = Navigation.findNavController( activity: this, R.id.nav_host_fragment)
39
40          bottom_navigation?.let {  it: BottomNavigationView
41              NavigationUI.setupWithNavController(it, navController)
42          }
43          askStoragePermissions()
44      }
```

Obr. 8 MainActivity.kt, nastavení bottomnavigation

2.2.1 Vyhledání hudby

První věc co je zapotřebí je vyhledat veškerou hudbu v telefonu. Poté ji uložíme jako ArrayList s objekty Music. Nejdříve však je zapotřebí zeptat se uživatele na oprávnění.

```
private fun askPermissions() {
    Dexter.withActivity( activity: this)
        .withPermissions(
                Manifest.permission.READ_EXTERNAL_STORAGE,
                Manifest.permission.WRITE_EXTERNAL_STORAGE
        )
        .withListener(object : MultiplePermissionsListener {
            @RequiresApi(Build.VERSION_CODES.Q)
            override fun onPermissionsChecked(report: MultiplePermissionsReport?) {
                if (report!!.areAllPermissionsGranted()) {
                    musicList = getAllAudio()
                    queue = ArrayList()
                }

                if (report.isAnyPermissionPermanentlyDenied)
                    Toast.makeText( context: this@MainActivity,  text: "Go" +
                            " to setting and grant permissions", Toast.LENGTH_LONG).show()
            }
            override fun onPermissionRationaleShouldBeShown(
                    permissions: MutableList<PermissionRequest>?,
                    token: PermissionToken?
            ) { token!!.continuePermissionRequest() }
        })
        .onSameThread()
        .check()
}
private fun getImage(path: String): ByteArray? {
    try {
        val mp3file = Mp3File(File(path))
        if (mp3file.hasId3v2Tag()) {
            val id3v2Tag = mp3file.id3v2Tag
            val imageData = id3v2Tag.albumImage
            if (imageData != null)
                return imageData
            return null
        }
    } catch (e: Exception) {
        Log.e( tag: "EXCEPTION", e.message.toString())
    }
    return null
}
fun setBottomBarVisibility(visible: Boolean) {
    bottom_navigation.visibility = if (visible) View.VISIBLE else View.GONE
}
```

Obr. 9 Získaní povolení uživatele

```
@RequiresApi(Build.VERSION_CODES.Q)
private fun getAllAudio(): ArrayList<Music> {
    val tempAudioList = ArrayList<Music>()
    val selection = MediaStore.Audio.Media.IS_MUSIC + " != 0"
    val uri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI
    val projection = arrayOf(
            MediaStore.Audio.Media.DATA,
            MediaStore.Audio.Media.DISPLAY_NAME,
            MediaStore.Audio.Media.ARTIST,
            MediaStore.Audio.Media.ALBUM,
            MediaStore.Audio.Media.DURATION,
            MediaStore.Audio.Media.SIZE,
            MediaStore.Audio.Media._ID,
            MediaStore.Audio.Media.ALBUM_ID,
    )
    val cursor = this.contentResolver.query(
            uri,
            projection,
            selection,
             selectionArgs: null,
             sortOrder: null
    )
    if (cursor != null) {
        while (cursor.moveToNext()) {
            val path = cursor.getString( columnIndex: 0)
            val title = cursor.getString( columnIndex: 1)
            val artist = cursor.getString( columnIndex: 2)
            val album = cursor.getString( columnIndex: 3)
            val duration = cursor.getString( columnIndex: 4)
            val size = cursor.getString( columnIndex: 5)
            val id = cursor.getString( columnIndex: 6)

            val file = File(path)
            if (file.exists()) {
                val art = getImage(path)
                val musicFiles = Music(path, title, artist, album, duration, size, id, art)
                tempAudioList.add(musicFiles)
            }
        }
        cursor.close()
    }
    tempAudioList.sortBy { it.title.toLowerCase() }
    return tempAudioList
}
}
```

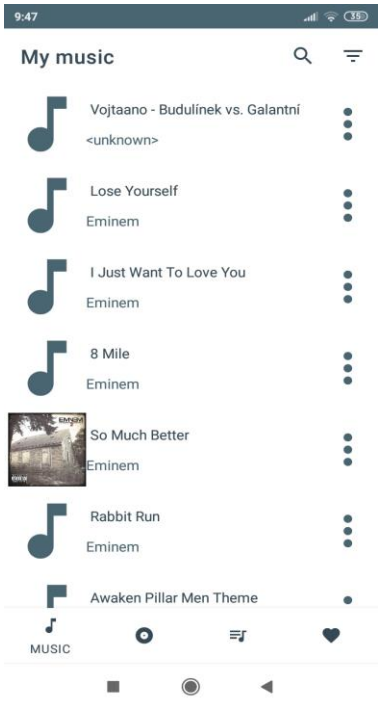
Obr 10 Vyhledání hudby a získání obrázku

2.2.2 Fragmenty

Fragment představuje opakovaně použitelnou část uživatelského rozhraní aplikace. Fragment definuje a spravuje vlastní rozvržení, má svůj vlastní životní cyklus a může zpracovávat své vlastní události. Fragmenty nemohou žít samy - musí být hostitelem aktivity nebo jiného fragmentu. Hierarchie zobrazení fragmentu se stává součástí hierarchie pohledu hostitele nebo se k ní připojuje. [11]

2.2.3 Music Fragment

V Music Fragmentu Budeme mít kompletní seznam hudby, ten se zobrazí pomocí UI prvku RecyclerView. Je také potřebné vytvořit UI item pro jednotlivou hudbu. Ten pojmenujeme music_item.xml. Každý item obsahuje ImageView (obrázek hudby a rozšiřovací menu) a TextView (Jeden pro název, druhý pro autora). Nedříve je zobrazena veškerá hudba. Po kliknutí na jednotlivou hudbu je uživatel přesměrován do Player Fragment, kde se hudba přehrává. Zároveň je z databáze získán seznam oblíbených písní a vytvořených playlistů. To je poté prováděno i v Album, Playlist a Favorite Fragmentech.

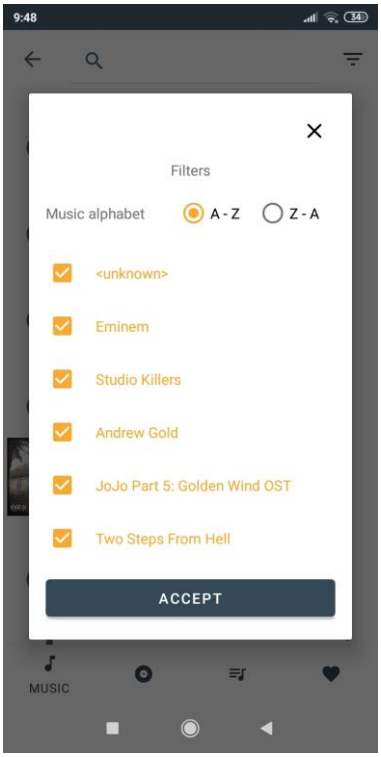


Obr. 11 Music Fragmen UI

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    (activity as MainActivity).setBottomBarVisibility(true)
    (activity as AppCompatActivity).setSupportActionBar(toolbar_music)
    (activity as AppCompatActivity).supportActionBar!!.setDisplayShowTitleEnabled(true)
    (activity as AppCompatActivity).supportActionBar!!.title = "My music"
    setHasOptionsMenu(true)

    recyclerView = view.findViewById(R.id.recycler_view_music)
    layoutNoMusic = view.findViewById(R.id.layout_no_music)
    navController = Navigation.findNavController(view)
    viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)

    viewModel.getFavorite.observe(requireActivity(), { it: List<FavoriteMusic>!
        favlist = it
        viewModel.getPlayList.observe(requireActivity(), { pl ->
            playlist = pl
            musicList = (activity as MainActivity).musicList
            if (musicList.isNotEmpty())
                for (music in musicList) {
                    music.fav = false
                    for (fav in favlist) if (fav.musicId == music.id) music.fav = true
                }
            if (!(activity as MainActivity).update) {
                createArtistsList()
                displayMusic(musicList)
            }
        })
    })
}

private fun createArtistsList() {
    artists = ArrayList()
    if (musicList.isNotEmpty()) {
        val artName = ArrayList<String>()
        for (i in musicList) {
            if (i.artist in artName) continue
            artName.add(i.artist)
            artists!!.add(Artists(i.artist))
        }
    }
}
```

Obr. 12 Vytváření listu

Music Fragment obsahuje toolbar, ten obsahuje v pravém rohu ikonku pro filtrování hudby dle umělce a ikonku pro vyhledávaní hudby pomocí názvu hudby nebo autora.



Obr. 13 Vyhledávaní hudby

Obr. 14 filtrování podle autora a abecedy

```kotlin
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
    super.onCreateOptionsMenu(menu, inflater)
    (activity as AppCompatActivity).menuInflater.inflate(R.menu.music_menu, menu)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    when (item.itemId) {
        R.id.menu_search -> searchMusic(item)
        R.id.menu_item_filters -> openFilterDialog()
    }
    return super.onOptionsItemSelected(item)
}
```

Obr. 15 Nastavení menu pro toolbar

```
private fun openFilterDialog() {
    val builder = AlertDialog.Builder(context)
    val mView: View = LayoutInflater.from(context).inflate(R.layout.dialog_music_filter,  root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()

    val recyclerView = mView.findViewById<RecyclerView>(R.id.recycler_view_filter)
    val checkAZ = mView.findViewById<RadioButton>(R.id.checkb_az)
    val checkZA = mView.findViewById<RadioButton>(R.id.checkb_za)
    val imgClose = mView.findViewById<ImageView>(R.id.img_close_dialog)
    val btnAccept = mView.findViewById<Button>(R.id.btn_dialog_filter_accept)

    if (aToZ) checkAZ.isChecked = true
    else checkZA.isChecked = true
    val artistCopy = ArrayList<Boolean>()
    for (artist in artists!!)
        artistCopy.add(artist.selected)
    recyclerView.layoutManager = LinearLayoutManager(context)
    val adapter = ArtistFilterAdapter(requireContext(), artists!!)
    recyclerView.adapter = adapter
    imgClose.setOnClickListener {  it: View!
        var x = 0
        for (i in artists!!) i.selected = artistCopy[x++]
        dialog.dismiss()
    }
    btnAccept.setOnClickListener {  it: View!
        val selectedArtists = ArrayList<String>()
        val newMusicList = ArrayList<Music>()
        aToZ = true
        for (artist in artists!!)
            if (artist.selected)
                selectedArtists.add(artist.name)
        for (i in musicList)
            if (i.artist in selectedArtists)
                newMusicList.add(i)
        newMusicList.sortBy { it.title.toLowerCase() }
        if (checkZA.isChecked) {
            aToZ = false
            newMusicList.reverse()
        }
        displayMusic(newMusicList)
        dialog.dismiss()
    }
}
```

Obr. 16 Filtrování hudby

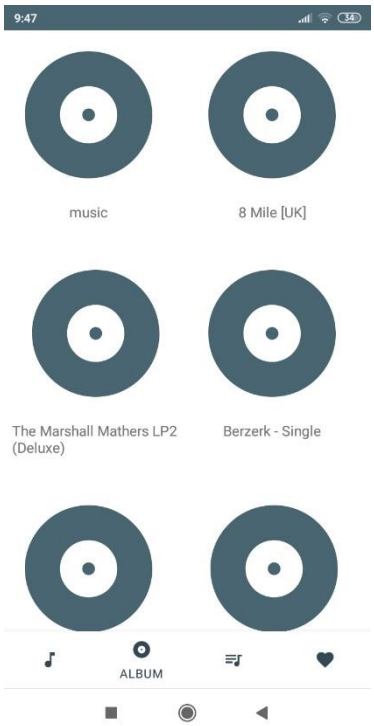
A nakonec vyhledání hudby. Pokud je shoda nalezena, zobrazí se list, jinak se zobrazí upozornění, že nebyla nalezena shoda.

```
private fun searchMusic(menuItem: MenuItem) {
    val searchView = menuItem.actionView as SearchView
    searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
        override fun onQueryTextSubmit(query: String?): Boolean {
            return false
        }

        override fun onQueryTextChange(newText: String?): Boolean {
            val newList = ArrayList<Music>()
            for (item in musicList!!)
                if (item.title.toLowerCase().contains(newText!!.toLowerCase())
                        || item.album.toLowerCase().contains(newText.toLowerCase())
                        || item.artist.toLowerCase().contains(newText.toLowerCase())) {
                    newList.add(item)
                }
            displayMusic(newList)
            return true
        }
    })
}
```

Obr. 17 Filtrování hudby

2.2.4 Album Fragment

Album Fragment obsahuje pouze recyclerview, jehož úkolem je zobrazit všechna možná alba, která jdou najít. Po kliknutí na jednotlivý item je uživatel přesměrován do Album Details Fragment. Album má každá hudba uložená jako id tag. Album Fragment má dost podobný kód jako Music Fragment.



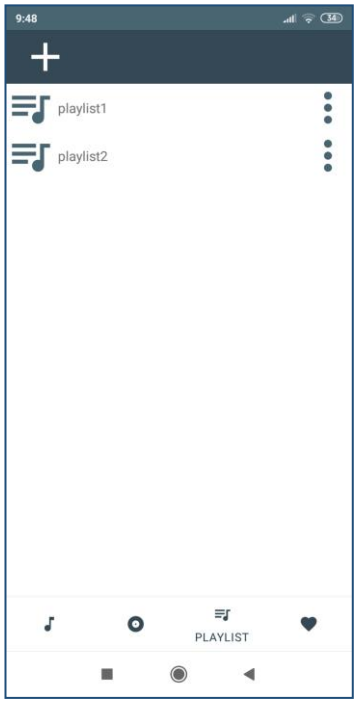
Obr. 18 Album Fragment, fragment_album.xml

```
private fun createAlbumList(): ArrayList<String> {
    val list = (activity as MainActivity).musicList
    val albumList = ArrayList<String>()
    for (music in list) {
        if (music.album in albumList) continue
        albumList.add(music.album)
    }
    return albumList
}
```

Obr. 19 Vytvoření album listu

2.2.5 PlayList Fragment

PlayList Fragment obsahuje recyclerview, jehož úkolem je zobrazit všechny playlisty, které byly uživatelem vytvořeny. Ty získává z SQLite databáze. Playlist je zobrazen pomocí itemu playlist_item.xml. Dále obsahuje toolbar s obrázkem. Ten slouží k vytvoření nového playlistu. Item obsahuje ImageView (jeden jako ikonka a druhý pro rozšiřovací menu) a TextView (název playlistu).



Obr. 20 Playlist Fragment, fragment_playlist.xml

Nastavení Playlist Fragmentu je dost podobné jako nastavení Music a Album Fragmentu

```
    img_add_white.setOnClickListener {  it: View!
        val builder = AlertDialog.Builder(context)
        val mView: View = LayoutInflater.from(context).inflate(R.layout.dialog_create_playlist,  root: null)
        builder.setView(mView)
        val dialog = builder.create()
        dialog.show()

        mView.btn_add.setOnClickListener {  it: View!
            val newName = mView.edt_txt_new_playlist.text.toString().trimEnd().trimStart()
            if (newName.isNotEmpty()) {
                val musics = ArrayList<Music>()
                val playList = PlayList( id: 0, newName, musics)
                viewModel.addPlaylist(playList)
                Toast.makeText(requireContext(),  text: "DONE", Toast.LENGTH_SHORT).show()
                dialog.dismiss()
            } else
                Toast.makeText(requireContext(),  text: "Enter playlist name!", Toast.LENGTH_SHORT).show()
        }

        mView.btn_cancel.setOnClickListener { dialog.dismiss() }
    }
}
```

Obr. 21 Přidání nového playlistu

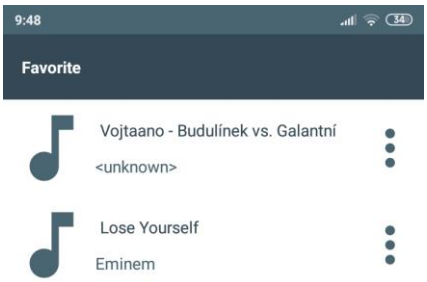
Poslední co je potřeba udělat, je nastavit možnost odstranit a přejmenovat playlist.

```
private fun delete(position: Int, adapter: PlaylistAdapter) {
    val builder = AlertDialog.Builder(context)
    val mView: View = LayoutInflater.from(context).inflate(R.layout.dialog_delete, root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()
    dialog.window!!.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

    mView.txt_delete_title.text = "Are you sure you want to delete this playlist"
    mView.btn_delete.setOnClickListener {  it: View!
        val playList = playlist[position]
        viewModel.removePlaylist(playList)
        adapter.notifyItemRemoved(position)
        dialog.dismiss()
    }

    mView.btn_cancel.setOnClickListener {  it: View!
        dialog.dismiss()
    }
}

private fun rename(position: Int, adapter: PlaylistAdapter) {
    val builder = AlertDialog.Builder(context)
    val mView: View = LayoutInflater.from(context).inflate(R.layout.dialog_rename, root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()
    dialog.window!!.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

    mView.btn_rename.setOnClickListener {  it: View!
        val playList = playlist[position]
        playList.playListName = mView.edit_txt_new_title.text.toString()
        viewModel.update(playList)
        adapter.notifyItemChanged(position)
        dialog.dismiss()
    }

    mView.btn_cancel.setOnClickListener {  it: View!
        dialog.dismiss()
    }
}
```

Obr. 22 Odstranění a přejmenování playlistů

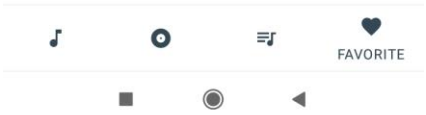
2.2.6 Favorite Fragment

Podobně jak Music Fragment, Favorit Fragment vypadá stejně, rozdíl je v tom, že zobrazuje pouze písně, které byly přidány uživatelem do oblíbených.



Obr. 23 Playlist Fragment, fragment_playlist.xml

```
class FavoriteFragment : Fragment() {
    private lateinit var navController: NavController
    private lateinit var viewModel: MainViewModel
    private lateinit var playlist: List<PlayListMusic>
    private lateinit var favlist: List<FavoriteMusic>

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.fragment_favorite, container, attachToRoot: false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        (activity as MainActivity).setBottomBarVisibility(true)
        navController = Navigation.findNavController(view)
        viewModel = ViewModelProvider( owner: this).get(MainViewModel::class.java)
        viewModel.getAll.observe(requireActivity(), Observer { it: List<PlayListMusic>!
            playlist = it
            viewModel.getFav.observe(requireActivity(), Observer { it: List<FavoriteMusic>!
                favlist = it
                if (favlist.isNotEmpty()) {
                    val favList = ArrayList<Music>()
                    for (music in (activity as MainActivity).musicList!!) {
                        music.fav = false
                        for (fav in favlist) {
                            if (fav.musicId == music.id) {
                                music.fav = true
                                favList.add(music)
                            }
                        }
                    }
                    if (!(activity as MainActivity).update) {
                        buildRecycleView(favList)
                    }
                    (activity as MainActivity).update = false

                } else recycler_view_fav.visibility = View.GONE
            })
        })
    }
```

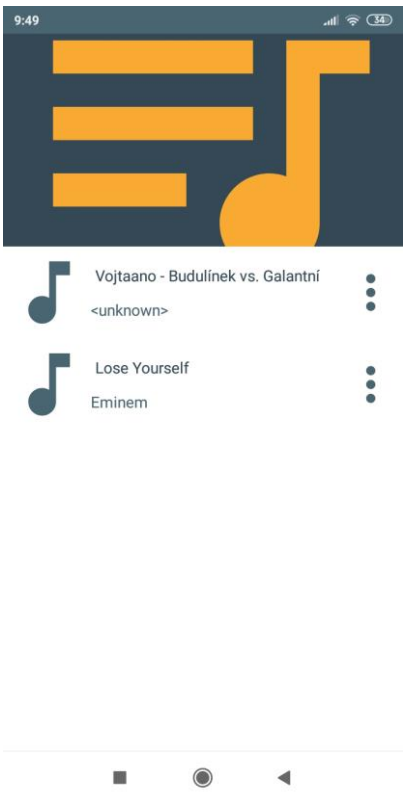


Obr. 24 Uložení všech písní, které jsou zapsané v databázi, do listu.

```
override fun onPause() {
    viewModel.getAll.removeObservers(requireActivity())
    viewModel.getFav.removeObservers(requireActivity())
    super.onPause()
}

private fun buildRecycleView(music: ArrayList<Music>?) {
    if (music != null && music.size != 0) {
        val adapter = MusicAdapter(requireContext(), music)
        recycler_view_fav!!.layoutManager = LinearLayoutManager(context)
        recycler_view_fav!!.adapter = adapter
        adapter.setOnItemClickListener(object : MusicAdapter.OnItemClickListener {
            override fun playSong(id: String) {
                val bundle = bundleOf( ...pairs: "id" to id, "fav" to "fav")
                navController.navigate(
                        R.id.action_favoriteFragment_to_playerFragment,
                        bundle
                )
            }

            @RequiresApi(Build.VERSION_CODES.M)
            override fun musicMenu(music: Music, view: View, position: Int) {
                val menuItem = MusicItemMenu(requireActivity(), requireContext(), view, playlist, favlist, viewModel, music, adapter, position)
                menuItem.createMenu()
            }
        })
    }
}
```

Obr. 25 Sestavení RecyclerView

2.2.7 PlayList Details Fragment



PlayList Details Fragment obsahuje všechny písně, které byly přidány do konkrétního playlistu. Layout obsahuje pouze recyclerview a collapsing toolbar. PlayList pro recyclerview používá music_item.xml. Po kliknutí na item je hudba přehrána.

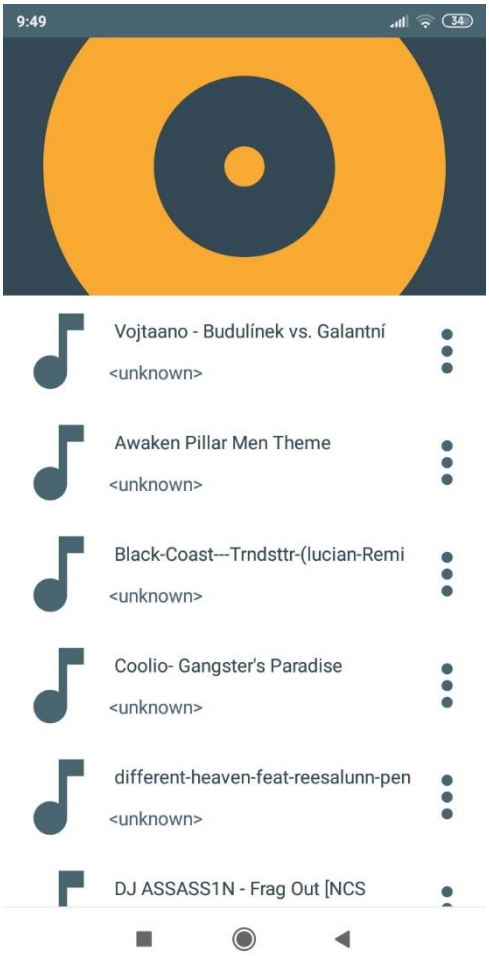
Obr. 26 Playlist Details Fragment, fragment_playlist_details.xml

Obr. 27 Uložení všech písní, které jsou zapsané v playlistu, do listu.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    (activity as MainActivity).setBottomBarVisibility(false)
    navController = Navigation.findNavController(view)
    recyclerView = view.findViewById(R.id.recycler_view_playlist_songs)
    viewModel = ViewModelProvider( owner: this).get(MainViewModel::class.java)
    viewModel.getFavorite.observe(requireActivity(), Observer { it: List<FavoriteMusic>!
        favlist = it
        viewModel.getPlayList.observe(requireActivity(), Observer { pl ->
            playlist = pl[position]
            playlists = pl
            if (playlist.musics.isNotEmpty()) {
                recyclerView.visibility = View.VISIBLE
                for (music in playlist.musics) {
                    music.fav = false
                    for (fav in favlist) if (fav.musicId == music.id) music.fav = true
                }
            } else recyclerView.visibility = View.GONE
            displayPlaylistSong(playlist.musics)
        })
    })

}
```

2.2.8 Album Details Fragment

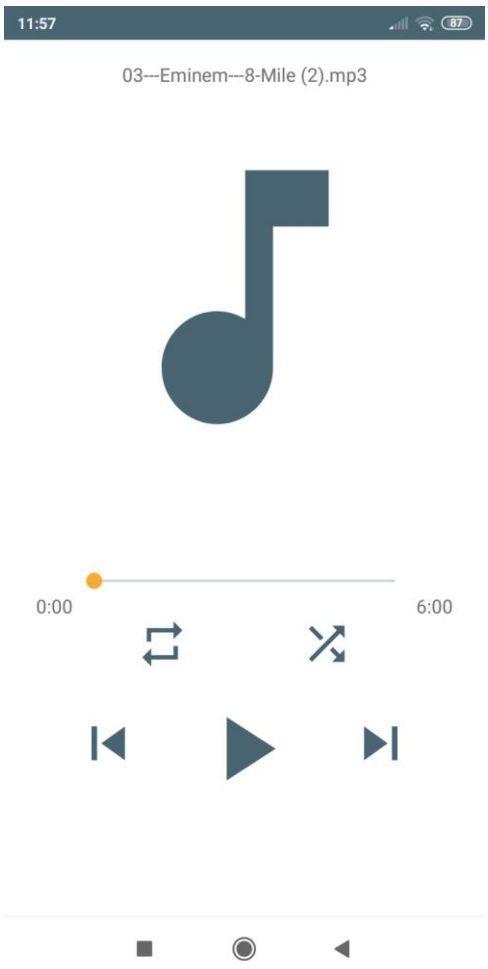
Album Details Fragment je doslova stejný jako PlayList Details Fragment. Jediný rozdíl je ten, že třídí hudbu dle vybraného alba.



Obr. 28 Album Details Fragment, fragment_album_details.xml

2.2.9 Player Fragment

Player Fragment je fragment, ve kterém se bude hudba přehrávat. Layout obsahuje imageview pro obrázek hudby, textview pro název skladby, progressbar, který zobrazuje progres hudby a celkem pět tlačítek. Tři hlavní tlačítka slouží pro přepínání mezi písněmi a pozastavení / spuštění. Dvě vedlejší tlačítka (nad hlavními tlačítky) slouží pro zapnutí opakování jedné skladby a zapnutí náhodného zamíchaní skladeb.



Obr. 29 Player Fragment, fragment_player.xml

Nejdříve je zapotřebí rozhodnout, jaká hudba se má přehrávat.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    id = requireArguments().getString( key: "id")!!
    album = requireArguments().getString( key: "album")
    playListPosition = requireArguments().getString( key: "playlist")
    fav = requireArguments().getString( key: "fav")
}
```

Obr. 30 Základní nastavení

Poté se spustí funkce initPlayer. Tato funkce se stará o přehrávání hudby a o progressbar.

```
private fun initPlayer() {
    job = Job()
    if (mediaPlayer != null) {
        mediaPlayer!!.stop()
        mediaPlayer!!.release()
    }

    mediaPlayer = MediaPlayer.create(requireContext(), Uri.parse(getMusic().path))
    mediaPlayer!!.start()

    setUI()

    mediaPlayer!!.setOnCompletionListener {  it: MediaPlayer!
        if (!repeat) {
            if (position < musicList.size - 1) position++
            else position = 0
        }
        initPlayer()
    }

    seekBar.setOnSeekBarChangeListener(object : SeekBar.OnSeekBarChangeListener {
        override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser: Boolean) {
            if (mediaPlayer != null && fromUser) {
                mediaPlayer!!.seekTo(progress)
                seekBar!!.progress = progress
            }
        }

        override fun onStartTrackingTouch(seekBar: SeekBar?) {

        }

        override fun onStopTrackingTouch(seekBar: SeekBar?) {

        }
    })

    musicProgress()
}
```

Obr. 31 Funkce pro přehrávaní hudby

```
private fun albumMusic(): ArrayList<Music> {
    val albumMusicList = ArrayList<Music>()
    for (music in (activity as MainActivity).musicList!!)
        if (music.album == album)
            albumMusicList.add(music)
    return albumMusicList
}

private fun playListMusic(): ArrayList<Music> {
    val playListMusicList = ArrayList<Music>()
    for (music in playlist[playListPosition!!.toInt()].musics)
        playListMusicList.add(music)
    return playListMusicList
}

private fun favListMusic(): ArrayList<Music> {
    val favListMusic = ArrayList<Music>()
    for (music in (activity as MainActivity).musicList!!)
        if (music.fav)
            favListMusic.add(music)
    return favListMusic
}

private fun setUI() {
    if (getMusic().art != null) {
        val bitmap = BitmapFactory.decodeByteArray(getMusic().art,  offset: 0, getMusic().art!!.size)
        img_music_photo.setImageBitmap(bitmap)
    } else img_music_photo.setImageDrawable(
        ContextCompat.getDrawable(
            requireContext(),
            R.drawable.ic_music
        )
    )
    createTimeLabel(mediaPlayer!!.duration)
    seekBar.max = mediaPlayer!!.duration
    txt_total_time.text = createTimeLabel(mediaPlayer!!.duration)
    txt_song_title.text = getMusic().title
}
```

Obr. 32 Nastavení UI Layoutu a listu

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    (activity as MainActivity).setBottomBarVisibility(false)
    viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)
    viewModel.getAll.observe(requireActivity(), Observer { playlist = it })
    musicList =
        when {
            album != null -> albumMusic()
            playListPosition != null -> playListMusic()
            fav != null -> favListMusic()
            else -> (activity as MainActivity).musicList!!
        }
    for (i in musicList.indices) if (musicList[i].id == id) position = i
    initPlayer()
    val animation = AnimationUtils.loadAnimation(requireContext(), R.anim.img_onclick_animation)
    img_play.setOnClickListener {  it: View!
        img_play.startAnimation(animation)
        play()
    }
    img_next.setOnClickListener {  it: View!
        job.cancel()
        img_next.startAnimation(animation)
        img_music_photo.startAnimation(animation)
        if (position < musicList.size - 1) position++
        else position = 0
        initPlayer()
    }
    img_previous.setOnClickListener {  it: View!
        job.cancel()
        img_previous.startAnimation(animation)
        img_music_photo.startAnimation(animation)
        if (position <= 0) position = musicList.size - 1
        else position--
        initPlayer()
    }
    img_shuffle.setOnClickListener {  it: View!
        img_shuffle.startAnimation(animation)
        shuffle = !shuffle
        shuffledMusic = musicList
        shuffledMusic!!.shuffle()
    }
    img_repeat.setOnClickListener {  it: View!
        img_repeat.startAnimation(animation)
        repeat = !repeat
    }
```

Obr. 33 Nastavení tlačítek

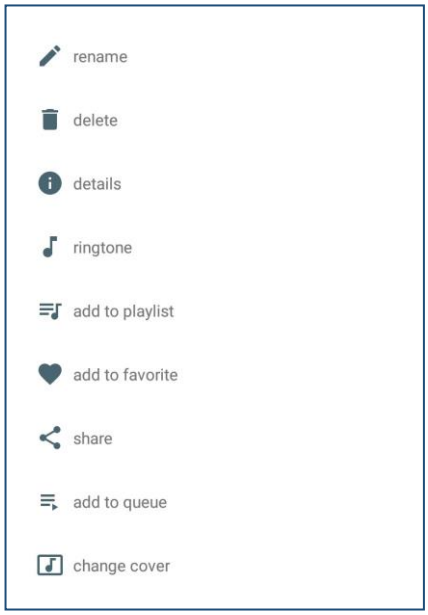
```
private fun musicProgress() = GlobalScope.launch {  this: CoroutineSc
    job = launch {  this: CoroutineScope
        while (mediaPlayer != null) {
            try {
                if (mediaPlayer!!.isPlaying) {
                    val msg = Message()
                    msg.what = mediaPlayer!!.currentPosition
                    handler.sendMessage(msg)
                    delay( timeMillis: 1000)
                }
            } catch (e: InterruptedException) {
                e.printStackTrace()
            } catch (e: IllegalStateException) {
                e.printStackTrace()
            }
        }
    }
}

@SuppressLint( ...value: "HandlerLeak")
private val handler = object : Handler() {
    override fun handleMessage(msg: Message) {
        val currentPosition: Int = msg.what
        if (seekBar != null && txt_current_time != null) {
            seekBar.progress = currentPosition
            val cTime = createTimeLabel(currentPosition)
            txt_current_time.text = cTime
        }
    }
}
```

Obr. 34 Zobrazení a funkce progressbaru

```
private fun play() {
    if (mediaPlayer != null && !mediaPlayer!!.isPlaying) {
        mediaPlayer!!.start()
        img_play.setImageResource(R.drawable.ic_pause)
    } else pause()
}

private fun pause() {
    if (mediaPlayer!!.isPlaying) {
        mediaPlayer!!.pause()
        img_play.setImageResource(R.drawable.ic_play)
    }
}

private fun getMusic(): Music {
    return if (!shuffle) musicList[position]
    else shuffledMusic!![position]
}
```

Obr. 44 Přehrát a pozastavit

2.3 Menu Hudby



Každá píseň lze přejmenovat, odstranit, přidat nebo odstranit z oblíbených a play listu. Též lze přidat nebo odstranit z fronty, změnit její obrázek, nebo nastavit jako vyzvánění. K tomu je vytvořena třída MusicItemMenu.

Obr. 45 Item menu

```
@Suppress( ...names: "DEPRECATION")
class MusicItemMenu(
        val activity: Activity,
        val playList: List<PlayList>,
        val favList: List<FavoriteMusic>,
        val viewModel: MainViewModel,
        val music: Music,
        val adapter: MusicAdapter,
        val position: Int,
        var album: ArrayList<Music>? = null
) : BottomSheetDialogFragment(){
    @RequiresApi(Build.VERSION_CODES.M)
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View? {
        val view = inflater.inflate(R.layout.dialog_music_menu, container,  attachToRoot: false)

        val btnRename = view.findViewById<LinearLayout>(R.id.layout_rename)
        val btnDelete = view.findViewById<LinearLayout>(R.id.layout_delete)
        val btnShowDetails = view.findViewById<LinearLayout>(R.id.layout_details)
        val btnSetRingtone = view.findViewById<LinearLayout>(R.id.layout_ringtone)
        val btnAddToPlayList = view.findViewById<LinearLayout>(R.id.layout_playlist)
        val btnAddToFav = view.findViewById<LinearLayout>(R.id.layout_fav)
        val btnShare = view.findViewById<LinearLayout>(R.id.layout_share)
        val btnQueue = view.findViewById<LinearLayout>(R.id.layout_queue)
        val btnCover = view.findViewById<LinearLayout>(R.id.layout_change_cover)
        val txtFav = view.findViewById<TextView>(R.id.txt_fav)
        val txtQueue = view.findViewById<TextView>(R.id.txt_queue)

        if (music.fav) txtFav.text = "remove from favorite"
        if (music.id in (activity as MainActivity).queue) txtQueue.text = "remove from queue"

        btnRename.setOnClickListener { rename() }
        btnDelete.setOnClickListener { delete() }
        btnShowDetails.setOnClickListener { showDetails() }
        btnSetRingtone.setOnClickListener { setRingtone() }
        btnAddToPlayList.setOnClickListener { addToPlayList() }
        btnAddToFav.setOnClickListener { addToFav() }
        btnShare.setOnClickListener { share() }
        btnQueue.setOnClickListener { addQueue() }
        btnCover.setOnClickListener { changeCover() }

        return view
    }
```

Obr. 46 Vytvoření MusicMenuItem

```
private fun rename() {
    val builder = AlertDialog.Builder(context)
    val mView: View = LayoutInflater.from(requireContext()).inflate(R.layout.dialog_rename,  root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()

    mView.btn_rename.setOnClickListener {  it: View!
        val newName = mView.edit_txt_new_title.text.toString()
        val path = ArrayList(music.path.split( ...delimiters: '/'))
        path[path.size - 1] = newName
        var newPath = ""
        for (i in 0 until path.size - 1) newPath += "${path[i]}/"

        newPath += "$newName.mp3"

        val file = File(music.path)
        if (file.renameTo(File(newPath))) {
            updateMediaStore(newPath, newName)
            music.title = newName
            music.path = newPath
            adapter.notifyItemChanged(position)
            Toast.makeText(requireContext(),  text: "DONE", Toast.LENGTH_LONG).show()
        }
        else
            Toast.makeText(requireContext(),  text: "ERROR", Toast.LENGTH_LONG).show()
        dialog.dismiss()
    }

    mView.btn_cancel.setOnClickListener { dialog.dismiss() }
}
```

Obr. 47 Přejmenování hudby

```
private fun delete() {
    val builder = AlertDialog.Builder(requireContext())
    val mView: View = LayoutInflater.from(requireContext()).inflate(R.layout.dialog_delete,  root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()

    mView.btn_delete.setOnClickListener {  it: View!
        val file = File(music.path)
        val deleted = file.delete()
        if (deleted) {
            (activity as MainActivity).update = true
            if (album != null) album!!.remove(music)
            (activity as MainActivity).musicList.remove(music)

            val pos = checkIfInFav()
            if (pos != -1) viewModel.removeFavorite(FavoriteMusic(pos, music.id))

            for (pl in playList) {
                for (song in pl.musics) {
                    if (song.id == music.id) {
                        pl.musics.remove(music)
                        viewModel.updatePlaylist(pl)
                        continue
                    }
                }
            }

            adapter.notifyItemRemoved(position)
            deleteMediaStore()
            Toast.makeText(requireContext(),  text: "DONE", Toast.LENGTH_SHORT).show()
        } else
            Toast.makeText(requireContext(),  text: "ERROR", Toast.LENGTH_SHORT).show()
        dialog.dismiss()
    }

    mView.btn_cancel.setOnClickListener {  it: View!
        dialog.dismiss()
    }
}
```

Obr. 48 Odstranění hudby

```
private fun showDetails() {
    val builder = AlertDialog.Builder(requireContext())
    val mView: View = LayoutInflater.from(requireContext()).inflate(R.layout.dialog_details,  root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()

    val txtName = mView.findViewById<TextView>(R.id.txt_song_name)
    val txtAlbum = mView.findViewById<TextView>(R.id.txt_album)
    val txtDuration = mView.findViewById<TextView>(R.id.txt_duration)
    val txtSize = mView.findViewById<TextView>(R.id.txt_size)

    txtName.text = music.title
    txtAlbum.text = music.album
    txtDuration.text = createTimeLabel(music.duration.toInt())
    txtSize.text = """${"%.1f".format( ...args: music.size.toFloat() / 1_000_000f)} MB"""
}
```

Obr. 49 Ukaž detaily

```
private fun addToPlayList() {
    val builder = AlertDialog.Builder(context)
    val mView: View = LayoutInflater.from(context).inflate(R.layout.dialog_add_to_playlist,  root: null)
    builder.setView(mView)
    val dialog = builder.create()
    dialog.show()

    val recyclerView = mView.findViewById<RecyclerView>(R.id.recycler_view_playlist)
    val txtNothing = mView.findViewById<TextView>(R.id.txt_no_playlist)
    val imgClose = mView.findViewById<ImageView>(R.id.img_close_dialog)

    if (playList.isNotEmpty()) {
        recyclerView.visibility = View.VISIBLE
        txtNothing.visibility = View.INVISIBLE
        recyclerView.LayoutManager = LinearLayoutManager(context)
        val ad = AddToPlayListAdapter(requireContext(), playList, music)
        recyclerView.adapter = ad
        ad.setOnItemClickListener(object : AddToPlayListAdapter.OnItemClickListener {
            override fun addToPlayList(data: PlayList) {
                var found = false
                for (i in data.musics)
                    if (i.id == music.id) {
                        found = true
                        break
                    }
                (activity as MainActivity).update = true
                if (found) data.musics.remove(music)
                else data.musics.add(music)
                viewModel.updatePlaylist(data)
                dialog.dismiss()
            }
        })
    } else {
        recyclerView.visibility = View.INVISIBLE
        txtNothing.visibility = View.VISIBLE
    }

    imgClose.setOnClickListener { dialog.dismiss() }
}
```

Obr. 50 Odstranění hudby

```
private fun addQueue() {
    if (music.id in (activity as MainActivity).queue)
        (activity as MainActivity).queue.remove(music.id)
    else (activity as MainActivity).queue.add(music.id)
    dismiss()
}
```

Obr. 51 Přidat do fronty

```
@RequiresApi(Build.VERSION_CODES.M)
private fun setRingtone() {
    try {
        if (!Settings.System.canWrite(requireContext())) {
            val intent = Intent(Settings.ACTION_MANAGE_WRITE_SETTINGS)
            intent.data = Uri.parse( uriString: "package:" + requireContext().packageName)
            requireContext().startActivity(intent)
        } else {
            val values = ContentValues()
            values.put(MediaStore.MediaColumns.DATA, music.path)
            values.put(MediaStore.MediaColumns.TITLE, music.title)
            values.put(MediaStore.MediaColumns.MIME_TYPE, "audio/mp3")
            values.put(MediaStore.MediaColumns.SIZE, music.size)
            values.put(MediaStore.Audio.Media.ARTIST, "Hudebni_Prehravac")
            values.put(MediaStore.Audio.Media.IS_RINGTONE, true)

            val uri = MediaStore.Audio.Media.getContentUriForPath(music.path)
            requireContext().contentResolver.delete(uri!!, where: MediaStore.MediaColumns.DATA + "=\"" + music.path + "\"", selectionArgs: null)
            val newUri: Uri = requireContext().contentResolver.insert(uri, values)!!

            RingtoneManager.setActualDefaultRingtoneUri(context, RingtoneManager.TYPE_RINGTONE, newUri)
            Toast.makeText(context, text: "DONE", Toast.LENGTH_LONG).show()
        }
    } catch (e: Exception) {
        Toast.makeText(context, text: "ERROR: ${e.message}", Toast.LENGTH_LONG).show()
    }
    dismiss()
}
```

Obr. 52 Nastavit jako vyzvánění

```
private fun addToFav() {
    if (favList.isNotEmpty()) {
        val position = checkIfInFav()
        if (position != -1) {
            (activity as MainActivity).update = true
            viewModel.removeFavorite(FavoriteMusic(position, music.id))
            music.fav = false
        } else {
            (activity as MainActivity).update = true
            viewModel.insertFavorite(FavoriteMusic( id: 0, music.id))
            music.fav = true
        }
    } else {
        (activity as MainActivity).update = true
        viewModel.insertFavorite(FavoriteMusic( id: 0, music.id))
        music.fav = true
    }
    dismiss()
    Toast.makeText(context, text: "DONE", Toast.LENGTH_LONG).show()
}

private fun checkIfInFav(): Int {
    for (fav in favList) if (fav.musicId == music.id) return fav.id
    return -1
}
```

Obr. 53 Přidat do oblíbených

```
private fun share() {
    val intent = Intent(Intent.ACTION_SEND)
    val file = File(music.path)
    val uri = FileProvider.getUriForFile(requireContext(),  authority: requireContext().applicationContext.packageName + ".provider", file)
    if (file.exists()) {
        intent.type = "application/mp3"
        intent.putExtra(Intent.EXTRA_STREAM, uri)
        intent.putExtra(Intent.EXTRA_SUBJECT,  value: "Share Song")
        intent.putExtra(Intent.EXTRA_TEXT,  value: "Share Song")
        intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
        requireContext().startActivity(Intent.createChooser(intent,  title: "Share Song"))
    }
}
```

Obr. 54 Sdílet

```
private fun changeCover() {
    Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI).also { intent ->
        intent.type = "image/*"
        val mimeTypes = arrayOf("image/jpeg", "image/png", "image/jpg")
        intent.putExtra(Intent.EXTRA_MIME_TYPES, mimeTypes)
        intent.flags = Intent.FLAG_GRANT_READ_URI_PERMISSION
        startActivityForResult(intent,  requestCode: 0)
    }
}
```

Obr. 55 Změnit obrázek 1/2

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    when (requestCode) {0 -> { try {
                if (resultCode == Activity.RESULT_OK)
                    data?.data?.let { uri ->
                        val imageStream: InputStream? = requireContext().contentResolver.openInputStream(uri)
                        val bitmap = BitmapFactory.decodeStream(imageStream)
                        val baos = ByteArrayOutputStream()
                        bitmap!!.compress(Bitmap.CompressFormat.JPEG,  quality: 100, baos)
                        val imageBytes = baos.toByteArray()
                        val newName = "Copy${(10000000..90000000).random()}"
                        val path = ArrayList(music.path.split( ...delimiters: '/'))
                        path[path.size - 1] = newName
                        var newPath = ""
                        for (i in 0 until path.size - 1) newPath += "${path[i]}/"
                        newPath += "$newName.mp3"
                        val file = File(music.path)
                        val mp3file = Mp3File(file)
                        val id3v2Tag: ID3v2
                        if (mp3file.hasId3v2Tag()) {
                            id3v2Tag = mp3file.id3v2Tag
                        } else {
                            id3v2Tag = ID3v24Tag()
                            mp3file.id3v2Tag = id3v2Tag
                        }
                        id3v2Tag.setAlbumImage(imageBytes,  mimeType: "image/jpeg")
                        mp3file.save(newPath)
                        file.delete()
                        deleteMediaStore()
                        val newFile = File(newPath)
                        newFile.renameTo(File(music.path))
                        val values = ContentValues()
                        values.put(MediaStore.Audio.Media.DATA, music.path)
                        MediaScannerConnection.scanFile(requireContext(), arrayOf(File(music.path).absolutePath),
                                mimeTypes: null) { _: String?, u: Uri ->
                            requireContext().contentResolver.update(u, values,  where: null,  selectionArgs: null)
                        }
                        music.art = imageBytes
                        adapter.notifyItemChanged(position)
                    }
                else Toast.makeText(requireContext(),  text: "You haven't picked Image", Toast.LENGTH_LONG).show()
            } catch (e: FileNotFoundException) {
                e.printStackTrace()
                Toast.makeText(requireContext(),  text: "File not found", Toast.LENGTH_LONG).show()
            } } } }
```

Obr. 55 Změnit obrázek 2/2

Když uživatel odstraní, nebo změní obrázek, je důležité pokaždé aktualizovat úložiště v mobilu. Pokud by tak neučinil, aplikace by zobrazovala nepřejmenovaný soubor, nebo by zobrazovala odstraněné soubory. [12]

```
private fun updateMediaStore(newPath: String, newName: String) {
    val values = ContentValues()
    values.put(MediaStore.Audio.Media.DATA, newPath)
    values.put(MediaStore.Audio.Media.DISPLAY_NAME, newName)
    val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, music.id.toLong())
    requireContext().contentResolver.update(uri, values,  where: null,  selectionArgs: null)
}

private fun deleteMediaStore() {
    val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, music.id.toLong())
    requireContext().contentResolver.delete(uri,  where: null,  selectionArgs: null)
}
```

Obr. 43 a 44 Aktualizace úložiště

ZÁVĚR

Výsledkem mojí ročníkové práce je základní aplikace pro přehrávání hudby. Projekt mi sice trval mnohem déle, kvůli problémům s android verzí a architekturou, a protože jsem zprvu přecenil svoje schopnosti a dovednosti. Nicméně se mi podařilo tyto problémy překonat a během vývoje aplikace jsem se naučil novým dovednostem v rámci programovacího jazyka Kotlin, vývojového prostředí Android Studio.

.

SEZNAM ZKRATEK

UI – User Interface
JSON – JavaScript Object Notation
MP3 – MPEG-2 Audio Layer III
XML – Extensible Markup Language

SEZNAM LITERATURY

1. Kotlin [online]. 2011 13. 3. 2021 v 07:21 [cit. 2021-03-20]. Dostupné z: https://cs.wikipedia.org/wiki/Kotlin_(programovac%C3%AD_jazyk)

2. JetBrain [online]. 2000, 20. 1. 2021 v 02:43 [cit. 2021-03-20]. Dostupné z: https://cs.wikipedia.org/wiki/JetBrains

3. XML [online]. 25. 3. 2021 v 21:32 [cit. 2021-03-20]. Dostupné z: https://en.wikipedia.org/wiki/XML

4. Gradle [online]. 5. 3. 2021 [cit. 2021-03-20]. Dostupné z: https://www.geeksforgeeks.org/android-build-gradle/

5. Jetpack Navigation [online]. 29. 3. 2021 [cit. 2021-03-20]. Dostupné z: https://developer.android.com/guide/navigation/

6. Dexter [online]. 2021 [cit. 2021-03-20]. Dostupné z: https://github.com/Karumi/Dexter

7. Coroutines [online]. 15. 3. 2021 [cit. 2021-03-20]. Dostupné z: https://en.wikipedia.org/wiki/Coroutine

8. Room [online]. 24. 3. 2021 [cit. 2021-03-20]. Dostupné z: https://developer.android.com/jetpack/androidx/releases/room

9. Gson [online]. 2021 [cit. 2021-03-20]. Dostupné z: https://github.com/google/gson

10. Mp3agic [online]. 2021 [cit. 2021-03-20]. Dostupné z: https://github.com/mpatric/mp3agic

11. Fragments [online]. 23. 11. 2021 [cit. 2021-03-20]. Dostupné z: https://developer.android.com/guide/fragments

12. MediaStore [online]. 17. 11. 2021 [cit. 2021-03-20]. Dostupné z: https://developer.android.com/reference/android/provider/MediaStore