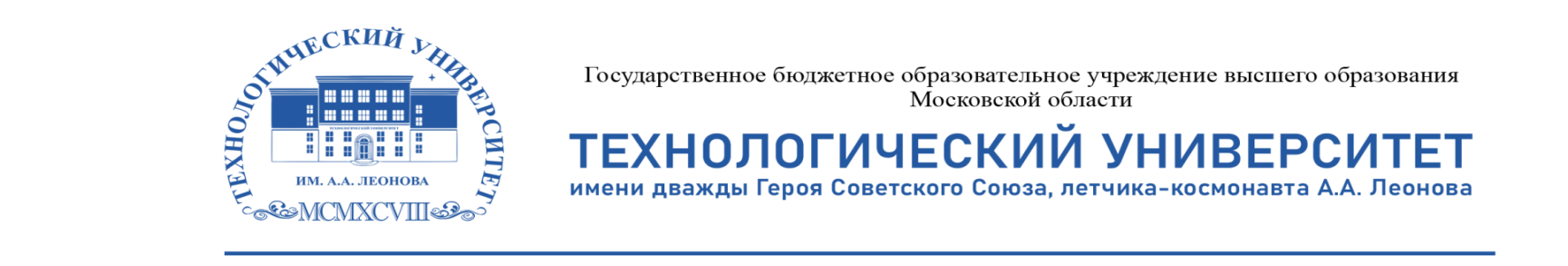
****

Колледж космического машиностроения и технологии

**Курсовой проект**

**Разработка андроид-приложения «Аудио-плеер».**

Пояснительная записка

КП.09.02.03.22.13ПЗ

Обучающийся группы П2-19 Оболонков А.В.

Руководитель курсового проекта Гусятинер Л.Б.

Результат защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Королев, 2022 г.

# Оглавление

[**Введение 3**](#_Toc106791889)

[**1. Теоретическая часть 4**](#_Toc106791890)

[**1.1. Описание предметной области 4**](#_Toc106791891)

[**1.2. Описание существующих разработок 6**](#_Toc106791892)

[**1.2.1. BlackPlayer 6**](#_Toc106791893)

[**1.2.2. AIMP 8**](#_Toc106791894)

[**2. Проектная часть 10**](#_Toc106791895)

[**2.1. Построение диаграммы прецедентов 10**](#_Toc106791896)

[**2.2. Выбор инструментов 11**](#_Toc106791897)

[**2.3. Проектирование сценария 12**](#_Toc106791898)

[**2.4. Диаграмма классов 13**](#_Toc106791899)

[**2.5. Описание главного модуля 15**](#_Toc106791900)

[**2.6. Описания спецификаций к модулям 24**](#_Toc106791901)

[**2.7. Описание модулей 25**](#_Toc106791902)

[**2.8. Описание тестовых наборов модулей 30**](#_Toc106791903)

[**2.9. Описание применения средств отладки 35**](#_Toc106791904)

[**3. Эксплуатационная часть 36**](#_Toc106791905)

[**3.1. Руководство оператора 36**](#_Toc106791906)

[**Заключение 42**](#_Toc106791907)

[**Список литературы и интернет-источников 43**](#_Toc106791908)

[**Приложение 1. Код главного модуля MainActivity.kt 44**](#_Toc106791909)

[**Приложение 2. Код модуля AudioClass.kt 54**](#_Toc106791910)

[**Приложение 3. Код модуля LibraryFragment.kt 56**](#_Toc106791911)

[**Приложение 4. Код модуля SearchFragment.kt 59**](#_Toc106791912)

[**Приложение 5. Код модуля SettingsFragment.kt 62**](#_Toc106791913)

# Введение

Целью данного курсового проекта является написание программы «Аудио-плеер» для прослушивания аудиофайлов на Android устройстве. Тема «Разработка андроид-приложения «Аудио-плеер»» была выбрана, поскольку не каждый производитель телефонов или сторонние аудиоплееры способны предоставить пользователю интуитивно понятный дизайн и простое управление воспроизведением, также многие добавляют в свои приложения рекламу, которая способна отвлекать пользователя от прослушивания музыки. Данный курсовой проект позволит пользователю приложения прослушать любые аудиофайлы на своем мобильном телефоне, такие как музыка или аудио эффекты. Также в данном проекте будет простой для понимания интерфейс и небольшой порог вхождения.

В первой части (теоретической) будет рассмотрена предметная область данной темы, а также несколько аналогичных продуктов, с их плюсами и минусами.

Во второй части (проектной) будет рассмотрен выбор инструментов и сценарий программы. Будет рассмотрен главный модуль, спецификации к модулям, дополнительные модули и применения средств откладки.

В третьей части (эксплуатационной) будет рассмотрено руководство оператора и To-Do лист.

В заключительной части будет представлен список используемой литературы и интернет ресурсов, а также сделаны общие выводы о получившемся проекте.

# Теоретическая часть

## Описание предметной области

Медиаплеер — компьютерная программа, предназначенная для воспроизведения файлов мультимедиа-содержимого.

Одна разновидность медиаплееров предназначена для воспроизведения только аудио- или же видеофайлов, которые называются, соответственно, — аудиоплеер и видеоплеер.

В основном все аудиоплееры делятся на несколько типов: простые (для работы на слабых ПК или мобильных устройствах), универсальные и специализированные (специфические и профессиональные программы для решения уникальных задач). Разработчики таких плееров стремятся сделать их как можно более удобными для воспроизведения соответствующих форматов.

Рынок аудио-проигрывателей изобилует множеством различающихся друг от друга плееров. Каждый пользователь выбирает аудиоплеер под свои нужды, поскольку различные аудиоплееры могут предложить различные функции, такие как:

1. Работа с различными плейлистами и списками. Формирование плейлистов на основании альбома, группы, музыканта, композитора, сборника, а также возможность пользователю создать свои собственные плейлисты.
2. Сортировка аудиофайлов в плейлистах.
3. Чтение или редактирование метаданных аудиофайлов.
4. Поиск определённого аудиофайла по его названию или по информации о файле (альбом, группа, музыкант, композитор)
5. Работа с потоковым звуком – интернет-радио или сервисы потокового вещания.

Еще могут существовать всевозможные дополнительные функции:

1. Изменение внешнего вида приложения,
2. Вывод всплывающих информационных окон.
3. Поддержка спецэффектов аудио.
4. Управление воспроизведением с помощью уведомлений, жестов или голосового помощника.
5. Визуализация звука спектроанализаторами или иными визуализаторами.

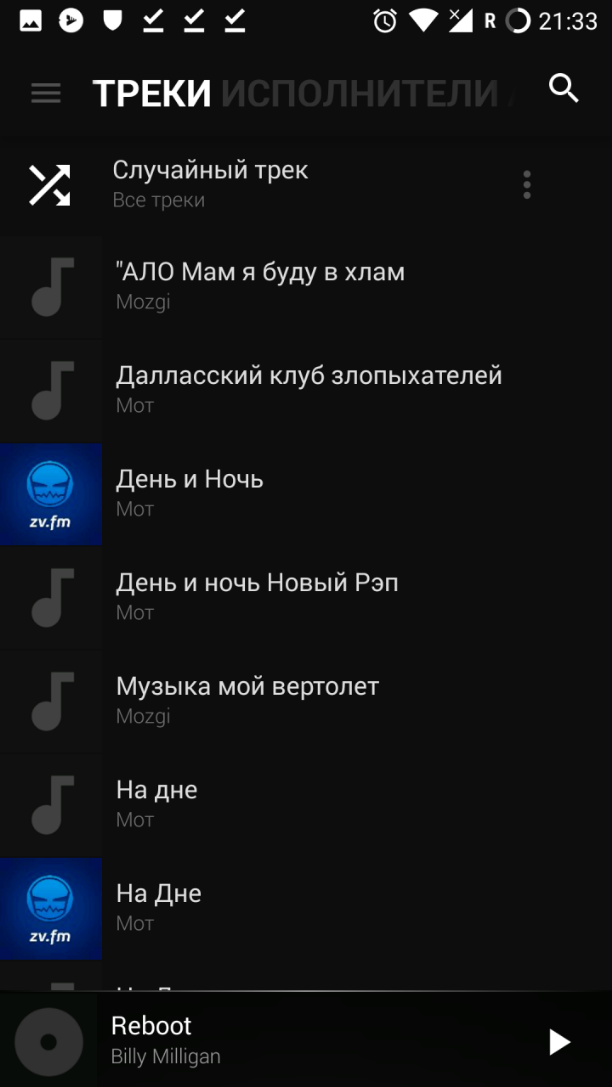
## Описание существующих разработок

В этом разделе рассмотрены некоторые уже имеющиеся программы для воспроизведения аудиофайлов.

Интерфейс аудиоплееров чаще всего простой и понятный для обычных пользователей, а продвинутые пользователи смогут найти дополнительные настройки и функции для улучшения качества звука или более удобного управления воспроизведением или плейлистами.

### BlackPlayer

BlackPlayer — минималистичный плеер с широкими возможностями настройки интерфейса. На рисунке 1 изображен интерфейс программы. BlackPlayer позволяет менять цвета, шрифты, а также настраивать варианты отображения обложек альбомов, жанров, композиторов или артистов. Есть поддержка автоматической загрузки метаданных, включая текст композиции. [[2]](https://overclockers.ru/lab/show/84348_3/vybiraem-audiopleer-dlya-android-ustrojstv-musixmatch-music-player-leopard-v7-i-blackplayer)

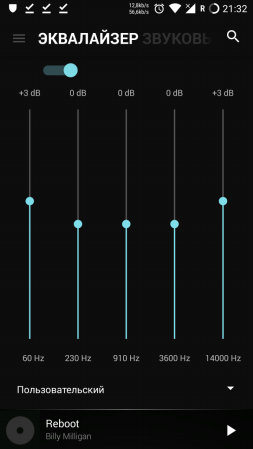


*Рисунок 1.* *Интерфейс BlackPlayer*

Ключевые функции BlackPlayer:

1. Поддерживает стандартные форматы аудиофайлов, такие как mp3, wav, ogg;
2. Включает встроенный эквалайзер, усилитель баса и 3D виртуалайзер.
3. Поддерживает воспроизведение формата сжатия без потерь FLAC (4.1+);
4. Возможность воспроизведения треков gapless (без прерываний).
5. Просмотр и редактирование встроенных слов песнен.
6. Эффективная система управления обложками альбомов, автоматическая и ручная.
7. Изменяемые темы, шрифты и цвета.
8. Три версии виджета для главного экрана.
9. Таймер сна.

Пользователь может скорректировать звучание через эквалайзер, воспользоваться таймером сна, создавать плейлисты, а также каталогизировать музыкальную библиотеку. Бесплатный BlackPlayer не имеет рекламы, разблокировка премиум версии с дополнительным функционалом стоит 319 рублей.

**

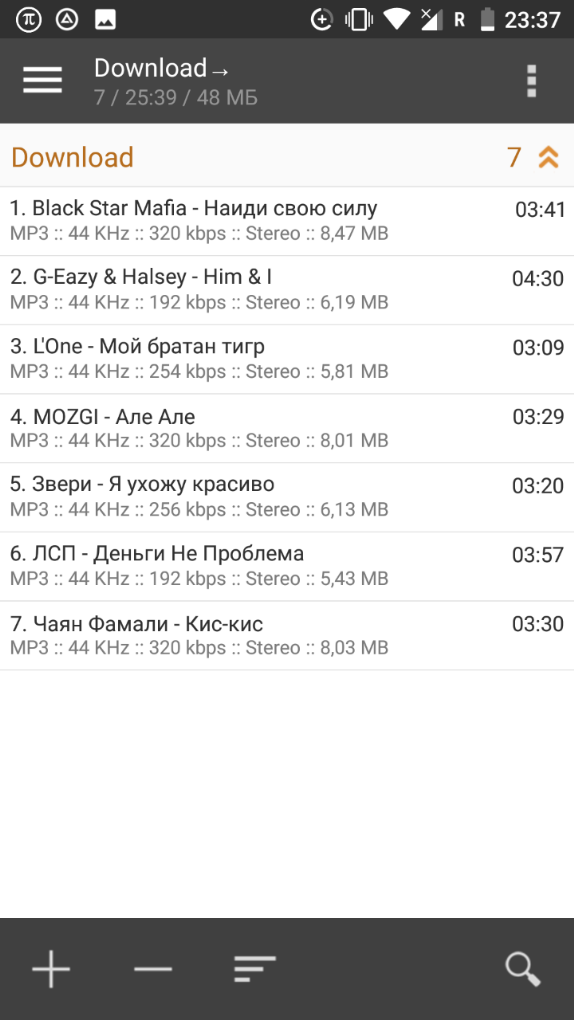
*Рисунок 2. Эквалайзер в BlackPlayer*

### AIMP

Один из самых популярных плееров, который можно скачать на телефон и на компьютер одновременно. Он поддерживает большинство распространенных музыкальных форматов (в том числе: FLAC, ALAC, MP4). Достоинства AIMP — простой интерфейс, который можно изменить с помощью различных тем от производителя или других пользователей, гибкий эквалайзер, возможность трансляции музыки по HTTP. Рекламы в AIMP нет, что делает плеер предпочтительнее большинства конкурентов. [[3]](https://aimp-download.ru/obzor-pleera-aimp-dlya-android)

Возможности AIMP:

1. Поддержка огромного количества форматов, в том числе «.mp3», «.wav», «.mp4», «.ogg» и так далее.
2. Возможность изменять тему программы для того, чтобы работа с программой доставляла больше комфорта.
3. Поддержка работы сразу с несколькими плейлистами.
4. Регулирование скорости воспроизведения.
5. Добавление отдельных файлов и папок в тот или иной плейлист.
6. Специальный виджет для экрана блокировки Android-устройства.
7. Виджеты для рабочего стола различных размеров.



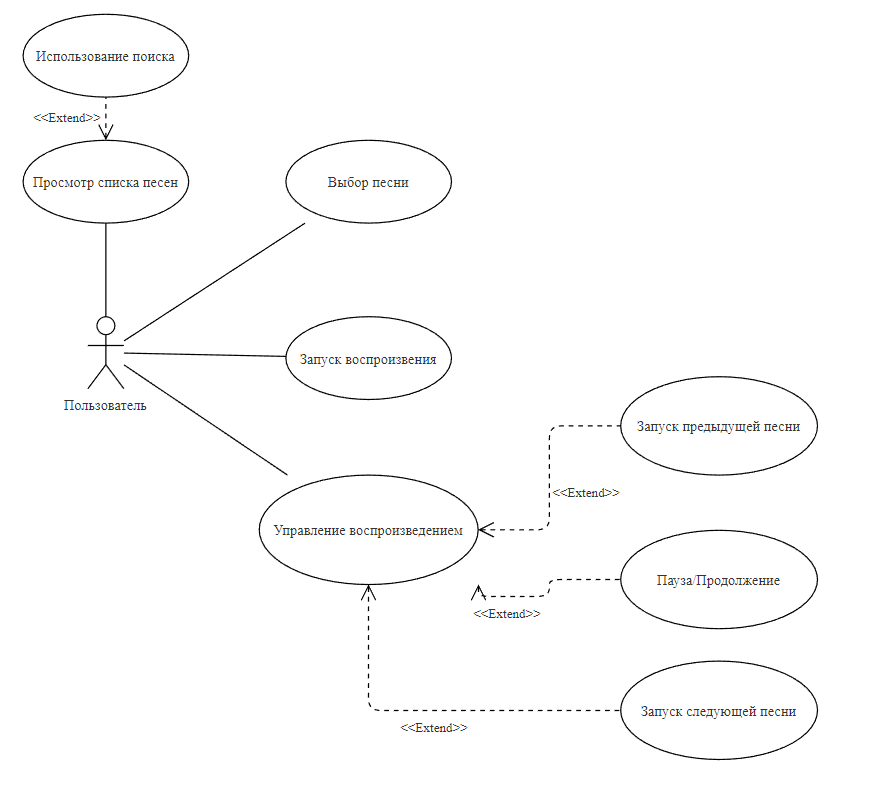
*Рисунок 3. Интерфейс AIMP*

# Проектная часть

## Построение диаграммы прецедентов

На рисунке 4 представлена диаграмма прецедентов. На ней показаны все возможные действия пользователя при использовании программы.

Пользователь при работе с приложением взаимодействует со списком песен. Через него он может включить воспроизведение желаемой песни. Также он может управлять воспроизведением с помощью кнопок управления, или перейти в меню поиска для того, чтобы найти необходимую песню.



*Рисунок 4. Диаграмма прецедентов работы с приложением.*

## Выбор инструментов

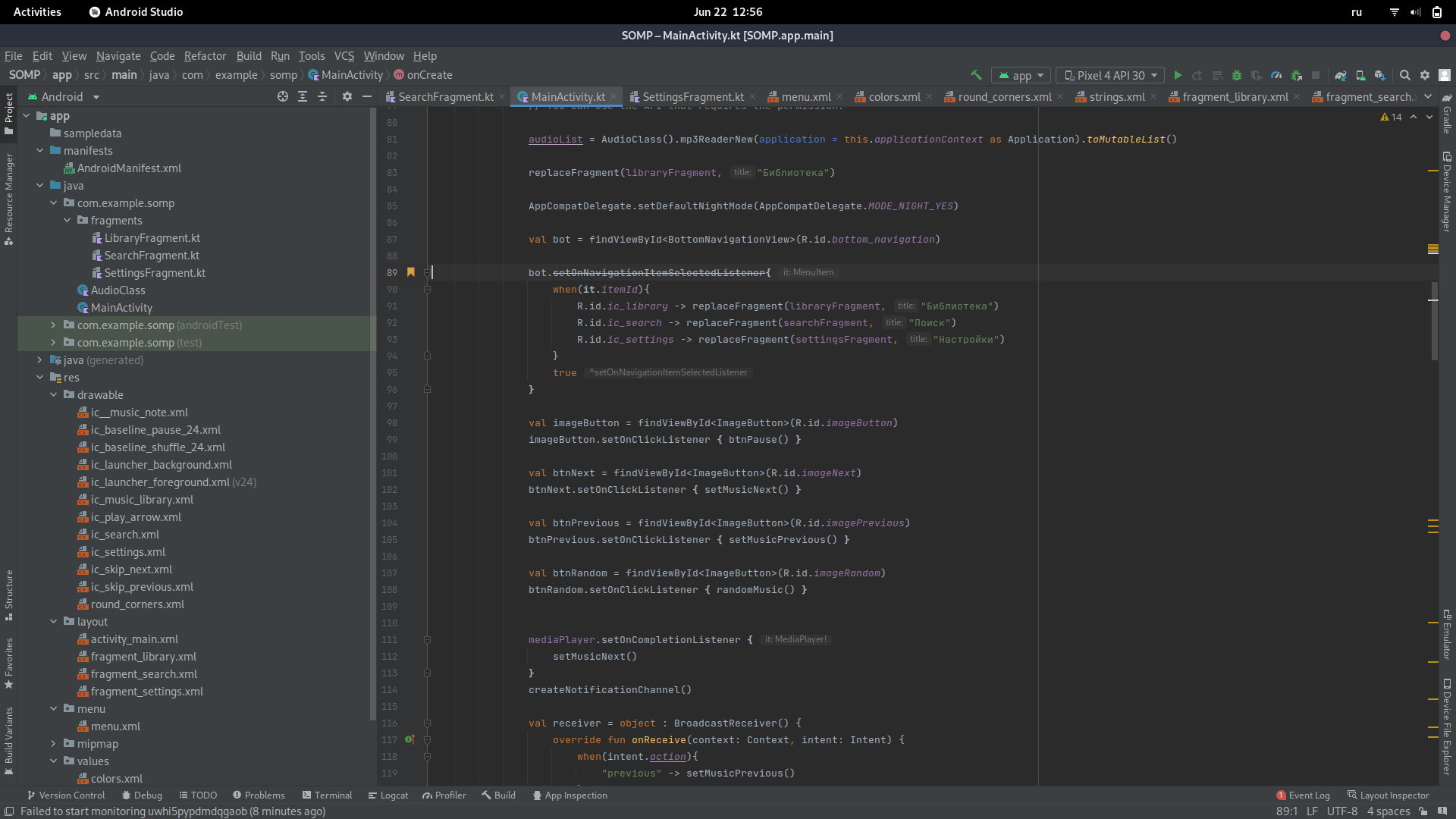
При выборе инструментов сравнивались все достоинства и недостатки каждого инструмента, знания и время на обучение.

Для разработки проекта был выбран язык программирования Kotlin. Он имеет схожий синтаксис как у Java и C#.

Также компания Google объявила, что язык программирования Kotlin является предпочтительным языком для разработчиков приложений под Android, что тоже было учтено при выборе языка программирования.

«На конференции для разработчиков I/O 2019 Google объявила, что язык программирования Kotlin теперь является предпочтительным языком для разработчиков приложений под Android — новые API и библиотеки Jetpack будут публиковаться сначала на Kotlin.» [[4]](https://vc.ru/dev/66728-google-obyavila-kotlin-prioritetnym-yazykom-programmirovaniya-dlya-razrabotki-android-prilozheniy)

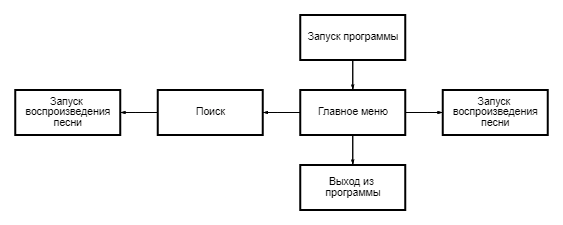
В качестве среды разработки была выбрана Android Studio, как так только на ней можно писать приложения под OC Android, и она полностью сделана под Android-разработку. На рисунке 5 показано окно разработки в Android Studio.



*Рисунок 5. Разработка в Android Studio*

## Проектирование сценария

В данном разделе приведен сценарий использования программы пользователем.



*Рисунок 6. Сценарий использования*

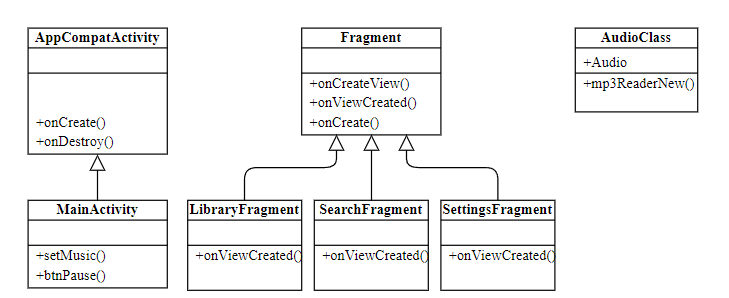
Пользователь запускает программу и открывается список всех песен, найденных на устройстве. При нажатии на название песни начнется её воспроизведение.

При нажатии на кнопку с иконкой лупы можно переключиться на окно с поиском. Точно также можно переключиться на окно со списком песен, нажав на кнопку с иконкой ноты.

Завершение программы происходит после закрытия из меню ранее запущенных приложений.

## Диаграмма классов

В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.



*Рисунок 7. Диаграмма классов для проекта.*

Класс «AppCompatActivity» является базовым классом для активностей, и имеет множество публичных полей и методов.

Класс «MainActivity» является наследником класса «AppCompatActivity» и содержит публичные методы setMusic(), который позволяет начать воспроизведение песни, а также btnPause(), который приостанавливает или продолжает воспроизведение.

Класс «Fragment» является базовым классом для фрагментов, и имеет множество публичных полей и методов.

Класс «LibraryFragment» является наследником класса «Fragment» и содержит публичный метод onViewCreated(), который вызывается при создании фрагмента “Библиотека”.

Класс «SearchFragment» является наследником класса «Fragment» и содержит публичный метод onViewCreated(), который вызывается при создании фрагмента “Поиск”.

Класс «SettingsFragment» является наследником класса «Fragment» и содержит публичный метод onViewCreated(), который вызывается при создании фрагмента “Настройки”.

Класс AudioClass содержит публичное поле Audio и публичный метод mp3ReaderNew, который считывает с Android-устройства все аудиофайлы.

## Описание главного модуля

В главный модуль входит класс MainActivity который отвечает за функционал программы, а также является первым экраном, появляющимся при запуске приложения пользователем. Полный код главного модуля находится в приложении 1.

В классе MainActivity реализованы следующие методы:

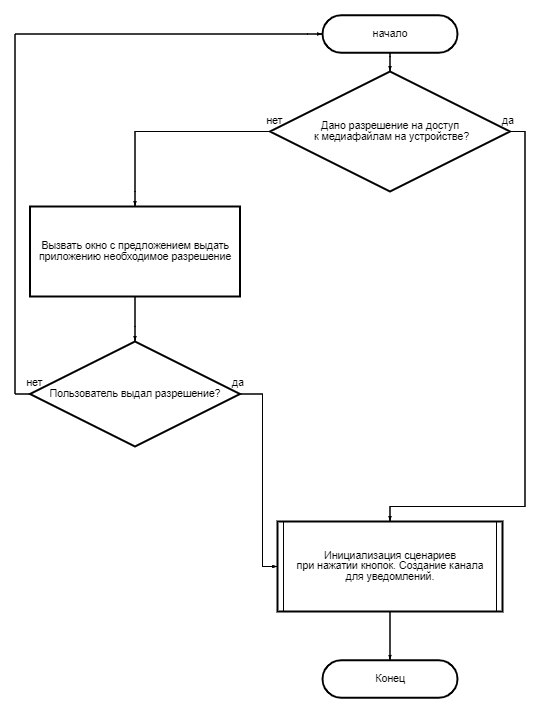
**Листинг 1. Обработка вызова состояния onCreate.**

Полный код метода находится в приложении 1.

Данный метод срабатывает, когда система создает приложение при запуске. В нем происходит проверка на наличие у приложения разрешения в системе для доступа к медиафайлам на устройстве. Если разрешения нет, метод вызовет другой метод для получения разрешения у пользователя, в ином случае сообщит, что пользоваться приложением невозможно без предоставленного разрешения.

После получения разрешения метод вызывает необходимые методы для работы приложения, такие как создание канала для вывода уведомлений, а также регистрирует в системе три BroadcastReceiver, которые получают и обрабатывают нажатия кнопок управления воспроизведением в уведомлении.

Данный метод отслеживает все действия пользователя и в зависимости от конкретного действия вызывает нужный метод. Блок-схема данного метода показана на рисунке 8.



*Рисунок 8. Блок-схема метода onCreate*

**Листинг 2. Отмена регистрации приемников.**

private fun unRegisterReceiver(r1:BroadcastReceiver, r2:BroadcastReceiver, r3:BroadcastReceiver){  
 this.unregisterReceiver(r1)  
 this.unregisterReceiver(r2)  
 this.unregisterReceiver(r3)  
 }

Данный метод отменяет регистрацию трех BroadcastReceiver, которые были зарегистрированы в системе при запуске программы.

**Листинг 3.** **Обработка вызова состояния onDestroy.**

override fun onDestroy() {  
 super.onDestroy()  
 unRegisterReceiver(receiver, receiver2, receiver3)  
}

Данный метод вызывается перед уничтожением приложения и вызывает метод unRegisterReceiver, о котором говорилось выше в Листинг 2.

**Листинг 4. Создание всплывающего окна с предупреждением о отсутствии разрешения на доступ к медиафайлам.**

private fun createAlertMessage(){  
 val builder: AlertDialog.Builder = this.*let* **{** AlertDialog.Builder(**it**)  
 **}** builder.setMessage("Без этого разрешения приложение не сможет просматривать музыку с вашего устройства. Вы хотите дать разрешение?")  
 ?.setTitle("Доступ запрещён")  
  
 builder.*apply* **{** setPositiveButton("Предоставить разрешение"  
 ) **{** \_, \_ **->** // Пользователь нажал кнопку ОК  
 val intent = Intent(*ACTION\_APPLICATION\_DETAILS\_SETTINGS*)  
 val uri = Uri.fromParts(  
 "package",  
 *packageName*, null  
 )  
 intent.*data* = uri  
 startActivity(intent)  
 **}** setNegativeButton("Отмена"  
 ) **{** \_, \_ **->** //Пользователь отменил диалог  
 **}  
 }** builder.create()  
 builder.show()  
}

Данный метод создает и вызывает всплывающее окно с предупреждением о отсутствии разрешения и двумя кнопками, кнопка “Предоставить разрешение” вызывает окно настроек Android и открывает настройки приложения.

**Листинг 5. Создание канала для уведомлений.**

private fun createNotificationChannel() {  
 // Создаем NotificationChannel, но только в API 26+, потому что  
 // класс NotificationChannel новый, его нет в библиотеке поддержки  
 if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  
 val name = "Music"  
 val descriptionText = "Music"  
 val importance = NotificationManager.*IMPORTANCE\_DEFAULT* val channel = NotificationChannel(CHANNEL\_ID, name, importance).*apply* **{** *description* = descriptionText  
 **}** // Регистрируем канал в системе  
 val notificationManager: NotificationManager =  
 getSystemService(Context.*NOTIFICATION\_SERVICE*) as NotificationManager  
 notificationManager.createNotificationChannel(channel)  
 }  
}

Данный метод регистрирует канал для вызова уведомлений, поскольку начиная с Android 8.0 (уровень API 26) все уведомления должны быть назначены каналу.

**Листинг 6. Создание уведомления с кнопками управления воспроизведением.**

private fun createMusicNotification(musicName: String, musicData: String){  
  
 val pauseIntent = Intent()  
 pauseIntent.*action* = "pause"  
 val pausePendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, pauseIntent, 0)  
 val previousIntent = Intent()  
 previousIntent.*action* = "previous"  
 val previousPendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, previousIntent, 0)  
 val nextIntent = Intent()  
 nextIntent.*action* = "next"  
 val nextPendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, nextIntent, 0)  
 val notification = NotificationCompat.Builder(this, CHANNEL\_ID)  
 // Show controls on lock screen even when user hides sensitive content.  
 .setVisibility(NotificationCompat.*VISIBILITY\_PUBLIC*)  
 .setSmallIcon(R.drawable.*ic\_\_music\_note*)  
 .addAction(R.drawable.*ic\_skip\_previous*, "previous", previousPendingIntent)  
 .addAction(R.drawable.*ic\_baseline\_pause\_24*, "pause", pausePendingIntent)  
 .addAction(R.drawable.*ic\_skip\_next*, "next", nextPendingIntent)  
 .setStyle(MediaNotificationCompat.MediaStyle()  
 .setShowActionsInCompactView(0, 1, 2))  
 // Apply the media style template  
 .setContentTitle(musicName)  
 .setContentText(musicData)  
 .setCategory(NotificationCompat.*CATEGORY\_TRANSPORT*)  
  
  
 *with*(NotificationManagerCompat.from(this)) **{** notify(NOTIFICATION\_ID, notification.build()) // посылаем уведомление  
 **}**}

Данный метод создает и вызывает уведомление с кнопками управления воспроизведением, в названии указывая название аудиофайла и в описании указывая имя музыканта из метаданных файла.

**Листинг 7. Включение и выключение случайного воспроизведения.**

private fun randomMusic(){  
 val btnRandom = findViewById<ImageButton>(R.id.*imageRandom*)  
 isRandom = if (!isRandom){  
 audioList.*shuffle*()  
 btnRandom.setColorFilter(Color.*GRAY*)  
 true  
 }  
 else{  
 audioList.*sortBy* **{ it**.name **}** btnRandom.setColorFilter(Color.*WHITE*)  
 false  
 }  
}

Данный метод случайно перемешивает список всех аудиофайлов при включении случайного воспроизведения, или сортирует его по названию файлов при выключении.

**Листинг 8. Включение следующего трека, относительно текущего.**

private fun setMusicNext(){  
 val musicName = textView.*text*.toString().plus(".mp3")  
 val num = audioList.*indexOf*(audioList.*find* **{ it**.name == musicName **}**)  
 if (num < audioList.*count*()-1){  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.*EXTERNAL\_CONTENT\_URI*, audioList[num+1].id)  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.*text* = audioList[num+1].name.*replace*(".mp3", "")  
  
 val drawable = getDrawable(R.drawable.*ic\_baseline\_pause\_24*)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num+1].name.*replace*(".mp3", ""), audioList[num+1].artist)  
 }  
 else{  
 setMusic(audioList[0].name, audioList)  
 }  
}

Данный метод запускает воспроизведение следующего по списку трека.

**Листинг 9. Включение предыдущего трека, относительно текущего.**

private fun setMusicPrevious(){  
 val musicName = textView.*text*.toString().plus(".mp3")  
 val num = audioList.*indexOf*(audioList.*find* **{ it**.name == musicName **}**)  
 if (num > 0){  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.*EXTERNAL\_CONTENT\_URI*, audioList[num-1].id)  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.*text* = audioList[num-1].name.*replace*(".mp3", "")  
  
  
 val drawable = getDrawable(R.drawable.*ic\_baseline\_pause\_24*)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num-1].name.*replace*(".mp3", ""), audioList[num-1].artist)  
 }  
 else{  
 setMusic(audioList[audioList.*count*()-1].name, audioList)  
 }  
}

Данный метод запускает воспроизведение предыдущего по списку трека.

**Листинг 10. Включение определенного трека.**

fun setMusic(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 val num = audioList.*indexOf*(audioList.*find* **{ it**.name == songName **}**)  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.*EXTERNAL\_CONTENT\_URI*, audioList[num].id)  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.*text* = audioList[num].name.*replace*(".mp3", "")  
  
 val drawable = getDrawable(R.drawable.*ic\_baseline\_pause\_24*)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num].name.*replace*(".mp3", ""), audioList[num].artist)  
 }  
 catch (e: Exception){  
 Toast.makeText(this, e.toString(), Toast.*LENGTH\_LONG*).show()  
 }  
}

Данный метод запускает воспроизведение трека по его названию.

**Листинг 11. Приостановка трека.**

fun btnPause(){  
 try {  
 isPaused = if(!mediaPlayer.*isPlaying* && mediaPlayer.*duration* != 0) {  
 val drawable = getDrawable(R.drawable.*ic\_baseline\_pause\_24*)  
 imageButton.setImageDrawable(drawable)  
 mediaPlayer.start()  
 false  
 } else {  
 val drawable = getDrawable(R.drawable.*ic\_play\_arrow*)  
 imageButton.setImageDrawable(drawable)  
 mediaPlayer.pause()  
 true  
 }  
 }  
 catch (e: Exception){  
 Toast.makeText(this, e.toString(), Toast.*LENGTH\_LONG*).show()  
 }  
}

Данный метод приостанавливает воспроизведение текущего трека, или включает воспроизведение, если текущий трек уже приостановлен.

**Листинг 12. Смена фрагмента экрана.**

private fun replaceFragment(fragment: Fragment, title: String){  
 val transaction = *supportFragmentManager*.beginTransaction()  
 transaction.replace(R.id.*frame\_container*, fragment)  
 transaction.commit()  
  
 *supportActionBar*?.*title* = title  
}

С помощью данного метода осуществляется переход из текущего фрагмента экрана в другие, такие как главное меню, поиск трека или настройки, при нажатии на кнопки выбора фрагмента.

## Описания спецификаций к модулям

В данном разделе описаны публичные члены модулей курсового проекта.

**MainActivity.kt**

В главном модуле (MainActivity.kt) содержатся публичные методы класса MainActivity, который наследуется от класса AppCompatActivity, являющимся базовым классом для AppCompatActivity: методы setMusic и btnPause, которые были описаны в разделе 2.5.

**AudioClass.kt**

В модуле, который реализует класс данных Audio и работу с получением всех аудиофайлов (AudioClass.kt) содержится один публичный метод класса AudioClass, который будет описан ниже в разделе 2.7, в данном разделе он описан не будет.

## Описание модулей

Кроме главного модуля программа содержит модули, который реализует класс данных Audio и чтение аудиофайлов с устройства пользователя.

Полный код модуля AudioClass.kt приведен в приложении 2.

Также интерфейс программы разбит на три фрагмента, реализованные в трех отдельных модулях:

Полный код модуля LibraryFragment.kt приведен в приложении 3.

Полный код модуля SeacrhFragment.kt приведен в приложении 4.

Полный код модуля SettingsFragment.kt приведен в приложении 5.

В этом разделе приведены все методы, которые содержатся в классе AudioClass из модуля AudioClass.kt, а также методы, которые содержатся в фрагментах LibraryFragment.kt и SearchFragment.kt.

**Модуль AudioClass.kt**

**Листинг 13. Метод чтения аудиофайлов с устройства.**

fun mp3ReaderNew(application: Context): List<Audio>{  
  
 val audioProjection = *arrayOf*(  
 MediaStore.Audio.Media.*\_ID*,  
 MediaStore.Audio.Media.*DISPLAY\_NAME*,  
 MediaStore.Audio.Media.*ARTIST*,  
 MediaStore.Audio.Media.*DATA*,  
 MediaStore.Audio.Media.*DURATION*,  
 )  
  
 val list = *mutableListOf*<Audio>()  
 application.*contentResolver*.query(  
 MediaStore.Audio.Media.*EXTERNAL\_CONTENT\_URI*,  
 audioProjection,  
 "${MediaStore.Audio.Media.*DURATION*} > 70000",  
 null,  
 "${MediaStore.Audio.Media.*DISPLAY\_NAME*} ASC"  
 )?.*use* **{** cursor **->** val columnId = cursor.getColumnIndexOrThrow(audioProjection[0])  
 val columnName = cursor.getColumnIndexOrThrow(audioProjection[1])  
 val columnArtist = cursor.getColumnIndexOrThrow(audioProjection[2])  
 val columnPath = cursor.getColumnIndexOrThrow(audioProjection[3])  
 val columnDur = cursor.getColumnIndexOrThrow(audioProjection[4])  
 while (cursor.moveToNext()) {  
 val id = cursor.getLong(columnId)  
 val name = cursor.getString(columnName)  
 val artist = cursor.getString(columnArtist)  
 val path = cursor.getString(columnPath)  
 val dur = cursor.getLong(columnDur)  
 list.add(Audio(id, name, artist, path, dur))  
 }  
 **}** return list.*toList*()  
}

Данный метод обращается к системе, которая автоматически сканирует внешний том хранилища и добавляет медиафайлы в четко определенные коллекции, из которых берутся только аудиофайлы.

**Модуль LibraryFragment.kt**

**Листинг 14. Обработка вызова состояния onViewCreated**

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 val clas = AudioClass()  
 val audioList = clas.mp3ReaderNew(application = *context*?.*applicationContext* as Application)  
  
 audioList.*forEach* **{** val ll = LinearLayout(*context*)  
 val img = ImageView(library\_layout.*context*)  
 val tv = TextView(library\_layout.*context*)  
 val drawable = library\_layout.*context*!!.*resources*.getDrawable(R.drawable.*ic\_\_music\_note*,library\_layout.*context*!!.*theme*)  
 img.setImageDrawable(drawable)  
 tv.*text* = **it**.name.*replace*(".mp3", "") + "\n"  
 val name = **it**.name  
 ll.addView(img)  
 ll.addView(tv)  
 ll.setOnClickListener **{**layoutClick(name, audioList)**}** library\_layout.addView(ll)  
 **}**}

Данный метод вызывается, когда фрагмент создается. Он выводит на экран все аудиофайлы, которые были найдены в системе и добавляет к каждому метод, который будет описан ниже, вызываемый при нажатии на название аудиофайла.

**Листинг 15. Запуск воспроизведения при нажатии на название аудиофайла.**

private fun layoutClick(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 (*activity* as MainActivity?)?.setMusic(songName, audioList)  
 }  
 catch (e: Exception){  
 Toast.makeText(*context*, e.toString(), Toast.*LENGTH\_LONG*).show()  
 }  
}

Данный метод вызывается, когда происходит нажатие на название аудиофайла и запускает воспроизведение данного файла, вызывая метод setMusic класса MainActivity.

**Модуль SearchFragment.kt**

**Листинг 16. Обработка вызова состояния onViewCreated**

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 val clas = AudioClass()  
 val audioList = clas.mp3ReaderNew(application = *context*?.*applicationContext* as Application)  
 editText.*doAfterTextChanged* **{** find(audioList, editText.*text*) **}**}

Данный метод вызывается, когда фрагмент создается. Происходит получение всех аудиофайлов и к полю для ввода текста применяется метод, который будет вызываться при каждом изменении текста пользователем.

**Листинг 17. Метод поиска аудиофайла**

private fun find(audioList: List<AudioClass.Audio>, text: Editable){  
 try {  
 search\_layout.removeAllViews()  
 audioList.*forEach* **{** if (**it**.name.*contains*(text)){  
 val ll = LinearLayout(*context*)  
 val img = ImageView(search\_layout.*context*)  
 val tv = TextView(search\_layout.*context*)  
 val drawable = *resources*.getDrawable(R.drawable.*ic\_\_music\_note*,search\_layout.*context*!!.*theme*)  
 img.setImageDrawable(drawable)  
 tv.*text* = **it**.name.*replace*(".mp3", "") + "\n"  
 val name = **it**.name  
 ll.addView(img)  
 ll.addView(tv)  
 ll.setOnClickListener **{**layoutClick(name, audioList)**}** search\_layout.addView(ll)  
 }  
 **}** }  
 catch (e: Exception){  
 Toast.makeText(search\_layout.*context*, e.toString(), Toast.*LENGTH\_SHORT*).show()  
 }  
}

Данный метод производит поиск аудиофайлов по названию, выводя на экран все аудиофайлы, в которых присутствует текст, который ввел пользователь в поле для ввода.

**Листинг 18.** **Запуск воспроизведения при нажатии на название аудиофайла.**

private fun layoutClick(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 (*activity* as MainActivity?)?.setMusic(songName, audioList)  
 }  
 catch (e: Exception){  
 Toast.makeText(*context*, e.toString(), Toast.*LENGTH\_LONG*).show()  
 }  
}

Данный метод вызывается, когда происходит нажатие на название аудиофайла и запускает воспроизведение данного файла, вызывая метод setMusic класса MainActivity.

## Описание тестовых наборов модулей

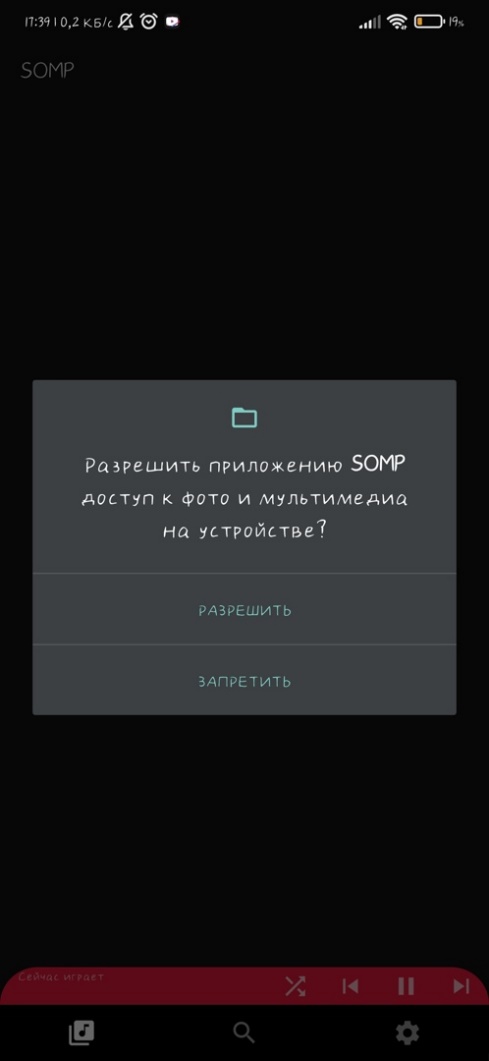
В этом разделе будут продемонстрированы результаты тестирования программы.

Тест 1. Получение разрешения к доступу к медиафайлам системы.

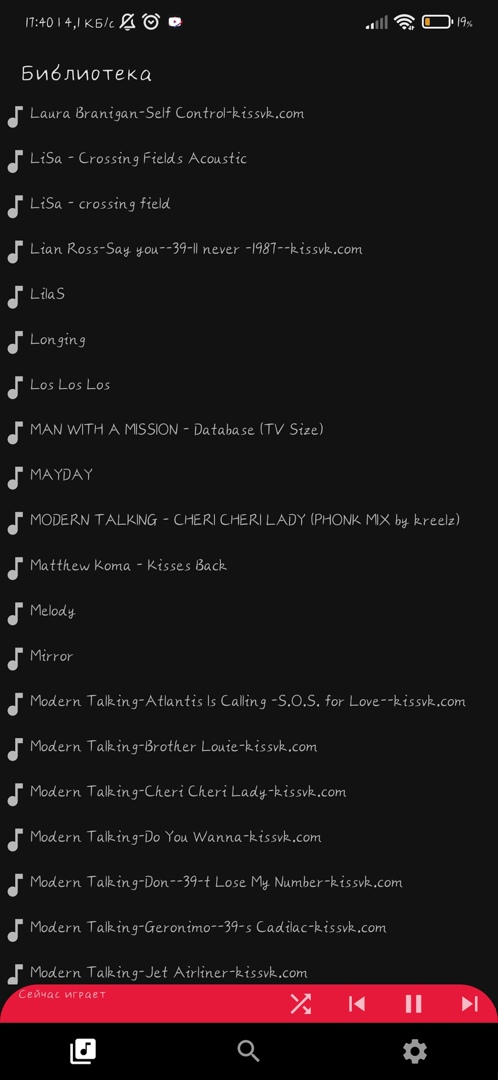
Действия: Запустить приложение. Нажать кнопку разрешить, чтобы получить доступ к медиафайлам.

Ожидаемый результат: Приложение автоматически перезапустится и выведет на экран список аудиофайлов.

Результат текста:



*Рисунок 9. Всплывающее окно предоставления разрешения*



*Рисунок 10. Главное меню*

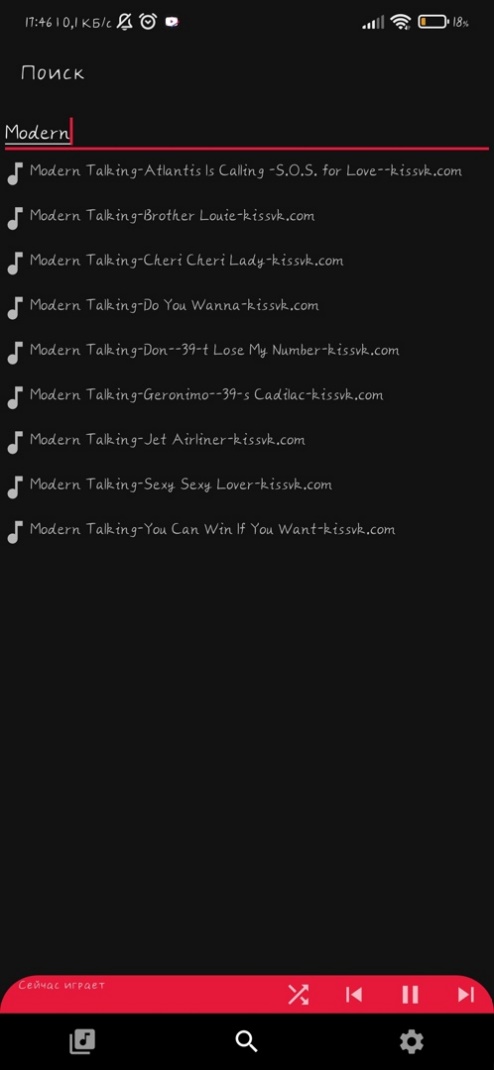
Тест 2. Поиск песни по названию.

Действия: Нажать на кнопку с иконкой лупы, чтобы перейти в меню поиска. Ввести название песни в поле для ввода. Запустить воспроизведение требуемой песни нажатием на её название.

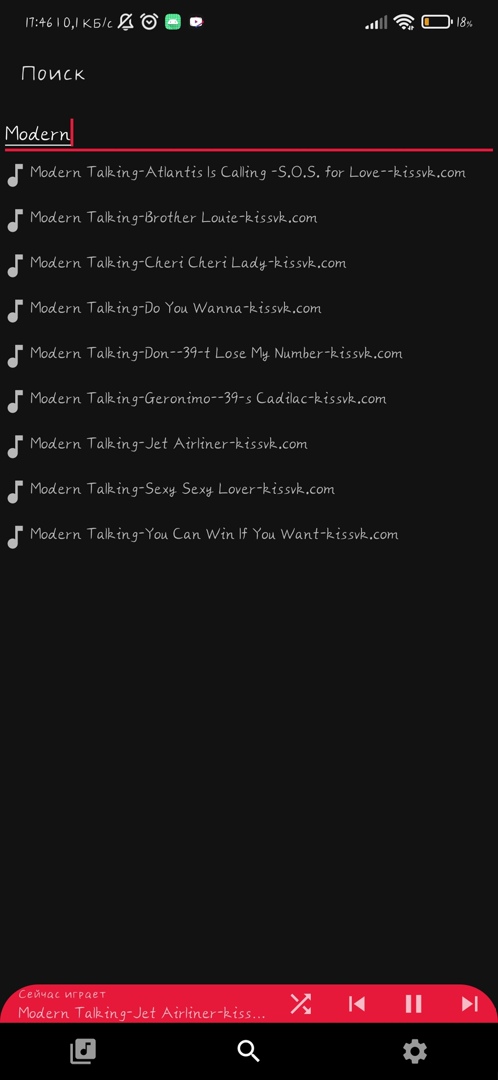
Ожидаемый результат: Найти необходимую песню и запустить её воспроизведение.



*Рисунок 11. Меню поиска песни.*



*Рисунок 12. Поиск необходимой песни*



*Рисунок 13. Воспроизведение песни запущено*

## Описание применения средств отладки

В этом разделе будет показано, как проводится откладка приложение и как находятся ошибки в коде.

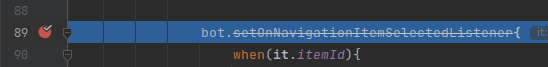
Откладка — это процесс поиска и исправления ошибок в коде программы. Откладка и сборка приложения в среде Microsoft Visual Studio проходят автоматически при её запуске. Отладка приложения обычно означает запуск приложения с подключенным отладчиком (то есть в режиме отладчика)

В режиме отладчика можно просматривать значения из переменных и задавать контрольные значения для переменных, чтобы отслеживать изменение значений. Также можно устанавливать точки остановки и запуска отладчика. Точки остановки указывают, где следует остановить программу.

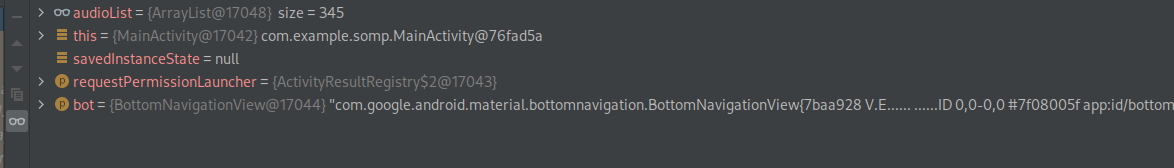
Для запуска приложения с откладкой следует нажать на сочетание клавиш Shift + F9. Задаем 89 строчку файла MainActivity.kt как точку остановки. Запускаем программу.

Программа проработала пару секунд и остановилась, так как дошла до точки остановки. Точка остановки отображена на рисунке 14.

В окне Variables отображены переменные, имеющие какое-либо значение на данный момент выполнения программы. Окно Variables показано на рисунке 15.



*Рисунок 14. Точка остановки*



*Рисунок 15. Окно* *Variables*

# Эксплуатационная часть

## Руководство оператора

**АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации программы «SOMP», предназначенной для прослушивания музыки с Android-устройства.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

1. **Назначение программы**
   1. **Функциональное назначение программы**

Специальное программное обеспечение «SOMP» используется для воспроизведения аудиофайлов, находящихся во внутренней памяти Android-устройства.

* 1. **Эксплуатационное назначение программы**

Специальное программное обеспечение «SOMP» может эксплуатироваться на любом устройстве под OC Android для прослушивания пользователем аудиофайлов.

* 1. **Состав функций**
     1. **Функция открытия окна поиска**

Эта функция позволяет перейти из главного меню в окно поиска аудиофайлов.

* + 1. **Функция запуска воспроизведения.**

Эта функция позволяет запустить воспроизведение аудиофайла.

* + 1. **Функция возвращения в главное меню.**

Эта функция позволяет вернуться в главное меню.

1. **Условия выполнения программы**
   1. **Минимальный состав аппаратнапых средств**

OC: Android 8.0

Процессор: 32-битные и 64-битные: ARM, MIPS, x86. Тактовая частота не ниже 1,8 ГГц.

ОЗУ: 2 ГБ

Место на диске: 16 МБ.

Дисплей: 720x1600

* 1. **Минимальный состав программных средств**

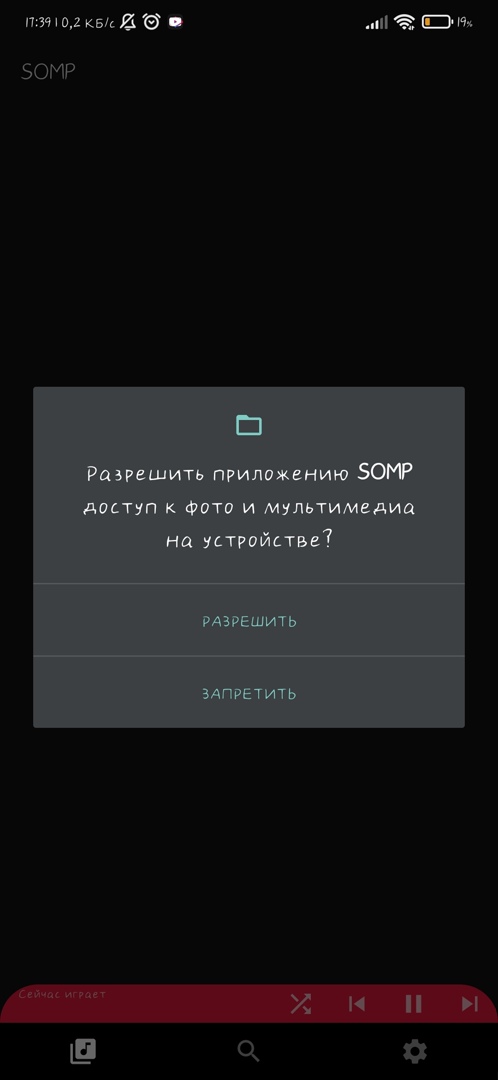
Дополнительные программные средства не требуются.

* 1. **Требование к пользователю**

Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

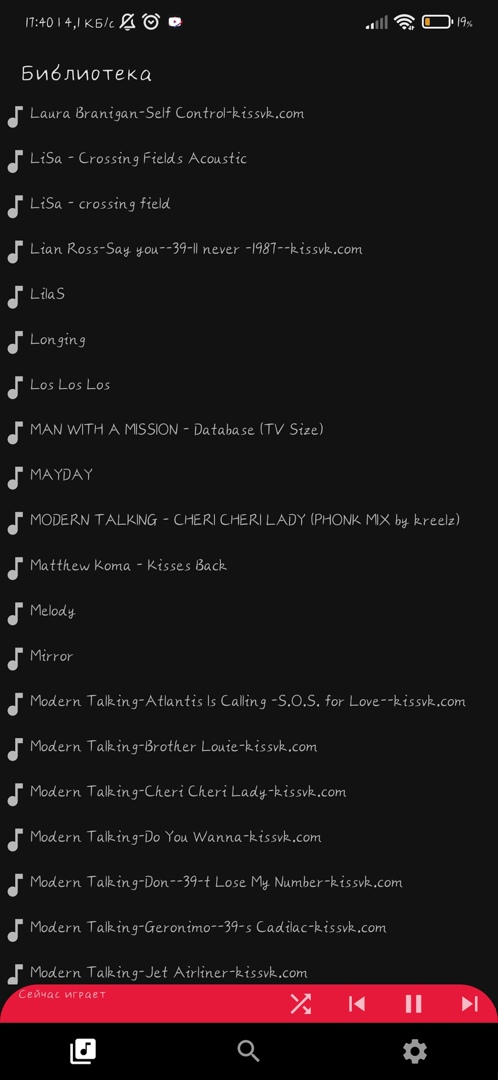
1. **Выполнение программы**
   1. **Загрузка и запуск программы**

**Важно!** После запуска программы вам необходимо выдать разрешение приложению на доступ к мультимедиа на устройстве.



*Рисунок 16. Окно выдачи разрешения*

Запустите программу «SOMP», откроется окно главного меню:



*Рисунок 17. Главное меню*

Кнопки меню имеют следующие функции:

Две пересекающиеся стрелки – включение режима случайного воспроизведения.

Стрелка влево – переключение текущего трека на предыдущий в списке.

Кнопка паузы – приостановить воспроизведение текущего трека.

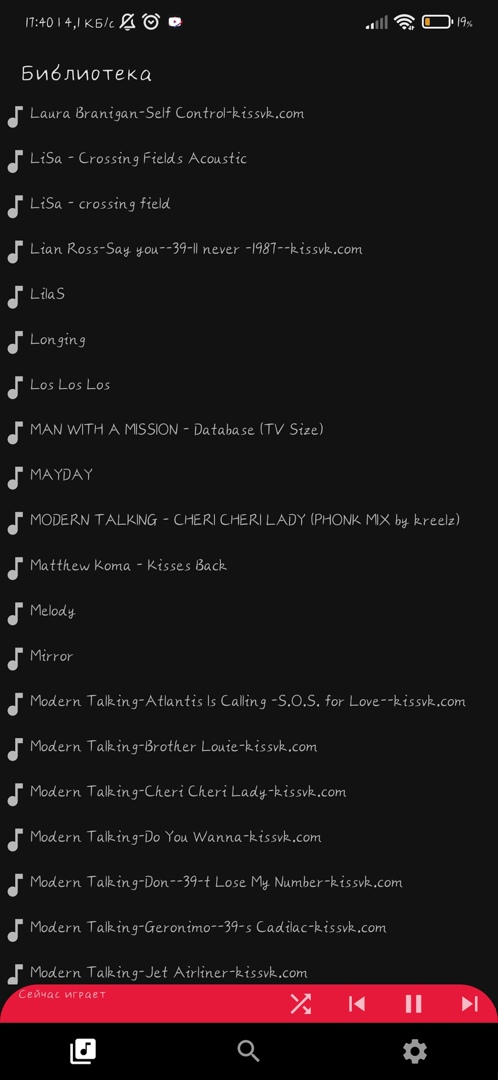
Стрелка вправо – переключение текущего трека на следующий в списке.

Иконка лупы – переход в окно поиска.

Иконка шестерёнки – переход в окно настроек.

1. **Состав функций**
   * 1. **Выполнение функции начала воспроизведения.**

Нажмите на название песни, после начнется воспроизведение данной песни.

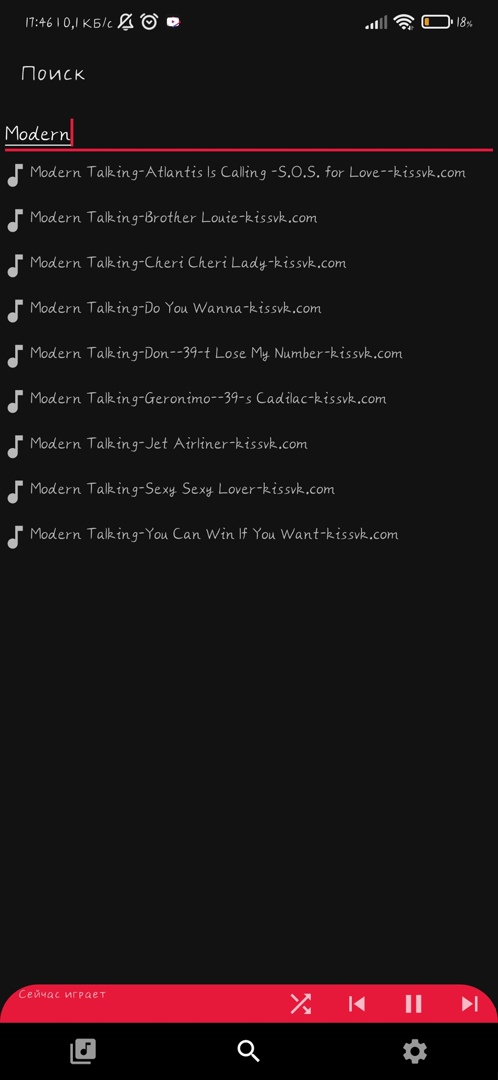


*Рисунок 18. Список песен*

* + 1. **Выполнение функции поиска песни.**

Нажмите на иконку лупы. В поле для ввода текста введите название песни, которую хотите найти.

Нажмите на название песни, после начнется воспроизведение данной песни.



*Рисунок 19. Поиск песни*

* 1. **Завершение работы программы**

Чтобы завершить работу программы закройте программу с помощью системной кнопки управления “Назад” или “Домой”.

## To-Do лист

1. Добавление создания и управления плейлистами.
2. Добавление пунктов настроек для удобной настройки приложения пользователем.
3. Доработка интерфейса с целью упрощения работы с программой.
4. Улучшение дизайна интерфейса.

# Заключение

Во время выполнения данной курсовой работы была исследована и написана программа «SOMP» для прослушивания песен с Android-устройства.

При написании программы была исследована среда разработки Android Studio. Был изучен язык программирования Kotlin, а также было проведено ознакомление с созданием графического интерфейса с помощью Android Studio.

Особенностью данной программы является возможность управления воспроизведением с помощью уведомлений, а также случайное воспроизведение очереди.

Данная программа может использоваться для прослушивания музыки обычными пользователями OC Android, которым не нужен широкий функционал и которым важны интуитивно понятный дизайн и простое управление воспроизведением.

В результате данного исследования был сделан вывод, что процесс написания Android-приложения на языке программирования Kotlin оказался достаточно трудоёмким.

# Список литературы и интернет-источников

1. Статья про медиаплеер:

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

1. Статья про аудиоплеер BlackPlayer:

<https://overclockers.ru/lab/show/84348_3/vybiraem-audiopleer-dlya-android-ustrojstv-musixmatch-music-player-leopard-v7-i-blackplayer>

1. Статья про медиаплеер AIMP:

<https://aimp-download.ru/obzor-pleera-aimp-dlya-android>

1. Евгений Делюкин*.* [Google объявила Kotlin приоритетным языком программирования для разработки Android-приложений — Разработка на vc.ru](https://vc.ru/dev/66728-google-obyavila-kotlin-prioritetnym-yazykom-programmirovaniya-dlya-razrabotki-android-prilozheniy). vc.ru (7 мая 2019).
2. SOMP - Android-приложение для прослушивания музыки

<https://github.com/ltachiUchiha/SOMP>

# Приложение 1. Код главного модуля MainActivity.kt

package com.example.somp  
  
import android.Manifest  
import android.app.\*  
import android.content.\*  
import android.content.pm.PackageManager  
import android.graphics.Color  
import android.media.MediaPlayer  
import android.net.Uri  
import android.os.Build  
import android.os.Bundle  
import android.provider.MediaStore  
import android.provider.Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS  
import android.widget.ImageButton  
import android.widget.Toast  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.appcompat.app.AppCompatActivity  
import androidx.appcompat.app.AppCompatDelegate  
import androidx.core.app.NotificationCompat  
import androidx.core.app.NotificationManagerCompat  
import androidx.core.content.ContextCompat  
import androidx.fragment.app.Fragment  
import androidx.localbroadcastmanager.content.LocalBroadcastManager  
import com.example.somp.fragments.LibraryFragment  
import com.example.somp.fragments.SearchFragment  
import com.example.somp.fragments.SettingsFragment  
import com.google.android.material.bottomnavigation.BottomNavigationView  
import kotlinx.android.synthetic.main.activity\_main.\*  
import androidx.media.app.NotificationCompat as MediaNotificationCompat  
  
  
class MainActivity : AppCompatActivity() {  
  
 private val libraryFragment = LibraryFragment()  
 private val searchFragment = SearchFragment()  
 private val settingsFragment = SettingsFragment()  
  
 private val mediaPlayer: MediaPlayer = MediaPlayer()  
  
 private var isPaused = false  
 private var isRandom = false  
  
 private lateinit var audioList : MutableList<AudioClass.Audio>  
  
 private lateinit var receiver : BroadcastReceiver  
 private lateinit var receiver2 : BroadcastReceiver  
 private lateinit var receiver3 : BroadcastReceiver  
  
 companion object {  
 const val NOTIFICATION\_ID = 101  
 const val CHANNEL\_ID = "1111"  
 }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_main)  
  
 val requestPermissionLauncher =  
 registerForActivityResult(  
 ActivityResultContracts.RequestPermission()  
 ) **{** isGranted: Boolean **->** if (isGranted) {  
 // Разрешение предоставлено.  
 finish();  
 overridePendingTransition(0, 0);  
 startActivity(intent);  
 overridePendingTransition(0, 0);  
 } else {  
 // Объяснение пользователю, что функция недоступна, потому что  
 // функции требуют разрешения, которое пользователь запретил.  
 createAlertMessage()  
 }  
 **}** when (PackageManager.PERMISSION\_GRANTED) {  
 ContextCompat.checkSelfPermission(  
 this,  
 Manifest.permission.READ\_EXTERNAL\_STORAGE  
 ) -> {  
  
 audioList = AudioClass().mp3ReaderNew(application = this.applicationContext as Application).toMutableList()  
  
 replaceFragment(libraryFragment, "Библиотека")  
  
 AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE\_NIGHT\_YES)  
  
 val bot = findViewById<BottomNavigationView>(R.id.bottom\_navigation)  
  
 bot.setOnNavigationItemSelectedListener**{** when(**it**.itemId){  
 R.id.ic\_library -> replaceFragment(libraryFragment, "Библиотека")  
 R.id.ic\_search -> replaceFragment(searchFragment, "Поиск")  
 R.id.ic\_settings -> replaceFragment(settingsFragment, "Настройки")  
 }  
 true  
 **}** val imageButton = findViewById<ImageButton>(R.id.imageButton)  
 imageButton.setOnClickListener **{** btnPause() **}** val btnNext = findViewById<ImageButton>(R.id.imageNext)  
 btnNext.setOnClickListener **{** setMusicNext() **}** val btnPrevious = findViewById<ImageButton>(R.id.imagePrevious)  
 btnPrevious.setOnClickListener **{** setMusicPrevious() **}** val btnRandom = findViewById<ImageButton>(R.id.imageRandom)  
 btnRandom.setOnClickListener **{** randomMusic() **}** mediaPlayer.setOnCompletionListener **{** setMusicNext()  
 **}** createNotificationChannel()  
  
 receiver = object : BroadcastReceiver() {  
 override fun onReceive(context: Context, intent: Intent) {  
 when(intent.action){  
 "previous" -> setMusicPrevious()  
 }  
 }  
 }  
 receiver2 = object : BroadcastReceiver() {  
 override fun onReceive(context: Context, intent: Intent) {  
 when(intent.action){  
 "pause" -> btnPause()  
 }  
 }  
 }  
 receiver3 = object : BroadcastReceiver() {  
 override fun onReceive(context: Context, intent: Intent) {  
 when(intent.action){  
 "next" -> setMusicNext()  
 }  
 }  
 }  
 this.registerReceiver(receiver, IntentFilter("previous"))  
 this.registerReceiver(receiver2, IntentFilter("pause"))  
 this.registerReceiver(receiver3, IntentFilter("next"))  
  
 }  
 else -> {  
 // Напрямую запрашиваем разрешение.  
 // Зарегистрированный ActivityResultCallback получает результат этого запроса.  
 requestPermissionLauncher.launch(  
 Manifest.permission.READ\_EXTERNAL\_STORAGE)  
 }  
 }  
 }  
 private fun unRegisterReceiver(r1:BroadcastReceiver, r2:BroadcastReceiver, r3:BroadcastReceiver){  
 this.unregisterReceiver(r1)  
 this.unregisterReceiver(r2)  
 this.unregisterReceiver(r3)  
 }  
 override fun onDestroy() {  
 super.onDestroy()  
 unRegisterReceiver(receiver, receiver2, receiver3)  
 }  
 private fun createAlertMessage(){  
 val builder: AlertDialog.Builder = this.let **{** AlertDialog.Builder(**it**)  
 **}** builder.setMessage("Без этого разрешения приложение не сможет просматривать музыку с вашего устройства. Вы хотите дать разрешение?")  
 ?.setTitle("Доступ запрещён")  
  
 builder.apply **{** setPositiveButton("Предоставить разрешение"  
 ) **{** \_, \_ **->** // Пользователь нажал кнопку ОК  
 val intent = Intent(ACTION\_APPLICATION\_DETAILS\_SETTINGS)  
 val uri = Uri.fromParts(  
 "package",  
 packageName, null  
 )  
 intent.data = uri  
 startActivity(intent)  
 **}** setNegativeButton("Отмена"  
 ) **{** \_, \_ **->** //Пользователь отменил диалог  
 **}  
 }** builder.create()  
 builder.show()  
 }  
  
 private fun createNotificationChannel() {  
 // Создаем NotificationChannel, но только в API 26+, потому что  
 // класс NotificationChannel новый, его нет в библиотеке поддержки  
 if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.O) {  
 val name = "Music"  
 val descriptionText = "Music"  
 val importance = NotificationManager.IMPORTANCE\_DEFAULT  
 val channel = NotificationChannel(CHANNEL\_ID, name, importance).apply **{** description = descriptionText  
 **}** // Регистрируем канал в системе  
 val notificationManager: NotificationManager =  
 getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager  
 notificationManager.createNotificationChannel(channel)  
 }  
 }  
 private fun createMusicNotification(musicName: String, musicData: String){  
  
 val pauseIntent = Intent()  
 pauseIntent.action = "pause"  
 val pausePendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, pauseIntent, 0)  
 val previousIntent = Intent()  
 previousIntent.action = "previous"  
 val previousPendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, previousIntent, 0)  
 val nextIntent = Intent()  
 nextIntent.action = "next"  
 val nextPendingIntent: PendingIntent = PendingIntent.getBroadcast(this, 0, nextIntent, 0)  
 val notification = NotificationCompat.Builder(this, CHANNEL\_ID)  
 // Показываем элементы управления на экране блокировки, даже если пользователь скрывает конфиденциальный контент.  
 .setVisibility(NotificationCompat.VISIBILITY\_PUBLIC)  
 .setSmallIcon(R.drawable.ic\_\_music\_note)  
 .addAction(R.drawable.ic\_skip\_previous, "previous", previousPendingIntent)  
 .addAction(R.drawable.ic\_baseline\_pause\_24, "pause", pausePendingIntent)  
 .addAction(R.drawable.ic\_skip\_next, "next", nextPendingIntent)  
 .setStyle(MediaNotificationCompat.MediaStyle()  
 .setShowActionsInCompactView(0, 1, 2))  
 // Применяем медиа стиль уведомления.  
 .setContentTitle(musicName)  
 .setContentText(musicData)  
 .setCategory(NotificationCompat.CATEGORY\_TRANSPORT)  
  
  
 with(NotificationManagerCompat.from(this)) **{** notify(NOTIFICATION\_ID, notification.build()) // посылаем уведомление  
 **}** }  
 private fun randomMusic(){  
 val btnRandom = findViewById<ImageButton>(R.id.imageRandom)  
 isRandom = if (!isRandom){  
 audioList.shuffle()  
 btnRandom.setColorFilter(Color.GRAY)  
 true  
 }  
 else{  
 audioList.sortBy **{ it**.name **}** btnRandom.setColorFilter(Color.WHITE)  
 false  
 }  
 }  
  
 private fun setMusicNext(){  
 val musicName = textView.text.toString().plus(".mp3")  
 val num = audioList.indexOf(audioList.find **{ it**.name == musicName **}**)  
 if (num < audioList.count()-1){  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI, audioList[num+1].id)  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.text = audioList[num+1].name.replace(".mp3", "")  
  
 val drawable = getDrawable(R.drawable.ic\_baseline\_pause\_24)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num+1].name.replace(".mp3", ""), audioList[num+1].artist)  
 }  
 else{  
 setMusic(audioList[0].name, audioList)  
 }  
 }  
  
 private fun setMusicPrevious(){  
 val musicName = textView.text.toString().plus(".mp3")  
 val num = audioList.indexOf(audioList.find **{ it**.name == musicName **}**)  
 if (num > 0){  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI, audioList[num-1].id)  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.text = audioList[num-1].name.replace(".mp3", "")  
  
  
 val drawable = getDrawable(R.drawable.ic\_baseline\_pause\_24)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num-1].name.replace(".mp3", ""), audioList[num-1].artist)  
 }  
 else{  
 setMusic(audioList[audioList.count()-1].name, audioList)  
 }  
 }  
  
 fun setMusic(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 mediaPlayer.stop()  
 mediaPlayer.reset()  
 val num = audioList.indexOf(audioList.find **{ it**.name == songName **}**)  
 val uri = ContentUris.withAppendedId(MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI, audioList[num].id)  
 mediaPlayer.setDataSource(this, uri)  
 mediaPlayer.prepare()  
 mediaPlayer.start()  
 textView.text = audioList[num].name.replace(".mp3", "")  
  
 val drawable = getDrawable(R.drawable.ic\_baseline\_pause\_24)  
 imageButton.setImageDrawable(drawable)  
  
 createMusicNotification(audioList[num].name.replace(".mp3", ""), audioList[num].artist)  
 }  
 catch (e: Exception){  
 Toast.makeText(this, e.toString(), Toast.LENGTH\_LONG).show()  
 }  
 }  
  
 fun btnPause(){  
 try {  
 isPaused = if(!mediaPlayer.isPlaying && mediaPlayer.duration != 0) {  
 val drawable = getDrawable(R.drawable.ic\_baseline\_pause\_24)  
 imageButton.setImageDrawable(drawable)  
 mediaPlayer.start()  
 false  
 } else {  
 val drawable = getDrawable(R.drawable.ic\_play\_arrow)  
 imageButton.setImageDrawable(drawable)  
 mediaPlayer.pause()  
 true  
 }  
 }  
 catch (e: Exception){  
 Toast.makeText(this, e.toString(), Toast.LENGTH\_LONG).show()  
 }  
 }  
  
 private fun replaceFragment(fragment: Fragment, title: String){  
 val transaction = supportFragmentManager.beginTransaction()  
 transaction.replace(R.id.frame\_container, fragment)  
 transaction.commit()  
  
 supportActionBar?.title = title  
 }  
  
}

# Приложение 2. Код модуля AudioClass.kt

package com.example.somp  
  
import android.content.Context  
import android.provider.MediaStore  
  
class AudioClass {  
 data class Audio(  
 var id: Long,  
 var name: String,  
 var artist: String,  
 var data: String,  
 var dur: Long  
 )  
 fun mp3ReaderNew(application: Context): List<Audio>{  
  
 val audioProjection = arrayOf(  
 MediaStore.Audio.Media.\_ID,  
 MediaStore.Audio.Media.DISPLAY\_NAME,  
 MediaStore.Audio.Media.ARTIST,  
 MediaStore.Audio.Media.DATA,  
 MediaStore.Audio.Media.DURATION,  
 )  
  
 val list = mutableListOf<Audio>()  
 application.contentResolver.query(  
 MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI,  
 audioProjection,  
 "${MediaStore.Audio.Media.DURATION} > 70000",  
 null,  
 "${MediaStore.Audio.Media.DISPLAY\_NAME} ASC"  
 )?.use **{** cursor **->** val columnId = cursor.getColumnIndexOrThrow(audioProjection[0])  
 val columnName = cursor.getColumnIndexOrThrow(audioProjection[1])  
 val columnArtist = cursor.getColumnIndexOrThrow(audioProjection[2])  
 val columnPath = cursor.getColumnIndexOrThrow(audioProjection[3])  
 val columnDur = cursor.getColumnIndexOrThrow(audioProjection[4])  
 while (cursor.moveToNext()) {  
 val id = cursor.getLong(columnId)  
 val name = cursor.getString(columnName)  
 val artist = cursor.getString(columnArtist)  
 val path = cursor.getString(columnPath)  
 val dur = cursor.getLong(columnDur)  
 list.add(Audio(id, name, artist, path, dur))  
 }  
 **}** return list.toList()  
 }  
}

# Приложение 3. Код модуля LibraryFragment.kt

package com.example.somp.fragments  
  
import android.app.Application  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.ImageView  
import android.widget.LinearLayout  
import android.widget.TextView  
import android.widget.Toast  
import androidx.fragment.app.Fragment  
import com.example.somp.AudioClass  
import com.example.somp.MainActivity  
import com.example.somp.R  
import kotlinx.android.synthetic.main.fragment\_library.\*  
  
// TODO: Rename parameter arguments, choose names that match  
// the fragment initialization parameters, e.g. ARG\_ITEM\_NUMBER  
private const val ARG\_PARAM1 = "param1"  
private const val ARG\_PARAM2 = "param2"  
  
/\*\*  
 \* A simple [Fragment] subclass.  
 \* Use the [LibraryFragment.newInstance] factory method to  
 \* create an instance of this fragment.  
 \*/  
class LibraryFragment : Fragment() {  
 // TODO: Rename and change types of parameters  
 private var param1: String? = null  
 private var param2: String? = null  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 arguments?.let **{** param1 = **it**.getString(ARG\_PARAM1)  
 param2 = **it**.getString(ARG\_PARAM2)  
 **}** }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // Inflate the layout for this fragment  
 return inflater.inflate(R.layout.fragment\_library, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 val clas = AudioClass()  
 val audioList = clas.mp3ReaderNew(application = context?.applicationContext as Application)  
  
 audioList.forEach **{** val ll = LinearLayout(context)  
 val img = ImageView(library\_layout.context)  
 val tv = TextView(library\_layout.context)  
 val drawable = library\_layout.context!!.resources.getDrawable(R.drawable.ic\_\_music\_note,library\_layout.context!!.theme)  
 img.setImageDrawable(drawable)  
 tv.text = **it**.name.replace(".mp3", "") + "\n"  
 val name = **it**.name  
 ll.addView(img)  
 ll.addView(tv)  
 ll.setOnClickListener **{**layoutClick(name, audioList)**}** library\_layout.addView(ll)  
 **}** }  
 private fun layoutClick(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 (activity as MainActivity?)?.setMusic(songName, audioList)  
 }  
 catch (e: Exception){  
 Toast.makeText(context, e.toString(), Toast.LENGTH\_LONG).show()  
 }  
 }  
  
 companion object {  
 /\*\*  
 \* Use this factory method to create a new instance of  
 \* this fragment using the provided parameters.  
 \*  
 \* **@param** param1 Parameter 1.  
 \* **@param** param2 Parameter 2.  
 \* **@return** A new instance of fragment LibraryFragment.  
 \*/  
 // TODO: Rename and change types and number of parameters  
 @JvmStatic  
 fun newInstance(param1: String, param2: String) =  
 LibraryFragment().apply **{** arguments = Bundle().apply **{** putString(ARG\_PARAM1, param1)  
 putString(ARG\_PARAM2, param2)  
 **}  
 }** }  
}

# Приложение 4. Код модуля SearchFragment.kt

package com.example.somp.fragments  
  
import android.app.Application  
import android.os.Bundle  
import android.text.Editable  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import android.widget.ImageView  
import android.widget.LinearLayout  
import android.widget.TextView  
import android.widget.Toast  
import androidx.core.widget.doAfterTextChanged  
import com.example.somp.AudioClass  
import com.example.somp.MainActivity  
import com.example.somp.R  
import kotlinx.android.synthetic.main.fragment\_search.\*  
import java.lang.Exception  
  
// TODO: Rename parameter arguments, choose names that match  
// the fragment initialization parameters, e.g. ARG\_ITEM\_NUMBER  
private const val ARG\_PARAM1 = "param1"  
private const val ARG\_PARAM2 = "param2"  
  
/\*\*  
 \* A simple [Fragment] subclass.  
 \* Use the [SearchFragment.newInstance] factory method to  
 \* create an instance of this fragment.  
 \*/  
class SearchFragment : Fragment() {  
 // TODO: Rename and change types of parameters  
 private var param1: String? = null  
 private var param2: String? = null  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 arguments?.let **{** param1 = **it**.getString(ARG\_PARAM1)  
 param2 = **it**.getString(ARG\_PARAM2)  
 **}** }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // Inflate the layout for this fragment  
 return inflater.inflate(R.layout.fragment\_search, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 val clas = AudioClass()  
 val audioList = clas.mp3ReaderNew(application = context?.applicationContext as Application)  
 editText.doAfterTextChanged **{** find(audioList, editText.text) **}** }  
 private fun find(audioList: List<AudioClass.Audio>, text: Editable){  
 try {  
 search\_layout.removeAllViews()  
 audioList.forEach **{** if (**it**.name.contains(text)){  
 val ll = LinearLayout(context)  
 val img = ImageView(search\_layout.context)  
 val tv = TextView(search\_layout.context)  
 val drawable = resources.getDrawable(R.drawable.ic\_\_music\_note,search\_layout.context!!.theme)  
 img.setImageDrawable(drawable)  
 tv.text = **it**.name.replace(".mp3", "") + "\n"  
 val name = **it**.name  
 ll.addView(img)  
 ll.addView(tv)  
 ll.setOnClickListener **{**layoutClick(name, audioList)**}** search\_layout.addView(ll)  
 }  
 **}** }  
 catch (e: Exception){  
 Toast.makeText(search\_layout.context, e.toString(), Toast.LENGTH\_SHORT).show()  
 }  
 }  
 private fun layoutClick(songName: String, audioList: List<AudioClass.Audio>){  
 try {  
 (activity as MainActivity?)?.setMusic(songName, audioList)  
 }  
 catch (e: Exception){  
 Toast.makeText(context, e.toString(), Toast.LENGTH\_LONG).show()  
 }  
 }  
 companion object {  
 /\*\*  
 \* Use this factory method to create a new instance of  
 \* this fragment using the provided parameters.  
 \*  
 \* **@param** param1 Parameter 1.  
 \* **@param** param2 Parameter 2.  
 \* **@return** A new instance of fragment SearchFragment.  
 \*/  
 // TODO: Rename and change types and number of parameters  
 @JvmStatic  
 fun newInstance(param1: String, param2: String) =  
 SearchFragment().apply **{** arguments = Bundle().apply **{** putString(ARG\_PARAM1, param1)  
 putString(ARG\_PARAM2, param2)  
 **}  
 }** }  
}

# Приложение 5. Код модуля SettingsFragment.kt

package com.example.somp.fragments  
  
import android.os.Bundle  
import androidx.fragment.app.Fragment  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import com.example.somp.R  
  
// TODO: Rename parameter arguments, choose names that match  
// the fragment initialization parameters, e.g. ARG\_ITEM\_NUMBER  
private const val ARG\_PARAM1 = "param1"  
private const val ARG\_PARAM2 = "param2"  
  
/\*\*  
 \* A simple [Fragment] subclass.  
 \* Use the [SettingsFragment.newInstance] factory method to  
 \* create an instance of this fragment.  
 \*/  
class SettingsFragment : Fragment() {  
 // TODO: Rename and change types of parameters  
 private var param1: String? = null  
 private var param2: String? = null  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 arguments?.let **{** param1 = **it**.getString(ARG\_PARAM1)  
 param2 = **it**.getString(ARG\_PARAM2)  
 **}** }  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // Inflate the layout for this fragment  
 return inflater.inflate(R.layout.fragment\_settings, container, false)  
 }  
  
 companion object {  
 /\*\*  
 \* Use this factory method to create a new instance of  
 \* this fragment using the provided parameters.  
 \*  
 \* **@param** param1 Parameter 1.  
 \* **@param** param2 Parameter 2.  
 \* **@return** A new instance of fragment SettingsFragment.  
 \*/  
 // TODO: Rename and change types and number of parameters  
 @JvmStatic  
 fun newInstance(param1: String, param2: String) =  
 SettingsFragment().apply **{** arguments = Bundle().apply **{** putString(ARG\_PARAM1, param1)  
 putString(ARG\_PARAM2, param2)  
 **}  
 }** }  
}

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)