

Колледж космического машиностроения и технологий

**КУРСОВОЙ ПРОЕКТ**

**По МДК.01.02 «Прикладное программирование»**

**Тема: «Разработка музыкального андроид-плеера»**

Выполнил студент

Константинович А.

Группа П1-18

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Принял преподаватель

Гусятинер Л.Б.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Подпись)

**Королёв 2021 г.**

Оглавление

[Введение 2](#_Toc69115482)

[1 Теоретическая часть 4](#_Toc69115483)

[1.1 Описание предметной области 4](#_Toc69115484)

[1.2 Описание существующих разработок 6](#_Toc69115485)

[2 Проектная часть 9](#_Toc69115486)

[2.1 Диаграмма прецедентов 9](#_Toc69115487)

[2.2 Выбор инструментов 9](#_Toc69115488)

[2.2.1 Язык программирования 9](#_Toc69115489)

[2.2.2 Среда разработки программного обеспечения 10](#_Toc69115490)

[2.2.3 База данных 10](#_Toc69115491)

[2.3 Проектирование сценария 11](#_Toc69115492)

[2.4 Диаграммы классов 12](#_Toc69115493)

[2.5 Описание главного модуля. 14](#_Toc69115494)

[2.6 Описание спецификаций к модулям 15](#_Toc69115495)

[2.7 Описание модулей 16](#_Toc69115496)

[2.8 Описание тестовых наборов модулей 19](#_Toc69115497)

[2.9 Описание применения средств отладки 20](#_Toc69115498)

[2.10 Анализ оптимальности использования памяти и быстродействия 21](#_Toc69115499)

[3 Эксплуатационная часть 23](#_Toc69115500)

[3.1 Руководство оператора 23](#_Toc69115501)

[3.1.1 Назначение программы 23](#_Toc69115502)

[3.1.2 Условия выполнения программы 23](#_Toc69115503)

[3.1.3 Выполнение программы 23](#_Toc69115504)

[3.1.4 Сообщение оператору 29](#_Toc69115505)

[3.2 To-Do лист 29](#_Toc69115506)

[Заключение 30](#_Toc69115507)

[Список литературы и интернет ресурсов 31](#_Toc69115508)

# Введение

Данный курсовой проект заключается в разработке приложения под операционную систему Android. Приложение представляет собой аудио проигрыватель, который воспроизводит аудиофайлы из памяти устройства. Разработка программы будет происходить на языке программирования Kotlin.

В первой части будут рассмотрены предметная область и существующие продукты по данной теме.

Во второй части будут описаны инструменты разработки программы и модули программы.

В третьей части описаны взаимодействие пользователя с программой и сообщения оператору.

В заключительной части будут приведены общие выводы по проекту.

# Теоретическая часть

## Описание предметной области

**Медиаплеер [3]** (также проигрыватель мультимедиа или мультимедиа-проигрыватель) — компьютерная программа, предназначенная для воспроизведения файлов мультимедиа-контента.

Одна разновидность медиаплееров предназначена для воспроизведения только аудио- или же видеофайлов, и которые называются, соответственно, — аудиоплеер и видеоплеер. В основном все видеоплееры делятся на несколько типов: Простые (для работы на слабых ПК), Универсальные и Специализированные (специфические и профессиональные программы для решения уникальных задач). Примерно по такому же принципу делятся и аудиоплееры. Разработчики таких плееров стремятся сделать их как можно более удобными для воспроизведения соответствующих форматов.

Другая разновидность программ-медиаплееров поддерживают как аудио так и видео (включая множество медиаконтейнеров) и называется мультимедиа-центры.

Большинство современных операционных систем по умолчанию со-держат в своём составе медиаплееры: например,Windows — Windows Media Player, Mac OS X — QuickTime Player (для воспроизведения видео в формате QuickTime) и iTunes (для некоторых других форматов), Linux — Amarok, Rhythmbox или иные (в зависимости от дистрибутива). Медиаплееры есть в телевизорах Smart TV и ресиверах цифрового телевидения.

Разрабатываемый проигрыватель поддерживает файлы в формате MP3. **MP3 [6]** — это разработанный командой MPEG формат файла для хранения аудиоинформации. MP3 является одним из самых распространённых и популярных форматов цифрового кодирования звуковой информации. Он широко используется в файлообменных сетях для оценочного скачивания музыкальных произведений. Формат может проигрываться практически во всех популярных операционных системах, на большинстве портативных аудиоплееров, а также поддерживается всеми современными моделями музыкальных центров и DVD-плееров.

Как и формат JPEG, MP3 использует спектральные отсечения, согласно психоакустической модели. Звуковой сигнал разбивается на равные по продолжительности отрезки, каждый из которых после обработки упаковывается в свой фрейм (кадр). Разложение в спектр требует непрерывности входного сигнала, в связи с этим для расчётов используется также предыдущий и следующий фрейм. В звуковом сигнале есть гармоники с меньшей амплитудой и гармоники, лежащие вблизи более интенсивных — такие гармоники отсекаются, так как среднестатистическое человеческое ухо не всегда сможет определить присутствие либо отсутствие таких гармоник. Такая особенность слуха называется эффектом маскировки. Также возможна замена двух и более близлежащих пиков одним усреднённым (что, как правило, и приводит к искажению звука). Критерий отсечения определяется требованием к выходному потоку. Поскольку весь спектр актуален, высокочастотные гармоники не отсекаются, как в JPEG, а только выборочно удаляются, чтобы уменьшить поток информации за счёт разрежения спектра. После спектральной «зачистки» применяются математические методы сжатия и упаковка во фреймы. Каждый фрейм может иметь несколько контейнеров, что позволяет хранить информацию о нескольких потоках (левый и правый канал либо центральный канал и разница каналов). Степень сжатия можно варьировать, в том числе в пределах одного фрейма. Интервал возможных значений битрейта составляет 8—320 кбит/c.

MP3-файл состоит из нескольких фрагментов (фреймов) MP3, которые, в свою очередь, состоят из заголовка и блока данных. Такая последовательность фрагментов называется элементарным потоком. Фрагменты не являются независимыми элементами («резервуар байт»), и поэтому не могут быть извлечены произвольно. Блок данных MP3-файла содержит сжатую аудиоинформацию в виде частот и амплитуд.

## Описание существующих разработок

На данный момент существует множество уже разработанных приложений на данную тематику. Практически все из них предоставляют похожий функционал:

* Поддержка многих аудиоформатов;
* 8-ми полосный эквалайзер с возможностью автоматической подстройки под жанр трека;
* Автоматическое определение кодировки данных в тегах;
* Поддержка обложек альбома;
* Возможность выборочного добавления файлов / папок в плейлист;
* Возможность быстрого добавления всей музыки с SD-карт;
* Возможность повтора плейлиста / повтора трека / проигрывания плей-листа без повтора;
* Последовательное проигрывание треков / проигрывание в случайном порядке;
* Управление из «шторки» (для Android v3.0 и новее) и с гарнитуры;
* Воспроизведение музыки из файловых менеджеров.

Примеры приложений:

* AIMP [7]

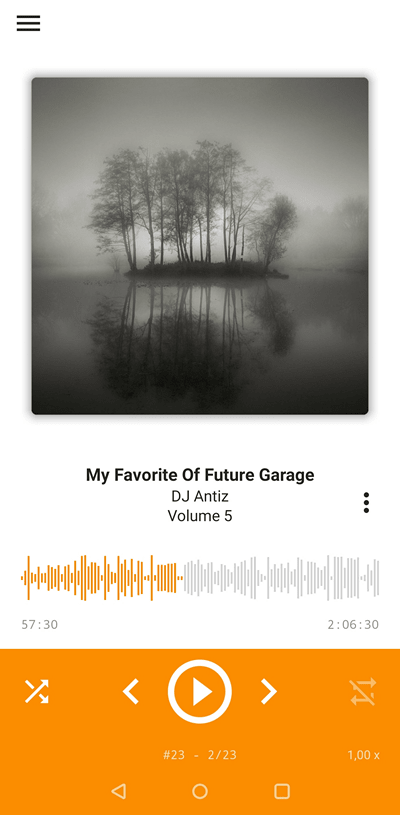


Рисунок 1. Аудиоплеер AIMP.

* Audify Music Player [8]



Рисунок 2. Аудиоплеер Audify Music Player.

* BlackPlayer [9]



Рисунок 3. Аудиоплеер BlackPlayer.

* jetAudio [10]

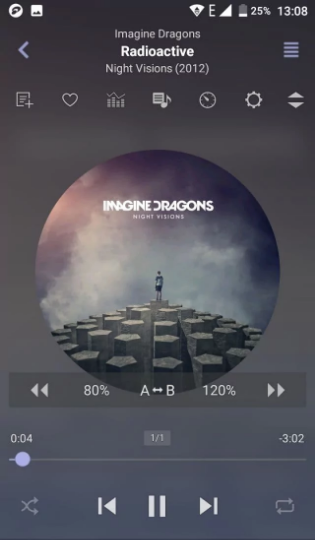


Рисунок 4. Аудиоплеер jetAudio.

# Проектная часть

## Диаграмма прецедентов

В данном разделе содержится диаграмма прецедентов для приложения, на которой показаны возможные функциональные отношения. (рис. 5)



Рисунок 5. Диаграмма прецедентов приложения.

## Выбор инструментов

### Язык программирования

В разработке приложений на Android используются в основном либо Java, либо Kotlin. Данный курсовой проект написан на Kotlin.

**Kotlin [1]** - язык программирования, разработанный компанией JetBrains, работающий на платформе Java. Он использует JDK, как и сама Java, но имеет другой синтаксис. Синтаксис языка использует элементы из Паскаля, TypeScript, Haxe, PL/SQL, F#, Go и Scala, C++, Java, C#, Rust и D. При объявлении переменных и параметров типы данных указываются после названия (разделитель - двоеточие). Точка с запятой как разделитель операторов так же необязательна. Кроме объектно-ориентированного подхода, Kotlin также поддерживает процедурный стиль с использованием функций. Как и в Си, C++ и D, точка входа в программу – функция main, принимающая массив параметров командной строки. Также поддерживается вывод типов.

В сравнении класса Java с эквивалентным классом Kotlin демонстрирует лаконичность кода Kotlin. Для той же операции, что выполняется в классе Java, класс Kotlin требует меньше кода. Одно из основных различий между Java и Kotlin заключается в том, что в последнем нет условий для проверяемых исключений (checked exception). Следовательно, нет необходимости отлавливать или объявлять какие-либо исключения. Несомненным преимуществом является полная совместимость с Java, в том числе и обратная. Все библиотеки для Java будут работать на Kotlin и наоборот. Также Kotlin позволяет разработчикам расширять класс новыми функциями с помощью функций расширения. Эти функции недоступны в Java. В отличие от Java, в Kotlin все типы по умолчанию являются не-nullable. Если разработчики попытаются присвоить или вернуть значение null в коде Kotlin, во время компиляции произойдет сбой.

Kotlin помог разработчикам писать программы с меньшим количеством кода. Помимо всего того, что есть в Java, он добавляют вещи из мира функционального программирования. Это значительно облегчает написание кода - делает его короче и выразительнее.

### Среда разработки программного обеспечения

**Android Studio [2]** — интегрированная среда разработки (IDE) для работы с платформой Android.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, — официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, macOS и GNU/Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал поддержку языка Kotlin, используемого в Android Studio, как официального языка программирования для платформы Android в дополнение к Java и С++.

### База данных

В разработке приложений на Android в основном в качестве базы данных используется **SQLite [11]**. Причины использования:

1. **Минимальные затраты ресурсов.** Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера: база данных SQLite представляет собой обычный файл. Когда БД не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.
2. **Оптимизация для одного пользователя.** С базой данных взаимодействует только наше приложение. Поэтому можно обойтись без идентификации с именем пользователя и паролем.
3. **Надежность и быстрота.** Базы данных SQLite поддерживают транзакции баз данных. Кроме того, операции чтения и записи данных реализуются на оптимизированном коде С.

## Проектирование сценария

В данном разделе приведен сценарий использования программы пользователем (Рисунок 6).



Рисунок 6. Сценарий использования приложения

При запуске приложения открывается экран управления текущим треком. По умолчанию текущего трека нет, поэтому при запуске плеера запустится первая песня из плейлиста. Также пользователь может выбрать песню из экрана управления плейлистом. Чтобы добавить новые песни следует перейти в экран добавления новых песен.

## Диаграммы классов

В данном разделе находятся диаграммы классов приложения. (Рисунки 7, 8, 9)

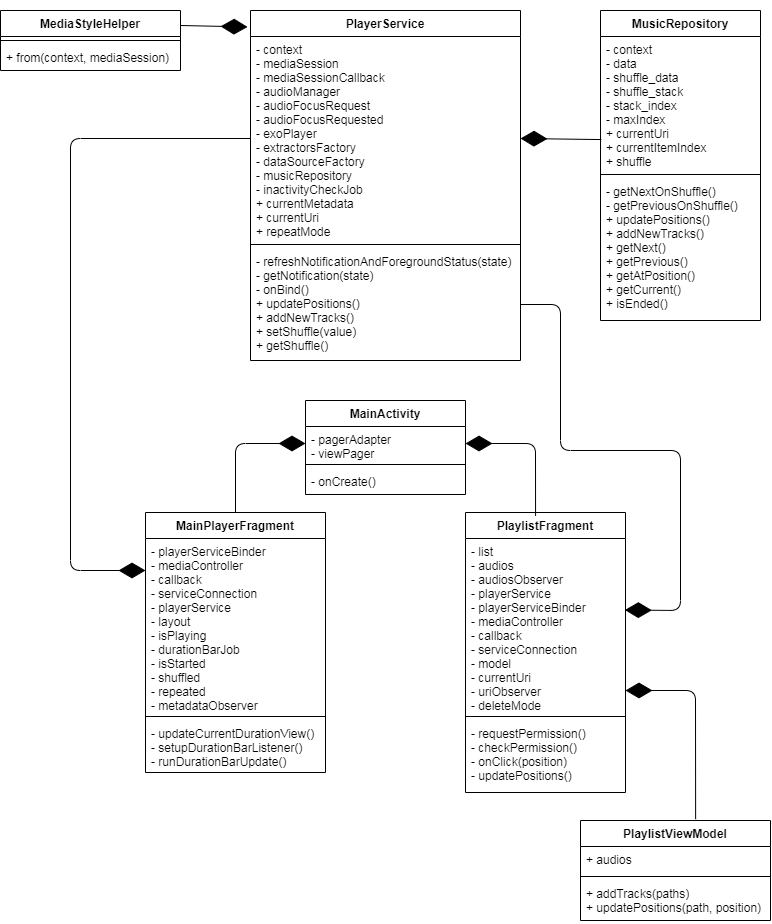


Рисунок 7. Диаграмма основных классов приложения

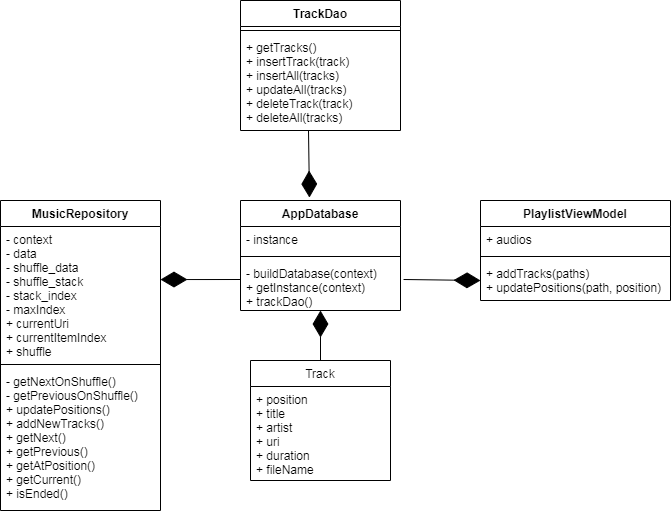
****

Рисунок 8. Диаграмма классов, связанных с БД.

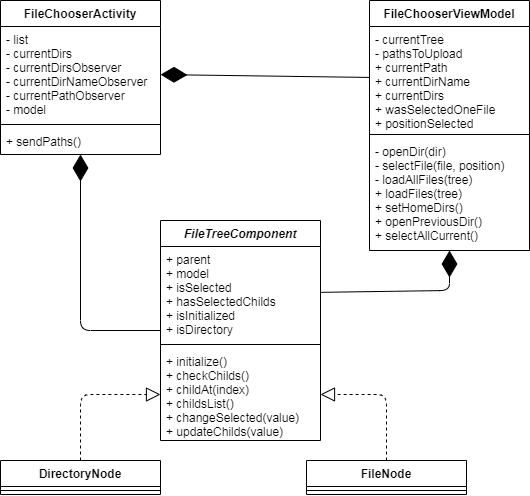


Рисунок 9. Диаграмма классов экрана добавления файлов.

## Описание главного модуля.

Главный модуль состоит из класса MainActivity. Он выполняется при запуске приложения. В данном модуле создаются основные экраны приложения и отображаются пользователю.

**Листинг 1. Главный модуль.**

package com.kmem.myplayer.ui.activities  
  
import android.content.Intent  
import android.os.Bundle  
import android.view.View  
import androidx.appcompat.app.AppCompatActivity  
import androidx.fragment.app.Fragment  
import androidx.fragment.app.FragmentManager  
import androidx.fragment.app.FragmentPagerAdapter  
import androidx.viewpager.widget.ViewPager  
import com.kmem.myplayer.ui.fragments.MainPlayerFragment  
import com.kmem.myplayer.ui.fragments.PlaylistFragment  
import com.kmem.myplayer.R  
import com.kmem.myplayer.service.PlayerService  
  
*/\*\*  
 \* Главная активность приложения. Она запускается при открытии приложения.  
 \* Отвечает за запуск остальных экранов приложения.  
 \*/*class MainActivity : AppCompatActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 val pagerAdapter = SectionsPagerAdapter(*supportFragmentManager*)  
 val viewPager = findViewById<View>(R.id.*pager*) as ViewPager  
 viewPager.*adapter* = pagerAdapter

// запуск сервиса  
 startService(Intent(*baseContext*, PlayerService::class.*java*))  
 }  
  
 private inner class SectionsPagerAdapter(fm: FragmentManager?) :  
 FragmentPagerAdapter(fm!!, *BEHAVIOR\_RESUME\_ONLY\_CURRENT\_FRAGMENT*) {  
 override fun getCount(): Int {  
 return 2  
 }  
  
 override fun getItem(position: Int): Fragment {  
 when (position) { // Создание отсальных экранов  
 0 -> return MainPlayerFragment()  
 1 -> return PlaylistFragment()  
 }  
 return MainPlayerFragment()  
 }  
  
 override fun getPageTitle(position: Int): CharSequence? {  
 when (position) {  
 0 -> return "main"  
 1 -> return "Playlist"  
 }  
 return null  
 }  
 }  
}

## Описание спецификаций к модулям

Всего разработано 5 модулей:

* Экран управления плеером и текущим треком (MainPlayerFragment)
* Экран управления плейлистом (PlaylistFragment)
* Экран добавления файлов в плейлист (FileChooserActivity)
* Сервис для проигрывания музыки (PlayerService)
* Репозиторий получения данных из БД для сервиса (MusicRepository)

Взаимодействие между модулями можно определить по диаграммам классов из раздела 2.4.

## Описание модулей

**Экран управления плеером и текущим треком** – на данном экране отображается информация о текущей проигрываемой песне, а также находятся кнопки для управления плеером (Приложение 1). Сведения о текущем треке, которые находятся на экране: обложка, автор, название, текущая длительность. Для управления плеером в нижней части экрана находятся следующие кнопки:

* Управление перемешиванием
* Управление повторением
* Предыдущая песня
* Следующая песня
* Воспроизведение/пауза текущей песни

**Экран управления плейлистом** – содержит информацию о проигрываемом списке песен. Позволяет добавлять и удалять песни. При нажатии на элемент списка (песню) запускает проигрывание песни. В нижней части экрана содержится панель управления с двумя кнопками: кнопка добавления и кнопка удаления. При нажатии на «плюс» (кнопка добавления) открывается новый экран для выбора песен из памяти устройства. При нажатии на «минус» (кнопка удаления) возле каждой песни появится флажок для выбора данной песни (Приложение 2).

**Экран добавления файлов** – внутренний обозреватель файлов. Позволяет проходить по папкам и выбирать файлы для добавления. Возле каждого файла и директории находится флажок для выбора. В верхней части экрана находится панель управления с кнопками для перехода к домашней и предыдущей директориям. Для загрузки файлов в нижней части экрана есть кнопка для завершения выбора файлов. (Приложение 3)

Рассмотрим функцию загрузки выбранных файлов, а также её блок-схему:

**Листинг 2. Функция загрузки выбранных файлов.**

// Загружает файлы, выбранные пользоваетелем

fun loadFiles(tree: FileChooserActivity.FileTreeComponent? = null) : ArrayList<String> {

var tmpTree = tree

if(tree == null) {

pathsToUpload.clear() // Список с выбранными путями

// Получаем главное дерево

while(currentTree?.parent != null)

currentTree = currentTree?.parent

tmpTree = currentTree

}

// Проходим по всем потомкам

for (child in tmpTree?.childsList()!!) {

if (child.isSelected) {

if (child.isDirectory)

loadAllFiles(child)

else

pathsToUpload.add(child.model?.file?.absolutePath!!)

} else if (child.hasSelectedChilds) {

loadFiles(child)

}

}

return pathsToUpload

}

// Загружает все файлы из директории

private fun loadAllFiles(tree: FileChooserActivity.FileTreeComponent?) {

for (child in tree?.childsList()!!) {

if (child.isDirectory)

loadAllFiles(child)

else

pathsToUpload.add(child.model?.file?.absolutePath!!)

}

}

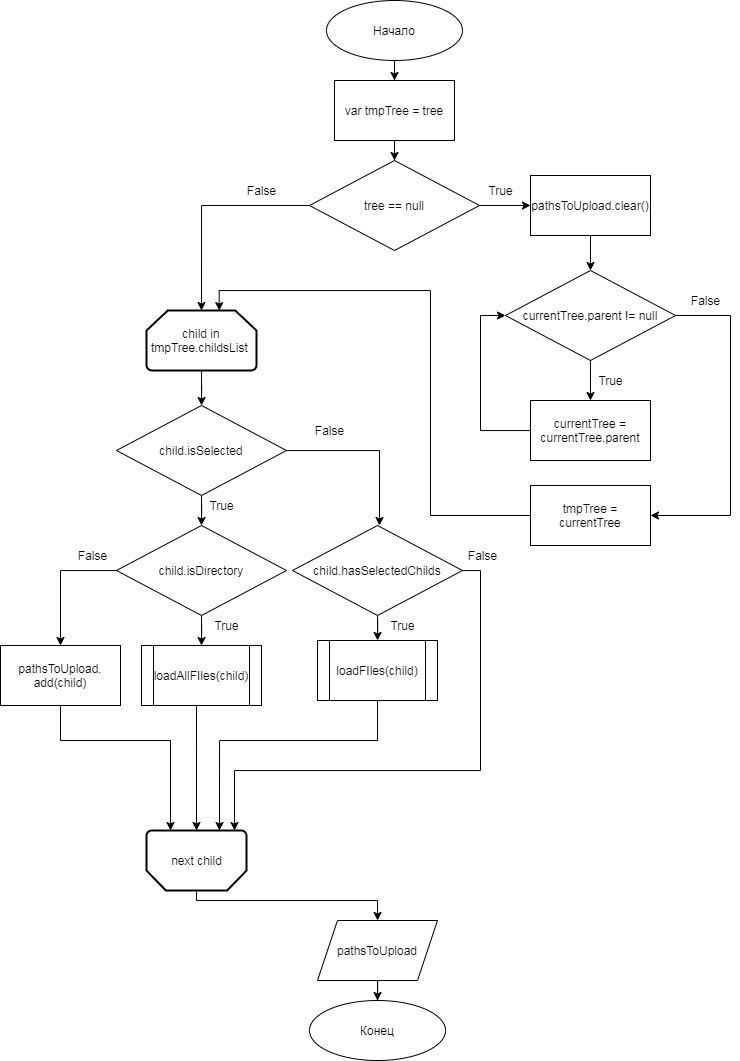


Рисунок 10. Блок схема функции loadFIles.

**Сервис для проигрывания музыки –** модуль, который непосредственно играет музыку и управляет плеером. Остальные модули получают от него информацию и через него взаимодействуют с плеером. Сервис также обрабатывает события от системы (например, поставить музыку на паузу при звонке). (Приложение 4)

**Репозиторий получения данных из БД для сервиса –** модуль, который отвечает за получение песен из БД. Сервис обращается к репозиторию для получения текущей/следующей/предыдущей песни. Также сервис может настраивать репозиторий. (например, перемешать песни). (Приложение 5)

## Описание тестовых наборов модулей

В некоторых модулях производится проверка данных на корректность, чтобы сообщить пользователю о проблеме и предотвратить аварийное завершение программы.

**Тест 1. Проверка на пустой плейлист.**

Если запустить плеер с пустым плейлистом, то в нижней части экрана появится соответствующее сообщение. (Рисунок 11)

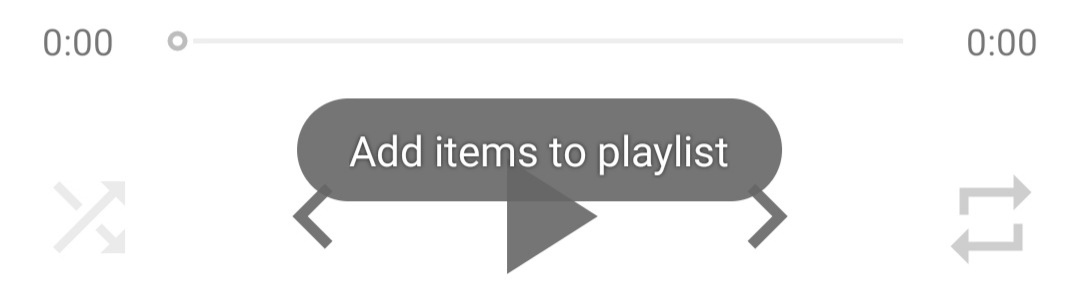


Рисунок 11. Ошибка при пустой плейлисте

**Тест 2. Проверка разрешения доступа к файлам.**

При запуске экрана выбора файлов, если пользователь до этого не давал разрешение доступа к файлам, приложение запросит соответствующее разрешение. (Рисунок 12)

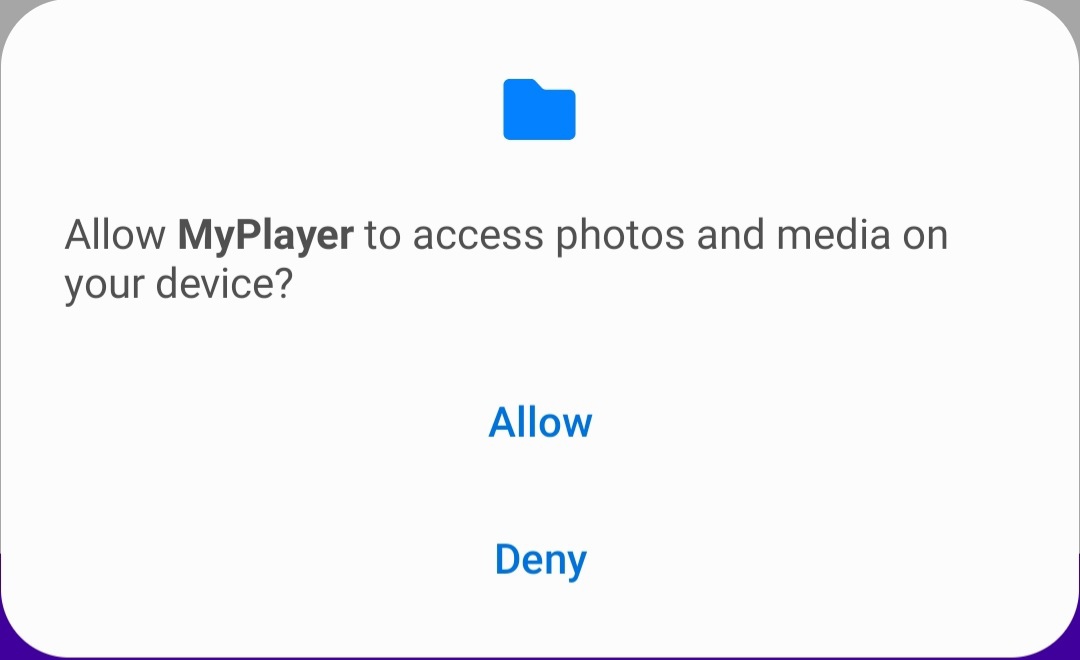


Рисунок 12. Запрос разрешения к файлам

Если пользователь отказал в доступе, то появится сообщение о необходимости разрешения для добавления файлов в плейлист. (Рисунок 13)

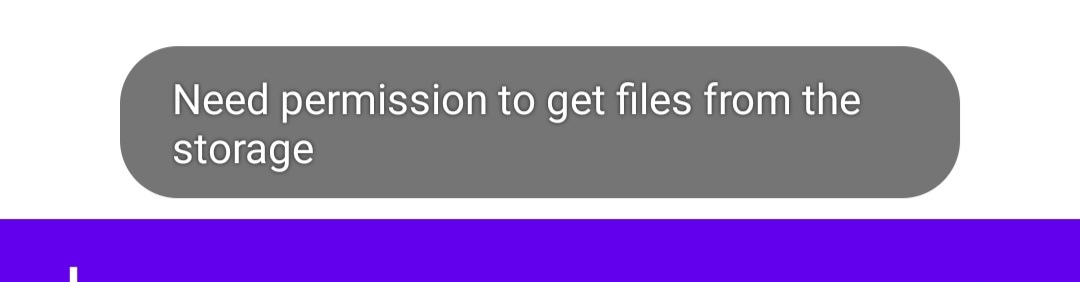


Рисунок 13. Ошибка получения разрешения

## Описание применения средств отладки

В ходе написания курсового проекта при попытке запустить программу были получены ошибки (Рисунок 14):

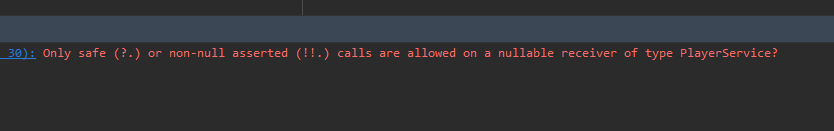


Рисунок 14. Сообщение об ошибке

При проверке кода были исправлены найденные ошибки, в результате при запуске программы ошибок не было (Рисунок 15):

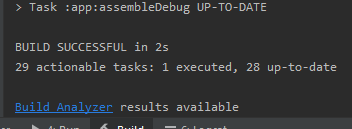


Рисунок 15. Успешная сборка

## Анализ оптимальности использования памяти и быстродействия

При анализе оптимальности использования памяти были обнаружены неиспользуемые файлы, которые увеличивали размер приложения (Рисунок 16):

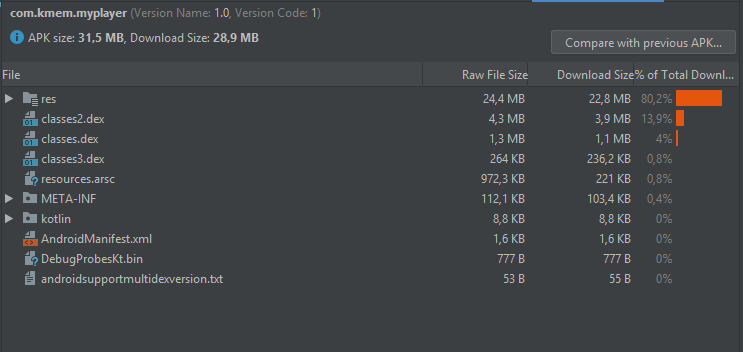


Рисунок 16. Размер приложения до оптимизации

После удаления ненужных файлов размер приложения уменьшился в 4 раза (Рисунок 17):

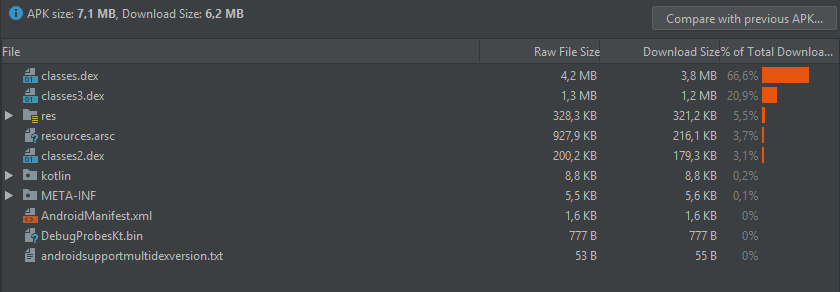


Рисунок 17. Размер после оптимизации

Также в класс DirectoryNode был добавлен метод для отложенной инициализации. Таким образом при создании дерева загружались не все узлы, а только необходимые.

**Листинг 3. Функция инициализации DirectoryNode.**

override fun initialize() {

if (isInitialized)

return

var node: FileTreeComponent // temporary variable

model?.file?.listFiles()?.forEach {

if (isAcceptable(it))

childModels.add(FileModel(it.name, it))

}

for (childModel in childModels) {

if (childModel.file.isDirectory) {

node = DirectoryNode(this, childModel)

} else {

node = FileNode(this, childModel)

}

node.isSelected = this.isSelected

childs.add(node)

}

isInitialized = true

}

# Эксплуатационная часть

## Руководство оператора

### Назначение программы

Разработанное приложение позволяет прослушивать аудио файлы в формате MP3 с памяти устройства Android. Добавляемые файлы сохраняются в БД приложения и данный список песен можно изменять.

### Условия выполнения программы

Операционная система: Android 4.4 и выше.

Оперативная память: 512 Мб.

Доступного места на диске: 50 Мб.

### Выполнение программы

При запуске программы откроется экран с информацией о текущем треке:

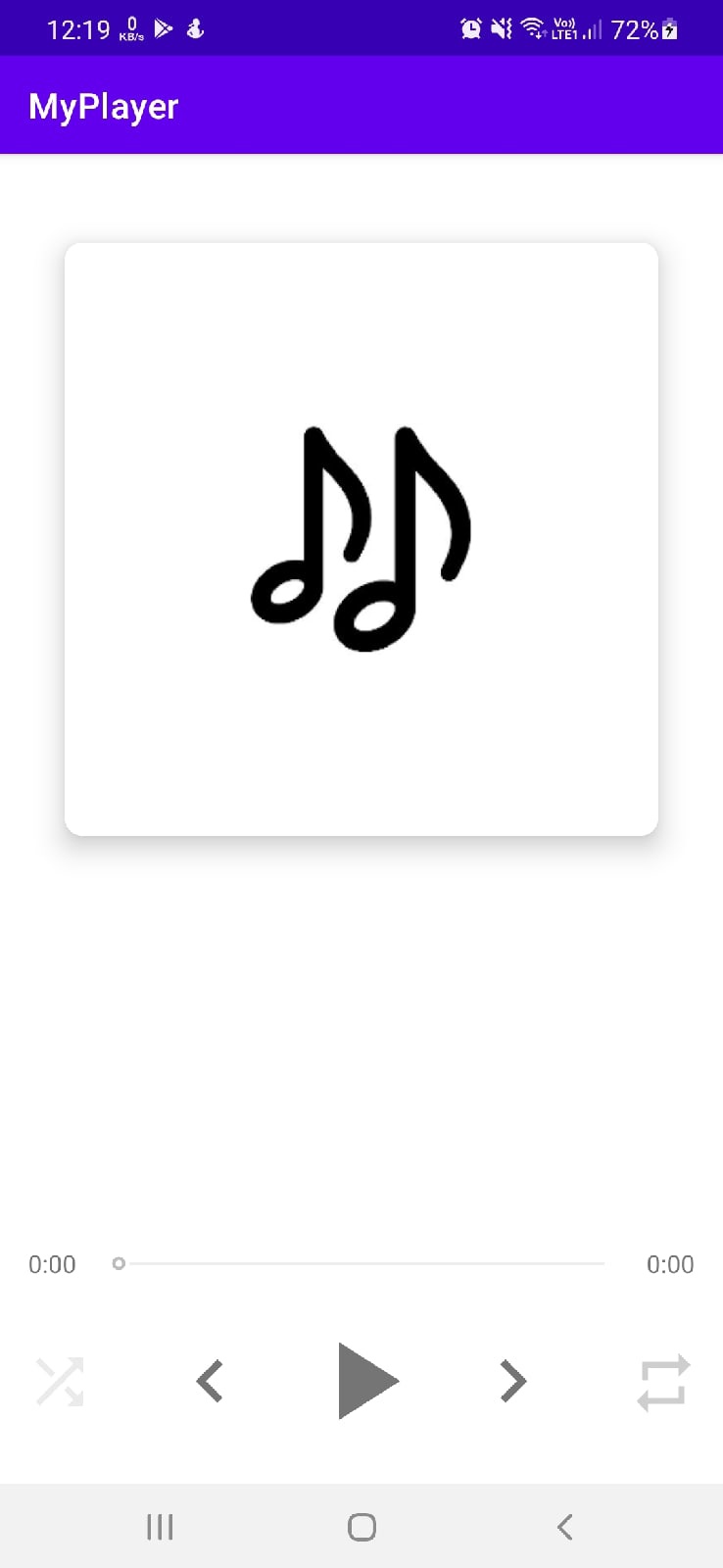


Рисунок 18. Запуск программы

Чтобы запустить плеер нужно добавить песни в плейлист. Для этого нужно провести влево для открытия экрана с плейлистом:



Рисунок 19. Экран с пустым плейлистом

Далее, чтобы добавить песни, нужно нажать на кнопку добавления в левом нижнем углу. При нажатии приложение запросит доступ к памяти и откроет экран добавления файлов:



Рисунок 20. Экран добавления файлов

При нажатии на директорию откроются файлы из этой директории. Чтобы выбрать файлы или директории рядом с ними находятся флажки:

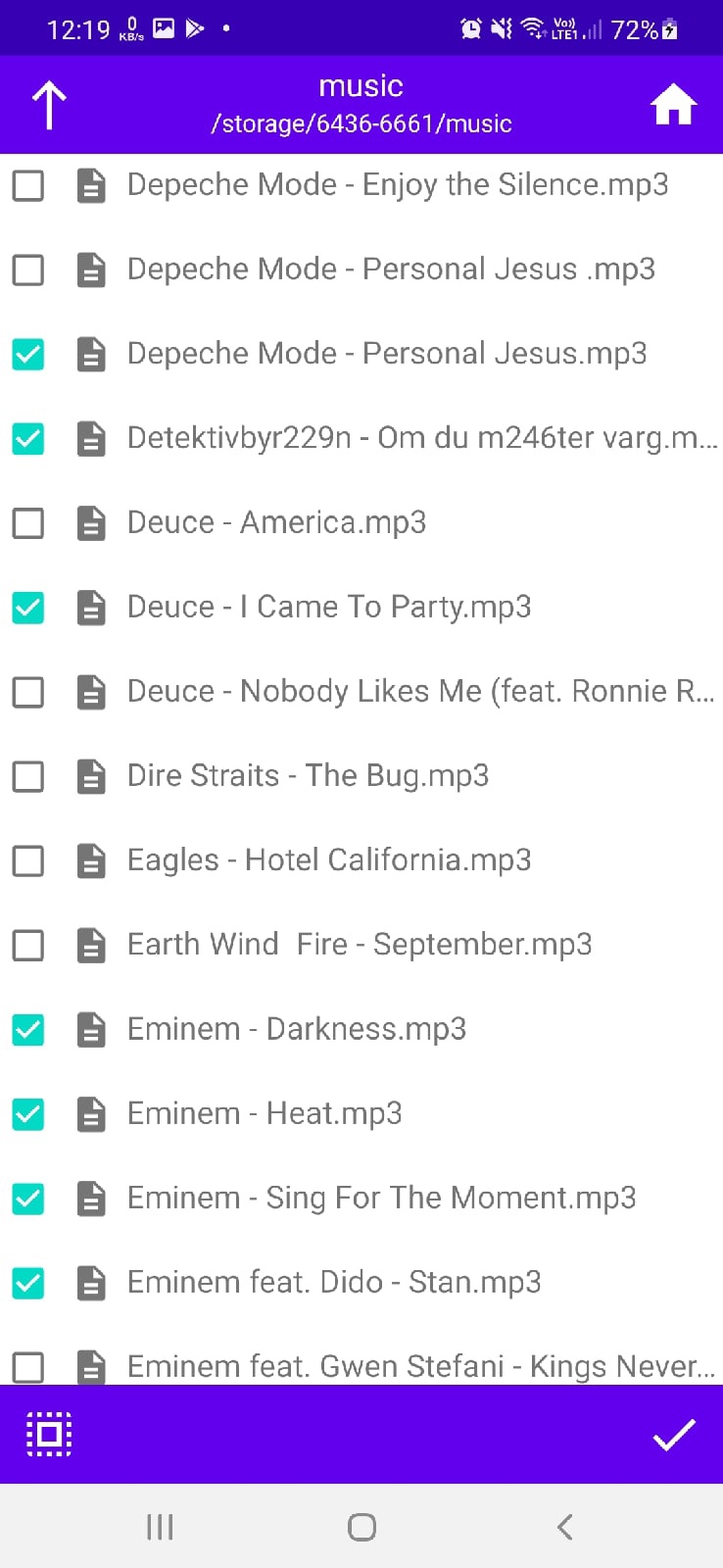


Рисунок 21. Выбор файлов

После выбора нужных файлов нужно нажать на кнопку подтверждения в правом нижнем углу. Откроется экран плейлиста с выбранными песнями:

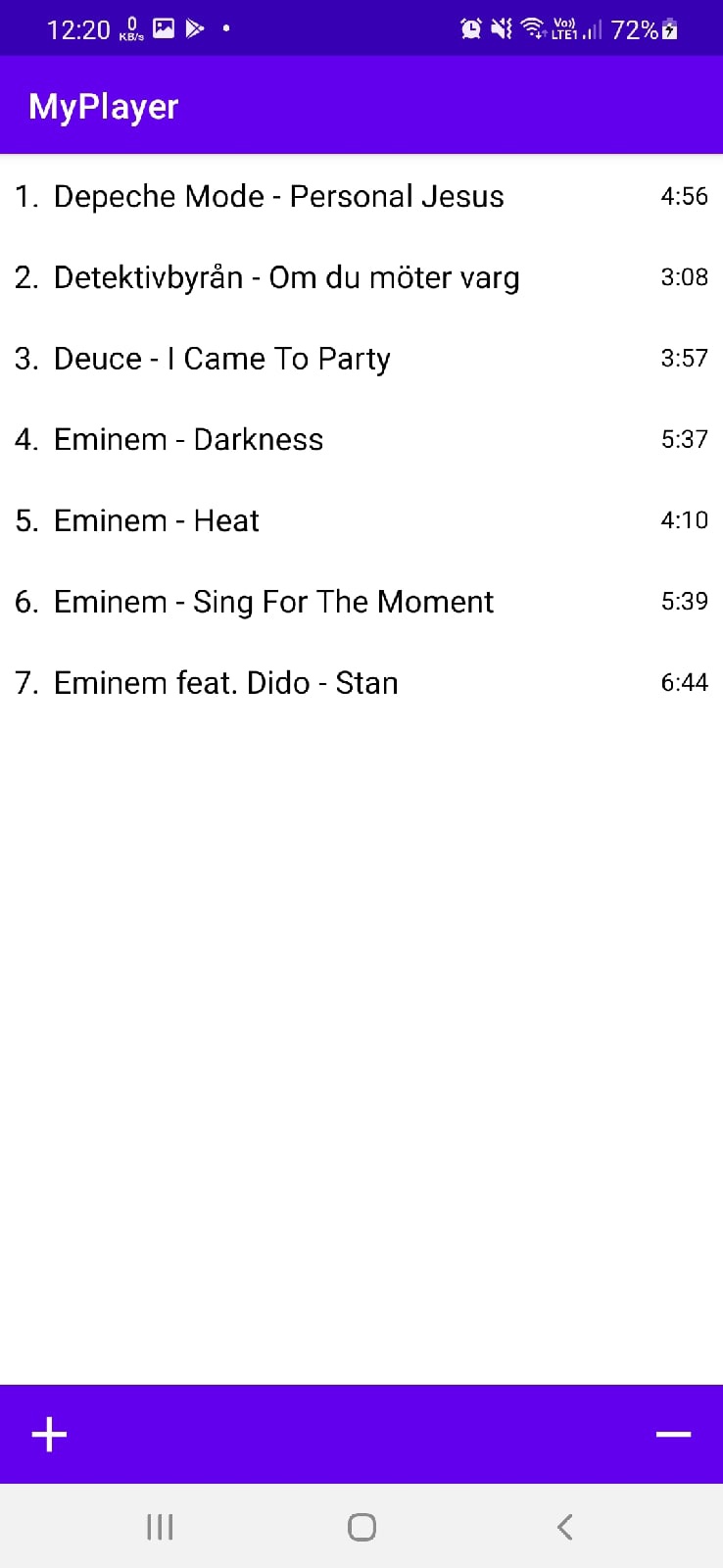


Рисунок 22. Плейлист с добавленными песнями

Чтобы начать проигрывание трека нужно нажать на элемент в списке песен. Он станет выделенным и экран текущего трека изменится:

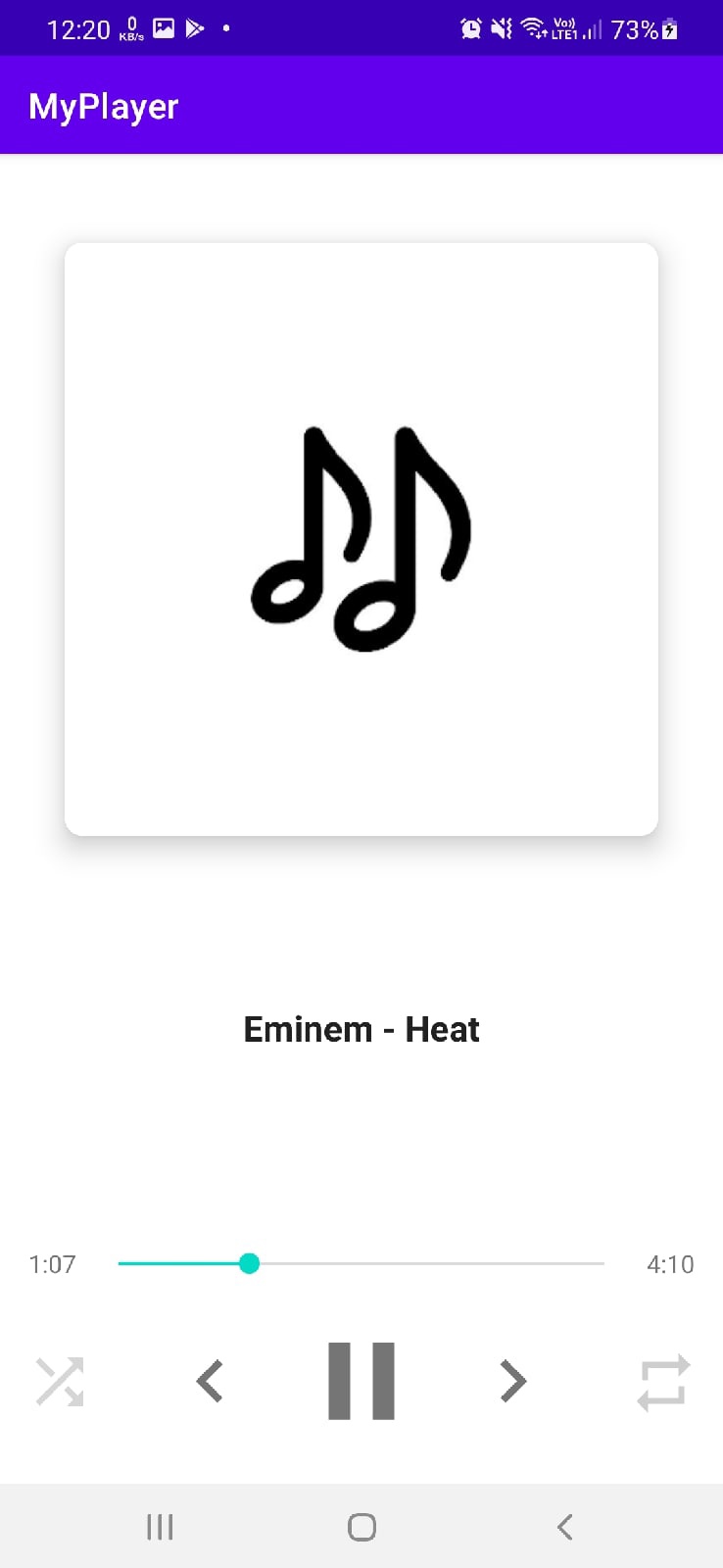


Рисунок 23. Текущая песня на главном экране

Чтобы **поменять позицию** трека в плейлисте нужно зажать на соответствующий элемент в списке и перетащить его на новую позицию.

Кнопка в правом нижнем углу на экране плейлиста позволяет **удалить песни** из плейлиста. При нажатии появятся флажки для выбора файлов, которые нужно удалить.

Для **перемешивания** песен или **повтора** песни используются соответствующие кнопки в нижней части экрана с текущей песней.

### Сообщение оператору

При попытке запуска плеера может появиться сообщение «Add items to playlist» (Рисунок 24):

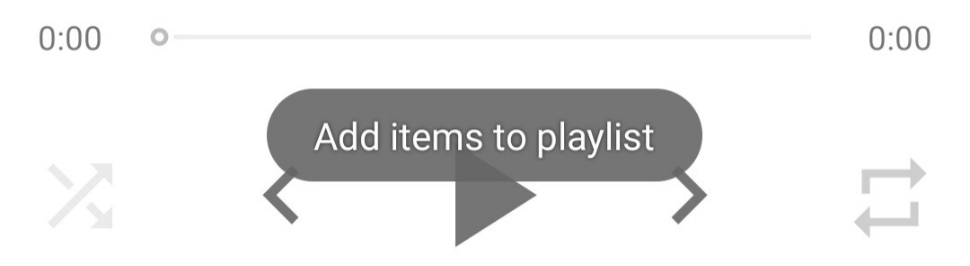


Рисунок 24. Сообщение об ошибке

**Причина:** отсутствие песен в плейлисте.

**Действия программы:** программа не запускает плеер.

**Действия оператора**: добавить песни в плейлист.

## To-Do лист

* Улучшить дизайн программы
* Добавить возможность создания нескольких плейлистов
* Увеличить функционал программы

# Заключение

Стандартный проигрыватель системы Android не очень удобен и имеет малый функционал. По этой причине в результате выполнения курсового проекта был создан музыкальный проигрыватель для системы Android. Данный проект может быть использован в учебных целях для обучения программированию приложений для Android.

# Список литературы и интернет ресурсов

1. Kotlin

<https://kotlinlang.ru/>

1. Android Studio

<https://developer.android.com/studio>

1. Медиапроигрыватель

<https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D0%B8%D0%B0%D0%BF%D1%80%D0%BE%D0%B8%D0%B3%D1%80%D1%8B%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C>

1. Head First Android Development: A Brain-Friendly Guide. Гриффитс Дэвид, Гриффитс Дон.
2. Kotlin for Android Developers А. Лейва
3. MP3

<https://ru.wikipedia.org/wiki/MP3>

1. AIMP

<https://play.google.com/store/apps/details?id=com.aimp.player&hl=ru&gl=US>

1. Audify Music Player

<https://play.google.com/store/apps/details?id=com.musicplayer.playermusic&hl=ru&gl=US>

1. BlackPlayer

<https://play.google.com/store/apps/details?id=com.musicplayer.blackplayerfree&hl=ru&gl=US>

1. jetAudio

<https://play.google.com/store/apps/details?id=com.jetappfactory.jetaudio&hl=ru&gl=US>

1. SQLite

<https://www.sqlite.org/index.html>

**Приложение 1. MainPlayerFragment.kt**

**package com.kmem.myplayer.ui.fragments**

**import android.animation.ObjectAnimator**

**import android.content.ComponentName**

**import android.content.Context.BIND\_AUTO\_CREATE**

**import android.content.Intent**

**import android.content.ServiceConnection**

**import android.os.Bundle**

**import android.os.IBinder**

**import android.os.RemoteException**

**import android.support.v4.media.MediaMetadataCompat**

**import android.support.v4.media.session.MediaControllerCompat**

**import android.support.v4.media.session.PlaybackStateCompat**

**import android.view.LayoutInflater**

**import android.view.View**

**import android.view.ViewGroup**

**import android.widget.ImageButton**

**import android.widget.ImageView**

**import android.widget.SeekBar**

**import android.widget.SeekBar.OnSeekBarChangeListener**

**import android.widget.TextView**

**import androidx.fragment.app.Fragment**

**import androidx.lifecycle.Observer**

**import com.kmem.myplayer.R**

**import com.kmem.myplayer.service.PlayerService**

**import kotlinx.coroutines.Job**

**import kotlinx.coroutines.MainScope**

**import kotlinx.coroutines.delay**

**import kotlinx.coroutines.launch**

**/\*\***

**\* Фрагмент экрана состояния плеера.**

**\* Отвечает за графические элементы и взаимодействие с пользователем.**

**\* Получает информацию от сервиса через PlayerServiceBinder.**

**\*/**

**class MainPlayerFragment : Fragment() {**

**private val FROM\_ALPHA = 0.3f**

**private val TO\_ALPHA = 1f**

**private var playerServiceBinder: PlayerService.PlayerServiceBinder? = null**

**private var mediaController: MediaControllerCompat? = null**

**private var callback: MediaControllerCompat.Callback? = null**

**private var serviceConnection: ServiceConnection? = null**

**private var playerService: PlayerService? = null**

**private lateinit var layout: View**

**private var isPlaying = false**

**private var durationBarJob: Job? = null**

**private var isStarted = false**

**private var shuffled = false**

**private var repeated = false**

**override fun onCreateView(**

**inflater: LayoutInflater, container: ViewGroup?,**

**savedInstanceState: Bundle?**

**): View {**

**// Inflate the layout for this fragment**

**layout = inflater.inflate(R.layout.fragment\_main\_player, container, false)**

**val prevButton = layout.findViewById<ImageView>(R.id.prev\_button)**

**val playButton = layout.findViewById<ImageView>(R.id.play\_button)**

**val nextButton = layout.findViewById<ImageView>(R.id.next\_button)**

**val shuffleButton = layout.findViewById<ImageButton>(R.id.shuffle\_button)**

**val repeatButton = layout.findViewById<ImageButton>(R.id.repeat\_button)**

**// select TextViews for sliding long text**

**layout.findViewById<TextView>(R.id.track\_title).isSelected = true**

**layout.findViewById<TextView>(R.id.artist).isSelected = true**

**callback = object : MediaControllerCompat.Callback() {**

**override fun onPlaybackStateChanged(state: PlaybackStateCompat?) {**

**if (state == null)**

**return**

**val positionView = layout.findViewById<SeekBar>(R.id.duration\_bar)**

**isPlaying = state.state == PlaybackStateCompat.STATE\_PLAYING**

**if (isPlaying) {**

**playButton.setImageResource(R.drawable.baseline\_pause\_24)**

**durationBarJob?.cancel()**

**if (isStarted)**

**runDurationBarUpdate()**

**} else {**

**playButton.setImageResource(R.drawable.baseline\_play\_arrow\_24)**

**durationBarJob?.cancel()**

**val position = mediaController?.playbackState?.position?.toInt() ?: 0**

**positionView.progress = position**

**updateCurrentDurationView(position)**

**}**

**}**

**}**

**serviceConnection = object : ServiceConnection {**

**override fun onServiceConnected(name: ComponentName?, service: IBinder?) {**

**playerServiceBinder = service as PlayerService.PlayerServiceBinder**

**try {**

**mediaController = playerServiceBinder!!.getMediaSessionToken()**

**?.let { MediaControllerCompat(activity, it) }**

**mediaController?.registerCallback(callback!!)**

**callback?.onPlaybackStateChanged(mediaController?.playbackState)**

**playerServiceBinder!!.getLiveMetadata()**

**.observe(viewLifecycleOwner, metadataObserver)**

**playerService = playerServiceBinder!!.getService()**

**shuffled = playerService?.getShuffle() ?: false**

**shuffleButton.alpha = if (shuffled) TO\_ALPHA else FROM\_ALPHA**

**} catch (e: RemoteException) {**

**mediaController = null**

**}**

**}**

**override fun onServiceDisconnected(name: ComponentName?) {**

**playerServiceBinder = null**

**if (mediaController != null) {**

**mediaController?.unregisterCallback(callback!!)**

**mediaController = null**

**}**

**}**

**}**

**activity?.bindService(**

**Intent(activity, PlayerService::class.java),**

**serviceConnection!!,**

**BIND\_AUTO\_CREATE**

**)**

**playButton.setOnClickListener {**

**if (mediaController != null) {**

**if (isPlaying)**

**mediaController?.transportControls?.pause()**

**else**

**mediaController?.transportControls?.play()**

**}**

**}**

**nextButton.setOnClickListener {**

**if (mediaController != null) {**

**// nullify duration**

**layout.findViewById<SeekBar>(R.id.duration\_bar).progress = 0**

**mediaController?.transportControls?.skipToNext()**

**}**

**}**

**prevButton.setOnClickListener {**

**if (mediaController != null) {**

**// nullify duration**

**layout.findViewById<SeekBar>(R.id.duration\_bar).progress = 0**

**mediaController?.transportControls?.skipToPrevious()**

**}**

**}**

**shuffleButton.isEnabled = false**

**shuffleButton.setOnClickListener {**

**var fromAlpha = FROM\_ALPHA**

**var toAlpha = TO\_ALPHA**

**if (shuffled)**

**fromAlpha = toAlpha.also { toAlpha = fromAlpha }**

**val alphaAnimator =**

**ObjectAnimator.ofFloat(shuffleButton, View.ALPHA, fromAlpha, toAlpha)**

**alphaAnimator.duration = 200**

**alphaAnimator.start()**

**shuffled = !shuffled**

**playerService?.setShuffle(shuffled)**

**}**

**repeatButton.isEnabled = false**

**repeatButton.setOnClickListener {**

**var fromAlpha = FROM\_ALPHA**

**var toAlpha = TO\_ALPHA**

**if (repeated)**

**fromAlpha = toAlpha.also { toAlpha = fromAlpha }**

**val alphaAnimator = ObjectAnimator.ofFloat(repeatButton, View.ALPHA, fromAlpha, toAlpha)**

**alphaAnimator.duration = 200**

**alphaAnimator.start()**

**repeated = !repeated**

**playerService?.repeatMode = repeated**

**}**

**setupDurationBarListener()**

**return layout**

**}**

**override fun onStart() {**

**super.onStart()**

**val durationBar = layout.findViewById<SeekBar>(R.id.duration\_bar)**

**// update duration if there is already playing track on pause**

**durationBar.progress = mediaController?.playbackState?.position?.toInt() ?: 0**

**val shuffleButton = layout.findViewById<ImageButton>(R.id.shuffle\_button) as ImageButton**

**shuffled = playerService?.getShuffle() ?: false**

**shuffleButton.alpha = if (shuffled) TO\_ALPHA else FROM\_ALPHA**

**val repeatButton = layout.findViewById<ImageButton>(R.id.repeat\_button) as ImageButton**

**repeated = playerService?.repeatMode ?: false**

**repeatButton.alpha = if (repeated) TO\_ALPHA else FROM\_ALPHA**

**if (isPlaying) runDurationBarUpdate()**

**isStarted = true**

**}**

**override fun onStop() {**

**super.onStop()**

**durationBarJob?.cancel()**

**isStarted = false**

**}**

**override fun onDestroy() {**

**super.onDestroy()**

**playerServiceBinder = null**

**if (mediaController != null) {**

**mediaController?.unregisterCallback(callback!!)**

**mediaController = null**

**}**

**durationBarJob?.cancel()**

**activity?.unbindService(serviceConnection!!)**

**}**

**private val metadataObserver = Observer<MediaMetadataCompat> { newMetadata ->**

**val artistView = layout.findViewById<TextView>(R.id.artist)**

**val titleView = layout.findViewById<TextView>(R.id.track\_title)**

**val albumImageView = layout.findViewById<ImageView>(R.id.album\_image)**

**val durationBar = layout.findViewById<SeekBar>(R.id.duration\_bar)**

**val maxDurationView = layout.findViewById<TextView>(R.id.max\_duration)**

**val shuffleButton = layout.findViewById<ImageButton>(R.id.shuffle\_button)**

**val repeatButton = layout.findViewById<ImageButton>(R.id.repeat\_button)**

**shuffleButton.isEnabled = true**

**repeatButton.isEnabled = true**

**if (newMetadata?.getString(MediaMetadataCompat.METADATA\_KEY\_ARTIST) == "Unknown") {**

**artistView.text = ""**

**} else {**

**artistView.text = newMetadata?.getString(MediaMetadataCompat.METADATA\_KEY\_ARTIST)**

**}**

**titleView.text = newMetadata?.getString(MediaMetadataCompat.METADATA\_KEY\_TITLE)**

**albumImageView.setImageBitmap(newMetadata?.getBitmap(MediaMetadataCompat.METADATA\_KEY\_ART))**

**durationBar.max =**

**newMetadata?.getLong(MediaMetadataCompat.METADATA\_KEY\_DURATION)?.toInt() ?: 1**

**durationBar.isEnabled = true**

**val mins = newMetadata!!.getLong(MediaMetadataCompat.METADATA\_KEY\_DURATION) / 1000 / 60**

**val secs = newMetadata.getLong(MediaMetadataCompat.METADATA\_KEY\_DURATION) / 1000 % 60**

**val duration = if (secs < 10) "$mins:0$secs" else "$mins:$secs" // "0:00" duration format**

**maxDurationView.text = duration**

**}**

**private fun updateCurrentDurationView(position: Int) {**

**val currDurationView = layout.findViewById<TextView>(R.id.curr\_duration)**

**val mins = position / 1000 / 60**

**val secs = position / 1000 % 60**

**val duration = if (secs < 10) "$mins:0$secs" else "$mins:$secs" // "0:00" duration format**

**currDurationView.text = duration**

**}**

**private fun setupDurationBarListener() {**

**val durationBar = layout.findViewById<SeekBar>(R.id.duration\_bar)**

**durationBar.isEnabled = false**

**durationBar.setOnSeekBarChangeListener(object : OnSeekBarChangeListener {**

**var changedDuration = 0**

**override fun onStopTrackingTouch(seekBar: SeekBar) {**

**mediaController?.transportControls?.seekTo(changedDuration.toLong())**

**durationBar.progress = changedDuration**

**if (isPlaying)**

**runDurationBarUpdate(true)**

**}**

**override fun onStartTrackingTouch(seekBar: SeekBar) {**

**durationBarJob?.cancel()**

**}**

**override fun onProgressChanged(seekBar: SeekBar, progress: Int, fromUser: Boolean) {**

**if (fromUser) {**

**changedDuration = progress**

**updateCurrentDurationView(changedDuration)**

**}**

**}**

**})**

**}**

**private fun runDurationBarUpdate(onSeek: Boolean = false) {**

**if (durationBarJob == null || durationBarJob!!.isCancelled) {**

**durationBarJob = MainScope().launch {**

**val positionView = layout.findViewById<SeekBar>(R.id.duration\_bar)**

**if (onSeek)**

**delay(100) // wait for exo player to seek**

**while (true) {**

**val position = mediaController?.playbackState?.position?.toInt() ?: 0**

**positionView.progress = position**

**updateCurrentDurationView(position)**

**delay(500)**

**}**

**}**

**}**

**}**

**}**

**Приложение 2. PlaylistFragment.kt**

**package com.kmem.myplayer.ui.fragments**

**import android.Manifest**

**import android.content.ComponentName**

**import android.content.Context.BIND\_AUTO\_CREATE**

**import android.content.Intent**

**import android.content.ServiceConnection**

**import android.content.pm.PackageManager**

**import android.net.Uri**

**import android.os.Build**

**import android.os.Bundle**

**import android.os.IBinder**

**import android.os.RemoteException**

**import android.support.v4.media.session.MediaControllerCompat**

**import android.support.v4.media.session.PlaybackStateCompat**

**import android.view.LayoutInflater**

**import android.view.View**

**import android.view.ViewGroup**

**import android.widget.ImageButton**

**import android.widget.ProgressBar**

**import android.widget.Toast**

**import androidx.annotation.RequiresApi**

**import androidx.core.content.ContextCompat**

**import androidx.fragment.app.Fragment**

**import androidx.lifecycle.Observer**

**import androidx.lifecycle.ViewModelProvider**

**import androidx.recyclerview.widget.ItemTouchHelper**

**import androidx.recyclerview.widget.LinearLayoutManager**

**import androidx.recyclerview.widget.RecyclerView**

**import com.kmem.myplayer.R**

**import com.kmem.myplayer.data.Track**

**import com.kmem.myplayer.service.PlayerService**

**import com.kmem.myplayer.ui.activities.FileChooserActivity**

**import com.kmem.myplayer.ui.adapters.PlaylistAdapter**

**import com.kmem.myplayer.ui.helpers.PlaylistItemTouchHelperCallback**

**import com.kmem.myplayer.viewmodels.PlaylistViewModel**

**import kotlinx.coroutines.\***

**/\*\***

**\* Фрагмент экрана управления плейлистом.**

**\* Отвечает за графические элементы и взаимодействие с пользователем.**

**\* Получает информацию от сервиса через PlayerServiceBinder.**

**\* Использует PlaylistViewModel для получения данных из БД.**

**\*/**

**class PlaylistFragment : Fragment(), PlaylistAdapter.Listener {**

**private var audios: ArrayList<Track> = ArrayList()**

**private lateinit var list: RecyclerView**

**private lateinit var touchHelper: ItemTouchHelper**

**private lateinit var layout: View**

**private var playerService: PlayerService? = null**

**private var playerServiceBinder: PlayerService.PlayerServiceBinder? = null**

**private var mediaController: MediaControllerCompat? = null**

**private var callback: MediaControllerCompat.Callback? = null**

**private var serviceConnection: ServiceConnection? = null**

**private lateinit var model: PlaylistViewModel**

**override var currentUri: Uri = Uri.EMPTY**

**override var deleteMode: Boolean = false**

**override var selectedCheckboxesPositions: ArrayList<Int> = ArrayList()**

**companion object {**

**const val PERMISSION\_STRING = Manifest.permission.READ\_EXTERNAL\_STORAGE**

**const val PERMISSION\_CODE = 596**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onCreateView(**

**inflater: LayoutInflater, container: ViewGroup?,**

**savedInstanceState: Bundle?**

**): View {**

**layout = inflater.inflate(R.layout.fragment\_playlist, container, false)**

**model = ViewModelProvider(requireActivity()).get(PlaylistViewModel::class.java)**

**model.audios.observe(viewLifecycleOwner, audiosObserver)**

**list = layout.findViewById<View>(R.id.songs\_list) as RecyclerView**

**val adapter = PlaylistAdapter(audios)**

**val touchCallback = PlaylistItemTouchHelperCallback(adapter)**

**touchHelper = ItemTouchHelper(touchCallback)**

**touchHelper.attachToRecyclerView(list)**

**adapter.setListener(this@PlaylistFragment)**

**list.adapter = adapter**

**list.layoutManager = LinearLayoutManager(context)**

**val addButton = layout.findViewById<ImageButton>(R.id.add\_tracks)**

**addButton.setOnClickListener {**

**val intent = Intent(context, FileChooserActivity::class.java)**

**if (checkPermission())**

**startActivityForResult(intent, 1)**

**else**

**requestPermission()**

**}**

**val removeButton = layout.findViewById<ImageButton>(R.id.remove\_tracks)**

**val deleteTracksButton = layout.findViewById<ImageButton>(R.id.delete\_tracks)**

**val selectAllButton = layout.findViewById<ImageButton>(R.id.select\_all)**

**removeButton.setOnClickListener {**

**deleteMode = !deleteMode**

**onDeleteModeChanged()**

**list.adapter?.notifyDataSetChanged()**

**}**

**deleteTracksButton.setOnClickListener {**

**val tracks = ArrayList<Track>()**

**for (pos in selectedCheckboxesPositions)**

**tracks.add(audios[pos])**

**deleteMode = false**

**onDeleteModeChanged()**

**// async update Database**

**MainScope().launch {**

**model.deleteTracks(tracks)**

**playerService?.deleteTracks(tracks)**

**}**

**}**

**selectAllButton.setOnClickListener {**

**if (selectedCheckboxesPositions.size == audios.size) {**

**selectedCheckboxesPositions.clear()**

**} else {**

**for (pos in 0 until audios.size)**

**if (pos !in selectedCheckboxesPositions)**

**selectedCheckboxesPositions.add(pos)**

**}**

**list.adapter?.notifyDataSetChanged()**

**}**

**callback = object : MediaControllerCompat.Callback() {**

**override fun onPlaybackStateChanged(state: PlaybackStateCompat?) {**

**if (state == null)**

**return**

**}**

**}**

**serviceConnection = object : ServiceConnection {**

**override fun onServiceConnected(name: ComponentName?, service: IBinder?) {**

**playerServiceBinder = service as PlayerService.PlayerServiceBinder**

**try {**

**mediaController = playerServiceBinder!!.getMediaSessionToken()**

**?.let { MediaControllerCompat(activity, it) }**

**mediaController?.registerCallback(callback!!)**

**callback?.onPlaybackStateChanged(mediaController?.playbackState)**

**playerServiceBinder!!.getLiveUri().observe(viewLifecycleOwner, uriObserver)**

**playerService = playerServiceBinder?.getService()**

**} catch (e: RemoteException) {**

**mediaController = null**

**}**

**}**

**override fun onServiceDisconnected(name: ComponentName?) {**

**playerServiceBinder = null**

**if (mediaController != null) {**

**mediaController?.unregisterCallback(callback!!)**

**mediaController = null**

**}**

**}**

**}**

**activity?.bindService(**

**Intent(activity, PlayerService::class.java),**

**serviceConnection!!,**

**BIND\_AUTO\_CREATE**

**)**

**return layout**

**}**

**private val audiosObserver = Observer<ArrayList<Track>> { newAudios ->**

**audios.clear()**

**audios.addAll(newAudios)**

**list.adapter?.notifyDataSetChanged()**

**}**

**override fun onDestroy() {**

**super.onDestroy()**

**playerServiceBinder = null**

**if (mediaController != null) {**

**mediaController?.unregisterCallback(callback!!)**

**mediaController = null**

**}**

**activity?.unbindService(serviceConnection!!)**

**}**

**override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {**

**if (data == null) return**

**val paths = data.getStringArrayListExtra(FileChooserActivity.PATHS) ?: ArrayList<String>()**

**if (paths.size == 0) return**

**MainScope().launch {**

**val loadingSpinner = layout.findViewById<ProgressBar>(R.id.progress\_bar)**

**loadingSpinner.visibility = View.VISIBLE**

**model.addTracks(paths)**

**playerService?.addNewTracks()**

**loadingSpinner.visibility = View.GONE**

**}**

**}**

**private val uriObserver = Observer<Uri> { newUri ->**

**currentUri = newUri**

**list.adapter?.notifyDataSetChanged()**

**}**

**private fun checkPermission(): Boolean {**

**return ContextCompat.checkSelfPermission(**

**requireContext(),**

**PERMISSION\_STRING**

**) == PackageManager.PERMISSION\_GRANTED**

**}**

**private fun requestPermission() {**

**if (ContextCompat.checkSelfPermission(requireContext(), PERMISSION\_STRING)**

**!= PackageManager.PERMISSION\_GRANTED**

**) {**

**requestPermissions(**

**arrayOf(PERMISSION\_STRING),**

**PERMISSION\_CODE**

**)**

**}**

**}**

**private fun onDeleteModeChanged() {**

**val addButton = layout.findViewById<ImageButton>(R.id.add\_tracks)**

**val selectAllButton = layout.findViewById<ImageButton>(R.id.select\_all)**

**val deleteTracksButton = layout.findViewById<ImageButton>(R.id.delete\_tracks)**

**if (deleteMode) {**

**touchHelper.attachToRecyclerView(null)**

**selectAllButton.visibility = View.VISIBLE**

**deleteTracksButton.visibility = View.VISIBLE**

**addButton.visibility = View.GONE**

**} else {**

**touchHelper.attachToRecyclerView(list)**

**selectAllButton.visibility = View.GONE**

**deleteTracksButton.visibility = View.GONE**

**addButton.visibility = View.VISIBLE**

**selectedCheckboxesPositions.clear()**

**}**

**}**

**override fun onRequestPermissionsResult(**

**requestCode: Int,**

**permissions: Array<out String>,**

**grantResults: IntArray**

**) {**

**super.onRequestPermissionsResult(requestCode, permissions, grantResults)**

**if (requestCode == PERMISSION\_CODE) {**

**if (grantResults.isNotEmpty() &&**

**grantResults[0] == PackageManager.PERMISSION\_GRANTED**

**) {**

**val intent = Intent(context, FileChooserActivity::class.java)**

**startActivityForResult(intent, 1)**

**} else {**

**Toast.makeText(**

**context,**

**getString(R.string.on\_permission\_denied),**

**Toast.LENGTH\_SHORT**

**).show()**

**}**

**}**

**}**

**override fun onClick(position: Int) {**

**val bundle = Bundle()**

**bundle.putInt(PlayerService.EXTRA\_POSITION, position)**

**mediaController?.transportControls?.sendCustomAction(**

**PlayerService.ACTION\_PLAY\_AT\_POSITION,**

**bundle**

**)**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun updatePositions() {**

**MainScope().launch {**

**withContext(Dispatchers.IO) {**

**model.updatePositions(audios)**

**playerService?.updatePositions()**

**}**

**}**

**}**

**}**

**Приложение 3. FileChooserActivity.kt**

**package com.kmem.myplayer.ui.activities**

**import android.content.Intent**

**import android.os.Bundle**

**import android.view.View**

**import android.widget.ImageButton**

**import android.widget.ProgressBar**

**import android.widget.TextView**

**import androidx.appcompat.app.AppCompatActivity**

**import androidx.appcompat.widget.Toolbar**

**import androidx.lifecycle.Observer**

**import androidx.lifecycle.ViewModelProvider**

**import androidx.recyclerview.widget.LinearLayoutManager**

**import androidx.recyclerview.widget.RecyclerView**

**import com.kmem.myplayer.R**

**import com.kmem.myplayer.ui.adapters.FileChooserAdapter**

**import com.kmem.myplayer.viewmodels.FileChooserViewModel**

**import kotlinx.coroutines.CoroutineScope**

**import kotlinx.coroutines.Dispatchers**

**import kotlinx.coroutines.Job**

**import kotlinx.coroutines.launch**

**import java.io.File**

**import java.io.Serializable**

**/\*\***

**\* Активность экрана добавления песен.**

**\* Отвечает за графические элементы и взаимодействие с пользователем.**

**\*/**

**class FileChooserActivity : AppCompatActivity(), FileChooserAdapter.Listener {**

**companion object {**

**const val PATHS = "paths"**

**}**

**private val list by lazy { findViewById<RecyclerView>(R.id.fileList) }**

**private val currentDirs: ArrayList<FileTreeComponent> = ArrayList()**

**private lateinit var model: FileChooserViewModel**

**private var scope = CoroutineScope(Dispatchers.Main + Job())**

**private lateinit var loadingSpinner: ProgressBar**

**override fun onCreate(savedInstanceState: Bundle?) {**

**super.onCreate(savedInstanceState)**

**setContentView(R.layout.activity\_file\_chooser)**

**model = ViewModelProvider(this).get(FileChooserViewModel::class.java)**

**model.currentDirs.observe(this, currentDirsObserver)**

**model.currentDirName.observe(this, currentDirNameObserver)**

**model.currentPath.observe(this, currentPathObserver)**

**loadingSpinner = findViewById<ProgressBar>(R.id.progress\_bar)**

**loadingSpinner.visibility = View.GONE**

**val toolbar = findViewById<Toolbar>(R.id.toolbar)**

**setSupportActionBar(toolbar)**

**supportActionBar?.setDisplayShowTitleEnabled(false)**

**val previousPathButton = findViewById<ImageButton>(R.id.prev\_path\_button)**

**previousPathButton.setOnClickListener { scope.launch { model.openPreviousDir() } }**

**val homeButton = findViewById<ImageButton>(R.id.home\_button)**

**homeButton.setOnClickListener { model.setHomeDirs() }**

**val selectAllButton = findViewById<ImageButton>(R.id.select\_all)**

**selectAllButton.setOnClickListener { model.selectAllCurrent() }**

**val loadButton = findViewById<ImageButton>(R.id.load\_files)**

**loadButton.setOnClickListener {**

**sendPaths(model.loadFiles())**

**}**

**val adapter = FileChooserAdapter(currentDirs)**

**adapter.setListener(this)**

**list.layoutManager = LinearLayoutManager(this)**

**list.adapter = adapter**

**}**

**private val currentDirsObserver = object : Observer<ArrayList<FileTreeComponent>> {**

**override fun onChanged(t: ArrayList<FileTreeComponent>?) {**

**if (t == null)**

**return**

**if (model.wasSelectedOneFile) {**

**currentDirs[model.positionSelected] = t[model.positionSelected]**

**list.adapter?.notifyItemChanged(model.positionSelected)**

**} else {**

**currentDirs.clear()**

**currentDirs.addAll(t)**

**list.adapter?.notifyDataSetChanged()**

**}**

**}**

**}**

**private val currentDirNameObserver = Observer<String> { t ->**

**val dirNameView = findViewById<TextView>(R.id.dir\_name)**

**if (t == "")**

**dirNameView.text = resources.getString(R.string.home\_screen)**

**else**

**dirNameView.text = t**

**}**

**private val currentPathObserver = Observer<String> { t ->**

**val pathView = findViewById<TextView>(R.id.dir\_path)**

**pathView.text = t**

**}**

**private fun sendPaths(paths: ArrayList<String>) {**

**intent = Intent()**

**intent.putStringArrayListExtra(PATHS, paths)**

**setResult(RESULT\_OK, intent)**

**finish()**

**}**

**override fun onClick(position: Int) {**

**scope.launch {**

**val isDirectory = currentDirs[position].isDirectory**

**if (isDirectory) {**

**loadingSpinner.visibility = View.VISIBLE**

**currentDirs.clear()**

**list.adapter?.notifyDataSetChanged()**

**}**

**model.onClick(position)**

**if (isDirectory) {**

**loadingSpinner.visibility = View.GONE**

**list.scrollToPosition(0)**

**}**

**}**

**}**

**override fun onCheckboxClick(position: Int, value: Boolean) {**

**model.onCheckboxClick(position, value)**

**}**

**abstract class FileTreeComponent : Serializable {**

**var parent: FileTreeComponent? = null**

**var model: FileModel? = null**

**var isSelected: Boolean = false**

**var hasSelectedChilds = false**

**var isInitialized = false**

**open var isDirectory = false**

**open fun initialize() {}**

**open fun checkChilds() {}**

**open fun childAt(index: Int): FileTreeComponent? {**

**return null**

**}**

**open fun childsList(): ArrayList<FileTreeComponent>? {**

**return null**

**}**

**open fun changeSelected(value: Boolean) {}**

**open fun updateChilds(value: Boolean) {}**

**}**

**class FileNode(parent: FileTreeComponent, model: FileModel) : FileTreeComponent(),**

**Serializable {**

**init {**

**this.parent = parent**

**this.model = model**

**}**

**override fun changeSelected(value: Boolean) {**

**isSelected = value**

**parent?.checkChilds()**

**}**

**}**

**class DirectoryNode(parent: FileTreeComponent?, model: FileModel?) : FileTreeComponent(),**

**Serializable {**

**var childs = ArrayList<FileTreeComponent>()**

**var childModels = ArrayList<FileModel>() // models for lately initializing**

**override var isDirectory = true**

**init {**

**this.parent = parent**

**this.model = model**

**}**

**override fun initialize() {**

**if (isInitialized)**

**return**

**var node: FileTreeComponent // temporary variable**

**model?.file?.listFiles()?.forEach {**

**if (isAcceptable(it))**

**childModels.add(FileModel(it.name, it))**

**}**

**for (childModel in childModels) {**

**if (childModel.file.isDirectory) {**

**node = DirectoryNode(this, childModel)**

**} else {**

**node = FileNode(this, childModel)**

**}**

**node.isSelected = this.isSelected**

**childs.add(node)**

**}**

**isInitialized = true**

**}**

**override fun changeSelected(value: Boolean) {**

**isSelected = value**

**parent?.checkChilds()**

**if (isInitialized)**

**updateChilds(value)**

**}**

**override fun checkChilds() {**

**if (childs.all { it.isSelected }) {**

**isSelected = true**

**} else if (childs.any { it.isSelected || it.hasSelectedChilds }) {**

**hasSelectedChilds = true**

**isSelected = false**

**} else {**

**isSelected = false**

**hasSelectedChilds = false**

**}**

**parent?.checkChilds()**

**}**

**override fun updateChilds(value: Boolean) {**

**childs.forEach { it.isSelected = value; it.updateChilds(value) }**

**}**

**override fun childAt(index: Int): FileTreeComponent {**

**return childs[index]**

**}**

**override fun childsList(): ArrayList<FileTreeComponent> {**

**initialize()**

**return childs**

**}**

**private fun isAcceptable(file: File): Boolean {**

**return (file.isDirectory && !file.name.startsWith('.'))**

**|| (!file.isDirectory && file.name.endsWith(".mp3"))**

**}**

**}**

**// file wrapper for custom name**

**data class FileModel(val name: String, val file: File) : Serializable**

**}**

**Приложение 4. PlayerService.kt**

**package com.kmem.myplayer.service**

**import android.annotation.SuppressLint**

**import android.app.\***

**import android.content.BroadcastReceiver**

**import android.content.Context**

**import android.content.Intent**

**import android.content.IntentFilter**

**import android.graphics.BitmapFactory**

**import android.media.\***

**import android.net.Uri**

**import android.os.Binder**

**import android.os.Build**

**import android.os.Bundle**

**import android.os.IBinder**

**import android.support.v4.media.MediaMetadataCompat**

**import android.support.v4.media.session.MediaSessionCompat**

**import android.support.v4.media.session.PlaybackStateCompat**

**import android.util.Log**

**import android.widget.Toast**

**import androidx.annotation.RequiresApi**

**import androidx.core.app.NotificationCompat**

**import androidx.core.app.NotificationManagerCompat**

**import androidx.core.content.ContextCompat**

**import androidx.lifecycle.LiveData**

**import androidx.lifecycle.MutableLiveData**

**import androidx.media.session.MediaButtonReceiver**

**import com.google.android.exoplayer2.\***

**import com.google.android.exoplayer2.extractor.DefaultExtractorsFactory**

**import com.google.android.exoplayer2.extractor.ExtractorsFactory**

**import com.google.android.exoplayer2.source.ProgressiveMediaSource**

**import com.google.android.exoplayer2.source.TrackGroupArray**

**import com.google.android.exoplayer2.trackselection.DefaultTrackSelector**

**import com.google.android.exoplayer2.trackselection.TrackSelectionArray**

**import com.google.android.exoplayer2.upstream.\***

**import com.kmem.myplayer.R**

**import com.kmem.myplayer.data.Track**

**import com.kmem.myplayer.ui.activities.MainActivity**

**import kotlinx.coroutines.\***

**import java.util.\***

**import kotlin.collections.ArrayList**

**/\*\***

**\* Сервис, который проигрывает музыку. Предоставляет API через PlayerServiceBinder.**

**\* Сервис имеет статус Foreground и может проигрывать музыку вне зависимости от проиложения.**

**\*/**

**class PlayerService : Service() {**

**companion object {**

**const val ACTION\_PLAY\_AT\_POSITION = "play\_at\_position"**

**const val EXTRA\_POSITION = "extra\_position"**

**private const val NOTIFICATION\_ID = 404**

**private const val NOTIFICATION\_DEFAULT\_CHANNEL\_ID = "default\_channel"**

**private const val INACTIVITY\_TIMEOUT = 600\_000L // 10 mins**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**private val metadataBuilder = MediaMetadataCompat.Builder()**

**private val stateBuilder = PlaybackStateCompat.Builder().setActions(**

**PlaybackStateCompat.ACTION\_PLAY or**

**PlaybackStateCompat.ACTION\_PAUSE or**

**PlaybackStateCompat.ACTION\_PLAY\_PAUSE or**

**PlaybackStateCompat.ACTION\_SKIP\_TO\_NEXT or**

**PlaybackStateCompat.ACTION\_SKIP\_TO\_PREVIOUS**

**)**

**private val context = this**

**private var mediaSession: MediaSessionCompat? = null**

**private var audioManager: AudioManager? = null**

**private var audioFocusRequest: AudioFocusRequest? = null**

**private var audioFocusRequested = false**

**private var exoPlayer: SimpleExoPlayer? = null**

**private var extractorsFactory: ExtractorsFactory? = null**

**private var dataSourceFactory: DataSource.Factory? = null**

**private var musicRepository: MusicRepository? = null**

**private var inactivityCheckJob: Job? = null**

**val currentMetadata: MutableLiveData<MediaMetadataCompat> =**

**MutableLiveData<MediaMetadataCompat>()**

**val currentUri: MutableLiveData<Uri> = MutableLiveData<Uri>()**

**var repeatMode: Boolean = false**

**@SuppressLint("WrongConstant")**

**override fun onCreate() {**

**super.onCreate()**

**if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {**

**val notificationChannel = NotificationChannel(**

**NOTIFICATION\_DEFAULT\_CHANNEL\_ID,**

**getString(R.string.notification\_channel\_name),**

**NotificationManagerCompat.IMPORTANCE\_DEFAULT**

**)**

**val notificationManager =**

**getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager**

**notificationManager.createNotificationChannel(notificationChannel)**

**val audioAttributes = AudioAttributes.Builder()**

**.setUsage(AudioAttributes.USAGE\_MEDIA)**

**.setContentType(AudioAttributes.USAGE\_MEDIA)**

**.build()**

**audioFocusRequest = AudioFocusRequest.Builder(AudioManager.AUDIOFOCUS\_GAIN)**

**.setOnAudioFocusChangeListener(audioFocusChangeListener)**

**.setAcceptsDelayedFocusGain(false)**

**.setWillPauseWhenDucked(true)**

**.setAudioAttributes(audioAttributes)**

**.build()**

**}**

**MainScope().launch {**

**withContext(Dispatchers.IO) {**

**musicRepository = MusicRepository(context)**

**}**

**}**

**audioManager = getSystemService(Context.AUDIO\_SERVICE) as AudioManager**

**mediaSession = MediaSessionCompat(this, "PlayerService")**

**mediaSession!!.setCallback(mediaSessionCallback)**

**val activityIntent = Intent(applicationContext, MainActivity::class.java)**

**mediaSession!!.setSessionActivity(**

**PendingIntent.getActivity(**

**applicationContext,**

**0,**

**activityIntent,**

**PendingIntent.FLAG\_UPDATE\_CURRENT**

**)**

**)**

**val mediaButtonIntent = Intent(**

**Intent.ACTION\_MEDIA\_BUTTON,**

**null,**

**applicationContext,**

**MediaButtonReceiver::class.java**

**)**

**mediaSession!!.setMediaButtonReceiver(**

**PendingIntent.getBroadcast(**

**applicationContext,**

**0,**

**mediaButtonIntent,**

**0**

**)**

**)**

**exoPlayer = SimpleExoPlayer.Builder(this)**

**.setTrackSelector(DefaultTrackSelector(this))**

**.setLoadControl(DefaultLoadControl())**

**.build()**

**exoPlayer?.addListener(exoPlayerListener)**

**val fileDataSource = FileDataSource.Factory()**

**this.dataSourceFactory = fileDataSource**

**this.extractorsFactory = DefaultExtractorsFactory()**

**}**

**override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {**

**MediaButtonReceiver.handleIntent(mediaSession, intent)**

**return START\_STICKY**

**}**

**override fun onDestroy() {**

**super.onDestroy()**

**mediaSession?.release()**

**exoPlayer?.release()**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**suspend fun updatePositions() {**

**musicRepository?.updatePositions()**

**}**

**suspend fun addNewTracks() {**

**musicRepository?.addNewTracks()**

**}**

**fun setShuffle(value: Boolean) {**

**musicRepository?.shuffle = value**

**}**

**fun getShuffle(): Boolean {**

**return musicRepository?.shuffle ?: false**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**suspend fun deleteTracks(tracks: ArrayList<Track>) {**

**musicRepository?.deleteTracks(tracks)**

**// refresh current track**

**if (musicRepository?.getCurrent()?.uri != currentUri.value**

**&& mediaSession?.isActive == true**

**) {**

**val state = mediaSessionCallback.currentState**

**mediaSessionCallback.onPause()**

**mediaSessionCallback.onPlay() // play for update track**

**if (state != PlaybackStateCompat.STATE\_PLAYING)**

**mediaSessionCallback.onPause()**

**}**

**}**

**private val mediaSessionCallback = object : MediaSessionCompat.Callback() {**

**var currentState = PlaybackStateCompat.STATE\_STOPPED**

**init {**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**currentState,**

**PlaybackStateCompat.PLAYBACK\_POSITION\_UNKNOWN,**

**1F**

**).build()**

**)**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onPlay() {**

**if (!exoPlayer!!.playWhenReady || repeatMode) {**

**startService(Intent(baseContext, PlayerService::class.java))**

**if (musicRepository == null) return**

**val track = musicRepository?.getCurrent()**

**if (track == null) {**

**Toast.makeText(**

**applicationContext,**

**"Swipe Right to Left to add items to playlist",**

**Toast.LENGTH\_SHORT**

**).show()**

**return**

**}**

**updateMetadataFromTrack(track)**

**prepareToPlay(track.uri)**

**if (!audioFocusRequested) {**

**audioFocusRequested = true**

**val audioFocusResult = if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {**

**audioManager?.requestAudioFocus(audioFocusRequest!!)**

**} else {**

**audioManager?.requestAudioFocus(**

**audioFocusChangeListener,**

**AudioManager.STREAM\_MUSIC,**

**AudioManager.AUDIOFOCUS\_GAIN**

**)**

**}**

**if (audioFocusResult != AudioManager.AUDIOFOCUS\_REQUEST\_GRANTED)**

**return**

**}**

**mediaSession?.isActive = true**

**registerReceiver(**

**becomingNoisyReceiver,**

**IntentFilter(AudioManager.ACTION\_AUDIO\_BECOMING\_NOISY)**

**)**

**exoPlayer!!.playWhenReady = true**

**}**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**PlaybackStateCompat.STATE\_PLAYING,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**currentState = PlaybackStateCompat.STATE\_PLAYING**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**override fun onPause() {**

**if (exoPlayer!!.playWhenReady) {**

**exoPlayer?.playWhenReady = false**

**unregisterReceiver(becomingNoisyReceiver)**

**}**

**if (audioFocusRequested)**

**audioFocusRequested = false**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**PlaybackStateCompat.STATE\_PAUSED,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**currentState = PlaybackStateCompat.STATE\_PAUSED**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**override fun onStop() {**

**if (exoPlayer!!.playWhenReady) {**

**exoPlayer?.playWhenReady = false**

**unregisterReceiver(becomingNoisyReceiver)**

**}**

**if (audioFocusRequested) {**

**audioFocusRequested = false**

**if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {**

**audioManager?.abandonAudioFocusRequest(audioFocusRequest!!)**

**} else {**

**audioManager?.abandonAudioFocus(audioFocusChangeListener)**

**}**

**}**

**mediaSession?.isActive = false**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**PlaybackStateCompat.STATE\_STOPPED,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**currentState = PlaybackStateCompat.STATE\_STOPPED**

**refreshNotificationAndForegroundStatus(currentState)**

**//stopSelf()**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onSkipToNext() {**

**if (musicRepository == null) return**

**val track = musicRepository?.getNext()**

**if (track == null) {**

**Toast.makeText(**

**applicationContext,**

**"Swipe Right to Left to add items to playlist",**

**Toast.LENGTH\_SHORT**

**).show()**

**return**

**}**

**updateMetadataFromTrack(track)**

**prepareToPlay(track.uri)**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**currentState,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onSkipToPrevious() {**

**if (musicRepository == null) return**

**val track = musicRepository?.getPrevious()**

**if (track == null) {**

**Toast.makeText(**

**applicationContext,**

**"Swipe Right to Left to add items to playlist",**

**Toast.LENGTH\_SHORT**

**).show()**

**return**

**}**

**updateMetadataFromTrack(track)**

**prepareToPlay(track.uri)**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**currentState,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**override fun onSeekTo(pos: Long) {**

**exoPlayer?.seekTo(pos)**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**currentState,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onCustomAction(action: String?, extras: Bundle?) {**

**if (action == ACTION\_PLAY\_AT\_POSITION && extras != null) {**

**startService(Intent(baseContext, PlayerService::class.java))**

**if (musicRepository == null) return**

**val position = extras.getInt(EXTRA\_POSITION)**

**val track = musicRepository?.getAtPosition(position)**

**if (track == null) {**

**Toast.makeText(**

**applicationContext,**

**"Swipe Right to Left to add items to playlist",**

**Toast.LENGTH\_SHORT**

**).show()**

**return**

**}**

**updateMetadataFromTrack(track)**

**prepareToPlay(track.uri)**

**if (!audioFocusRequested) {**

**audioFocusRequested = true**

**val audioFocusResult = if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {**

**audioManager?.requestAudioFocus(audioFocusRequest!!)**

**} else {**

**audioManager?.requestAudioFocus(**

**audioFocusChangeListener,**

**AudioManager.STREAM\_MUSIC,**

**AudioManager.AUDIOFOCUS\_GAIN**

**)**

**}**

**if (audioFocusResult != AudioManager.AUDIOFOCUS\_REQUEST\_GRANTED)**

**return**

**}**

**mediaSession?.isActive = true**

**registerReceiver(**

**becomingNoisyReceiver,**

**IntentFilter(AudioManager.ACTION\_AUDIO\_BECOMING\_NOISY)**

**)**

**exoPlayer!!.playWhenReady = true**

**mediaSession?.setPlaybackState(**

**stateBuilder.setState(**

**PlaybackStateCompat.STATE\_PLAYING,**

**exoPlayer?.contentPosition!!,**

**1F**

**).build()**

**)**

**currentState = PlaybackStateCompat.STATE\_PLAYING**

**refreshNotificationAndForegroundStatus(currentState)**

**}**

**}**

**private fun prepareToPlay(uri: Uri) {**

**if (uri != currentUri.value || repeatMode) {**

**currentUri.value = uri**

**val mediaItem = MediaItem.fromUri(uri)**

**val mediaSource =**

**ProgressiveMediaSource.Factory(dataSourceFactory!!, extractorsFactory!!)**

**.createMediaSource(mediaItem)**

**exoPlayer?.setMediaSource(mediaSource)**

**exoPlayer?.prepare()**

**}**

**}**

**@SuppressLint("WrongConstant")**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**private fun updateMetadataFromTrack(track: Track) {**

**metadataBuilder.putString(MediaMetadataCompat.METADATA\_KEY\_TITLE, track.title)**

**metadataBuilder.putString(MediaMetadataCompat.METADATA\_KEY\_ALBUM, track.artist)**

**metadataBuilder.putString(MediaMetadataCompat.METADATA\_KEY\_ARTIST, track.artist)**

**metadataBuilder.putLong(MediaMetadataCompat.METADATA\_KEY\_DURATION, track.duration)**

**var metadata = metadataBuilder.build()**

**currentMetadata.value = metadata**

**mediaSession?.setMetadata(metadata)**

**MainScope().launch {**

**withContext(Dispatchers.IO) {**

**val mmr = MediaMetadataRetriever().apply { setDataSource(context, track.uri) }**

**val art = mmr.embeddedPicture**

**if (art != null)**

**metadataBuilder.putBitmap(**

**MediaMetadataCompat.METADATA\_KEY\_ART, BitmapFactory.decodeByteArray(**

**art, 0, art.size**

**)**

**)**

**else**

**metadataBuilder.putBitmap(**

**MediaMetadataCompat.METADATA\_KEY\_ART,**

**BitmapFactory.decodeResource(resources, R.drawable.without\_album)**

**)**

**}**

**metadata = metadataBuilder.build()**

**currentMetadata.value = metadata**

**mediaSession?.setMetadata(metadata)**

**}**

**}**

**}**

**private var audioFocusChangeListener: AudioManager.OnAudioFocusChangeListener =**

**object : AudioManager.OnAudioFocusChangeListener {**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onAudioFocusChange(focusChange: Int) {**

**when (focusChange) {**

**AudioManager.AUDIOFOCUS\_GAIN -> {**

**exoPlayer?.volume = 1F**

**}**

**AudioManager.AUDIOFOCUS\_LOSS\_TRANSIENT\_CAN\_DUCK -> {**

**exoPlayer?.volume = 0.3F**

**}**

**else -> mediaSessionCallback.onPause()**

**}**

**}**

**}**

**private val becomingNoisyReceiver: BroadcastReceiver = object : BroadcastReceiver() {**

**override fun onReceive(context: Context?, intent: Intent?) {**

**// Disconnecting headphones - stop playback**

**if (AudioManager.ACTION\_AUDIO\_BECOMING\_NOISY == intent?.action)**

**mediaSessionCallback.onPause()**

**}**

**}**

**private var exoPlayerListener: Player.EventListener = object : Player.EventListener {**

**override fun onTracksChanged(**

**trackGroups: TrackGroupArray,**

**trackSelections: TrackSelectionArray**

**) {**

**}**

**override fun onLoadingChanged(isLoading: Boolean) {}**

**@RequiresApi(Build.VERSION\_CODES.LOLLIPOP)**

**override fun onPlayerStateChanged(playWhenReady: Boolean, playbackState: Int) {**

**if (playWhenReady && playbackState == ExoPlayer.STATE\_ENDED) {**

**if (repeatMode) {**

**mediaSessionCallback.onPlay()**

**} else {**

**val isEnded = musicRepository?.isEnded()**

**mediaSessionCallback.onSkipToNext()**

**if (isEnded == true)**

**mediaSessionCallback.onPause()**

**}**

**}**

**}**

**override fun onPlayerError(error: ExoPlaybackException) {}**

**override fun onPlaybackParametersChanged(playbackParameters: PlaybackParameters) {}**

**}**

**override fun onBind(intent: Intent?): IBinder {**

**return PlayerServiceBinder()**

**}**

**inner class PlayerServiceBinder : Binder() {**

**fun getMediaSessionToken(): MediaSessionCompat.Token? {**

**return mediaSession?.sessionToken**

**}**

**fun getLiveMetadata(): LiveData<MediaMetadataCompat> {**

**return currentMetadata**

**}**

**fun getLiveUri(): LiveData<Uri> {**

**return currentUri**

**}**

**fun getService(): PlayerService {**

**return this@PlayerService**

**}**

**}**

**private fun refreshNotificationAndForegroundStatus(playbackState: Int) {**

**when (playbackState) {**

**PlaybackStateCompat.STATE\_PLAYING -> {**

**startForeground(NOTIFICATION\_ID, getNotification(playbackState))**

**NotificationManagerCompat.from(baseContext)**

**.notify(NOTIFICATION\_ID, getNotification(playbackState))**

**// cancel INACTIVITY\_TIMEOUT**

**inactivityCheckJob?.cancel()**

**}**

**PlaybackStateCompat.STATE\_PAUSED -> {**

**NotificationManagerCompat.from(baseContext)**

**.notify(NOTIFICATION\_ID, getNotification(playbackState))**

**// wait INACTIVITY\_TIMEOUT and stop player**

**inactivityCheckJob = MainScope().launch {**

**withContext(Dispatchers.Default) {**

**delay(INACTIVITY\_TIMEOUT)**

**}**

**mediaSessionCallback.onStop()**

**}**

**}**

**else -> {**

**}**

**}**

**}**

**private fun getNotification(playbackState: Int): Notification {**

**Log.d("state", playbackState.toString())**

**val builder = MediaStyleHelper.from(baseContext, mediaSession!!)**

**builder.addAction(**

**NotificationCompat.Action(**

**android.R.drawable.ic\_media\_previous, getString(R.string.previous),**

**MediaButtonReceiver.buildMediaButtonPendingIntent(**

**this,**

**PlaybackStateCompat.ACTION\_SKIP\_TO\_PREVIOUS**

**)**

**)**

**)**

**if (playbackState == PlaybackStateCompat.STATE\_PLAYING) {**

**builder.addAction(**

**NotificationCompat.Action(**

**android.R.drawable.ic\_media\_pause, getString(R.string.pause),**

**MediaButtonReceiver.buildMediaButtonPendingIntent(**

**this,**

**PlaybackStateCompat.ACTION\_PLAY\_PAUSE**

**)**

**)**

**)**

**} else {**

**builder.addAction(**

**NotificationCompat.Action(**

**android.R.drawable.ic\_media\_play, getString(R.string.play),**

**MediaButtonReceiver.buildMediaButtonPendingIntent(**

**this,**

**PlaybackStateCompat.ACTION\_PLAY\_PAUSE**

**)**

**)**

**)**

**}**

**builder.addAction(**

**NotificationCompat.Action(**

**android.R.drawable.ic\_media\_next, getString(R.string.next),**

**MediaButtonReceiver.buildMediaButtonPendingIntent(**

**this,**

**PlaybackStateCompat.ACTION\_SKIP\_TO\_NEXT**

**)**

**)**

**)**

**builder.setStyle(**

**androidx.media.app.NotificationCompat.MediaStyle()**

**.setShowActionsInCompactView(0, 1, 2)**

**.setShowCancelButton(true)**

**.setCancelButtonIntent(**

**MediaButtonReceiver.buildMediaButtonPendingIntent(**

**this,**

**PlaybackStateCompat.ACTION\_STOP**

**)**

**)**

**.setMediaSession(mediaSession?.sessionToken) // setMediaSession required for Android Wear**

**)**

**builder.setSmallIcon(R.mipmap.ic\_launcher)**

**builder.color = ContextCompat.getColor(**

**this,**

**R.color.design\_default\_color\_primary\_dark**

**) // The whole background (in MediaStyle), not just icon background**

**builder.setShowWhen(false)**

**builder.setContentTitle(resources.getString(R.string.app\_name))**

**builder.priority = NotificationCompat.PRIORITY\_HIGH**

**builder.setNotificationSilent()**

**builder.setChannelId(NOTIFICATION\_DEFAULT\_CHANNEL\_ID)**

**return builder.build()**

**}**

**}**

**Приложение 5. MusicRepository.kt**

**package com.kmem.myplayer.service**

**import android.content.Context**

**import android.net.Uri**

**import com.kmem.myplayer.data.AppDatabase**

**import com.kmem.myplayer.data.Track**

**import kotlinx.coroutines.Dispatchers**

**import kotlinx.coroutines.withContext**

**/\*\***

**\* Предоставляет данные для сервиса. Сервис делегирует запросы к БД этому классу.**

**\*/**

**class MusicRepository(val context: Context) {**

**private var data: ArrayList<Track> = ArrayList()**

**private var shuffle\_data: ArrayList<Track> = ArrayList() // additional copy for shuffle mode**

**private var shuffle\_stack: ArrayList<Track> = ArrayList() // stack trace for shuffle mode**

**private var stack\_index = 0**

**private var maxIndex = 0**

**var currentUri: Uri? = null**

**var currentItemIndex = 0**

**var shuffle = false**

**set(value) {**

**field = value**

**val isAlreadyShuffled = shuffle\_stack.size == 1 && shuffle\_stack[0].uri == currentUri**

**if (value && !isAlreadyShuffled) {**

**shuffle\_data.clear()**

**shuffle\_data.addAll(data)**

**shuffle\_data.shuffle()**

**shuffle\_stack.clear()**

**shuffle\_stack.add(data.first { it.uri == currentUri })**

**shuffle\_data.remove(data.first { it.uri == currentUri })**

**stack\_index = 0**

**}**

**}**

**init {**

**data.addAll(AppDatabase.getInstance(context).trackDao().getTracks())**

**maxIndex = data.lastIndex**

**}**

**suspend fun updatePositions() {**

**withContext(Dispatchers.IO) {**

**data.clear()**

**data.addAll(AppDatabase.getInstance(context).trackDao().getTracks())**

**currentItemIndex = data.indexOfFirst { it.uri == currentUri }**

**if (currentItemIndex == -1)**

**currentItemIndex = 0**

**}**

**}**

**suspend fun addNewTracks() {**

**withContext(Dispatchers.IO) {**

**data.clear()**

**data.addAll(AppDatabase.getInstance(context).trackDao().getTracks())**

**maxIndex = data.lastIndex**

**val isAlreadyShuffled = shuffle\_stack.size == 1 && shuffle\_stack[0].uri == currentUri**

**if (shuffle || isAlreadyShuffled) {**

**shuffle\_data.clear()**

**shuffle\_data.addAll(data)**

**shuffle\_data.removeAll(shuffle\_stack)**

**shuffle\_data.shuffle()**

**}**

**}**

**}**

**suspend fun deleteTracks(tracks: ArrayList<Track>) {**

**data.clear()**

**withContext(Dispatchers.IO) {**

**data.addAll(AppDatabase.getInstance(context).trackDao().getTracks())**

**}**

**maxIndex = data.lastIndex**

**var indexOfCurrentTrack = data.indexOfFirst { it.uri == currentUri }**

**// track that was playing is deleted**

**if (indexOfCurrentTrack == -1) {**

**if (currentItemIndex > maxIndex)**

**currentItemIndex = maxIndex**

**} else {**

**currentItemIndex = indexOfCurrentTrack**

**}**

**val isAlreadyShuffled = shuffle\_stack.size == 1 && shuffle\_stack[0].uri == currentUri**

**if (shuffle || isAlreadyShuffled) {**

**shuffle\_data.removeAll(tracks)**

**shuffle\_stack.removeAll(tracks)**

**indexOfCurrentTrack = shuffle\_stack.indexOfFirst { it.uri == currentUri }**

**if (indexOfCurrentTrack == -1) {**

**if (stack\_index > shuffle\_stack.lastIndex)**

**stack\_index = shuffle\_stack.lastIndex**

**} else {**

**stack\_index = indexOfCurrentTrack**

**}**

**}**

**}**

**fun getNext(): Track? {**

**if (shuffle) {**

**return getNextOnShuffle()**

**}**

**if (currentItemIndex == maxIndex)**

**currentItemIndex = 0**

**else**

**currentItemIndex++**

**return getCurrent()**

**}**

**private fun getNextOnShuffle(): Track {**

**val track: Track?**

**if (stack\_index == shuffle\_stack.size - 1) {**

**if (shuffle\_data.isEmpty())**

**refreshShuffleData()**

**track = shuffle\_data.removeAt(0)**

**shuffle\_stack.add(track)**

**stack\_index++**

**} else {**

**track = shuffle\_stack[++stack\_index]**

**}**

**currentUri = track.uri**

**currentItemIndex = data.indexOf(track)**

**return track**

**}**

**private fun refreshShuffleData() {**

**shuffle\_data.addAll(shuffle\_stack)**

**shuffle\_data.shuffle()**

**shuffle\_stack.clear()**

**shuffle\_stack.add(shuffle\_data.removeAt(0))**

**stack\_index = 0**

**}**

**fun getPrevious(): Track? {**

**if (shuffle)**

**return getPreviousOnShuffle()**

**if (currentItemIndex == 0)**

**currentItemIndex = maxIndex**

**else**

**currentItemIndex--**

**return getCurrent()**

**}**

**private fun getPreviousOnShuffle(): Track {**

**if (stack\_index != 0) stack\_index--**

**val track = shuffle\_stack[stack\_index]**

**currentUri = track.uri**

**currentItemIndex = data.indexOf(track)**

**return track**

**}**

**fun getAtPosition(pos: Int): Track {**

**val track = data[pos]**

**if (shuffle) {**

**if (track in shuffle\_stack) {**

**shuffle\_stack.remove(track)**

**shuffle\_stack.add(track)**

**} else {**

**shuffle\_data.remove(track)**

**shuffle\_stack.add(track)**

**}**

**stack\_index = shuffle\_stack.lastIndex**

**}**

**currentItemIndex = pos**

**currentUri = track.uri**

**return track**

**}**

**fun getCurrent(): Track? {**

**if (maxIndex == -1) return null**

**currentUri = data[currentItemIndex].uri**

**return data[currentItemIndex]**

**}**

**fun isEnded(): Boolean {**

**return if (shuffle) {**

**shuffle\_data.isEmpty()**

**} else {**

**currentItemIndex == maxIndex**

**}**

**}**

**}**