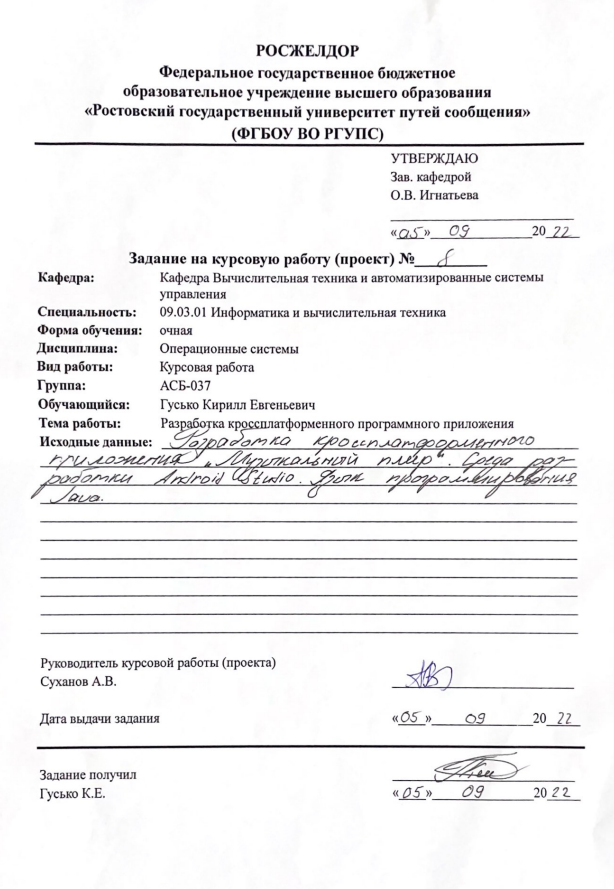
****

**Содержание**

[Введение 4](#_Toc122973567)

[1 Анализ предметной области 5](#_Toc122973568)

[1.2 Кроссплатформенный подход к разработке 6](#_Toc122973569)

[1.3 Гибридные приложения 8](#_Toc122973570)

[1.4 Приложения для разработки 9](#_Toc122973571)

[1.4 Описание языка программирования Java 10](#_Toc122973572)

[1.6 Описание языка программирования Kotlin 11](#_Toc122973573)

[Выводы к первой главе 12](#_Toc122973574)

[2 Разработка приложения «Музыкальный плеер» 13](#_Toc122973575)

[2.1 Разработка структуры приложения 13](#_Toc122973576)

[2.2 Разработка программного кода 15](#_Toc122973577)

[2.3 Описание приложения 16](#_Toc122973578)

[2.4 Руководство по использованию 16](#_Toc122973579)

[2.5 Контрольный запуск 21](#_Toc122973580)

[2.5 Выводы ко второй главе 24](#_Toc122973581)

[Заключение 25](#_Toc122973582)

[Список использованных источников 26](#_Toc122973583)

[Приложение А 27](#_Toc122973584)

## Введение

В современном мире, когда технологии используются во всех сферах человеческой жизни и уже занимают неотьемлемую часть нашей жизни, разработка мобильных приложений играет большую роль. Такие приложения облегчают жизнь обычных людей, помогая с решением каких либо задач. Ведь совсем недавно мы и представить не могли, что при помощи гаджетов можно будет дистанционно покупать билеты, оплачивать счета, заказывать еду и другое.

Данное приложение по прослушиванию музыки тоже относится к таким приложениям. Ведь с его помощью люди могут отдохнуть в свободную минуту или параллельно работе слушать музыку для разгрузки и лучшей работоспособности. Также данное приложение является очень простым в иприменении, и любой, от мала до велика сможет без труда пользоваться им.

Эта курсовая работа посвящена разработке приложения «Музыкальный плеер».

## 1 Анализ предметной области

* 1. **Нативный подход к разработке**

Каждый из подходов обладает своей спецификой, критически влияющей на конечный результат.

Нативные приложения – созданное с использованием инструментов разработки вендоров: Apple/Google и распространяемое через магазины приложений. Для разработки под Apple актуальны технологии: UIKit, SwiftUI + богатый iOS SDK, язык программирования Swift (и для особых случаев старичок Objective-C). Для Андроид соответственно – Android SDK, Jetpack Compose, языки: Java, Kotlin.

Нативные приложения могут использовать программное и аппаратное обеспечение конкретного устройства, так как оно создано для использования на конкретном устройстве и его ОС. «Нативки» могут получать доступ ко всем службам, сервисам и примочкам телефона: камере, микрофону, геолокатору, акселерометру, календарю, медиафайлам, уведомлениям и так далее  – в общем, полноценно обживаются и чувствуют себя как дома.

Нативными приложениями являются те, с которыми пользователь сталкивается с первого дня использования устройства. Это установленные по умолчанию браузер, почтовый клиент, адресная книга, будильник, календарь и другие стандартные программы.

*Плюсы и минусы нативных приложений.*

Плюсы:

1.Система лучше понимает свой язык. С приложением, написанным специально под iOS или Android, будет меньше технических сложностей, в том числе с обновлениями. Его проще оптимизировать, сделать быстрее или легче. А чем меньше весит приложение, тем охотнее его скачивают пользователи.

2.Никаких ограничений: можно смело браться за реализацию любых идей, связанных с работой устройства – камерой, GPS, сенсорами, файловой системой устройства и так далее.

3.В нативной разработке намного больше специалистов – нет проблем с тем, чтобы найти сотрудников на проект или просто с кем-то посоветоваться.

Минусы:

1.Главный минус это цена.

Если Вам нужно приложение на iOS (iPhone, iPad) и Android, то придется нанимать минимум двух программистов. Один будет писать на Kotlin (для Android), другой на Swift (для iOS).

2.Второй минус это время.

Обычно языки программирования, которые используют для нативного подхода, сами по себе сложные поэтому требуют чуть больше времени на написание какого-то функционала. Например, сделать интерфейс приложения быстрее всего будет с помощью JavaScript, а не C# или Swift. А вот сам JavaScript, к сожалению, не подходит для нативной разработки (потому что это не язык “системы”).

## Кроссплатформенный подход к разработке

Кроссплатформенные приложения – это программы, созданные по концепции «скомпилируй один раз, запусти везде», поскольку они предполагают разработку одной кодовой базы для работы на разных платформах / операционных системах. Это достигается за счет применения таких инструментов разработки, как React Native и Xamarin, которые позволяют написать код программы и развернуть его на разных платформах с близким к родному для выбранной платформы UI/UX-дизайну.

Благодаря кроссплатформенному подходу к созданию программ можно существенно сократить время и стоимость разработки. Однако при этом не стоит рассчитывать на выдающуюся производительность и / или максимальную интеграцию с функциями и сервисами платформы и устройства. Настройка также может стать проблемой, поскольку она будет ограничена используемой структурой.

*Плюсы кроссплатформенной разработки приложений*

**1.Единая база кода для всех платформ.**

Возможно, главный плюс кроссплатформенной разработки – использование одной базы кода для разных мобильных платформ. Благодаря этому команда разработчиков может использовать лишь один технологический стек (и при этом его гораздо проще подобрать), а не сразу несколько технологических стеков для каждой мобильной платформы / операционной системы. **Более простое и быстрое развертывание**. Поскольку для создания кроссплатформенных решений не нужно изучать несколько технологических стеков, разработчики могут сконцентрироваться на одном технологическом стеке, что позволяет изучить и затем реализовать все его преимущества. Кроме того, одна кодовая база также позволяет сильно ускорить начальное развертывание приложения сразу на нескольких платформах, а также сократить время и сложность апдейтов.

**2.Сокращение времени и стоимости разработки.**

Это еще одно важное преимущество кроссплатформенной разработки, которое достигается за счет сокращения времени на написание кода для двух и больше баз кода. Сколько именно можно сэкономить времени и денег, зависит от задачи проекта и фреймворка. Например, благодаря React Native вы можете создавать приложения на 50% быстрее, чем при нативной разработке. Кроме того, React Native позволяет использовать готовые компоненты, которые уже были созданы кем-то другим, легко интегрируя их в свой собственный проект. Именно поэтому мы в Merehead создаем кроссплатформенные приложения, используя React Native.

**3.Покрытие более широкой аудитории пользователей**.

Так как кроссплатформенные решения разрабатываются для двух и более платформ / операционных систем, то и охват целевой аудитории у них больше. Это позволяет представить приложение на большем количестве мобильных устройств для большего числа пользователей, что делает кроссплатформу более выгодной для бизнеса, чем присутствие лишь на одной платформе.

**4.Допустимость одинакового интерфейса и UX.**

Одна команда разработчиков и общая кодовая база дает возможность создать одинаковый пользовательский интерфейс для всех платформ, что очень важно для брендов. Кроме того, это позволяет реализовать «бесшовный» переход с одной платформы на другую, например, со смартфона на планшет, ноутбук, компьютер или телевизор (игровую приставку или смарт-тв).

*Минусы кроссплатформенной разработки приложений*

**1.Меньшая гибкость, чем у нативных приложений.**

Унифицированный стек технологий не дает такой же высокой гибкости, как стек, заточенный под конкретную платформу. Из-за чего кроссплатформенные решения в подавляющем большинстве не могут обладать такой же степенью интеграции с платформой и устройством, как нативные решения.

**2.Более низкая производительность и надежность**.

Потеря гибкости в разработке также будет означать потерю в возможностях по улучшению и оптимизации производительности. Другими словами, при прочих равных кроссплатформенные решения никогда не будут работать так же быстро, надежно, безопасно и плавно, как нативные приложения.

**3.Возможное несоответствие в дизайне на двух платформах.**

Еще одна проблема, которая может у вас возникнуть, – наличие некоторых отличий между пользовательскими интерфейсами и пользовательским опытом на разных платформах / операционных системах (например, на Windows есть кнопка «Пуск», а на macOS у нее нет аналога). Это может быть обусловлено как разными требованиями к дизайну пользовательских интерфейсов на разных платформах, так и специфическими ограничениями этих платформ.

**4.Возможные сложности с маркетплейсами приложений.**

Механизм и требования к кроссплатформенным приложениям в маркетплейсах Apple App Store и в Google Play Store отличаются. Прохождение всех бюрократических процедур, проверок и тестов может занять много времени и вызвать некоторые трудности.

## Гибридные приложения

Гибрид –это скорее подход к программированию, чем какой-то особенный вид приложения. Его принцип заключается в том, что всё что можно (читай "все функции не связанные с железом") программируется на стороне сервера, а на стороне клиента остаётся только необходимый минимум. Если рассматривать типовой мобильный магазин, то GPS и функция отображения данных будет на стороне приложения, а каталог, карточки товаров и т.д. будут грузиться с сервера.

*Плюсы гибридной разработки приложений*

1.Используется только один язык программирования, поэтому проще найти нужного специалиста и не нужно привлекать к работе дополнительных сотрудников.

2.Демократичная стоимость.

3.Кроссплатформенность.

Не нужно создавать несколько программ под разные ОС. Приложение будет поддерживаться на всех устройствах: IOS, Android, Windows Phone.

4.Возможность работы в оффлайн режиме.

5.Доступ к основным данным мобильного устройства (камера, отпечаток пальца).

6.Возможность размещения приложения в официальных магазинах ОС (Google Play Market, App Store).

*Минусы гибридной разработки приложений*

1.Трудности с версткой адаптивного дизайна.

Несмотря на использование фреймворков, многие из них могут некорректно отображаться на некоторых устройствах.

2.Скорость работы будет ниже, чем при нативном подходе.

3.Есть ограничение по внедрению визуальных и графических элементов.

4.Возможно появление багов, которые трудно обнаружить.

Это связно с тем, что при тестировании программы, ошибка может быть только на одной ОС, а при ее исправлении может возникнуть еще одна, но уже на другой операционной системе.

5.Мало возможностей для интеграции с внутренними инструментами мобильного девайса.

## 1.4 Приложения для разработки

В создании своего проекта я пользовался такой средой разработки как Android Studio.

Android Studio – программа, являющаяся средой разработки приложений для мобильной платформы Android.

Это универсальная среда разработки, так как она позволяет оптимизировать работу будущих приложения для работы не только на смартфонах, но и на планшетах, портативных ПК, которые работают на основе рассматриваемой операционной системы.

Во время создания приложений и утилит для операционной системы Android, пользователь программного обеспечения может наблюдать за изменениями в проекте, в режиме реального времени.

*Достоинства утилиты*

1.Среда разработки поддерживает работу с несколькими языками программирования, к которым относятся самые популярные – C/C++, Java.

2.Редактор кода, с которым удобно работать;

3.Позволяет разрабатывать приложения не только для смартфонов/планшетов, а и для портативных ПК, приставок для телевизоров Android TV, устройств Android Wear, новомодных мобильных устройств с необычным соотношением сторон экрана;

4.Тестирование корректности работы новых игр, утилит, их производительности на той или иной системе, происходит непосредственно в эмуляторе;

5.Рефакторинг уже готового кода;

6.Достаточно большая библиотека с готовыми шаблонами и компонентами для разработки ПО;

7.Разработка приложения для Android N – самой последней версии операционной системы;

8.Предварительная проверка уже созданного приложения на предмет ошибок в нем;

9.Большой набор средств инструментов для тестирования каждого элемента приложения, игры;

10.Для неопытных/начинающих разработчиков специально создано руководство по использованию Android Studio, размещенное на официальном сайте утилиты.

Смотря на все вышеперечисленные преимущества и удобства, я выбрал именно эту IDE для создания своего приложения на Android.

## Описание языка программирования Java

В Android Studio включены такие языки как Java и Kotlin(о нем в 1.6).

Приложения для Android разрабатываются с использованием языка Java. На данный момент это действительно ваш единственный вариант для нативных приложений. Java –очень популярный язык программирования, разработанный Sun Microsystems (теперь принадлежащий Oracle). Разработанные намного позже после C и C ++, Java включает в себя многие из мощных функций этих мощных языков программирования и устраняет некоторые из их недостатков. Тем не менее, языки программирования настолько же сильны, как и их библиотеки. Эти библиотеки существуют, чтобы помочь разработчикам создавать приложения.

*Плюсы:*

1.Подходит для разработки как нативных, так и кроссплатформенных приложений.

2.Огромное количество готовых библиотек, в том числе для проектов, ориентированных на Android.

3.Приложения, созданные с помощью Java, легче и компактнее по сравнению с их аналогами на основе Kotlin, что улучшает общий пользовательский опыт.

4.Высокая скорость разработки.

*Минусы:*

1.Это довольно сложный язык в плане синтаксиса, что увеличивает вероятность ошибок и багов.

2.Java-разработчики сталкиваются с проблемами при разработке приложений на базе Android API из-за определённых ограничений в коде.

3.Требуется больше памяти по сравнению с приложениями на Kotlin.

## 1.6 Описание языка программирования Kotlin

Kotlin появился, когда Android начал требовать более современных программных решений и тех возможностей, которые Java просто не был в состоянии охватить. JetBrains – создатели одной из самых популярных IDE под названием Intelliji IDEA – также являются «отцами-основателями» Kotlin.

Специально для Android они создали новый язык программирования с открытым исходным кодом на базе JVM (виртуальной машины Java). Это даёт разработчикам возможность обновлять устаревшие приложения Java, используя инструменты Kotlin и переносить старые проекты, ранее выполненные на Java, в новую среду, не переписывая полностью.

*Плюсы:*

1.Главным плюсом Kotlin, бесспорно, является минимализм. То, что потребовало бы 50 строк кода в Java, в Kotlin можно уместить в 2–3 строки. Именно поэтому этот язык стремительно набирает популярность среди разработчиков во всём мире. Минималистичный код также означает, что при создании приложений с помощью Kotlin гораздо меньше вероятность допустить и не заметить ошибки.

2.Kotlin позволяет разрабатывать чистые API.

3.С помощью байт-кода Java библиотеки Java вместе с его фреймворками можно легко использовать в Kotlin, что делает переход с одного языка программирования на другой довольно плавным.

4.Kotlin включает null в свою систему типов, что невозможно в среде Java.

5.Разработчикам Kotlin доступна широчайшая библиотека Anko, включающая сотни проектов, доступных на GitHub.

*Минусы:*

1.Крутая кривая обучения. Стоит знать, что Kotlin использует довольно сложный синтаксис, к которому непросто привыкнуть.

2.Kotlin в некоторых случаях имеет более медленную скорость компиляции по сравнению с Java, хотя во многих других аспектах этот язык превосходит своего предшественника.

3.Сообщество Kotlin довольно молодое, поэтому количество доступных ресурсов для обучения довольно ограничено, что может затруднить поиск решений сложных проблем. Однако по мере развития языка количество тематических ресурсов также растёт.

4.Из-за сравнительной новизны Kotlin может быть сложно найти опытных разработчиков, которые могли бы выступать в роли наставников для команд.

5.В Android Studio есть такие функции, как автозаполнение и компиляция, которые работают медленнее в приложениях на основе Kotlin, чем в их аналогах, созданных с использованием Java.

## 1.7 Выводы к первой главе

Таким образом для разработки приложения для моей курсовой работы были выбраны: IDE – Android Studio и язык программирования Java.

## 2 Разработка приложения «Музыкальный плеер»

## 2.1 Разработка структуры приложения

Создание приложения включает в себя следующие основные этапы:

1. Разработка визуального интерфейса;
2. Написание кода;
3. Отладка программы, устранение ошибок и т.д.

При надобности стоит воспользоваться документацией выбранного фреймворка.

Для разработки всех этапов была использована среда Android Studio.

Интерфейс создан из нескольких элементов: активного дисплея (activity\_main.xml), поля с элементами данными – песнями(recycler\_item.xml) и активное окно проигрывателя(activity\_misic\_player.xml) (рисунки 2.1-2.3).

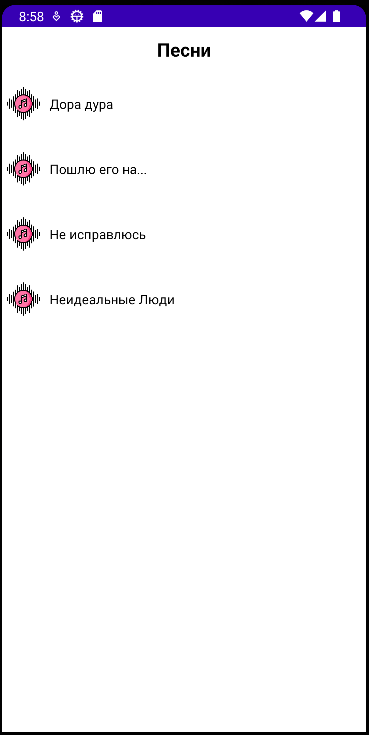


Рисунок 2.1 – Активное музыкальное окно

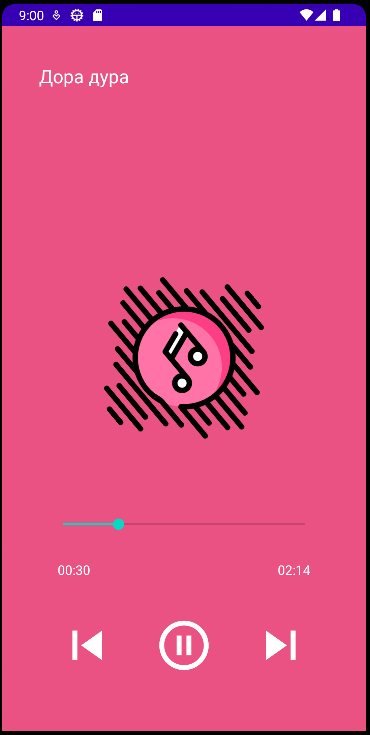


Рисунок 2.2 – Окно музыкального проигрывателя

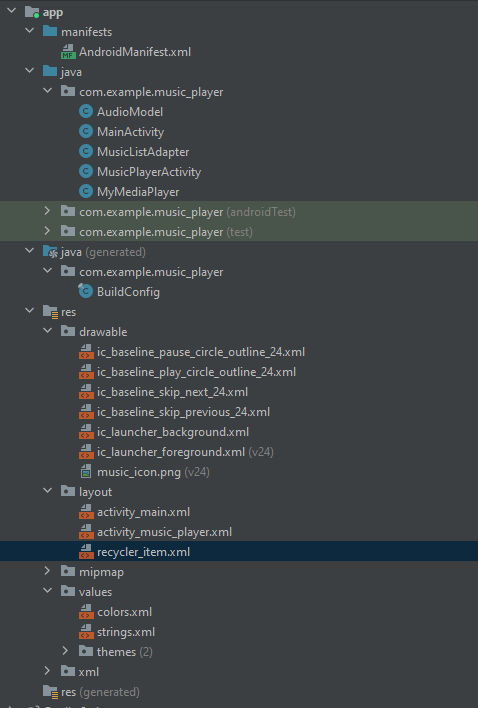


Рисунок 2.3 – Все элементы приложения

­

На этом интерфейсе располагаются:

* главный класс
* класс, в котором определены все переменные используемые в приложении
* класс адаптер, связывающий данные с компонентами и отслеживающий изменения.

## 2.2 Разработка программного кода

Проект состоит из следующих компонентов (рисунок 2.3):

1. Класс AudioModel содержит список песен и имеет поля: данные, название песни и длительность. Мы добавляем список музыки в список аудио моделей.
2. Класс MainActivity – главный класс. В нем происходит основное действие, а именно проверка разрешение на доступ к внешнему хранилищу устройства(checkPermission) и получение разрешения на доступ к хранилищу(requestPermission), также тут происходит добавление аудио моделей в список песен если они существуют.
3. Класс MusicListAdapter. Здесь мы создаем публичный класс ViewHolder, который будет расширять RecyclerView. Также в данном классе находится держатель вида(onCreateViewHolder), который возвращает класс ViewHolder, отмена привязки держателя просмотра(onBindViewHolder), где мы привяжим все данные из списка представления(в этом классе мы устанавливаем значек и название песни.
4. Класс MusicPlayerActivit создан для активного музыкального окна. Здесь проработаны такие кнопки как: воспроизведение, пауза, воспроизведение следующего трека и предыдущего. А также здесь прописано время в MMSS, поэтому мы можем увидеть длительность композиции и текущий момент воспроизведения.
5. Класс MyMediaPlayer – вспомогательный класс. Он создается для того чтобы мы могли получить доступ к медиа плееру из любого вида деятельности, здесь у нас есть статические переменные и метод для осуществления этого.

## 2.3 Описание приложения

Приложение создано на основе Android Studio технологии. Использовался современный язык программирования Java. Интерфейс прост и понятен.

Приложение создано для удобного прослушивания треков и композиций в любом месте.

Программа выполняет следующие базовые функции:

1.Воспроизведение песни.

2.Постановка на паузу песни.

3.Воспроизведение следующей песни.

4.Воспроизведение предыдущей песни.

Благодаря правильно составленной логике приложения, в него легко можно добавлять другие функции.

Приложение создано для системы Android. Это позволяет пользоваться приложением с устройств, на которых реализована система Android.

## 2.4 Руководство по использованию

Впервые запустив приложение пользователь видит данный экран с предоставлением разрешения(рисунок 2.4)

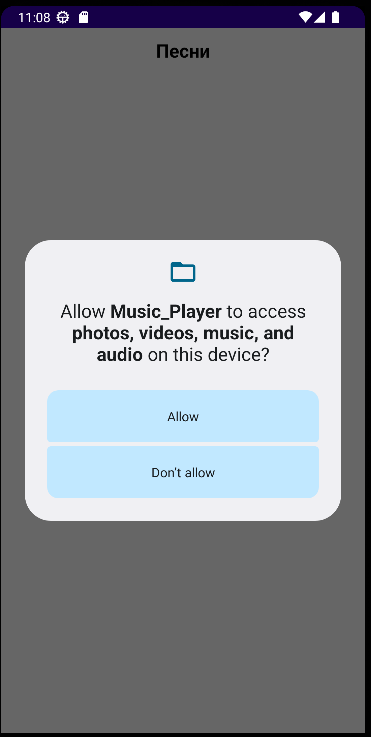


Рисунок 2.4 – Запрос разрешения

Если разрешение не предоставили, то списка песен не будет(рисунок 2.5), пользователю будет предоставлена подсказка(рисунок 2.6) и придется предоставлять разрешение из настроек(рисунок 2.7)

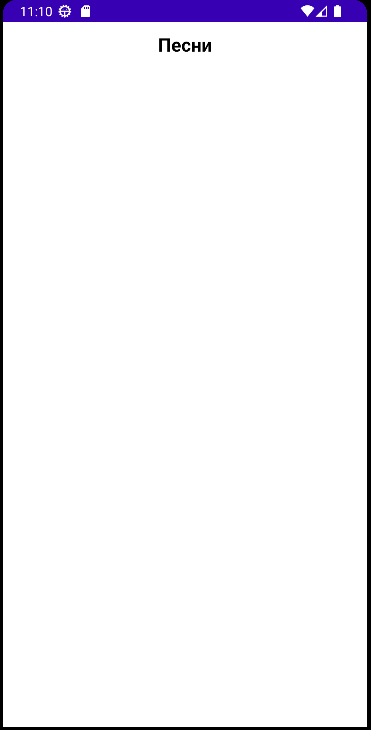


Рисунок 2.5 – Вид списка без разрешения

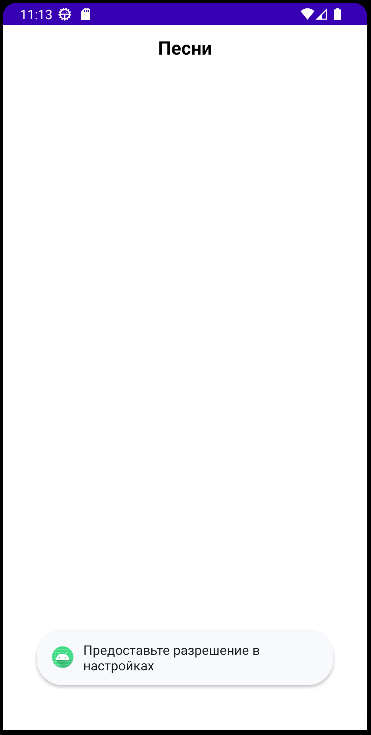


Рисунок 2.6 – Подсказка о предоставлении разрешения

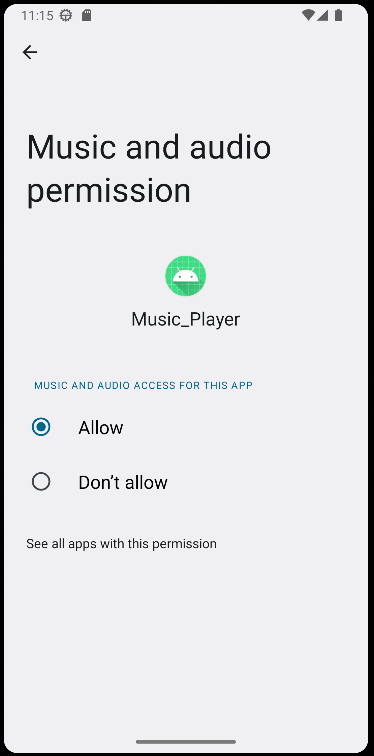


Рисунок 2.7 – Разрешение в настройках

После предоставления разрешения перед пользователем появиться список песен доступных для воспроизведения(рисунок 2.8)

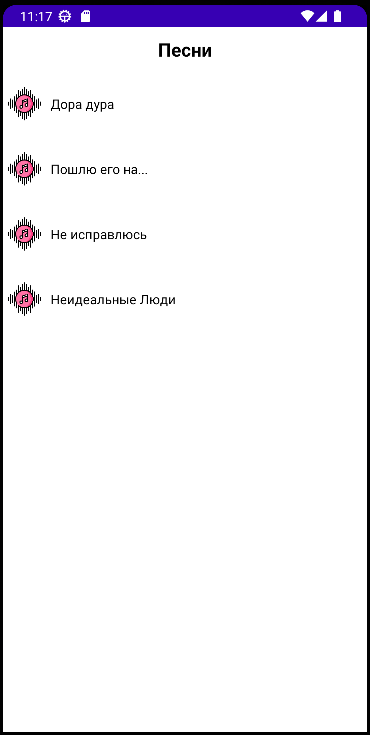


Рисунок 2.8 – Список песен

После нажатия на любую композицию перед вами выпадет активное окно проигрывателя(рисунок 2.9)

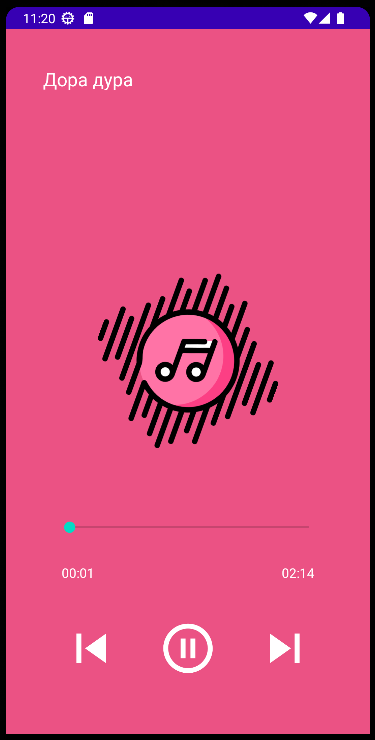


Рисунок 2.9 – Активное окно проигрывателя

Здесь пользователю доступны 3 кнопки: пауза, следующий трек и предыдущий(рисунок 2.10)



Рисунок 2.10 – Доступные кнопки

## 2.5 Контрольный запуск

На данном этапе убедимся, что все доступные кнопки пользователя работают исправно(рисунки 2.11-2.14)

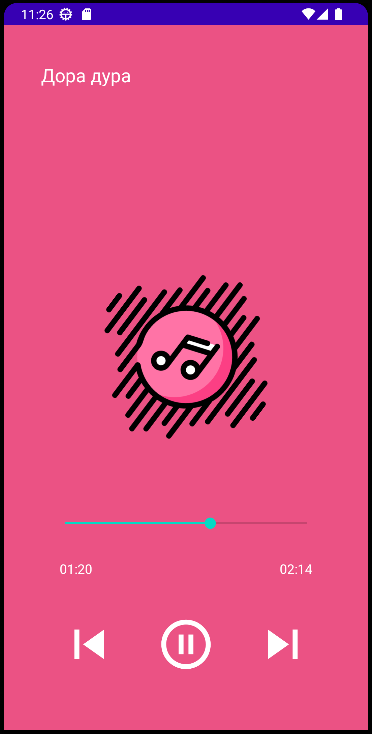


Рисунок 2.11 – Текущий трек

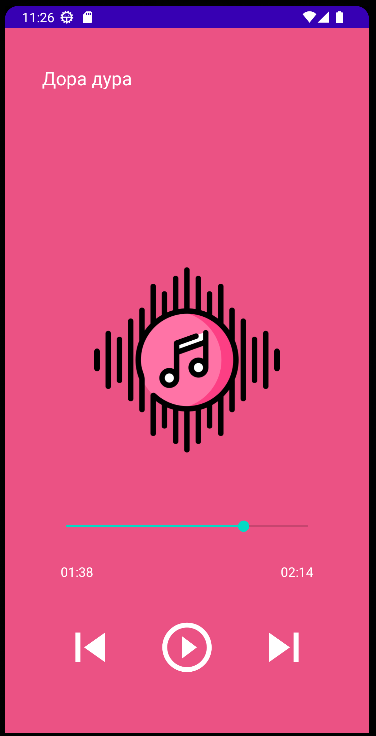


Рисунок 2.12 – Постановка текущего трека на паузу

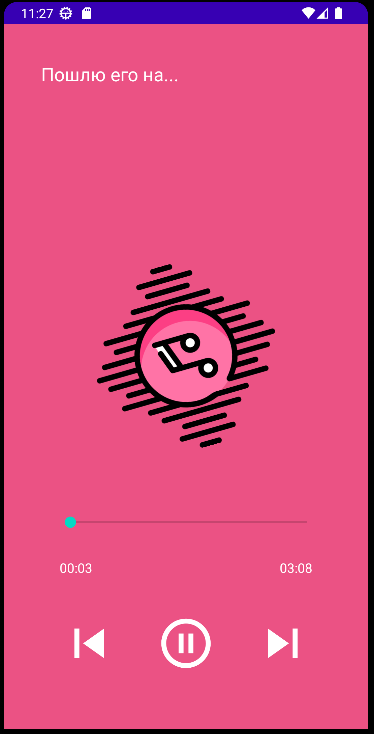


Рисунок 2.13 Воспроизведение следующего трека

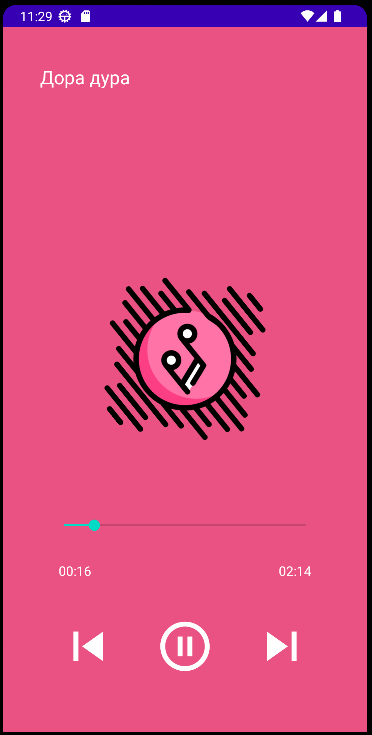


Рисунок 2.14 – Воспроизведение предыдущего трека

## 2.5 Выводы ко второй главе

В данной главе удалось полностью проверить приложение, написанное на языке программирования Java в среде Android Studio. Были проведены тесты, исходя из которых видна правильность выполняемых действий. Это приложение легко и понятно в использовании. Также оно может сделать жизнь некоторых людей проще. Приложение разработано для платформы Android.

## Заключение

В данной работе рассмотрены подходы к разработке мобильных приложений, обсуждены преимущества и недостатки каждого из них. Приложение, которое создано в ходе данной работы это «Музыкальный плеер». Приложение доступно для использования на платформе Android.

Приложение реализовано с помощью среды разработки Android Studio на языке Java. Данное приложение предусматривает добавление новых функций и улучшение старых.

## Список использованных источников

1. **Шилдт Г. Java. Полное руководство. –10-e изд./** **Г. Шилдт –СПб.: Диалектика-Вильямс, 2018 . –1488 с.**
2. **Блох Д.** Java. Эффективное программирование. –3-e изд./Д. Блох –СПб.: Диалектика-Вильямс, 2019. –464 с.
3. **Гриффитс Д**. Head First. Kotlin. –3-e изд./Д. Гриффитс Д. Гриффитс –М.: Питер, 2020. –464 с.
4. **Ян Ф.** Дарвин Android. Сборник рецептов. Задачи и решения для разработчиков приложений./Ф. Ян–2-e изд. –СПб.: Диалектика, 2018. –768 с.
5. **Дейтел П.** Android для разработчиков. –3-e изд./П. Дейтел, Х. Дейтел, А. Уолд –СПб.: Питер, 2016. –512 с.

# 

# Приложение А

(обязательное)

Листинг кода программы

**MainActivity //Основной код**

package com.example.music\_player;

import androidx.appcompat.app.AppCompatActivity;

import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;

import android.Manifest;

import android.content.pm.PackageManager;

import android.database.Cursor;

import android.media.MediaPlayer;

import android.os.Bundle;

import android.provider.MediaStore;

import android.view.View;

import android.widget.TextView;

import android.widget.Toast;

import java.io.File;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

RecyclerView recyclerView;

TextView noMusicTextView;

ArrayList<AudioModel> songsList = new ArrayList<>();

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

recyclerView = findViewById(R.id.recycler\_view);

noMusicTextView = findViewById(R.id.no\_songs\_text);

if (checkPermission() == false)

{

requestPermission();

return;

}

String[] projection =

{

MediaStore.Audio.Media.TITLE,

MediaStore.Audio.Media.DATA,

MediaStore.Audio.Media.DURATION

};

String selection = MediaStore.Audio.Media.IS\_MUSIC + "!= 0";

Cursor cursor = getContentResolver().query(MediaStore.Audio.Media.EXTERNAL\_CONTENT\_URI, projection, selection, null, null);

while(cursor.moveToNext())

{

AudioModel songData = new AudioModel(cursor.getString(1), cursor.getString(0), cursor.getString(2) );

if(new File (songData.getPath()).exists()) {

songsList.add(songData);

}

}

if(songsList.size() == 0)

{

noMusicTextView.setVisibility(View.VISIBLE);

}

else

{

recyclerView.setLayoutManager(new LinearLayoutManager(this));

recyclerView.setAdapter(new MusicListAdapter(songsList, getApplicationContext()));

}

}

boolean checkPermission()

{

int result = ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.READ\_EXTERNAL\_STORAGE);

if(result == PackageManager.PERMISSION\_GRANTED)

{

return true;

}

else

{

return false;

}

}

void requestPermission()

{

if(ActivityCompat.shouldShowRequestPermissionRationale(MainActivity.this, Manifest.permission.READ\_EXTERNAL\_STORAGE))

{

Toast.makeText(MainActivity.this, "Предоставьте разрешение в настройках", Toast. LENGTH\_SHORT).show();

}

else

{

ActivityCompat.requestPermissions(MainActivity.this, new String[]{Manifest.permission.READ\_EXTERNAL\_STORAGE}, 123);

}

}

protected void onResume()

{

super.onResume();

if(recyclerView!=null)

{

recyclerView.setAdapter(new MusicListAdapter(songsList, getApplicationContext()));

}

}

}

**AudioModel**

package com.example.music\_player;

import java.io.Serializable;

public class AudioModel implements Serializable {

String path;

String title;

String duration;

public String getPath() {

return path;

}

public void setPath(String path) {

this.path = path;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getDuration() {

return duration;

}

public void setDuration(String duration) {

this.duration = duration;

}

public AudioModel(String path, String title, String duration) {

this.path = path;

this.title = title;

this.duration = duration;

}

}

**MusicListAdapter**

package com.example.music\_player;

import android.content.Context;

import android.content.Intent;

import android.graphics.Color;

import android.media.MediaPlayer;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.ImageView;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;

public class MusicListAdapter extends RecyclerView.Adapter<MusicListAdapter.ViewHolder>{

ArrayList<AudioModel> songsList;

Context context;

public MusicListAdapter(ArrayList<AudioModel> songsList, Context context) {

this.songsList = songsList;

this.context = context;

}

@Override

public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

View view = LayoutInflater.from(context).inflate(R.layout.recycler\_item,parent, false);

return new MusicListAdapter.ViewHolder(view);

}

@Override

public void onBindViewHolder(MusicListAdapter.ViewHolder holder, int position) {

AudioModel songData = songsList.get(position);

holder.titleTextView.setText(songData.getTitle());

if(MyMediaPlayer.currentIndex == position)

{

holder.titleTextView.setTextColor(Color.parseColor("#EB5284"));

}

else

{

holder.titleTextView.setTextColor(Color.parseColor("#000000"));

}

holder.itemView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

MyMediaPlayer.getInstance().reset();

MyMediaPlayer.currentIndex = position;

Intent intent = new Intent(context, MusicPlayerActivity.class);

intent.putExtra("LIST", songsList);

intent.setFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK);

context.startActivity(intent);

}

});

}

@Override

public int getItemCount() {

return songsList.size();

}

public class ViewHolder extends RecyclerView.ViewHolder

{

TextView titleTextView;

ImageView iconImageView;

public ViewHolder(View itemView)

{

super(itemView);

titleTextView = itemView.findViewById(R.id.music\_title\_text);

iconImageView = itemView.findViewById(R.id.icon\_view);

}

}

}

**MyMediaPlayer**

package com.example.music\_player;

import android.media.MediaPlayer;

public class MyMediaPlayer {

static MediaPlayer instance;

public static MediaPlayer getInstance()

{

if(instance == null)

{

instance = new MediaPlayer();

}

return instance;

}

public static int currentIndex = -1;

}

**MusicPlayerActivity**

package com.example.music\_player;

import androidx.appcompat.app.AppCompatActivity;

import android.media.MediaPlayer;

import android.os.Bundle;

import android.os.Handler;

import android.widget.ImageView;

import android.widget.SeekBar;

import android.widget.TextView;

import java.io.IOException;

import java.util.ArrayList;

import java.util.concurrent.TimeUnit;

public class MusicPlayerActivity extends AppCompatActivity {

TextView titleTv, currentTimeTv, totalTimeTv;

SeekBar seekBar;

ImageView pausePlay, nextBtn, previousBtn, musicIcon;

ArrayList<AudioModel> songsList;

AudioModel currentSong;

MediaPlayer mediaPlayer = MyMediaPlayer.getInstance();

int x = 0;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_music\_player);

titleTv = findViewById(R.id.song\_title);

currentTimeTv = findViewById(R.id.current\_time);

totalTimeTv = findViewById(R.id.total\_time);

seekBar = findViewById(R.id.seek\_bar);

pausePlay = findViewById(R.id.pause\_play);

previousBtn = findViewById(R.id.previous);

musicIcon = findViewById(R.id.music\_icon\_big);

nextBtn = findViewById(R.id.next);

titleTv.setSelected(true);

songsList = (ArrayList<AudioModel>) getIntent().getSerializableExtra("LIST");

setResourcesWithMusic();

MusicPlayerActivity.this.runOnUiThread(new Runnable() {

@Override

public void run() {

if(mediaPlayer != null)

{

seekBar.setProgress(mediaPlayer.getCurrentPosition());

currentTimeTv.setText(convertToMMSS(mediaPlayer.getCurrentPosition()+""));

if(mediaPlayer.isPlaying())

{

pausePlay.setImageResource(R.drawable.ic\_baseline\_pause\_circle\_outline\_24);

musicIcon.setRotation(x++);

}

else

{

pausePlay.setImageResource(R.drawable.ic\_baseline\_play\_circle\_outline\_24);

musicIcon.setRotation(0);

}

}

new Handler().postDelayed(this, 100);

}

});

seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {

@Override

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {

if(mediaPlayer != null && fromUser)

{

mediaPlayer.seekTo(progress);

}

}

@Override

public void onStartTrackingTouch(SeekBar seekBar) {

}

@Override

public void onStopTrackingTouch(SeekBar seekBar) {

}

});

}

void setResourcesWithMusic()

{

currentSong = songsList.get(MyMediaPlayer.currentIndex);

titleTv.setText(currentSong.getTitle());

totalTimeTv.setText(convertToMMSS(currentSong.getDuration()));

pausePlay.setOnClickListener(v-> pausePlay());

nextBtn.setOnClickListener(v-> playNextSong());

previousBtn.setOnClickListener(v-> playPreviousSong());

playMusic();

}

private void playMusic()

{

mediaPlayer.reset();

try {

mediaPlayer.setDataSource(currentSong.getPath());

mediaPlayer.prepare();

mediaPlayer.start();

seekBar.setProgress(0);

seekBar.setMax(mediaPlayer.getDuration());

}catch(IOException e)

{

e.printStackTrace();

}

}

private void playNextSong()

{

if(MyMediaPlayer.currentIndex == songsList.size()-1)

return;

MyMediaPlayer.currentIndex +=1;

mediaPlayer.reset();

setResourcesWithMusic();

}

private void playPreviousSong()

{

if(MyMediaPlayer.currentIndex == 0)

return;

MyMediaPlayer.currentIndex -=1;

mediaPlayer.reset();

setResourcesWithMusic();

}

private void pausePlay()

{

if(mediaPlayer.isPlaying())

{

mediaPlayer.pause();

}

else

{

mediaPlayer.start();

}

}

public static String convertToMMSS(String duration)

{

Long millis = Long.parseLong(duration);

return String.format("%02d:%02d",

TimeUnit.MILLISECONDS.toMinutes(millis) % TimeUnit.HOURS.toMinutes(1),

TimeUnit.MILLISECONDS.toSeconds(millis) % TimeUnit.MINUTES.toSeconds(1));

}

}