

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

АУДИОПЛЕЕР ДЛЯ ANDROID

Студент Беляков А.М. 20.06.24
(подпись, дата)

Руководитель Слепцов Н.В. 20.06.24
(подпись, дата)

Нормоконтролёр Бычков А.С. 20.06.24
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 г. № 11)

Заведующий кафедрой Митрохин М.А.
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
24.06.2024г. № 37)

Секретарь ГЭК Гурин Е.И.
(подпись)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«19» апреля 2024 г.

ЗАДАНИЕ НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Беляков Арсений Михайлович гр. 20ВВП1 факультета Вычислительной техники.

2. Тема работы: «Аудиоплеер для Android»

Тема утверждена приказом ПГУ № 405/о от "19" апреля 2024 г.

3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Слепцов Н.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать и реализовать аудиоплеер для платформы Android. Данное приложение должно предоставлять пользователям удобные и многофункциональные инструменты для воспроизведения различных аудиофайлов. Аудиоплеер должен поддерживать автоматическую обработку и организацию аудиофайлов, обеспечивая пользователям быстрый доступ к их библиотеке. Программа должна улучшить пользовательский опыт, ускорить и облегчить процесс прослушивания музыки, что обеспечит более качественное и эффективное использование мобильного устройства.

5. Перечень подлежащих разработке вопросов
Требуется провести анализ предметных областей: изучить существующие аналоги, выделить достоинства и недостатки. Выбрать язык программирования и другие составляющие разработки, подходящие для оптимальной реализации задачи. Определить архитектуру приложения и описать внутреннюю организацию. Изучить функционал библиотек для работы с Android и базой данных SQLite.

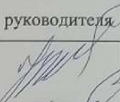
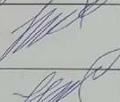
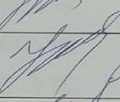
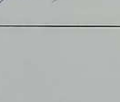
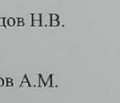
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

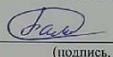
2

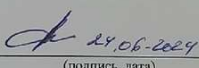
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	20	26.04.2024 – 27.04.2024	
2	Анализ требований	32	28.04.2024 – 30.04.2024	
3	Разработка основной части	82	31.04.2024 – 14.05.2024	
4	Разработка логической модели	56	15.05.2024 – 19.05.2024	
5	Отладка и тестирование	50	19.05.2024 – 23.05.2024	
6	Оформление пояснительной записки	54	24.05.2024 – 31.05.2024	

Дата выдачи задания "19" апреля 2024 г.

Руководитель бакалаврской работы  Слепцов Н.В.
(подпись, дата)

Задание к исполнению принял студент  Беляков А.М.
(подпись, дата)

Работу к защите допустить
Декан факультета  24.06.2024 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 105 листов, 28 рисунков, 7 источников, 3 приложения

АУДИОПЛЕЕР, ANDROID, KOTLIN, MVVM

Цель работы – разработать программу, обеспечивающую создание плейлистов и проигрывание музыки из данных плейлистов. Для достижения цели была разработана программа, отвечающее следующим поставленным задачам: создание плейлиста, удаление плейлиста, изменение громкости воспроизводимой музыки, остановка/продолжение воспроизводимой музыки, переход к предыдущему и следующему треку перемещение по воспроизводимой музыке.

В результате проведенной работы было разработано приложение аудиоплеер под операционную систему Android в среде программирования Android Studio JellyFish. В качестве языка программирования был выбран Kotlin.

					ПГУ 1.090301.03.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Беляков А. М.	<i>[Подпись]</i>	10.06.19	Лит.	Лист	Листов
Провер.		Слепцова Н.В.	<i>[Подпись]</i>	10.06.19		2	105
Н. Контр		Беляков А.С.	<i>[Подпись]</i>	10.06.19	ФВТ, каф ВТ		
Утверд.		Митрохин М.А.	<i>[Подпись]</i>	10.06.19	гр. 20ВВП1		

Аудиоплеер для Android
Пояснительная записка

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Аудиоплеер для Android.....	7
1.1 Анализ темы	7
1.2 Анализ аналогов.....	8
1.2.1 Spotify	9
1.2.2 YouTube Music.....	12
1.2.3 Poweramp.....	14
1.2.4 VLC for Android.....	16
1.2.5 Вывод.....	18
1.3 Постановка задачи	18
2 Выбор инструментов разработки	20
2.1 Выбор языка программирования	20
2.1.1 Язык программирования «Python»	20
2.1.2 Язык программирования «C/C++»	21
2.1.3 Язык программирования «Java»	22
2.1.4 Язык программирования «Kotlin»	22
2.1.5 Язык программирования «JavaScript».....	23
2.1.6 Вывод.....	24
2.2 Выбор среды разработки.....	24
2.3 Выбор базы данных	25
2.3.1 Sqlite.....	25
2.3.2 Room	25
2.3.3 Firebase Firestore	25
3 Архитектура приложения.....	27
3.1 Архитектурный паттерн «MVVM».....	27
4 Реализация приложения	29
4.1 Описание функционала	29
4.2 Описание работы приложения	32

4.3 Основная реализация приложения.....	33
4.3.1 Блок Model	34
4.3.2 Блок View	37
4.3.3 Блок ViewModel.....	41
4.3.4 Вспомогательные классы	43
5 Тестирование	46
5.1 Создание плейлиста.....	46
5.2 Восстановление сессии	50
5.3 Управление воспроизводимой музыкой.....	51
5.4 Вывод	54
6 Безопасность жизнедеятельности.....	55
6.1 Правила и техника безопасности при работе с ПК	55
6.2 Правила и техника безопасности при работе со смартфоном	56
Список использованных источников	59
Приложение А	60
Приложение Б	66
Приложение В.....	96

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

ОС – операционная система

Фреймворк – программная платформа, которая упрощает разработку программного продукта, определяет структуру проекта

DAO - (Data Access Object) - объект доступа к данным

IDE – (Integrated Development Environment) – интегрированная среда разработки

MVVM (Model View ViewModel) – шаблон проектирования архитектуры приложения

UI – (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		5

Введение

В настоящее время мобильные телефоны и планшеты занимают важную роль в жизни человека. Различные устройства на системе Android предоставляют широкий спектр возможностей для пользователя. Довольно востребованным классом приложений являются различные аудио плееры, так как мало кто из людей хочет покупать и иметь отдельный гаджет, когда всегда под рукой есть смартфон.

Важность создания приложения для проигрывания аудио файлов обусловлена постоянным развитием мобильных технологий и популярностью различных мультимедийных приложения среди пользователей мобильных устройств.

Целью данной выпускной квалификационной работы является разработка аудио плеера для операционной системы Android, который будет отвечать современным требованиям пользователей и давать им доступ к комфортному инструменту для прослушивания аудио контента.

					ПГУ 1.090301.03.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

1 Аудиоплеер для Android

1.1 Анализ темы

В современном мире всё больше граждан ищут способ занять себя в транспорте, в котором они могут находиться довольно долго, особенно в крупных городах, таких как Москва или Санкт-Петербург. Время, проведенное отдельным человеком в дороге, например, на работу или учёбу, может достигать одного-двух, а то и трёх часов в день, что толкает людей искать различные пути сделать его более приятным и продуктивным. Одним из самых частых вариантов досуга в данной ситуации является прослушивание различных аудиофайлов, это может быть, как музыка, так и аудиокниги или радиопрограммы. Это не только скрасит время поездки, но и позволит получить новую информацию или развлечься.

Раньше для прослушивания аудиофайлов использовались плееры — специально предназначенные для этой цели отдельные устройства. Аудио плееров было множество видов - кассетные, дисковые, с наушниками, с динамиком. Однако с развитием технологий и увеличением функциональности смартфонов плееры как отдельное устройство постепенно отходят на второй план. Сегодня их место успешно занимают различные приложения для смартфонов, предоставляющие гораздо больше функционала и удобства для пользователя.

Таким образом, разработка аудиоплеера для операционной системы Android является не только актуальной, но и довольно важной задачей. Из года в год популярность мобильных устройств продолжает расти, и вместе с тем растет потребность пользователей в удобных и функциональных приложениях для воспроизведения аудиозаписей. В тоже время Android как операционная система занимает больше половины рынка мобильных гаджетов. Пользователи в свою очередь хотят получить доступ к своему любимому контенту в любое время и в любом месте, а приложения аудиоплееры дают им такую возможность. Помимо

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

этого, из-за увеличения разнообразия аудиоконтента возникает потребность в приложениях, которые имеют возможность управлять большими библиотеками файлов, предлагать различные способы поиска и организации контента, а также обеспечивать высокое качество звука и дополнительные функции, такие как создание списков воспроизведения, рекомендаций на основе предпочтений пользователя и многое другое.

В результате создание аудио плееров под операционную систему Android — это не только ответ на современные тенденции и запросы пользователей, но и шаг в развитии мобильных технологий и улучшении пользовательского опыта.

1.2 Анализ аналогов

Современные смартфоны превратились в универсальные портативные гаджеты, с помощью которых можно слушать музыку, радиопрограммы и аудиокниги. Именно поэтому разработка хороших плееров продолжает оставаться важной задачей для мобильных и не только разработчиков. Распространённые в настоящее время аудиоплееры, такие как Spotify, Apple Music, YouTube Music и многие другие, предлагают клиентам множество полезных и не очень функций, а также зачастую удобный интерфейс, что делает их чрезвычайно популярными на рынке.

Перед тем как создавать приложение, которое будет конкурировать с представленными выше представителями аудиоплееров, нам нужно провести детальную оценку существующих на рынке решений. Это поможет нам определить, что нужно людям от плеера, обнаружить преимущества и недостатки популярных на сегодня аудиоплееров и установить основные требования к разрабатываемой нами программе. Далее мы рассмотрим ряд популярных на данный момент плееров для операционной системы Android, таких как Spotify, YouTube Music, Poweramp и VLC для Android, чтобы определить их основные плюсы и минусы.

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

1.2.1 Spotify

Spotify – крайне популярный стриминговый сервис имеющий охват в более чем 30% общемирового аудиостриминга, дающий доступ к обширной библиотеке музыки, подкастов и аудиокниг. Клиенты могут слушать музыку как онлайн, так и офлайн, создавать и делиться плейлистами, получать персонализированные рекомендации.

Преимущества:

- обширная библиотека контента, включающая миллионы треков и подкастов;
- алгоритмы, которые предлагают музыку на основе предпочтений пользователя;
- возможность делиться плейлистами, следить за друзьями;
- интуитивно понятный и удобный дизайн.

Недостатки:

- в бесплатной версии имеется реклама и различные другие ограничения;
- для использования большинства функций требуется подключение к сети;
- может потреблять много ресурсов устройства;
- не доступен в нашем регионе.

На рисунке 1 мы можем видеть интерфейс приложения Spotify, а именно страницу с плейлистом, где мы можем видеть обложку плейлиста, список композиций в плейлисте, а также навигационное меню. На рисунке 2 мы можем видеть страницу управления проигрываемым треком

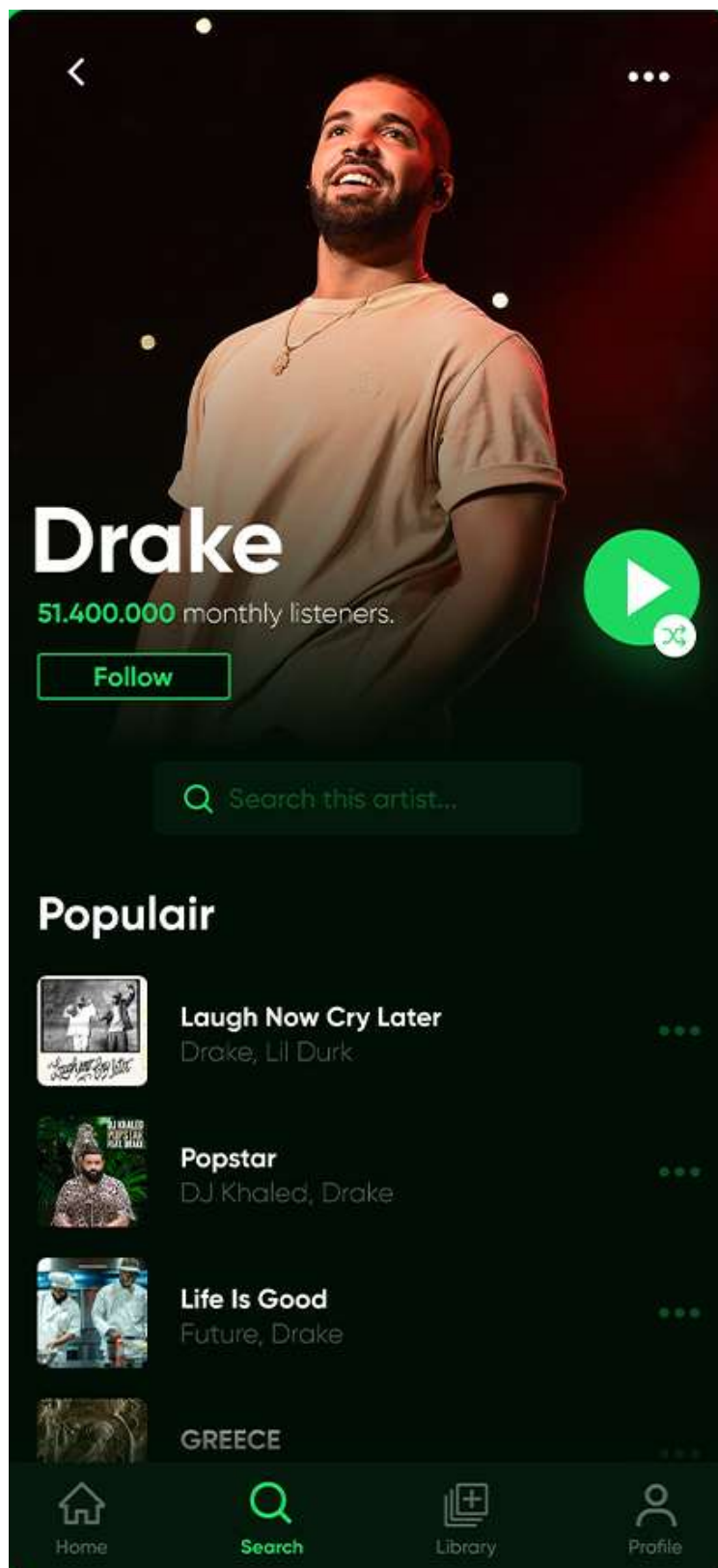


Рисунок 1 – Страница с плейлистом

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

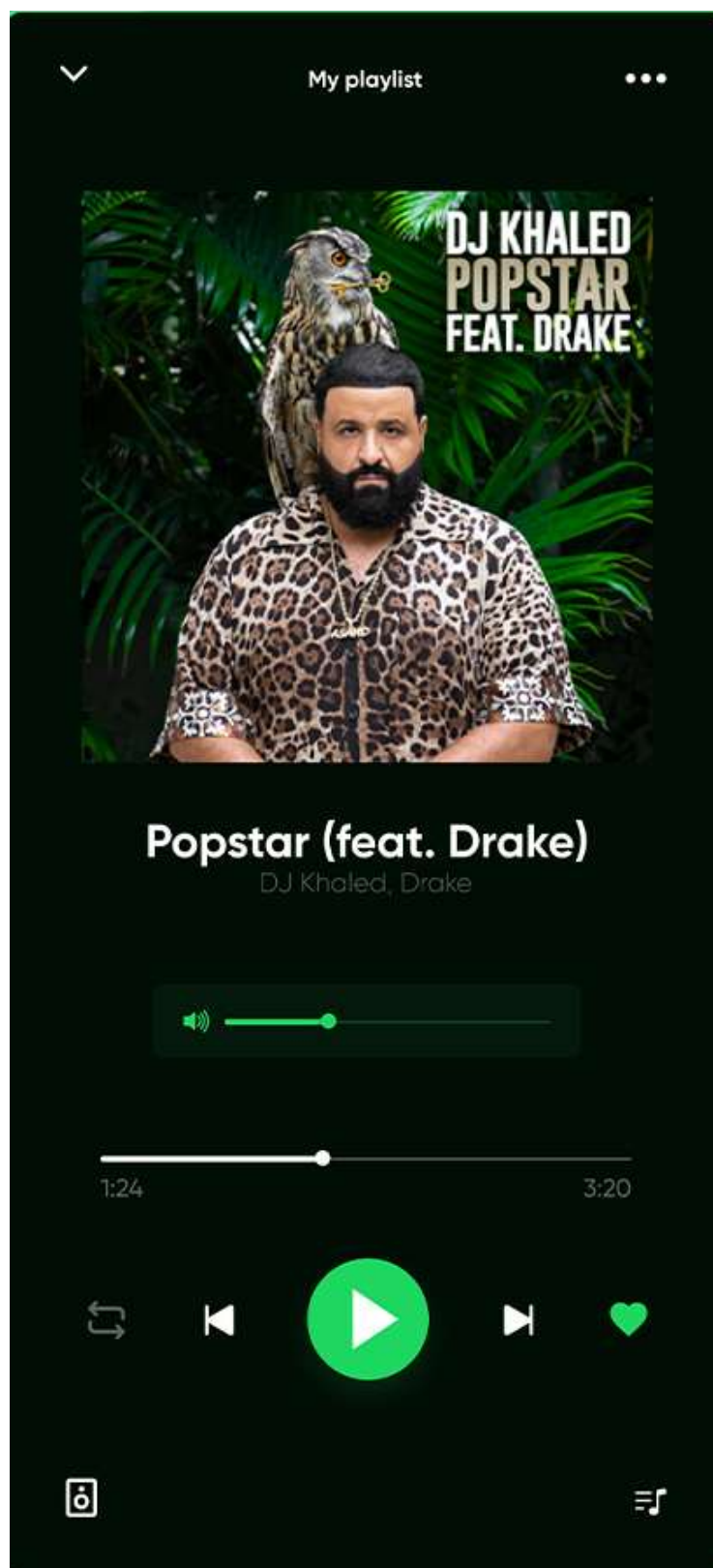


Рисунок 2 – Страница с управлением
проигрываемого трека

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

11

1.2.2 YouTube Music

YouTube Music – это стриминговый сервис, дающий доступ к большой библиотеке музыкальных треков и видеоклипов, а также подкастов. Он заменил Google Play Music и интегрирован с YouTube, что позволяет пользователям переключаться между аудио и видео режимами.

Преимущества:

- есть возможность проигрывать как треки на устройстве, так и из онлайн библиотеки;
- обычные пользователи также могут загружать песни и не только;
- интеграция с YouTube даёт возможность просмотра видеоклипов и концертных записей;
- рекомендации специально для вас подборки на основе прослушанных треков и просмотров;
- возможность скачивания треков для прослушивания без подключения к интернету.

Недостатки:

- в бесплатной версии имеется реклама и невозможность фонового воспроизведения;
- для некоторых пользователей интерфейс может быть перегруженным;
- многие функции требуют постоянного подключения к сети.

На рисунке 3 мы видим страницу плейлиста с выбором песни и навигационным меню в низу. На рисунке 4 мы видим страницу управления треком, а также рекомендации и при наличии видеоролик с платформы YouTube.

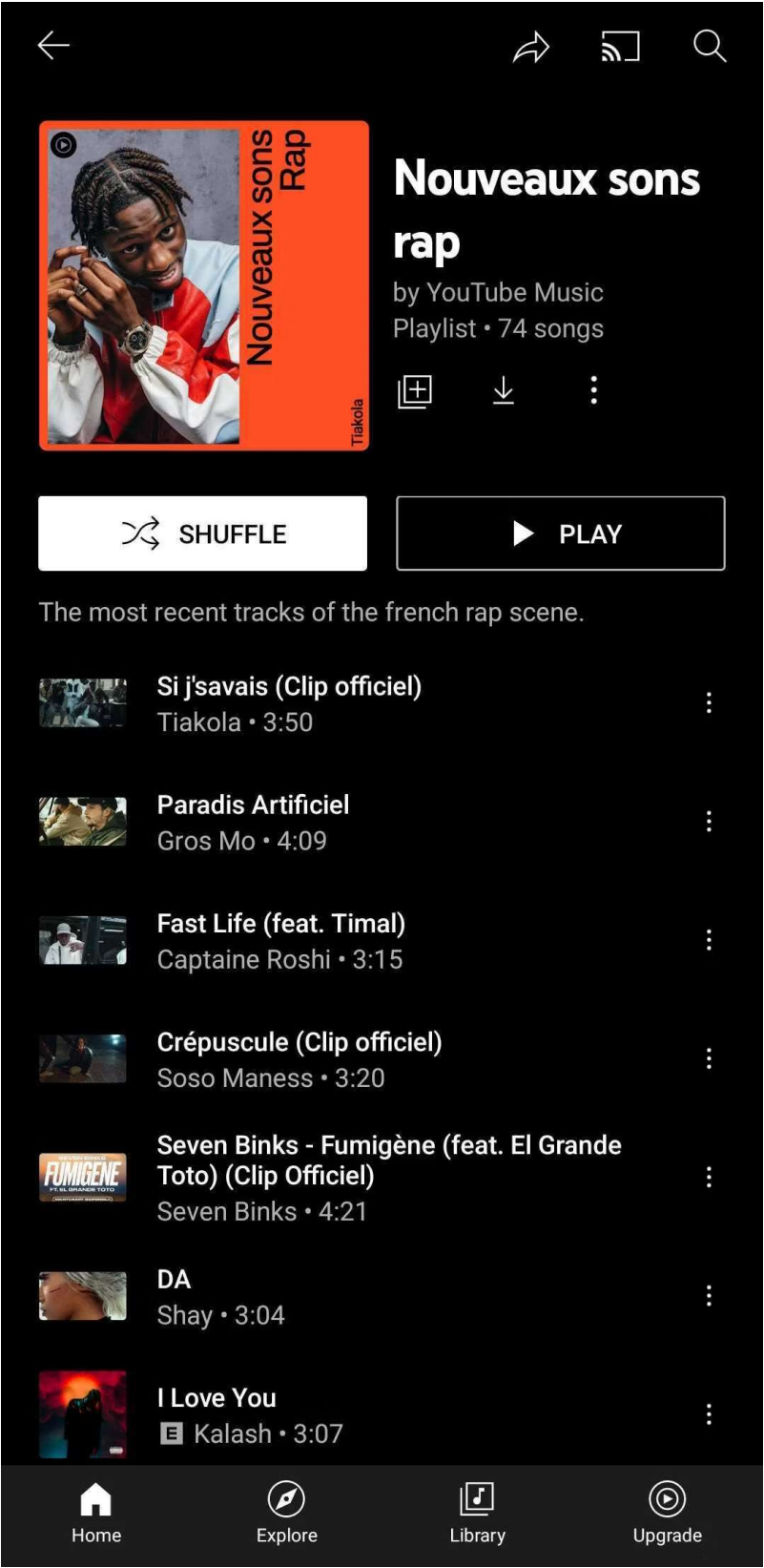


Рисунок 3 – Страница с плейлистом

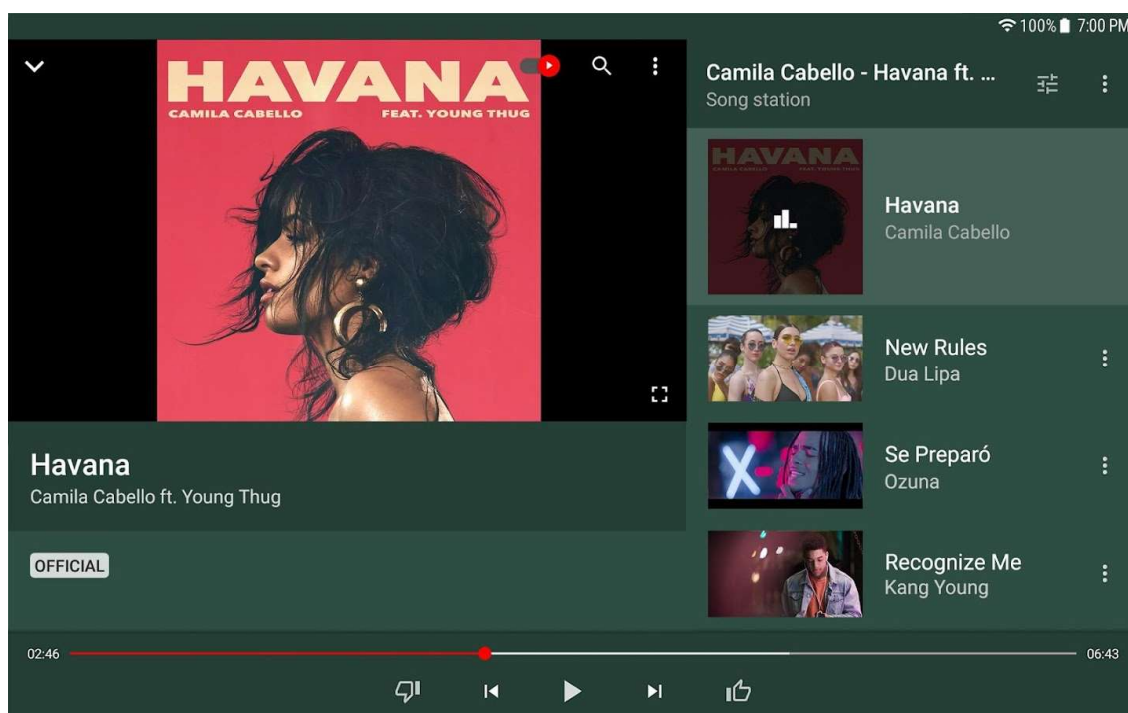


Рисунок 4 – Страница с управлением проигрываемого трека

1.2.3 Poweramp

Poweramp – это мощный аудиоплеер, который поддерживает большое количество типов аудиофайлов, а также предоставляет пользователям мощный эквалайзер. Приложение предназначено исключительно для воспроизведения локальных музыкальных файлов.

Преимущества:

- воспроизводит множество популярных и редких форматов (MP3, WAV, FLAC, APE и т. д.);
- мощный 10-полосный эквалайзер, басы и эффекты объемного звучания;
- оптимизирован для быстрого чтения и низкого потребления ресурсов.

Недостатки:

- бесплатная версия имеет ограниченные возможности;
- предназначен для воспроизведения локальных файлов без поддержки онлайн-трансляции;

– большое количество настроек может вызвать затруднения у мало опытных пользователей.

На рисунке 5 мы можем видеть управление эквалайзером. Управление воспроизводимым треком, а также его описание мы можем видеть на рисунке 6.



Рисунок 5 – Управление эквалайзером

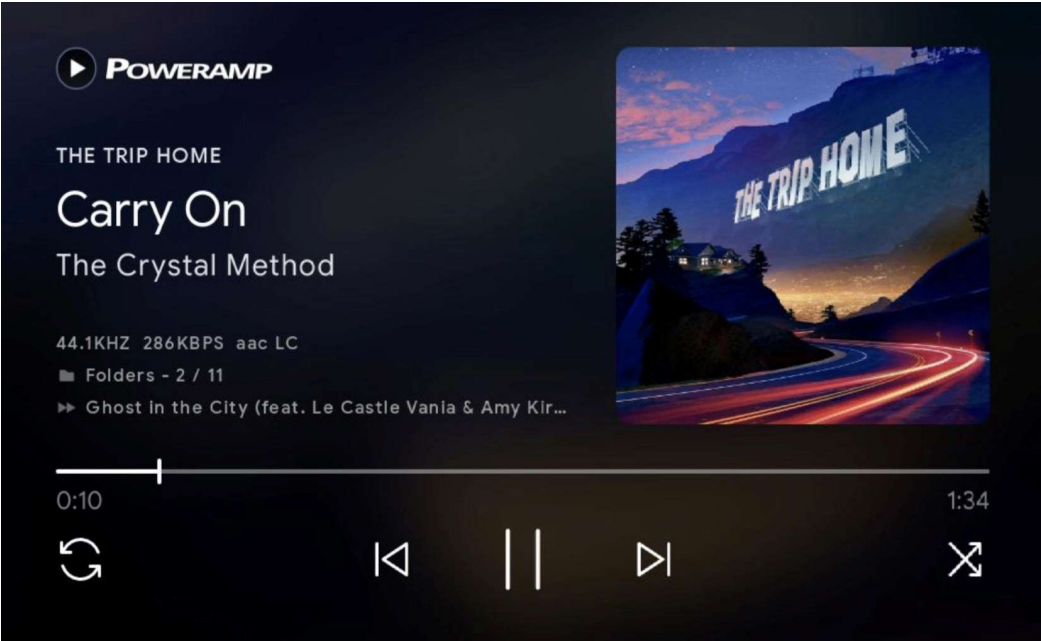


Рисунок 6 – Управление проигрываемой музыки

1.2.4 VLC for Android

VLC for Android – универсальный медиаплеер, поддерживающий воспроизведение как аудио-, так и видеофайлов. Приложение предлагает множество функций и поддерживает практически все существующие форматы медиафайлов.

Преимущества:

- воспроизводит любые аудио- и видеоформаты без необходимости в дополнительных кодеках;
- большая функциональность: потоковое воспроизведение, эквалайзер, поддержка субтитров и плейлистов;
- открытый исходный код: бесплатное приложение без рекламы и встроенных покупок;
- удобный и интуитивно понятный дизайн.

Недостатки:

- некоторые пользователи могут найти интерфейс перегруженным из-за большого количества функций;
- может потреблять много ресурсов при воспроизведении высококачественных видеофайлов;
- нет функции подбора контента на основе предпочтений пользователя.

На рисунке 7 мы видим файловый браузер приложения и нижнюю навигационную панель. На рисунке 8 мы можем наблюдать окно управления проигрываемой музыкой, а также плейлист.

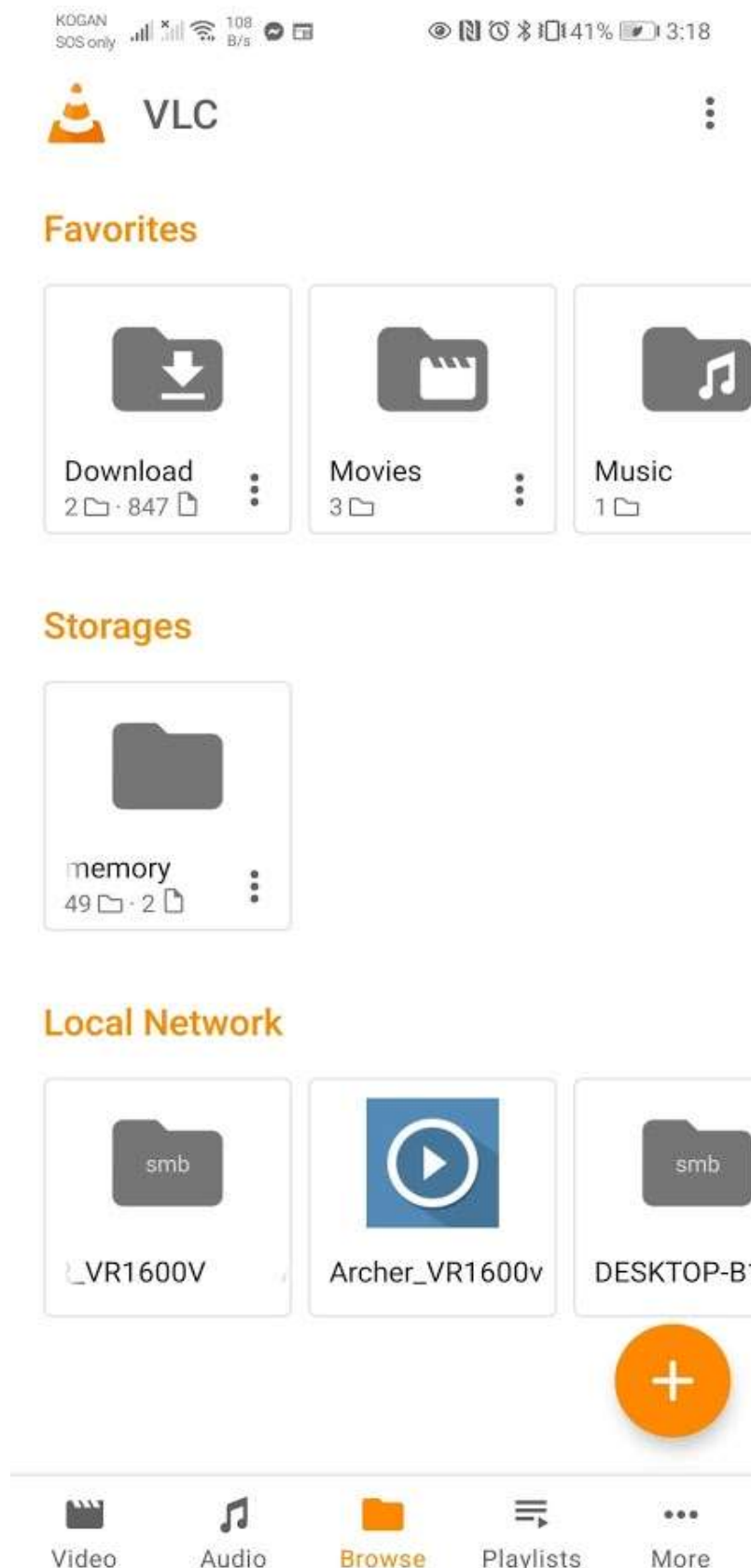


Рисунок 7 – Окно браузера файлов

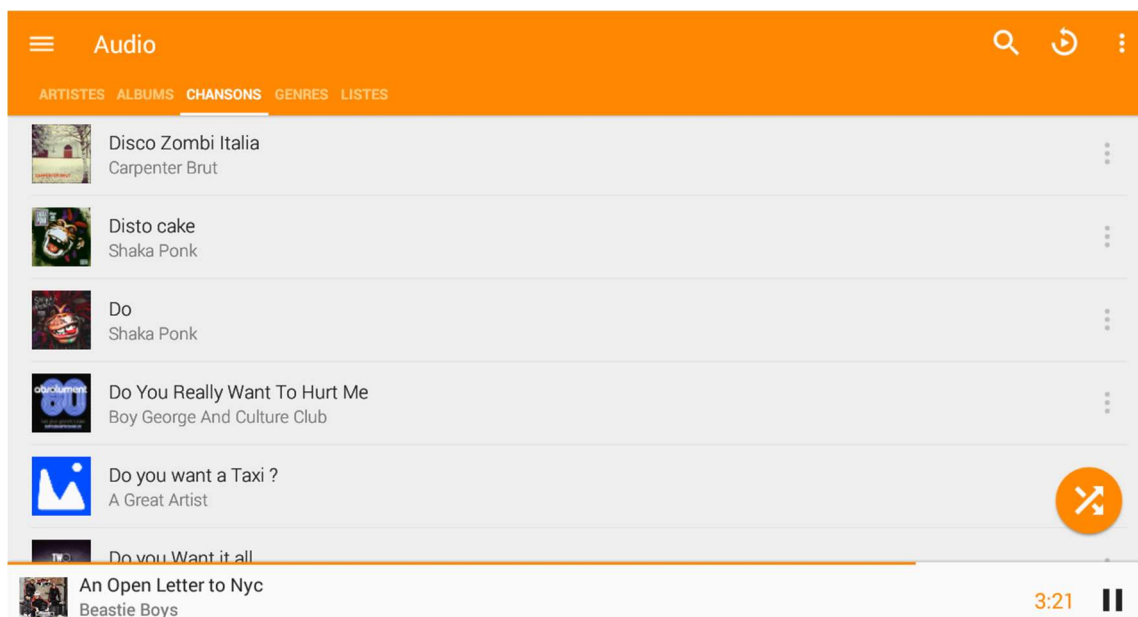


Рисунок 8 – Окно выбора трека и управления воспроизводимым

1.2.5 Вывод

Проведя анализ существующие в данный момент приложения, можно отметить, что основной функционал у большинства программ близок и почти не меняется, основные отличия в интерфейсе и из каких источников берётся музыка.

Нужно осознавать, что большинство приложений потребляют довольно много ресурсов из-за чего будут плохо работать на старых или бюджетных устройствах, поэтому создание простого и минималистичного аудиплеера является оправданным.

1.3 Постановка задачи

Необходимо разработать приложение аудиоплеер для мобильных устройств под управлением операционной системы Android. По общемировой статистике более 60% пользователей смартфонов пользуются Android [1], а с учётом общего количества людей, имеющих мобильный телефон в примерно четыре целых и три десятых миллиарда [2], даёт нам более двух миллиардов

возможных пользователей нашего приложения, именно поэтому была выбрана ОС Android.

В приложении должно быть реализовано:

- создание плейлиста;
- удаление плейлиста;
- изменение громкости воспроизводимой музыки;
- остановка/продолжение воспроизводимой музыки;
- переход к предыдущему и следующему треку;
- перемещение по воспроизводимой музыке.

Минимальные системные требования:

- операционная система Android 11;
- доступ к файловому хранилищу для чтения.

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		19

2 Выбор инструментов разработки

Для того, чтобы избежать проблем, а также ускорить и упростить разработку приложения, необходимо выбрать набор правильных инструментов, таких как среда разработки, язык и фреймворки с библиотеками.

2.1 Выбор языка программирования

Первым делом необходимо выбрать язык программирования, при не правильном выборе разработка может сильно затянуться или приложение может иметь проблемы с производительностью. В качестве критериев для выбора языка будут:

- производительность языка;
- встроенные в язык средства безопасности;
- популярность языка для данной задачи;
- различные другие преимущества и недостатки.

На оценку возьмём следующие языки:

- Python;
- C/C++;
- Java;
- Kotlin;
- JavaScript.

2.1.1 Язык программирования «Python»

Python – высокоуровневый интерпретируемый язык общего назначения. Самый популярный язык программирования в данный момент [3], известен за свою простоту и читаемость, является полностью объектно-ориентированным в том плане, что всё в языке является объектами. Также является

мультипарадигменным языком программирования, за счёт динамической типизации декораторов и различных фреймворков.

Перейдём к нашим критериям.

Производительность - являясь компилируемым языком с динамической типизацией Python менее производителен по сравнению с компилируемыми языками, такими как Java и C/C++. Поэтому он не подходит для задач, требующих высокой производительности, таких как мобильные приложения.

Безопасность - динамическая типизация и автоматическое управление памятью обеспечивают безопасность доступа к памяти.

Популярность - Python не популярен в разработке Android-приложений так как мало фреймворков и статей, но широко используется во различных других областях, таких как веб-разработка, научные вычисления и автоматизация.

Преимущества - прост в изучении и использовании, стандартная библиотека имеет множество модулей для широкого круга задач, большое сообщество разработчиков.

Недостатки - описаны выше.

2.1.2 Язык программирования «C/C++»

C/C++ – это компилируемый язык программирования, широко используемый для системного программирования, разработки игр и приложений, требующих высокой производительности. Так же данный язык является вторым по популярности в мире [3], обрёл свою известность за счёт того, что на момент появления был передовым языком программирования с большим функционалом и удерживает популярность до сих пор.

Производительность - очень высокая, за счёт сочетания возможностей высокоуровневых и низкоуровневых языков, что делает его идеальным для ресурсоемких приложений, таких как 3D-игры и операционные системы.

Безопасность - отсутствие автоматического управления памятью, наличие небезопасных указателей и каламбуров типизации приводит к ошибкам и уязвимостям.

Популярность - в разработке под Android используются для специализированных задач, в основном это игры и низкоуровневые компоненты.

Преимущества - ручной контроль над памятью может оказаться полезным при умении им пользоваться.

Недостатки - сложность разработки, отсутствие сборщика мусора.

2.1.3 Язык программирования «Java»

Java – это компилируемый в байт код для JVM, объектно-ориентированный язык программирования, хоть он и является официальным языком для разработки Android-приложений, но Google не рекомендует его использовать.

Производительность - высокая, благодаря использованию виртуальной машины (JVM) и компиляции байт-кода.

Безопасность - встроенные механизмы безопасности, такие как автоматическое управление памятью, строгая типизация и обработка исключений.

Популярность - широко используется для разработки Android-приложений, имеет большое сообщество и богатую экосистему библиотек и фреймворков, документация от Google написана также и для Java.

Преимущества - широкая поддержка, стабильность, большое количество ресурсов и сообществ.

Недостатки - избыточный код.

2.1.4 Язык программирования «Kotlin»

Kotlin – это статически типизированный язык программирования, являющийся оберткой над Java и компилируемый в байт-код для JVM, который является официальным языком для разработки Android-приложений и рекомендуется Google. Данный язык был разработан международной компанией

JetBrains и назван в честь российского острова Котлин. Исходя из выше сказанного, Kotlin является хорошим выбором и идеально подойдет для разработки мобильного приложения [4].

Производительность - сопоставима с Java, так как компилируется в байт-код и работает на JVM.

Безопасность - улучшенные средства безопасности, включая нулевую безопасность (null-safety), неизменяемые коллекции и корутины для асинхронного программирования.

Популярность - быстро набирает популярность среди разработчиков Android-приложений, поддерживается Google.

Преимущества - современный синтаксис, совместимость с Java, улучшенная безопасность, компактный и выразительный код.

Недостатки - меньшее количество библиотек по сравнению с Java, более молодой язык, поэтому большинство инструментов и ресурсов встречаются менее зрелые.

2.1.5 Язык программирования «JavaScript»

JavaScript – это мультипарадигменный интерпретируемый динамически типизированный язык программирования, широко используемый для веб-разработки и кроссплатформенной мобильной разработки с помощью различных фреймворков, таких как React Native. В настоящий момент имеет 6 место среди всех языков по популярности [3].

Производительность - меньше по сравнению с компилируемыми языками, такими как Java и C++, особенно для нативных приложений.

Безопасность - динамическая типизация, автоматическое управление памятью, строгие ограничения делают его относительно безопасным, но не стоит забывать о возможностях производить различные операции над разными типами данных, что может приводить к не запланированным результатам.

Популярность - популярен для кроссплатформенной разработки мобильных приложений, но не является нативным языком для Android.

Недостатки - сложность при доступе к нативным функциям платформы, зависимости от фреймворков.

2.1.6 Вывод

Проанализировав варианты, было принято решение выбрать Kotlin в качестве основного языка программирования для разработки аудиоплеера, данный язык множество преимуществ таких как:

- поддержка и рекомендация от Google как разработчика платформы Android;
- компиляция в байт-код JVM, и работа на данной виртуальной машине даёт высокую производительность и различные встроенные защиты;
- легко читаемый и не избыточный код.

2.2 Выбор среды разработки

Основной IDE для разработки под операционную систему Android является Android Studio созданная Google совместно с JetBrains. Данная IDE имеет множество встроенных полезных функций, таких как:

- эмулятор телефона;
- доступ к файлам устройства;
- возможность подключить физическое устройство и проверять на нём;
- встроенный Gemini;
- просмотр логов в реальном времени;
- профайлер.

Так как за основу IDE была взята IntelliJ IDEA то она имеет тот же интерфейс и те же преимущества, что и IDE от JetBrains, а именно:

- встроенный контроль версий;
- возможность установки плагинов;
- удобное дерево проекта;

- встроенный сборщик проекта;
- хорошая система подсказок кода и многое другое.

Существует множество других IDE для разработки, но они либо специализированные, например, Unity и UnrealEngine, или не имеют столь широкого встроенного функционала как Android Studio.

2.3 Выбор базы данных

Для разработки приложений под ОС Android существует несколько баз данных для хранения и управления данными приложения. Разберём некоторые из этих баз данных.

2.3.1 Sqlite

Это встроенная в систему база данных. Из преимуществ можно выделить малый размер, отсутствие необходимости настройки сервера и простота использования. Недостатки: ограниченные возможности масштабирования, отсутствие встроенной поддержки синхронизации с удаленными базами данных.

2.3.2 Room

ORM (Object-Relational Mapping) библиотека, предоставляемая Google как часть Android Jetpack. Плюсам данной библиотеки является поддержка RxJava и LiveData, автоматическая обработка запросов и безопасная работа с SQLite. Минусы — это дополнительно изучение концепций ORM.

2.3.3 Firebase Firestore

Новая облачная база данных от Google с поддержкой документов и коллекций. Преимуществами данной базы являются: высокая гибкость, поддержка сложных запросов, оффлайн-синхронизация. Минусами же служат: зависимость от интернет-соединения, потенциальные затраты при масштабировании.

Рассмотрев вышеперечисленные базы данных, было решено использовать Room так как оффлайн синхронизация в нашем приложении не нужна, а

безопасность работы с SQLite, и поддержка LiveData важные функции для приложения.

					ПГУ 1.090301.03.001 ПЗ	Лист
						26
Изм.	Лист	№ докум.	Подпись	Дата		

3 Архитектура приложения

3.1 Архитектурный паттерн «MVVM»

Для написания современного приложения, которое будет легко тестировать и расширять необходимо делить программу на логические части. Именно про данный принцип проектирования приложений написал Джон Госсман в своей статье [5]. Он разделяет приложение на три основных блока:

- Модель (Model);
- Представление (View);
- Модель Представления (ViewModel).

В блоке модель находятся различные модели данных и логика работы с этими данными, запись и чтение базы данных, запросы в Интернет и так далее.

Предствление хранит в себе графический интерфейс нашего приложения. Также оно является подписчиком на события изменения состояний свойств в Модель Предствление. Если пользователь приложения начнёт взаимодействовать с графическим интфейсом, например, нажмёт на кнопку, то Предствление вызовет команду из Модель Представление.

Модель Представление содержит в себе бизнес-логику, а с одной стороны, абстракция Представления, а с другой — обёртка данных из Модели, подлежащих связыванию.

Разделение кода программы на такие логические блоки позволяет добиться независимости графического интерфейса от данных и на оборот, что помогает разработчика в тестировании, поддержании продукта, улучшает безопасность данных и позволяет довольно легко заменять части приложения без влияния на остальную программу.

Также не стоит забывать, что компания Google которая является разработчиком операционной системы Android, рекомендует использовать именно архитектуру MVVM для разработки приложений[6]. Исходя из всего

вышеперечисленного было принято решение использовать архитектурный паттерн MVVM.

4 Реализация приложения

Перед тем как начать разрабатывать наше приложение необходимо разобраться в архитектуре и функционале, который мы хотим получить. Начнём изучение данных тем с описания функционала, а после рассмотрим структуру приложения.

4.1 Описание функционала

Данная программа должна иметь следующие возможности:

- создание плейлиста;
- удаление плейлиста;
- изменение громкости воспроизводимой музыки;
- остановка/продолжение воспроизводимой музыки;
- переход к предыдущему и следующему треку;
- перемещение по воспроизводимой музыке.

Данные варианты использования представлены на диаграмме на рисунке 9.

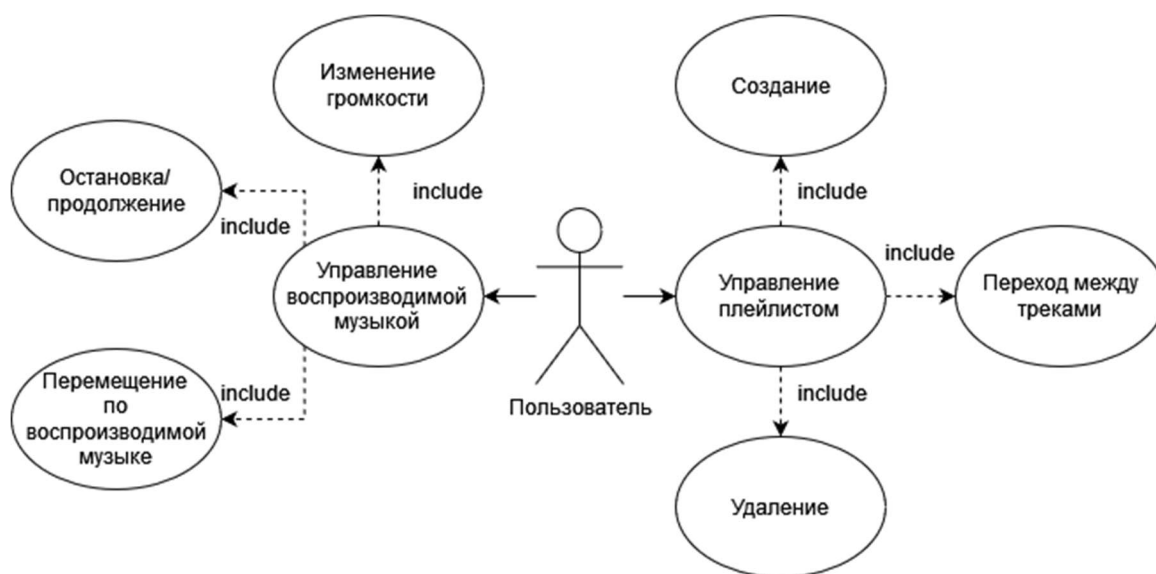


Рисунок 9 – UML–диаграмма вариантов использования для пользователя

Чтобы более точно проанализировать взаимодействие пользователя с приложением составим диаграммы деятельности, представленные на рисунках 10 - 12.

Пользователь начинает взаимодействие с приложением через его запуск, после чего приложение проверяет был ли запущен плейлист раньше и если да, то восстанавливает трек и плейлист который проигрывался раньше (рисунок 10).

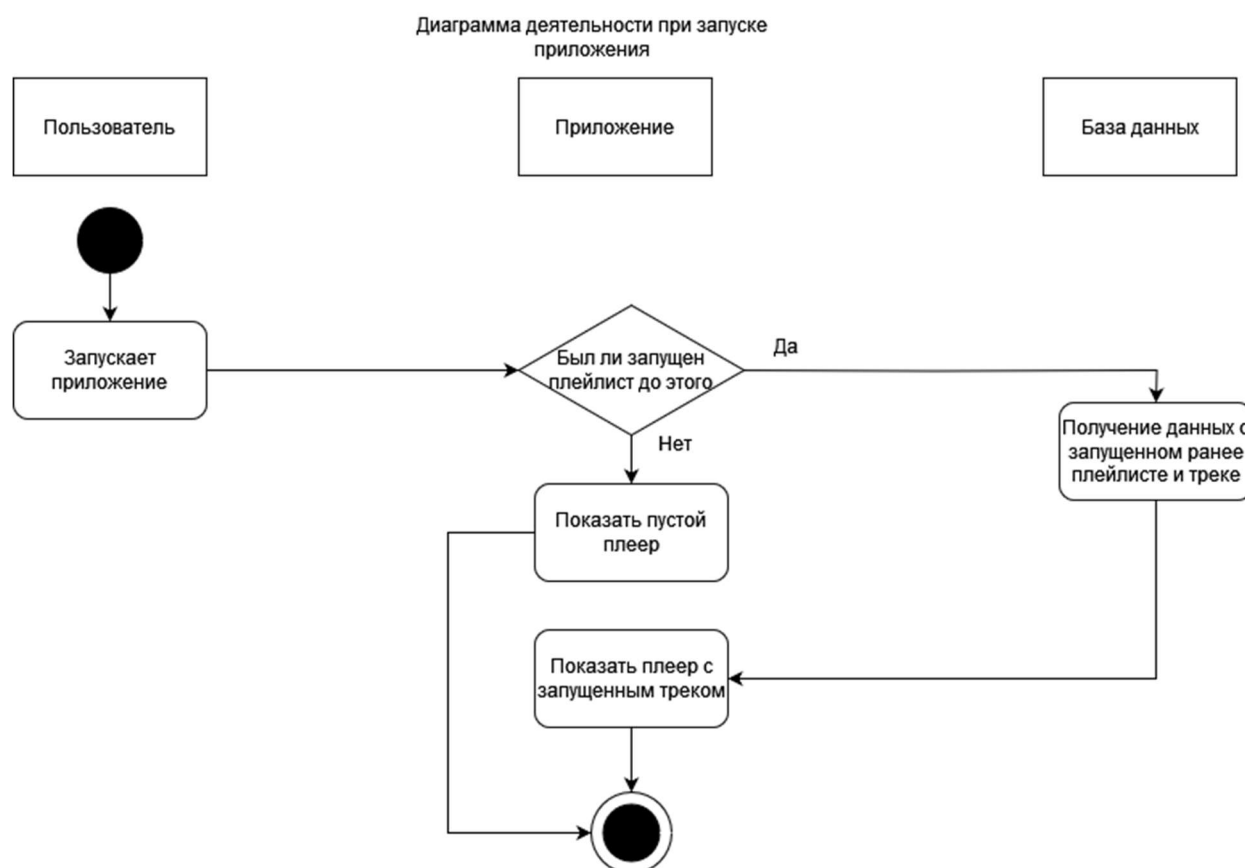


Рисунок 10 – UML–диаграмма деятельности при запуске приложения

После пользователь может либо взаимодействовать с плейлистами (Рисунки 11, 12) или с треком который воспроизводится в настоящий момент.

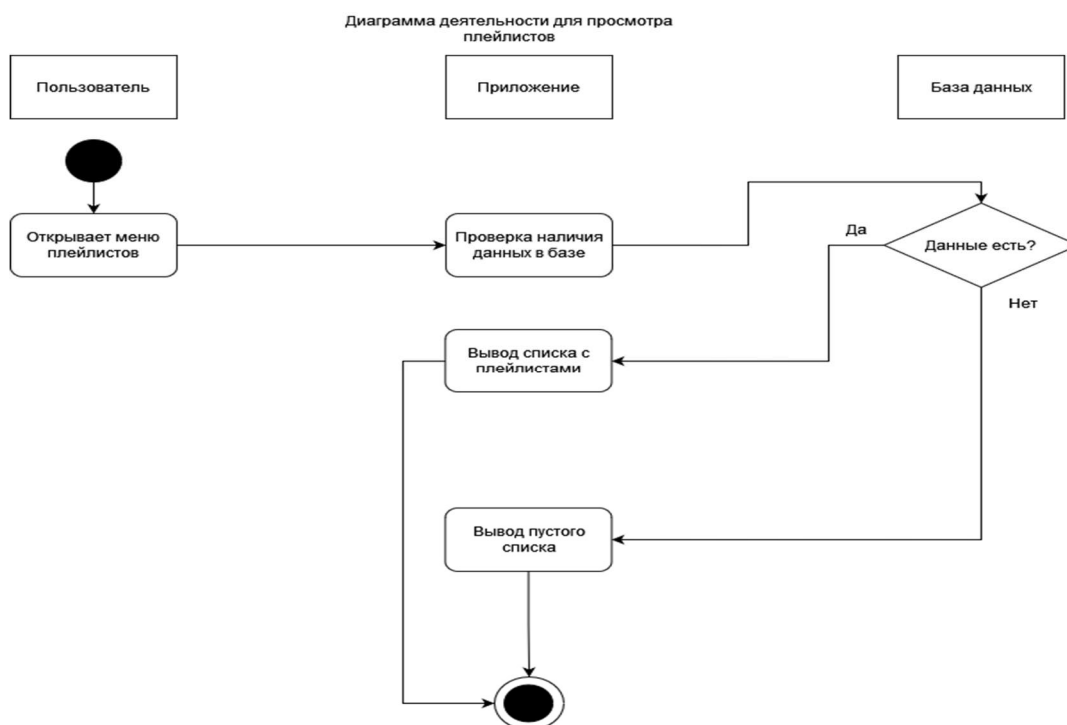


Рисунок 11 – UML–диаграмма деятельности для просмотра плейлистов

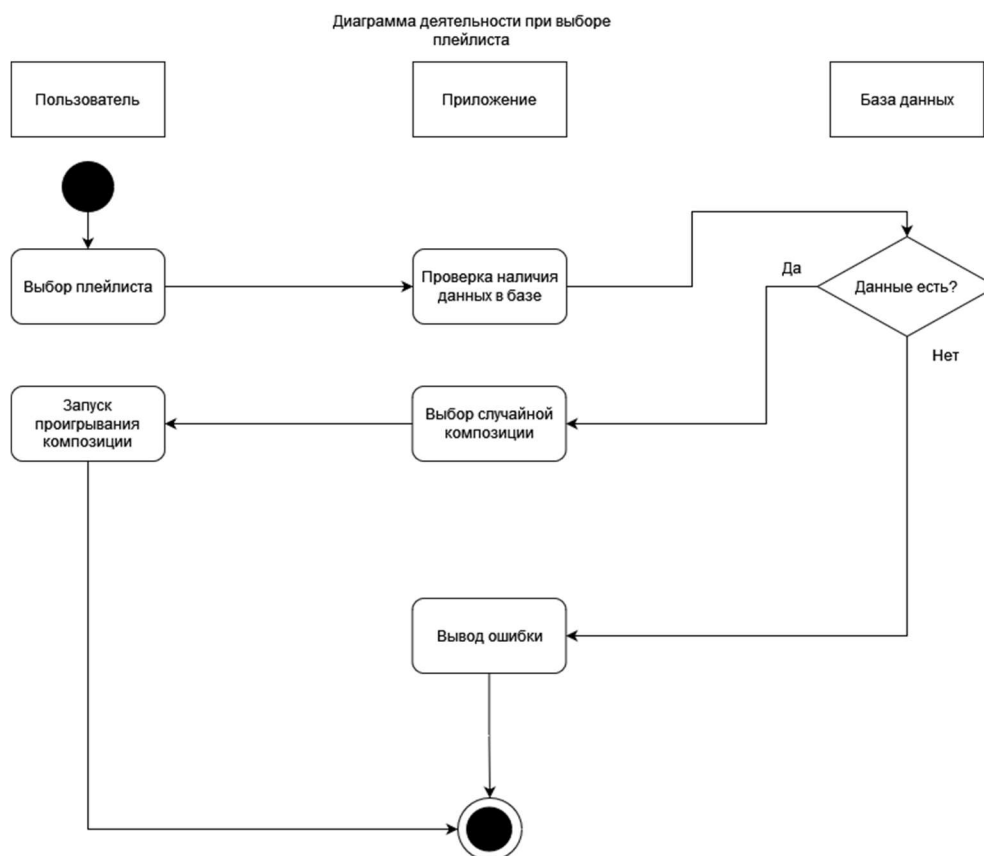


Рисунок 12 – UML–диаграмма деятельности при выборе плейлиста

4.2 Описание работы приложения

Для того чтобы начать пользоваться ПО требуется установить и запустить его на подходящей версии Android, далее приложение проверит был ли запущен какой-либо плейлист ранее, если да, то запросит у базы данных список плейлистов и воспроизводимую в последний раз композицию для восстановления предыдущей сессии. После пользователь может начать пользоваться приложением, а именно выбирать плейлисты и управлять воспроизводимой музыкой.

Все действия, которые совершает пользователь, приложение и база данных показаны на диаграмме последовательности на рисунке 13. Физическое расположение компонентов системы изображено на рисунке 14.

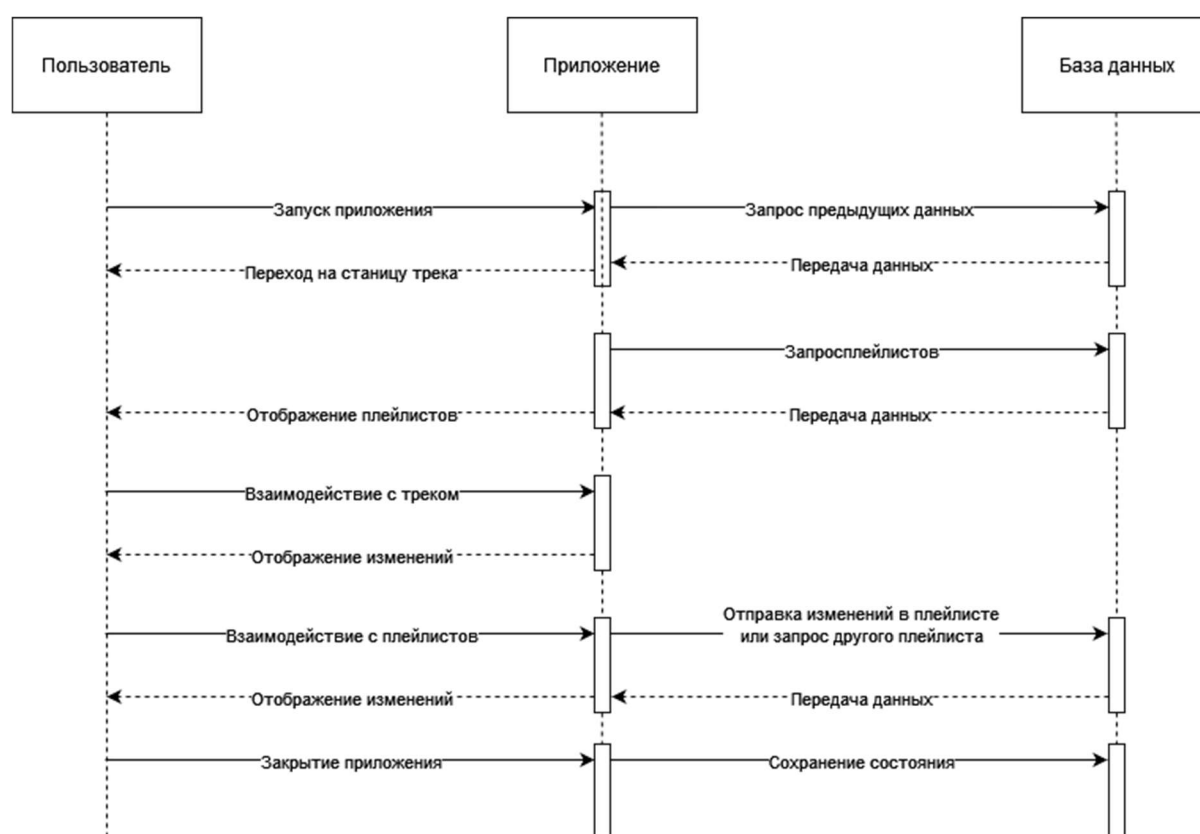


Рисунок 13 – UML–диаграмма последовательности приложения

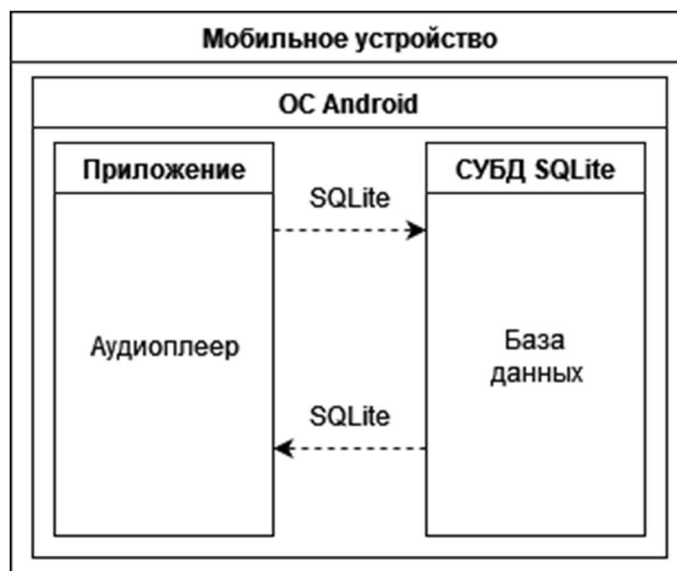


Рисунок 14 – UML–диаграмма развертывания

4.3 Основная реализация приложения

Раз была выбрана архитектура «MVVM», то нам необходимо разбить программу на три основных блока, а именно:

- Model;
- View;
- ViewModel.

В блоке Model находятся все данные, логика работы с базой данных для доступа к этим данным, а также работа с сетью Интернет и так далее. Данный слой включает в себя Entities, DAO, Converters и Database.

Блок View отвечает за графический интерфейс приложения, который взаимодействует с блоком ViewModel используя подписки на изменения и вызов методов. В данном слое находятся Activity и Adapter.

Последний блок содержит всю бизнес-логику, а также является прослойкой между Model и View, благодаря чему мы можем легко менять как View, так и Model.

Далее рассмотрим каждый блок на примере.

4.3.1 Блок Model

В данном блоке имеется большое количество файлов, а именно: IntListConverter, UriConverter, PlaylistEntity, AppDatabase, AudioFileDAO, PlayListDao, AudioFile, PlayList и PlayListWithAudioFiles. Возьмём для рассмотрения по одному файлу из каждой группы.

```
class UriConverter {
    @TypeConverter
    fun fromUri(uri: Uri?): String? {
        return uri?.toString()
    }

    @TypeConverter
    fun toUri(uriString: String?): Uri? {
        return uriString?.let { Uri.parse(it) }
    }
}
```

Как мы видим класс UriConverter имеет два метода fromUri и toUri, первый метод конвертирует uri в string, а второй метод конвертирует string в uri. Таким образом мы можем понять, что группа классов Converters необходима для конвертации одних типов данных в другие. Так же можно заметить аннотации @TypeConverter, они необходимы библиотеке Room для автоматического использования данных конвертеров при работе с базой данных.

```
@Entity(tableName = "playlists")
data class PlayListEntity(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val title: String,
    val nowPlaying: Int,
    val queue: List<Long>
)
```

Класс PlayListEntity является классом описания таблицы в базе данных. По аннотации @Entity Room определяет, что это класс описания таблицы, если мы хотим что-бы название таблицы отличалось от названия класса, то мы, как видно на этом примере, используем аргумент tableName у аннотации и прописываем в него название таблицы. Аннотация @PrimaryKey(autoGenerate = true) сообщает библиотеке Room, что поле id будет использоваться в качестве уникального идентификатора для каждой записи в таблице, аргумент autoGenerate =

true указывает, что значение для id будет автоматически генерироваться Room при добавлении новой записи.

```
data class PlayListWithAudioFiles(  
    @Embedded val playListEntity: PlayListEntity,  
    @Relation(  
        parentColumn = "id",  
        entityColumn = "playlistId"  
    )  
    val audioFiles: List<AudioFile>  
)
```

Данный класс хоть и не является Entity, но всё равно крайне важен при работе с базой данных, так как содержит в себе отношение "один ко многим" между сущностью плейлиста (PlayListEntity) и списком аудиофайлов (AudioFile). Класс использует аннотации из библиотеки Room для связывания данных из двух таблиц в базе данных. Аннотация @Embedded указывает, что объект playListEntity будет встроен в класс PlayListWithAudioFiles как составное поле. Аннотация @Relation указывает какие столбцы базы данных использовать для определения отношения. Так аргумент parentColumn = "id" указывает что мы используем столбец id в родительской таблице для связи, а аргумент entityColumn = "playlistId" указывает из какого столбца дочерней таблицы брать внешний ключ для создания отношений.

```
@Dao  
interface PlayListDao {  
    @Insert(onConflict = OnConflictStrategy.REPLACE)  
    suspend fun insert(playList: PlayListEntity): Long  
  
    @Delete  
    suspend fun delete(playList: PlayListEntity)  
    @Transaction  
    @Query("SELECT * FROM playlists WHERE id = :id")  
    suspend fun getPlayList(id: Long): PlayListEntity?  
    @Transaction  
    @Query("SELECT * FROM playlists")  
    suspend fun getAllPlayLists(): List<PlayListEntity>  
  
    @Transaction  
    @Query("SELECT * FROM playlists WHERE id = :id")  
    suspend fun getPlayListWithAudioFiles(id: Long):  
    PlayListWithAudioFiles?
```

```

        @Query("UPDATE playlists SET nowPlaying = :nowPlaying WHERE
id = :id")
        suspend fun updateNowPlaying(id: Long, nowPlaying: Int)

        @Query("UPDATE playlists SET queue = :queue WHERE id = :id")
        suspend fun updateQueue(id: Long, queue: List<Long>)
    }

```

DAO файлы являются интерфейсами так как Room сгенерирует их самостоятельно основываясь на аннотациях и типах которые должен принимать и возвращать метод. Аннотация `@Dao` помечает данный интерфейс как Dao для Room, аннотации `@Delete` и `@Insert` указывают что данные методы должны удалять и вставлять данные в таблицу, а `@Transaction` указывает на то, что данный метод будет обернут в транзакцию. Аннотация `@Query` позволяет нам прописывать запросы к базе данных самостоятельно.

```

@Database(entities = [AudioFile::class, PlayListEntity::class],
version = 1)
@TypeConverters(IntListConverter::class, UriConverter::class)
abstract class AppDatabase : RoomDatabase() {
    abstract fun audioFileDao(): AudioFileDao
    abstract fun playListDao(): PlayListDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "app_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}

```

Данный класс необходим для создания базы данных с использованием библиотеки Room. Аннотация `@Database` указывает Room, что это класс базы данных и в аргументах принимает `entities = [AudioFile::class, PlayListEntity::class]`, который показывает какие таблицы должны быть в базе,

version = 1, указывает версию базы для миграций. Аннотация @TypeConverters(IntListConverter::class, UriConverter::class) служит для указания классов конвертеров для конвертации типов с которыми SQLite не может работать на прямую. abstract fun audioFileDao(): AudioFileDao и abstract fun playlistDao(): PlaylistDao методы для получения DAO с которыми будет работать база данных. Companion object нужен для получения нашей базы данных в коде.

4.3.2 Блок View

В данном блоке находятся файлы разметки экранов и элементов, а также классы Adapter и Activity. В нашем случае имеется: PlaylistAdapter, AudioControlActivity, audio_control.xml и playlist_item.xml. Рассмотрим эти файлы подробнее.

Разберём класс PlaylistAdapter подробно, что бы в класс AudioControlActivity разобрать только методы этапов жизненного цикла.

```
class PlaylistAdapter(
    private val viewModel: AudioControlViewModel
) : RecyclerView.Adapter<PlaylistAdapter.PlaylistViewHolder>() {

    private var playlists: List<PlayListEntity> = emptyList()

    fun setPlaylists(playlists: List<PlayListEntity>) {
        this.playlists = playlists
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
    Int): PlaylistViewHolder {
        val binding =
        PlaylistItemBinding.inflate(LayoutInflater.from(parent.context),
        parent, false)
        return PlaylistViewHolder(binding)
    }

    override fun onBindViewHolder(holder: PlaylistViewHolder,
    position: Int) {
        val playlist = playlists[position]
        holder.bind(playlist)
    }

    override fun getItemCount(): Int {
```

```

        return playlists.size
    }

    inner class PlaylistViewHolder(private val binding:
PlaylistItemBinding) : RecyclerView.ViewHolder(binding.root) {

        fun bind(playlist: PlayListEntity) {
            binding.playlistTitle.text = playlist.title

            binding.root.setOnClickListener {
                viewModel.onPlaylistClick(playlist)
            }

            binding.deleteButton.setOnClickListener {
                showDeleteConfirmationDialog(binding.root.context, playlist)
            }

            private fun showDeleteConfirmationDialog(context:
Context, playlist: PlayListEntity) {
                AlertDialog.Builder(context)
                    .setTitle(R.string.delete_playlist)

                .setMessage(R.string.delete_playlist_confirmation)
                    .setPositiveButton(R.string.delete) { _, _ ->
                        viewModel.deletePlayList(playlist)
                    }
                    .setNegativeButton(R.string.cancel, null)
                    .show()
            }
        }
    }
}

```

Данный адаптер связывает данные из списка плейлистов с пользовательским интерфейсом. Метод `setPlayLists` обновляет список плейлистов и говорит адаптеру о необходимости обновить графический интерфейс. Когда адаптер узнаёт о необходимости обновить интерфейс, то вызывает метод `onBindViewHolder` который был переопределён нами. Данный метод вызывает метод `bind` подкласса, который, в свою очередь, используя механизм `viewBinding` вставляет текст в текстовое поле и вешает слушателей на кнопки. Для работы `viewBinding` нужно включить его в файле `build.gradle.kts` на уровне модуля следующим образом:

```
buildFeatures {
    viewBinding = true
}
```

Из класса `AudioControlView` возьмём для примера только один метод жизненного цикла, а именно `onCreate`.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = DataBindingUtil.setContentView(this,
R.layout.audio_control)
    binding.viewModel = viewModel
    binding.lifecycleOwner = this
    binding.menuButton.setOnClickListener {
        binding.drawerLayout.open()
    }

    binding.addPlaylistButton.setOnClickListener {
        selectFolderLauncher.launch(null)
    }

    val adapter = PlaylistAdapter(viewModel)
    binding.playlistRecyclerView.layoutManager =
LinearLayoutManager(this)
    binding.playlistRecyclerView.adapter = adapter

    viewModel.playlists.observe(this, Observer { playlists -
>
        adapter.setPlaylists(playlists)
    })

    viewModel.trackImage.observe(this, Observer { image->
        if(image == null){

binding.audioImage.setImageResource(R.drawable.ic_audio_image_pl
aceholder)
        }
        else{
            Glide.with(this)
                .load(image)
                .into(binding.audioImage)
        }
    })
}
```

Методы жизненного цикла зачастую начинаются одинаково, с вызова метода суперкласса для выполнения базовой логики. Далее мы видим установку содержимого активности с привязкой данных используя механизм `DataBinding`. Заканчиваем первичную настройку установкой `viewModel` и владельца

жизненного цикла. Для работы кнопок есть два механизма установки действий по нажатию, в данный момент мы рассмотрим только один:

```
binding.buttonId.setOnClickListener {func()}.
```

Метод `setOnClickListener` принимает на вход или метод, или `lambda`-функцию, которую будет вызывать при нажатии на данную кнопку. Далее мы можем видеть создание и установку адаптера для списка плейлистов. Последним идут установки наблюдателей за `LiveData` в `viewModel`:

```
viewModel.variableName.observe(this, Observer { it → func(it)})
```

Как мы можем заметить установка наблюдателя за переменной очень похоже на установку наблюдателя за нажатием на кнопку, только используется метод `observe` и как с нажатием на кнопку это не единственный способ установить наблюдателя.

Теперь рассмотрим небольшую часть файла `audio_control.xml`:

```
<data>
    <variable
        name="viewModel"

type="com.example.shuffleit.viewModels.AudioControlViewModel" />
</data>

<ImageButton
    android:id="@+id/volumeDownButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_below="@id/playPauseButton"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:background="@drawable/ic_minus"
    android:onClick="@{() ->
viewModel.volumeDown()}" />

<TextView
    android:id="@+id/timeNow"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="10dp"
    android:layout_marginBottom="32dp"
    android:text="@{viewModel.currentTimeString}" />
```

Здесь мы можем увидеть еще по одному способу установки наблюдателей о которых говорилось ранее, а именно сначала нам необходимо создать переменную в .xml файле используя теги data и variable и уже после мы можем в поле android:onClick тега button и android:text тега textview установить либо какой текст писать, либо какой метод вызывать при нажатии на кнопку. Данный способ требует механизма dataBinding, который можно включить в файле build.gradle.kts на уровне модуля следующим образом:

```
buildFeatures {
    dataBinding = true
}
```

4.3.3 Блок ViewModel

Это последний, но не по значению, блок который будем рассматривать. ViewModel содержит в себе всю бизнес-логику приложения, а также связывает данные из Model с графическим отображением в View. В проекте аудиоплеера содержится файл AudioControlViewModel, части которого мы рассмотрим далее:

```
private val sharedPreferences = application.getSharedPreferences
("LastPlayedPlayList", Context.MODE_PRIVATE)
private val database =
AppDatabase.getDatabase(application.applicationContext)
private val playListDao = database.playListDao()
private val audioFileDao = database.audioFileDao()
private var audioManager: AudioManager =
application.getSystemService (Context.AUDIO_SERVICE) as
AudioManager
private var _playlists = MutableLiveData(getAllPlayLists())
val playlists: LiveData<ArrayList<PlayListEntity>> get() =
_playlists
```

Здесь мы можем видеть получение доступа к shared preferences приложения, доступа к базе данных, а также создание LiveData переменных.

```
_trackLen.value = 0
_currentTime.value = 0
_playlists.value?.addAll(getAllPlayLists())
val playlistId = sharedPreferences.getLong("PlaylistId", -1)
val nowPlaying = sharedPreferences.getInt("nowPlaying", -1)
val currentTime = sharedPreferences.getInt("currentTime", 0)
if (playlistId.toInt() != -1) {
    viewModelScope.launch {
```

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		41

```

        val retrievedPlaylist = getPlayList(playlistId)
        retrievedPlaylist?.let {
            playlist = it
            if (nowPlaying != -1) {
                playlist.setNowPlaying(nowPlaying)
                initMediaPlayer(playlist.getAudio().path)
                if (currentTime != 0) {
                    mediaPlayer?.seekTo(currentTime)
                    updateCurrentTime()
                }
            }
        }
    }
}

```

Этот код запускается при создании активности и проверяет shared preferences и базу данных на наличие данных о предыдущей сессии.

```

fun onStop() {
    val editor = sharedPreferences.edit()
    if (::playlist.isInitialized) {
        editor.putLong("PlaylistId", playlist.id)
        editor.apply()
        editor.putInt("nowPlaying",
            playlist.getNowPlaying())
        editor.apply()
        editor.putInt("currentTime",
            mediaPlayer!!.currentPosition)
        editor.apply()
        updateQueue(playlist.id, playlist.getQueue())
    }
    mediaPlayer?.release()
    handler.removeCallbacks(updateCurrentTimeRunnable)
}

```

Данный метод вызывается при прекращении работы приложения и сохраняет текущую сессию в shared preferences и базу данных для последующего восстановления.

```

fun addPlaylist(uri: Uri, name: String) {
    viewModelScope.launch {
        val newPlaylistEntity = PlayListEntity(
            title = name,
            nowPlaying = 0,
            queue = emptyList()
        )

        val playlistId =
            playListDao.insert(newPlaylistEntity)
    }
}

```

```

        val audioFiles =
Utils.getAudioFilesFromUri(application.applicationContext, uri,
playlistId)
        val audioFileIds:ArrayList<Long> = arrayListOf()
        audioFiles.forEach { audioFile ->
            audioFileIds.add(audioFileDao.insert(audioFile))
        }
        val queue = shuffle(audioFileIds)
        updateQueue(playlistId, queue)

        _playlists.value?.add(playListDao.getPlayList(playlistId)!!)
        _playlists.value = _playlists.value
    }
}

```

Метод создания плейлиста, сначала создаёт плейлист в базе данных для получения его id, после чего сохраняет все пути треков в базу данных и создаёт очередь треков для последующего воспроизведения, а также обновляет значение LiveData переменной `_playlist` для изменения списка в пользовательском интерфейсе.

4.3.4 Вспомогательные классы

Также в нашем проекте имеются вспомогательные файлы которые содержат константы или методы которые могут быть вызваны в любом месте программы:

```

class Const {
    companion object Requests{
        val REQUEST_CODE_OPEN_DOCUMENT_TREE = 1
    }
}

```

Класс констант содержит в себе различные константы, для того что бы в коде не было «магических чисел»

```

class Utils {
    companion object {
        fun intToTime(time: Int): String {
            var hours = (time/3600).toString()
            var mins = (time/60).toString()
            var secs = (time%60).toString().padStart(2, '0')
            if(hours!="0"){
                mins = mins.padStart(2, '0')
                return "$hours: $mins : $secs"
            }
            return "$mins : $secs"
        }
    }
}

```

```

        fun getAudioFilesFromUri(context: Context, uri: Uri,
playlistId:Long): List<AudioFile> {
            val audioFiles = mutableListOf<AudioFile>()
            val docUri =
DocumentsContract.buildDocumentUriUsingTree(uri,
DocumentsContract.getTreeDocumentId(uri))
            collectAudioFiles(context, docUri, audioFiles,
playlistId)
            return audioFiles
        }

        private fun collectAudioFiles(context: Context, uri: Uri,
audioFiles: MutableList<AudioFile>, playlistId:Long) {
            val childrenUri =
DocumentsContract.buildChildDocumentsUriUsingTree(uri,
DocumentsContract.getDocumentId(uri))
            val cursor: Cursor? =
context.contentResolver.query(childrenUri, arrayOf(
DocumentsContract.Document.COLUMN_DOCUMENT_ID,
DocumentsContract.Document.COLUMN_DISPLAY_NAME,
DocumentsContract.Document.COLUMN_MIME_TYPE,
DocumentsContract.Document.COLUMN_SIZE
), null, null, null)

            cursor?.use {
                val idColumn =
cursor.getColumnIndexOrThrow(DocumentsContract.Document.COLUMN_D
OCUMENT_ID)
                val mimeTypeColumn =
cursor.getColumnIndexOrThrow(DocumentsContract.Document.COLUMN_M
IME_TYPE)

                while (cursor.moveToNext()) {
                    val documentId = cursor.getString(idColumn)
                    val mimeType =
cursor.getString(mimeTypeColumn)

                    val documentUri =
DocumentsContract.buildDocumentUriUsingTree(uri, documentId)

                    if (mimeType ==
DocumentsContract.Document.MIME_TYPE_DIR) {
                        collectAudioFiles(context, documentUri,
audioFiles, playlistId)
                    } else if (mimeType.startsWith("audio/")) {
                        val audioFile = AudioFile(path =
documentUri, playlistId = playlistId)
                        audioFiles.add(audioFile)
                    }
                }
            }
        }

```

```

    }
}
}
}

```

Класс Utils содержит в себе различные вспомогательные методы, например метод перевода времени из числа секунд в строку формата ЧЧ:ММ:СС. Метод getAudioFilesFromUri получает Uri папки из которой надо достать все аудиофайлы и рекурсивно проходит по всему дереву в поисках аудиофайлов. Что бы не запутаться можно посмотреть на диаграмму классов изображённую на рисунке 15.

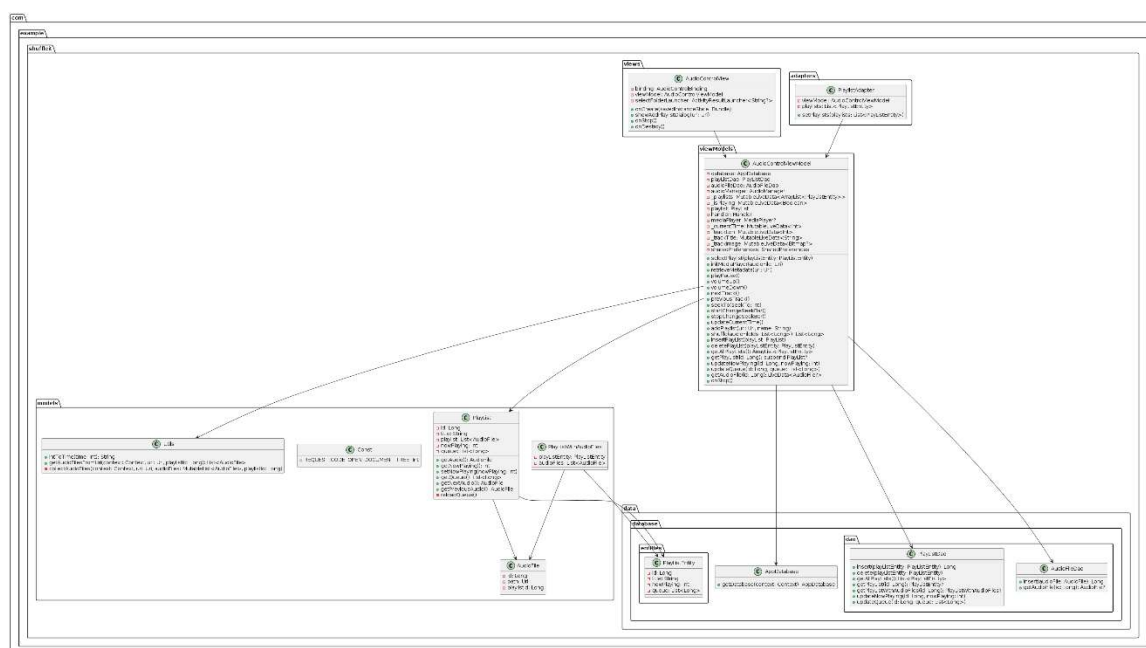


Рисунок 15 – UML–Диаграмма классов

5 Тестирование

После реализации основного функционала приложения необходимо провести тестирование продукта. Этап тестирования позволяет выявить критические проблемы и ошибки в программе, что улучшит опыт пользователя и безопасность приложения, поэтому данный этап является важным этапом при разработке ПО.

В данном пункте мы проведём тестирование основных функций приложения, таких как:

- Создание плейлиста;
- Выбор плейлиста;
- Сохранение и восстановление данных сессии;
- Перемотка музыки;
- Изменение громкости;
- Переход по трекам.

5.1 Создание плейлиста

Создание плейлиста важная часть приложения так как без этого невозможно запустить трек на проигрывание. Для этого запустим приложение, откроем боковое навигационное меню, нажмём на кнопку “Add playlist”, выберем папку, треки в которой будем использовать в плейлисте, даём доступ приложению к папке, вводим название плейлиста и проверяем его работоспособность. На рисунке 16 мы видим запуск приложения, на рисунке 17 можем видеть открытие бокового навигационного меню. После нажатия на кнопку “Add playlist” мы можем видеть открывшееся меню выбора папки на рисунке 18. После выбора папки мы вводим название, рисунок 19 далее на рисунке 20 мы можем видеть, что наш плейлист появился в боковом навигационном меню. При нажатии на плейлист мы видим открывшуюся песню.

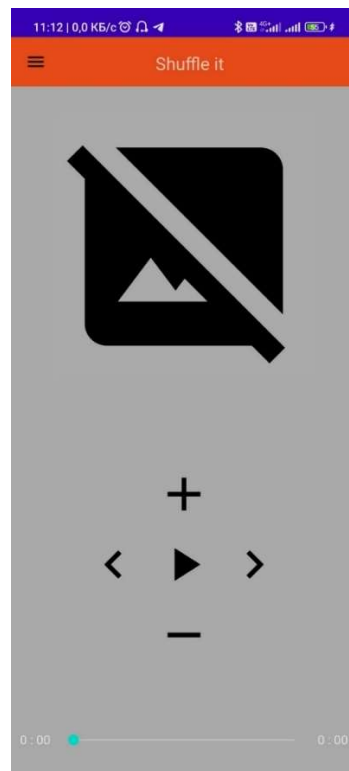


Рисунок 16 – Запустили приложение



Рисунок 17 – Боковое меню

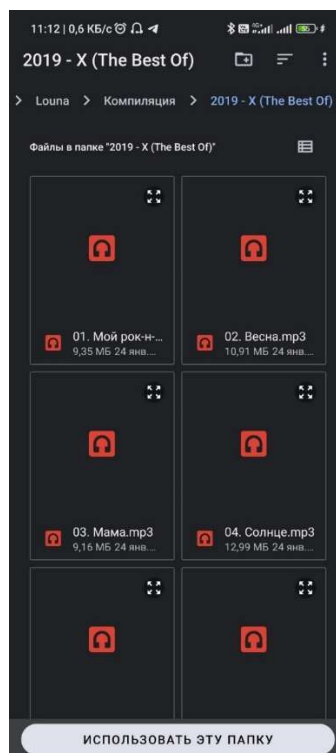


Рисунок 18 – Выбор папки

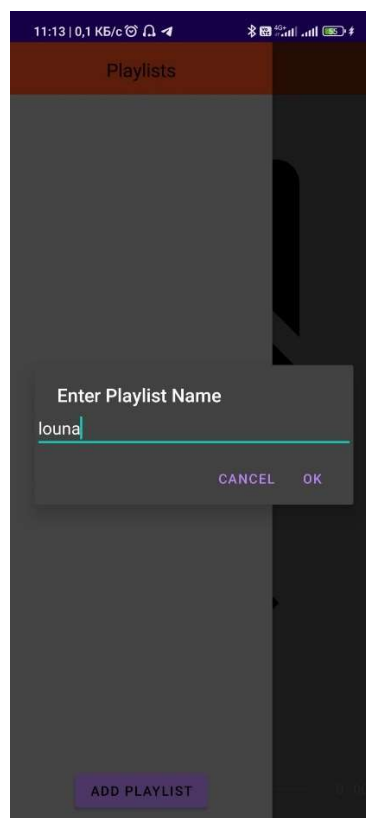


Рисунок 19 –Ввод названия

					ПГУ 1.090301.03.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 20 – Плейлист появился в списке плейлистов

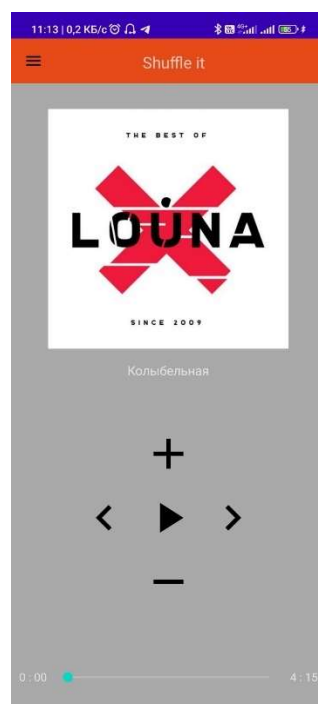


Рисунок 21 – Плеер готов к работе

5.2 Восстановление сессии

Для проверки восстановления сессии нам необходимо закрыть приложение, когда какой-либо плейлист воспроизводился, а после открыть приложение.

На рисунке 22 мы видим состояние приложения до закрытия.



Рисунок 22 – Во время работы закрываем приложение

На рисунке 23 мы видим что при открытии приложения состоянии предыдущей сессии полностью восстановлено.



Рисунок 23 – Открыли приложение

5.3 Управление воспроизводимой музыкой

Управление воспроизводимой музыкой для плеера является крайне важной функцией, которая обязана работать. Необходимо проверить остановка/продолжение трека, смена трека и изменение громкости.

На рисунке 24 и рисунке 25 мы видим что при нажатии на кнопку Play/Pause пригреваемый трек останавливается и продолжает работу. На рисунках 26, 27 мы видим переходы на следующий и предыдущий треки при нажатии на соответствующие кнопки.



Рисунок 24 – трек остановлен

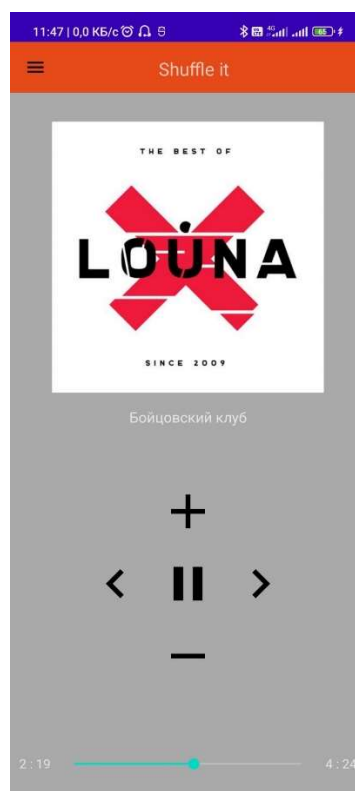


Рисунок 25 – Трек продолжен



Рисунок 26 – Следующий трек

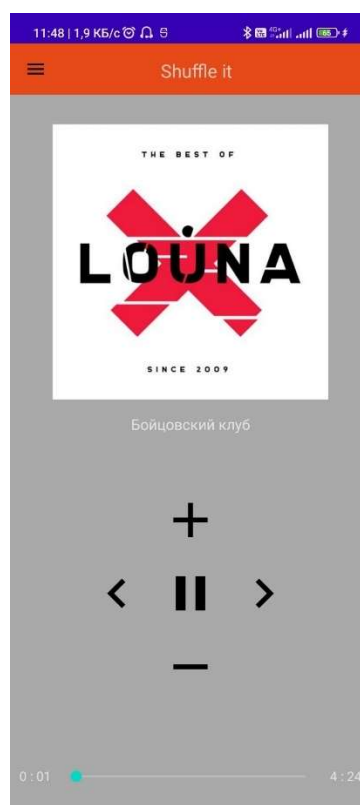


Рисунок 27 – Предыдущий трек

Проверив остановку/продолжение трека, смену трека и изменение громкости мы закончили проверку всех основных функций аудиоплеера.

5.4 Вывод

Завершив проверку приложения, были обнаружены и исправлены несколько проблем. Данный этап позволил нас создать более отказоустойчивое приложение, что улучшит опыт пользователя.

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		54

6 Безопасность жизнедеятельности

Для того чтобы обеспечить безопасность жизнедеятельности программистов и простых людей что проводят за компьютером много времени были разработаны различные нормативы и правила для обеспечения безопасности за персональным компьютером и мобильными устройствами. Пренебрежение данными правилами может влечь за собой негативные последствия как для здоровья пользователя, так и для его имущества.

В этом пункте рассматриваются основные правила и инструкции по работе человека с компьютером и мобильным телефоном.

6.1 Правила и техника безопасности при работе с ПК

При взаимодействии с вычислительной техникой человек может подвергаться различными вредными воздействиями такими как:

- поражение электрическим током;
- электромагнитное излучение;
- опасность пожара;
- повышенный уровень шума;
- механические повреждения.

Для того чтобы уменьшить влияние требуется соблюдать ряд определенных правил.

В первую очередь необходимо пользоваться только исправной техникой, запрещается пользоваться техникой с внешними повреждениями. Важно следить за техническим состоянием персональной электронной вычислительной машины и не оставлять включённой без присмотра.

Также важную роль играет посадка и мебель. Необходимо обзавестись эргономичной офисной мебелью, как креслом, так и компьютерным столом. Правильную посадку можно посмотреть на Рисунке

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

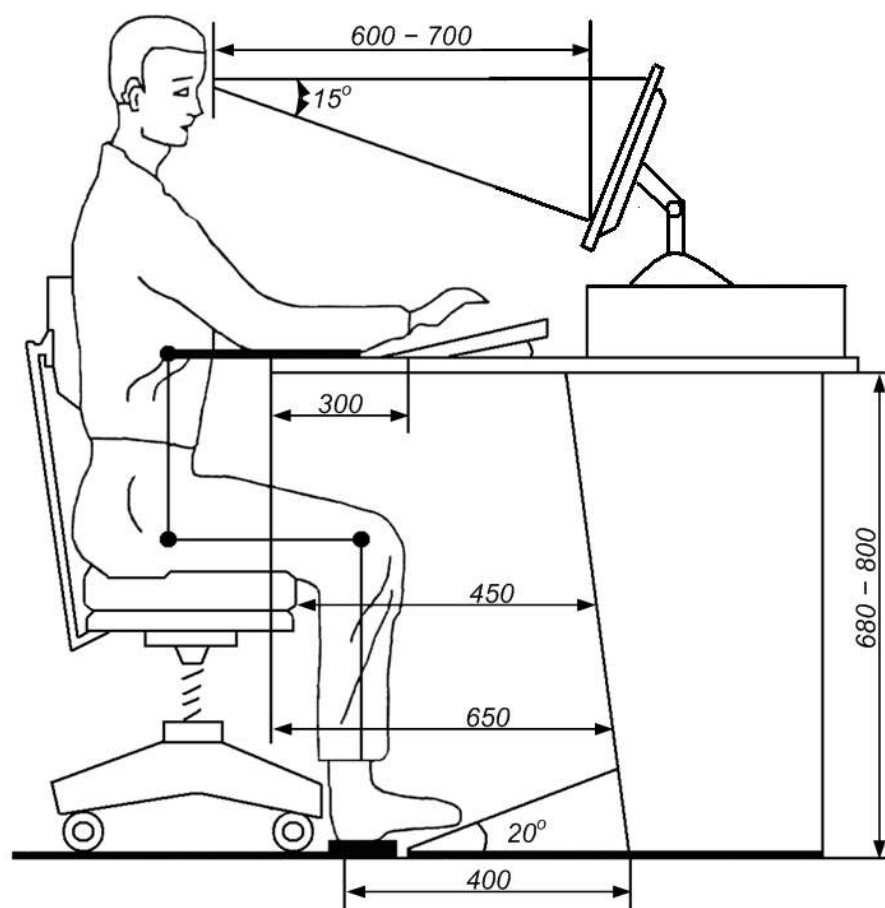


Рисунок 28 Правильная посадка за компьютером

Чтобы снизить как физическую, так и психологическую нагрузку требуется периодически отдыхать от работы. При 8-часовой смене общая продолжительность перерывов должна составлять от 30 до 90 минут. Также для профилактики заболевания глаз и опорно-двигательного аппарата работнику следует проводить короткие упражнения, нацеленные на снятия нагрузок с собственного тела.

6.2 Правила и техника безопасности при работе со смартфоном

Так как основным устройством, где будет работать приложение, является телефон, то следует также проанализировать и учесть все правила безопасности по обращению с ним.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

56

Не следует пытаться вскрывать, разбирать или модифицировать устройство самостоятельно. Также не следует подвергать устройство физическим нагрузкам (ронять, сгибать, бросать, протыкать) или воздействию огня или воды.

Если мобильное или зарядное устройство неисправно использовать его строго запрещено. Также следует пользоваться только оригинальными зарядными устройствами и аксессуарами.

Мобильное устройство постоянно выделяет радиоизлучение, поэтому в целях безопасности не стоит класть телефон близко к жизненно важным органам, а также при разговоре периодически менять положение смартфона относительно головы. Также нужно учитывать, что данных и качественных исследований относительно пагубного влияния радиоизлучения пока что нет [7].

					ПГУ 1.090301.03.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно разработанное мобильное приложение «Shuffleit», полностью соответствующее заданию. Также были выполнены следующие задачи:

- полный анализ темы и аналогов;
- выбор инструментов разработки;
- анализ архитектуры разрабатываемого приложения;
- непосредственная разработка приложения;
- тестирование ПО.

В результате было создано полностью протестированное приложение для мобильных устройств, написанное на языке программирования Kotlin. Также в ходе разработки были получены навыки работы с Room, Glide, databinding и viewBinding. Таким образом результаты показали, что приложение полностью соответствует всем поставленным задачам, имеет удобный пользовательский интерфейс и выполняет свою работу корректно.

В перспективе приложение можно будет улучшить, добавив медиа-уведомление, новый функционал и доработав пользовательский интерфейс для более комфортного использования со стороны пользователей.

Список использованных источников

1. Share of Android vs iOS users worldwide as of June 2023, by generation: [Электронный ресурс]. URL: <https://www.statista.com/forecasts/1406595/android-vs-ios-users-worldwide-by-generation> (дата обращения: 19.05.2024)
2. Gsma Smartphone owners are now the global majority, New GSMA report reveals: [Электронный ресурс]. URL: <https://www.gsma.com/newsroom/press-release/smartphone-owners-are-now-the-global-majority-new-gsma-report-reveals/> (дата обращения: 19.05.2024)
3. TIOBE: [Электронный ресурс]. URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 25.05.2024)
4. Kotlin programming language : [Электронный ресурс]. URL: <https://kotlinlang.org/> (дата обращения: 25.05.2024)
5. Introduction to Model/View/ViewModel pattern for building WPF apps | Microsoft learn: [Электронный ресурс]. URL: <https://learn.microsoft.com/en-us/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps> (дата обращения: 25.05.2024)
6. Android | Больше возможностей: [Электронный ресурс]. URL: <https://developer.android.com/topic/architecture> (дата обращения: 25.05.2024)
7. Меры безопасности при использовании мобильных устройств [Электронный ресурс]. URL: <https://stupinoadm.ru/novosti/detail/mery-bezopasnosti-pri-ispolzovanii-mobilnykh-ustroystv/> (дата обращения: 26.05.2024)

ПРИЛОЖЕНИЕ А

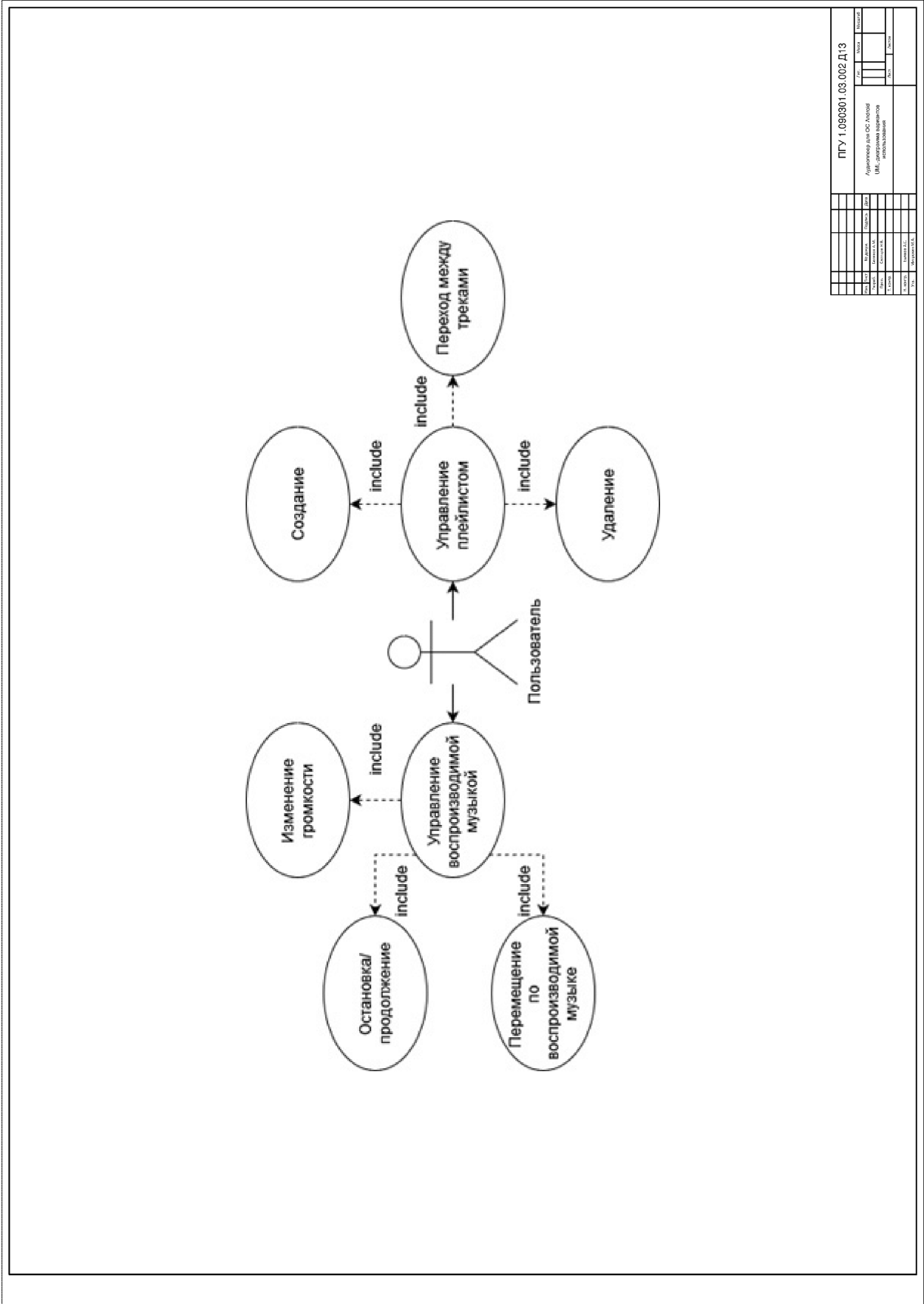
(Обязательное)

Аудиоплеер для Android

UML-диаграммы

					ПГУ 1.090301.03.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

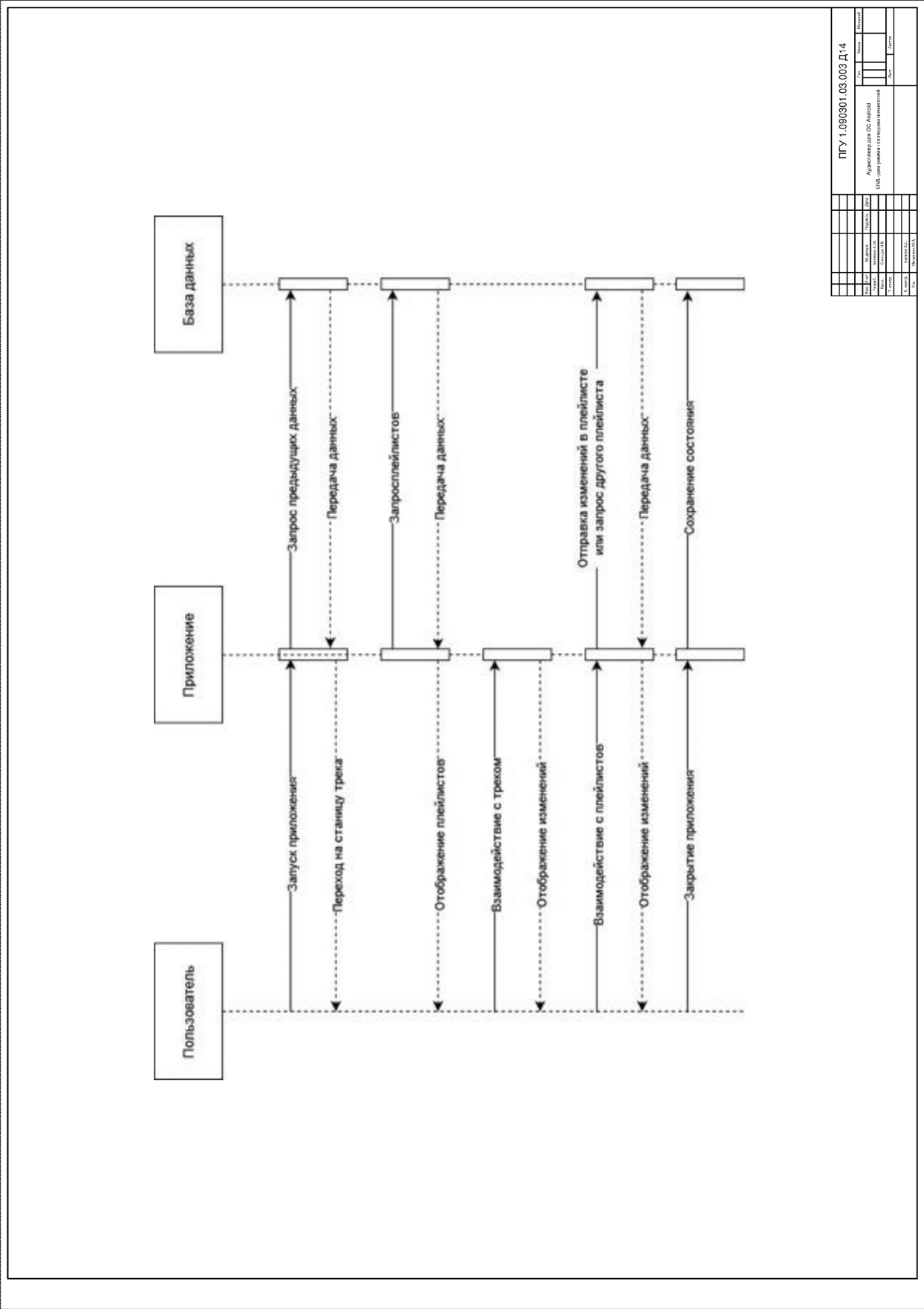
Приложение А.1-Диаграмма прецедентов



ПГУ 1.090301.03.002 Д13									
Универсальный документ ИБД для разработки ИБД									
№ докум.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Изм.	Лист	№ докум.	Подпись	Дата

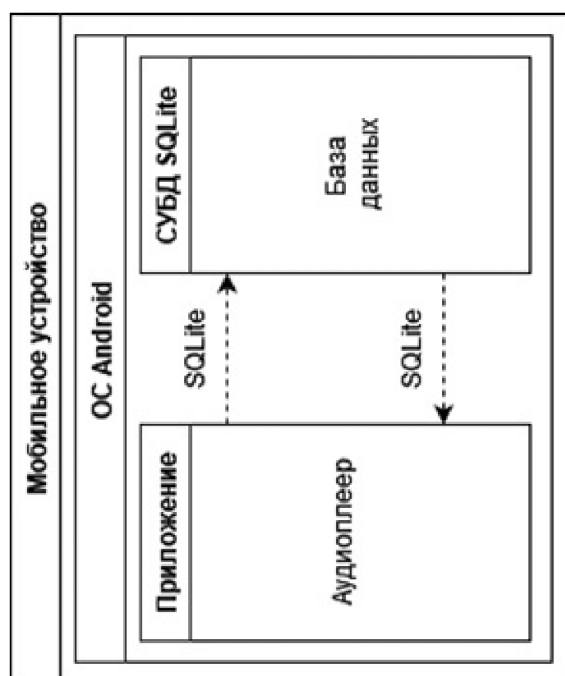
Приложение А.2-Диаграмма последовательностей



ПГУ 1.090301.03.003 Д14									
Адресован для ОС Android									
УМ, для работы (поддержка платформ)									
№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.	№ докум.
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14
1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14	1.090301.03.003	Д14

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3-Диаграмма развертывания

[illegible]

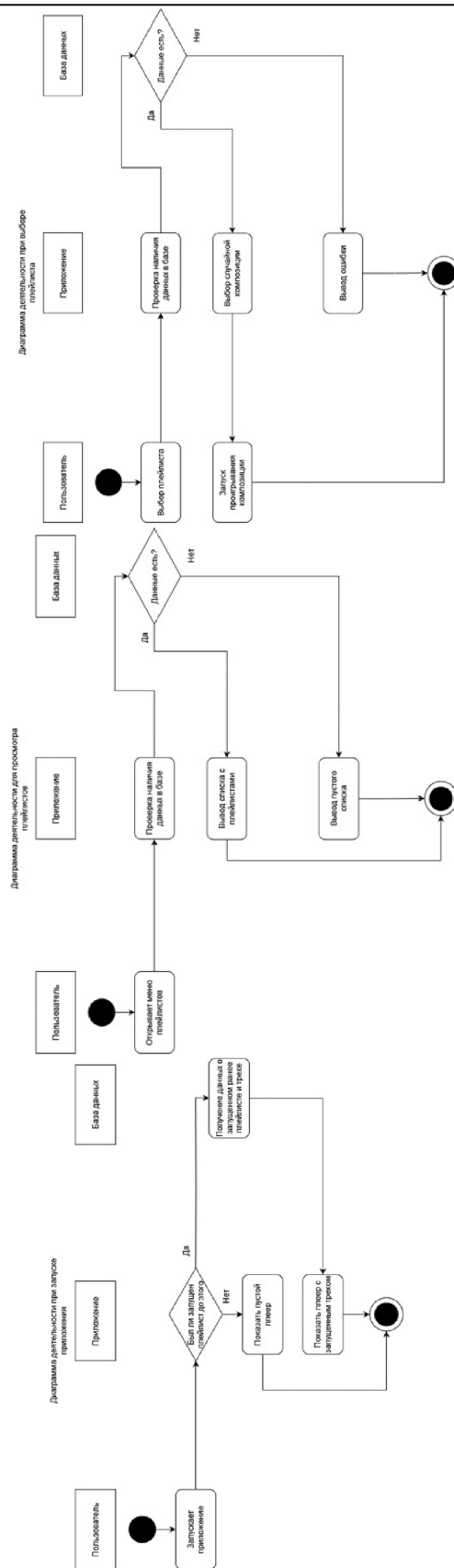
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

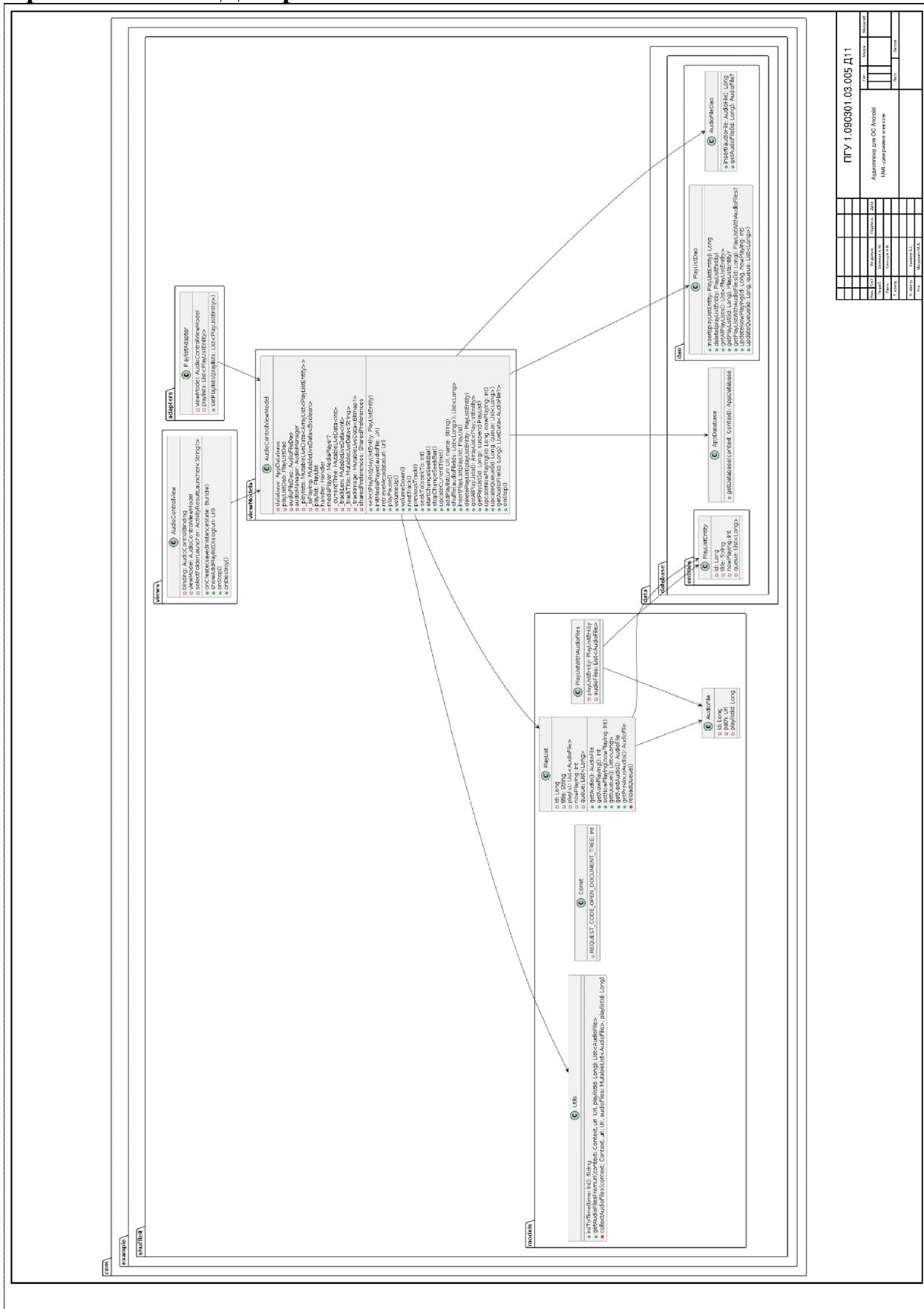
Лист

63

Приложение А.4-Диаграмма деятельности

[illegible]

Приложение А.5-Диаграмма классов



ПРИЛОЖЕНИЕ Б

(Обязательное)

Аудиоплеер для Android

Листинг программы

					ПГУ 1.090301.03.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

```

PlaylistWithAudioFiles.kt
package com.example.shuffleit.models

import androidx.room.Embedded
import androidx.room.Relation
import
com.example.shuffleit.data.database.entities.PlayListEntity

data class PlaylistWithAudioFiles(
    @Embedded val playListEntity: PlayListEntity,
    @Relation(
        parentColumn = "id",
        entityColumn = "playlistId"
    )
    val audioFiles: List<AudioFile>
)
Playlist.kt
package com.example.shuffleit.models

data class Playlist(val id:Long, val title:String, var
playlist:List<AudioFile>, private var nowPlaying:Int, private
var queue:List<Long>) {

    fun getAudio():AudioFile{
        return getAudioFromQueue()
    }
    private fun getAudioFromQueue():AudioFile{
        return playlist.filter { queue[nowPlaying] == it.id
    }.first()
    }

    fun getNowPlaying():Int{
        return nowPlaying
    }

    fun setNowPlaying(nowPlaying: Int){
        this.nowPlaying = nowPlaying
    }

    fun getQueue():List<Long>{
        return queue
    }

    fun getNextAudio():AudioFile{
        if(nowPlaying+1 == playlist.size){
            reloadQueue()
            nowPlaying = 0
            return getAudioFromQueue()
        }
        nowPlaying++
    }
}

```

```

        return getAudioFromQueue()
    }

    fun getPreviousAudio():AudioFile{
        if(nowPlaying == 0){
            reloadQueue()
            nowPlaying = 0
            return getAudioFromQueue()
        }
        nowPlaying--
        return getAudioFromQueue()
    }

    private fun reloadQueue(){
        queue = queue.shuffled()
    }
}
AudioControlVewModel.kt
package com.example.shuffleit.viewModels

import android.app.Application
import android.content.Context
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.media.AudioManager
import android.media.MediaMetadataRetriever
import android.media.MediaPlayer
import android.os.Handler
import android.os.Looper
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.MutableLiveData
import com.example.shuffleit.models.PlayList
import com.example.shuffleit.Utills
import android.widget.SeekBar
import androidx.lifecycle.LiveData
import androidx.lifecycle.map
import android.net.Uri
import android.util.Log
import androidx.lifecycle.viewModelScope
import com.example.shuffleit.data.database.AppDatabase
import
com.example.shuffleit.data.database.entities.PlayListEntity
import com.example.shuffleit.models.AudioFile
import com.google.gson.Gson
import com.mikepenz.fastadapter.dsl.genericFastAdapter
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.runBlocking
import kotlinx.coroutines.withContext

```

```

import java.io.File
import java.io.FileWriter
import java.io.IOException

class AudioControlViewModel(private val application:
Application) : AndroidViewModel(application) {
    private val database =
AppDatabase.getDatabase(application.applicationContext)
    private val playListDao = database.playListDao()
    private val audioFileDao = database.audioFileDao()

    private var audioManager: AudioManager =
application.getSystemService(Context.AUDIO_SERVICE) as
AudioManager

    private var _playlists = MutableLiveData(getAllPlayLists())
    val playlists: LiveData<ArrayList<PlayListEntity>> get() =
_playlists

    private val _isPlaying = MutableLiveData<Boolean>().apply {
value = false }
    val isPlaying: MutableLiveData<Boolean> get() = _isPlaying

    private lateinit var playlist: PlayList
    private val handler: Handler =
Handler(Looper.getMainLooper())
    private var mediaPlayer: MediaPlayer? = null

    private val _currentTime = MutableLiveData<Int>()
    val currentTime: LiveData<Int> get() = _currentTime
    val currentTimeString = _currentTime.map {
Utils.intToTime(it) }

    private val _trackLen = MutableLiveData<Int>()
    val trackLen: LiveData<Int> get() = _trackLen
    val trackLenString = _trackLen.map { Utils.intToTime(it) }

    private val _trackTitle = MutableLiveData<String>()
    val trackTitle: LiveData<String> get() = _trackTitle

    private val _trackImage = MutableLiveData<Bitmap?>()
    val trackImage: LiveData<Bitmap?> get() = _trackImage

    private val sharedPreferences =
application.getSharedPreferences("LastPlayedPlayList",
Context.MODE_PRIVATE)

    val seekBarChangeListener = object :
SeekBar.OnSeekBarChangeListener {

```

```

        override fun onProgressChanged(seekBar: SeekBar,
progress: Int, fromUser: Boolean) {
            if (fromUser) {
                seekTo(progress)
            }
        }

        override fun onStartTrackingTouch(seekBar: SeekBar) {
            startChangeSeekBar()
        }

        override fun onStopTrackingTouch(seekBar: SeekBar) {
            stopChangeSeekBar()
            seekTo(seekBar.progress)
        }
    }

    init {
        _trackLen.value = 0
        _currentTime.value = 0
        _playlists.value?.addAll(getAllPlayLists())
        val playlistId = sharedPreferences.getLong("PlaylistId",
-1)
        val nowPlaying = sharedPreferences.getInt("nowPlaying",
-1)
        val currentTime =
sharedPreferences.getInt("currentTime",0)
        if (playlistId.toInt() != -1) {
            viewModelScope.launch {
                val retrievedPlaylist = getPlayList(playlistId)
                retrievedPlaylist?.let {
                    playlist = it
                    if (nowPlaying != -1) {
                        playlist.setNowPlaying(nowPlaying)
                    }
                }
            }
        }

        initMediaPlayer(playlist.getAudio().path)
        if (currentTime != 0) {
            mediaPlayer?.seekTo(currentTime)
            updateCurrentTime()
        }
    }

    fun onPlaylistClick(playlist: PlayListEntity) {
        selectPlaylist(playlist)
    }

```

```

fun initMediaPlayer(audioFile: Uri) {
    mediaPlayer?.release()
    mediaPlayer = MediaPlayer().apply {
        setDataSource(application.applicationContext,
audioFile)
        setAudioStreamType(AudioManager.STREAM_MUSIC)
        prepare()
        setOnPreparedListener {
            _trackLen.value = it.duration / 1000
        }
        setOnCompletionListener {
            nextTrack()
        }
    }
    retrieveMetadata(audioFile)
}

private fun retrieveMetadata(uri: Uri) {
    CoroutineScope(Dispatchers.IO).launch {
        val retriever = MediaMetadataRetriever()

retriever.setDataSource(getApplication<Application>().applicatio
nContext, uri)
        val title =
retriever.extractMetadata(MediaMetadataRetriever.METADATA_KEY_TI
TLE)

        val albumArt = retriever.embeddedPicture
retriever.release()

        withContext(Dispatchers.Main) {
            _trackTitle.value = title ?: "Unknown"
            _trackImage.value = albumArt?.let{
BitmapFactory.decodeByteArray(it, 0, it.size) }
        }
    }
}

fun selectPlaylist(playListEntity: PlayListEntity){
    if (::playlist.isInitialized) {
        updateQueue(playlist.id, playlist.getQueue())
    }
    viewModelScope.launch {
        playlist = getPlayList(playListEntity.id)!!
        initMediaPlayer(playlist.getAudio().path)
    }
}

fun playPause() {
    if (_isPlaying.value == true) {
        stop()
    }
}

```

```

        } else {
            play()
        }
    }

private fun play() {
    updateCurrentTime()
    mediaPlayer?.start()
    _isPlaying.value = true
}

private fun stop() {
    handler.removeCallbacks(updateCurrentTimeRunnable)
    mediaPlayer?.pause()
    _isPlaying.value = false
}

fun volumeUp() {
    audioManager.adjustVolume(AudioManager.ADJUST_RAISE,
AudioManager.FLAG_PLAY_SOUND)
}

fun volumeDown() {
    audioManager.adjustVolume(AudioManager.ADJUST_LOWER,
AudioManager.FLAG_PLAY_SOUND)
}

fun nextTrack(){
    initMediaPlayer(playlist.getNextAudio().path)
    play()
}

fun previousTrack(){
    initMediaPlayer(playlist.getPreviousAudio().path)
    play()
}

fun seekTo(seekTo: Int) {
    mediaPlayer?.seekTo(seekTo * 1000)
    updateCurrentTime()
}

fun startChangeSeekBar() {
    handler.removeCallbacks(updateCurrentTimeRunnable)
}

fun stopChangeSeekBar() {
    handler.removeCallbacks(updateCurrentTimeRunnable)
}

```

```

        private fun updateCurrentTime() {
            _currentTime.value = (mediaPlayer?.currentPosition ?: 0)
/ 1000
            handler.postDelayed(updateCurrentTimeRunnable, 1000)
        }

        private val updateCurrentTimeRunnable = object : Runnable {
            override fun run() {
                updateCurrentTime()
            }
        }

        fun addPlaylist(uri: Uri, name: String) {
            viewModelScope.launch {
                val newPlaylistEntity = PlayListEntity(
                    title = name,
                    nowPlaying = 0,
                    queue = emptyList()
                )

                val playlistId =
playListDao.insert(newPlaylistEntity)
                val audioFiles =
Utils.getAudioFilesFromUri(application.applicationContext, uri,
playlistId)

                val audioFileIds:ArrayList<Long> = arrayListOf()
                audioFiles.forEach { audioFile ->
                    audioFileIds.add(audioFileDao.insert(audioFile))
                }
                val queue = shuffle(audioFileIds)
                updateQueue(playlistId, queue)

                _playlists.value?.add(playListDao.getPlayList(playlistId)!!)
                _playlists.value = _playlists.value
            }
        }

        private fun shuffle(audioFileIds : List<Long>): List<Long>{
            val queue = audioFileIds.shuffled()
            return queue
        }

        fun insertPlayList(playList: PlayList) {
            val playListEntity = PlayListEntity(
                title = playList.title,
                nowPlaying = playList.getNowPlaying(),
                queue = playList.getQueue()
            )
            viewModelScope.launch {
                playListDao.insert(playListEntity)
            }
        }

```

```

        playlist.playlist.forEach { audioFile ->
            audioFileDao.insert(AudioFile(path =
audioFile.path, playlistId = playlistEntity.id))
        }
    }
}

fun deletePlayList(playlistEntity: PlayListEntity) {
    viewModelScope.launch {
        playlistDao.delete(playlistEntity)

        if(playlist.id == playlistEntity.id){
            mediaPlayer?.release()
            mediaPlayer = null
            _trackTitle.value = ""
            _trackImage.value = null
            _trackLen.value = 0
        }
        val updatedPlaylists = withContext(Dispatchers.IO) {
            playlistDao.getAllPlayLists()
        }
        _playlists.value = updatedPlaylists as
ArrayList<PlayListEntity>
    }
}

fun getAllPlayLists(): ArrayList<PlayListEntity> {
    var playLists = arrayListOf<PlayListEntity>()
    viewModelScope.launch {
        playLists.addAll(playlistDao.getAllPlayLists())
    }
    return playLists
}

suspend fun getPlayList(id: Long): PlayList? {
    val playListWithAudioFiles =
playlistDao.getPlayListWithAudioFiles(id)
    return playListWithAudioFiles?.let {
        val audioFiles = it.audioFiles
        PlayList(it.playlistEntity.id,
it.playlistEntity.title, audioFiles,
it.playlistEntity.nowPlaying, it.playlistEntity.queue)
    }
}

fun updateNowPlaying(id: Long, nowPlaying: Int) {
    viewModelScope.launch {
        playlistDao.updateNowPlaying(id, nowPlaying)
    }
}

```

```

fun updateQueue(id: Long, queue: List<Long>) {
    viewModelScope.launch {
        playlistDao.updateQueue(id, queue)
    }
}

fun getAudioFile(id: Long): LiveData<AudioFile?> {
    val audioFile = MutableLiveData<AudioFile?>()
    viewModelScope.launch {
        audioFile.postValue(audioFileDao.getAudioFile(id))
    }
    return audioFile
}

fun onStop(){
    val editor = sharedPreferences.edit()
    if(::playlist.isInitialized) {
        editor.putLong("PlaylistId", playlist.id)
        editor.apply()
        editor.putInt("nowPlaying",
playlist.getNowPlaying())
        editor.apply()
        editor.putInt("currentTime",
mediaPlayer!!.currentPosition)
        editor.apply()
        updateQueue(playlist.id, playlist.getQueue())
    }
    mediaPlayer?.release()
    handler.removeCallbacks(updateCurrentTimeRunnable)
}
}
AudioControlView.kt
package com.example.shuffleit.views

import android.annotation.SuppressLint
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.util.Log
import android.widget.EditText
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.viewModels
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.databinding.DataBindingUtil
import androidx.lifecycle.Observer

```

```

import androidx.recyclerview.widget.LinearLayoutManager
import com.bumptech.glide.Glide
import com.example.shuffleit.R
import com.example.shuffleit.adapters.PlaylistAdapter
import com.example.shuffleit.databinding.AudioControlBinding
import com.example.shuffleit.viewModels.AudioControlViewModel

class AudioControlView: AppCompatActivity() {
    private lateinit var binding: AudioControlBinding
    private val viewModel: AudioControlViewModel by viewModels()

    private val selectFolderLauncher =
registerForActivityResult(
    ActivityResultContracts.OpenDocumentTree()
) { uri: Uri? ->
    uri?.let {
        contentResolver.takePersistableUriPermission(
            it,
            Intent.FLAG_GRANT_READ_URI_PERMISSION or
Intent.FLAG_GRANT_WRITE_URI_PERMISSION
        )
        showAddPlaylistDialog(it)
    }
}

@SuppressLint("NotifyDataSetChanged")
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = DataBindingUtil.setContentView(this,
R.layout.audio_control)
    binding.viewModel = viewModel
    binding.lifecycleOwner = this
    binding.menuButton.setOnClickListener {
        binding.drawerLayout.open()
    }

    binding.addPlaylistButton.setOnClickListener {
        selectFolderLauncher.launch(null)
    }

    val adapter = PlaylistAdapter(viewModel)
    binding.playlistRecyclerView.layoutManager =
LinearLayoutManager(this)
    binding.playlistRecyclerView.adapter = adapter

    viewModel.playlists.observe(this, Observer { playlists -
>
        adapter.setPlaylists(playlists)
    })
}

```

```

        viewModel.trackImage.observe(this, Observer { image->
            if(image == null){

binding.audioImage.setImageResource(R.drawable.ic_audio_image_pl
aceholder)
            }
            else{
                Glide.with(this)
                    .load(image)
                    .into(binding.audioImage)
            }
        })
    }

    private fun showAddPlaylistDialog(uri: Uri) {
        val builder = AlertDialog.Builder(this)
        builder.setTitle("Enter Playlist Name")

        val input = EditText(this)
        builder.setView(input)

        builder.setPositiveButton("OK") { dialog, which ->
            val playlistName = input.text.toString()
            viewModel.addPlaylist(uri, playlistName)
        }
        builder.setNegativeButton("Cancel") { dialog, which ->
            dialog.cancel() }

        builder.show()
    }

    override fun onStop() {
        super.onStop()
        Log.d("AudioControlView", "onStop called")
        viewModel.onStop()
    }

    override fun onDestroy() {
        super.onDestroy()
    }
}
Utils.kt
package com.example.shuffleit

import android.content.Context
import android.database.Cursor
import android.net.Uri
import android.provider.DocumentsContract
import com.example.shuffleit.models.AudioFile

```

```

class Utils {
    companion object {
        fun intToTime(time: Int): String {
            var hours = (time/3600).toString()
            var mins = (time/60).toString()
            var secs = (time%60).toString().padStart(2, '0')
            if(hours!="0"){
                mins = mins.padStart(2, '0')
                return "$hours: $mins : $secs"
            }
            return "$mins : $secs"
        }

        fun getAudioFilesFromUri(context: Context, uri: Uri,
            playlistId:Long): List<AudioFile> {
            val audioFiles = mutableListOf<AudioFile>()
            val docUri =
DocumentsContract.buildDocumentUriUsingTree(uri,
DocumentsContract.getTreeDocumentId(uri))
            collectAudioFiles(context, docUri, audioFiles,
            playlistId)
            return audioFiles
        }

        private fun collectAudioFiles(context: Context, uri:
Uri, audioFiles: MutableList<AudioFile>, playlistId:Long) {
            val childrenUri =
DocumentsContract.buildChildDocumentsUriUsingTree(uri,
DocumentsContract.getDocumentId(uri))
            val cursor: Cursor? =
context.contentResolver.query(childrenUri, arrayOf(
DocumentsContract.Document.COLUMN_DOCUMENT_ID,
DocumentsContract.Document.COLUMN_DISPLAY_NAME,
DocumentsContract.Document.COLUMN_MIME_TYPE,
DocumentsContract.Document.COLUMN_SIZE
), null, null, null)

            cursor?.use {
                val idColumn =
cursor.getColumnIndexOrThrow(DocumentsContract.Document.COLUMN_D
OCUMENT_ID)
                val mimeTypeColumn =
cursor.getColumnIndexOrThrow(DocumentsContract.Document.COLUMN_M
IME_TYPE)

                while (cursor.moveToNext()) {
                    val documentId = cursor.getString(idColumn)
                    val mimeType =
cursor.getString(mimeTypeColumn)

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

78


```

        return getAudioFromQueue()
    }
    private fun getAudioFromQueue():AudioFile{
        return playlist.filter { queue[nowPlaying] == it.id
    }.first()
    }

    fun getNowPlaying():Int{
        return nowPlaying
    }

    fun setNowPlaying(nowPlaying: Int){
        this.nowPlaying = nowPlaying
    }

    fun getQueue():List<Long>{
        return queue
    }

    fun getNextAudio():AudioFile{
        if(nowPlaying+1 == playlist.size){
            reloadQueue()
            nowPlaying = 0
            return getAudioFromQueue()
        }
        nowPlaying++
        return getAudioFromQueue()
    }

    fun getPreviousAudio():AudioFile{
        if(nowPlaying == 0){
            reloadQueue()
            nowPlaying = 0
            return getAudioFromQueue()
        }
        nowPlaying--
        return getAudioFromQueue()
    }

    private fun reloadQueue(){
        queue = queue.shuffled()
    }
}
AudioFile.kt
package com.example.shuffleit.models

import android.net.Uri
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.ForeignKey

```

```

import androidx.room.Index
import androidx.room.PrimaryKey
import
com.example.shuffleit.data.database.entities.PlayListEntity

@Entity(
    tableName = "audio_files",
    foreignKeys = [ForeignKey(
        entity = PlayListEntity::class,
        parentColumns = ["id"],
        childColumns = ["playlistId"],
        onDelete = ForeignKey.CASCADE
    )],
    indices = [Index(value = ["playlistId"])]
)
data class AudioFile(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    @ColumnInfo(name = "path") val path: Uri,
    @ColumnInfo(name = "playlistId") val playlistId: Long
)
PlaylistAdapte.kt
package com.example.shuffleit.adapters

import android.content.Context
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.appcompat.app.AlertDialog
import androidx.recyclerview.widget.RecyclerView
import com.example.shuffleit.R
import com.example.shuffleit.databinding.PlaylistItemBinding
import com.example.shuffleit.viewModels.AudioControlViewModel
import
com.example.shuffleit.data.database.entities.PlayListEntity

class PlaylistAdapter(
    private val viewModel: AudioControlViewModel
) : RecyclerView.Adapter<PlaylistAdapter.PlaylistViewHolder>() {

    private var playlists: List<PlayListEntity> = emptyList()

    fun setPlaylists(playlists: List<PlayListEntity>) {
        this.playlists = playlists
        notifyDataSetChanged()
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): PlaylistViewHolder {
        val binding =
PlaylistItemBinding.inflate(LayoutInflater.from(parent.context),
parent, false)

```

```

        return PlaylistViewHolder(binding)
    }

    override fun onBindViewHolder(holder: PlaylistViewHolder,
position: Int) {
        val playlist = playlists[position]
        holder.bind(playlist)
    }

    override fun getItemCount(): Int {
        return playlists.size
    }

    inner class PlaylistViewHolder(private val binding:
PlaylistItemBinding) : RecyclerView.ViewHolder(binding.root) {

        fun bind(playlist: PlayListEntity) {
            binding.playlistTitle.text = playlist.title

            binding.root.setOnClickListener {
                viewModel.onPlaylistClick(playlist)
            }

            binding.deleteButton.setOnClickListener {
showDeleteConfirmationDialog(binding.root.context, playlist)
            }

            private fun showDeleteConfirmationDialog(context:
Context, playlist: PlayListEntity) {
                AlertDialog.Builder(context)
                    .setTitle(R.string.delete_playlist)

                .setMessage(R.string.delete_playlist_confirmation)
                    .setPositiveButton(R.string.delete) { _, _ ->
                        viewModel.deletePlayList(playlist)
                    }
                    .setNegativeButton(R.string.cancel, null)
                    .show()
            }
        }
    }
}

IntListConverters.kt
package com.example.shuffleit.data.database.Converters

import androidx.room.TypeConverter
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken

```

```

class IntListConverter {
    @TypeConverter
    fun fromIntList(value: List<Long>): String {
        val gson = Gson()
        val type = object : TypeToken<List<Long>>() {}.type
        return gson.toJson(value, type)
    }

    @TypeConverter
    fun toIntList(value: String): List<Long> {
        val gson = Gson()
        val type = object : TypeToken<List<Long>>() {}.type
        return gson.fromJson(value, type)
    }
}

UriConverter.kt
package com.example.shuffleit.data.database.Converters

import android.net.Uri
import androidx.room.TypeConverter

class UriConverter {
    @TypeConverter
    fun fromUri(uri: Uri?): String? {
        return uri?.toString()
    }

    @TypeConverter
    fun toUri(uriString: String?): Uri? {
        return uriString?.let { Uri.parse(it) }
    }
}

PlaylistEntity.kt
package com.example.shuffleit.data.database.entities

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "playlists")
data class PlaylistEntity(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val title: String,
    val nowPlaying: Int,
    val queue: List<Long>
)

AppDatabase.kt
package com.example.shuffleit.data.database

```

```

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import
com.example.shuffleit.data.database.Converters.IntListConverter
import
com.example.shuffleit.data.database.Converters.UriConverter
import
com.example.shuffleit.data.database.entities.PlayListEntity
import com.example.shuffleit.models.AudioFile

@Database(entities = [AudioFile::class, PlayListEntity::class],
version = 1, exportSchema = false)
@TypeConverters(IntListConverter::class, UriConverter::class)
abstract class AppDatabase : RoomDatabase() {
    abstract fun audioFileDao(): AudioFileDao
    abstract fun playListDao(): PlayListDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                    "app_database"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}

AudioFileDao.kt
package com.example.shuffleit.data.database

import androidx.room.*
import
com.example.shuffleit.data.database.entities.PlayListEntity
import com.example.shuffleit.models.AudioFile
import com.example.shuffleit.models.PlayListWithAudioFiles

@Dao
interface AudioFileDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(audioFile: AudioFile): Long

```

```

        @Delete
        suspend fun delete(audioFile: AudioFile)

        @Query("SELECT * FROM audio_files WHERE id = :id")
        suspend fun getAudioFile(id: Long): AudioFile?
    }
}
PlaylistDao.kt
package com.example.shuffleit.data.database

import androidx.room.Dao
import androidx.room.Delete
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query
import androidx.room.Transaction
import com.example.shuffleit.data.database.entities.PlayListEntity
import com.example.shuffleit.models.PlayListWithAudioFiles

@Dao
interface PlayListDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(playList: PlayListEntity): Long

    @Delete
    suspend fun delete(playList: PlayListEntity)
    @Transaction
    @Query("SELECT * FROM playlists WHERE id = :id")
    suspend fun getPlayList(id: Long): PlayListEntity?
    @Transaction
    @Query("SELECT * FROM playlists")
    suspend fun getAllPlayLists(): List<PlayListEntity>

    @Transaction
    @Query("SELECT * FROM playlists WHERE id = :id")
    suspend fun getPlayListWithAudioFiles(id: Long):
    PlayListWithAudioFiles?

    @Query("UPDATE playlists SET nowPlaying = :nowPlaying WHERE
    id = :id")
    suspend fun updateNowPlaying(id: Long, nowPlaying: Int)

    @Query("UPDATE playlists SET queue = :queue WHERE id = :id")
    suspend fun updateQueue(id: Long, queue: List<Long>)
}
audio_control.xml
<!-- res/layout/activity_main.xml -->
<layout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"

```

```

xmlns:tools="http://schemas.android.com/tools">

<data>
    <variable
        name="viewModel"

type="com.example.shuffleit.viewModels.AudioControlViewModel" />
</data>
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#A9A9A9"
        tools:context=".views.AudioControlView">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#E64A19"

android:theme="@style/ThemeOverlay.AppCompat.ActionBar"

app:popupTheme="@style/ThemeOverlay.AppCompat.Light">
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#A9A9A9"
        tools:context=".views.AudioControlView">

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#E64A19"

android:theme="@style/ThemeOverlay.AppCompat.ActionBar"

app:popupTheme="@style/ThemeOverlay.AppCompat.Light">

```

```

        <ImageButton
            android:id="@+id/menuButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

android:background="?attr/selectableItemBackgroundBorderless"
            android:src="@drawable/ic_menu" />

    </androidx.appcompat.widget.Toolbar>

    <ImageView
        android:id="@+id/audioImage"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_below="@id/toolbar"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:background="@android:color/darker_gray"
        android:scaleType="centerCrop"

android:src="@drawable/ic_audio_image_placeholder" />

    <RelativeLayout
        android:id="@+id/controlPad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/seekBar"
        android:layout_below="@+id/trackName"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp">

        <ImageButton
            android:id="@+id/volumeUpButton"
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:layout_above="@id/playPauseButton"
            android:layout_centerHorizontal="true"
            android:layout_marginBottom="16dp"
            android:background="@drawable/ic_plus"
            android:onClick="@{() ->
viewModel.volumeUp()} " />

        <ImageButton
            android:id="@+id/playPauseButton"
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:layout_centerVertical="true"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="16dp"
            android:layout_toEndOf="@id/prevButton"

```

```

        android:background="@{viewModel.isPlaying ?
@drawable/ic_pause : @drawable/ic_play}"
        android:onClick="@{() ->
viewModel.playPause()}"/>

<ImageButton
    android:id="@+id/prevButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_centerVertical="true"
    android:background="@drawable/ic_prev"
    android:onClick="@{() ->
viewModel.previousTrack()}"/>

<ImageButton
    android:id="@+id/nextButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_centerVertical="true"
    android:layout_toEndOf="@id/playPauseButton"
    android:background="@drawable/ic_next"
    android:onClick="@{() ->
viewModel.nextTrack()}"/>

<ImageButton
    android:id="@+id/volumeDownButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_below="@id/playPauseButton"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:background="@drawable/ic_minus"
    android:onClick="@{() ->
viewModel.volumeDown()}"/>
</RelativeLayout>

<TextView
    android:id="@+id/timeNow"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="10dp"
    android:layout_marginBottom="32dp"
    android:text="@{viewModel.currentTimeString}"/>

<TextView
    android:id="@+id/timeFull"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

```

```

        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginEnd="10dp"
        android:layout_marginBottom="32dp"
        android:text="@{viewModel.trackLenString}" />

<SeekBar
    android:id="@+id/seekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="10dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="10dp"
    android:layout_marginBottom="32dp"
    android:layout_toStartOf="@+id/timeFull"
    android:layout_toEndOf="@+id/timeNow"
    android:max="@{viewModel.trackLen}"

    android:onSeekBarChangeListener="@{viewModel.seekBarChangeListener}"

    android:progress="@{viewModel.currentTime}" />

<TextView
    android:id="@+id/appName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginStart="164dp"
    android:layout_marginTop="16dp"
    android:text="@string/shuffle_it"
    android:textSize="20dp" />

<TextView
    android:id="@+id/trackName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/audioImage"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:textSize="16sp"
    android:text="@{viewModel.trackTitle}"/>

</RelativeLayout>

<com.google.android.material.navigation.NavigationView
    android:id="@+id/navigation_view"
    android:layout_width="wrap_content"

```

```

android:layout_height="match_parent"
android:layout_gravity="start"
android:background="#A9A9A9"
android:fitsSystemWindows="false"
app:headerLayout="@layout/playlist_menu_header"
app:menu="@menu/playlist_menu">

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/add_playlist_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp"
        android:text="Add Playlist" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/playlist_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="48dp" />
</RelativeLayout>
<ImageView
    android:id="@+id/audioImage"
    android:layout_width="300dp"
    android:layout_height="300dp"
    android:layout_below="@id/toolbar"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="32dp"
    android:background="@android:color/darker_gray"
    android:scaleType="centerCrop"

    android:src="@drawable/ic_audio_image_placeholder" />

    <RelativeLayout
        android:id="@+id/controlPad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/seekBar"
        android:layout_below="@+id/trackName"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp">

        <ImageButton
            android:id="@+id/volumeUpButton"
            android:layout_width="64dp"

```

```

        android:layout_height="64dp"
        android:layout_above="@id/playPauseButton"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="16dp"
        android:background="@drawable/ic_plus"
        android:onClick="@{() ->
viewModel.volumeUp()}"/>

<ImageButton
    android:id="@+id/playPauseButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_centerVertical="true"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp"
    android:layout_toEndOf="@id/prevButton"
    android:background="@{viewModel.isPlaying ?
@drawable/ic_pause : @drawable/ic_play}"
    android:onClick="@{() ->
viewModel.playPause()}"/>

<ImageButton
    android:id="@+id/prevButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_centerVertical="true"
    android:background="@drawable/ic_prev"
    android:onClick="@{() ->
viewModel.previousTrack()}"/>

<ImageButton
    android:id="@+id/nextButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_centerVertical="true"
    android:layout_toEndOf="@id/playPauseButton"
    android:background="@drawable/ic_next"
    android:onClick="@{() ->
viewModel.nextTrack()}"/>

<ImageButton
    android:id="@+id/volumeDownButton"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_below="@id/playPauseButton"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="16dp"
    android:background="@drawable/ic_minus"
    android:onClick="@{() ->
viewModel.volumeDown()}"/>

```

```

</RelativeLayout>

<TextView
    android:id="@+id/timeNow"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="10dp"
    android:layout_marginBottom="32dp"
    android:text="@{viewModel.currentTimeString}" />

<TextView
    android:id="@+id/timeFull"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true"
    android:layout_marginEnd="10dp"
    android:layout_marginBottom="32dp"
    android:text="@{viewModel.trackLenString}" />

<SeekBar
    android:id="@+id/seekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="10dp"
    android:layout_marginTop="32dp"
    android:layout_marginEnd="10dp"
    android:layout_marginBottom="32dp"
    android:layout_toStartOf="@+id/timeFull"
    android:layout_toEndOf="@+id/timeNow"
    android:max="@{viewModel.trackLen}"

    android:onSeekBarChangeListener="@{viewModel.seekBarChangeListener}"

    android:progress="@{viewModel.currentTime}" />

<TextView
    android:id="@+id/appName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_marginStart="164dp"
    android:layout_marginTop="16dp"
    android:text="@string/shuffle_it"
    android:textSize="20dp" />

```

```

        <TextView
            android:id="@+id/trackName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@+id/audioImage"
            android:layout_centerHorizontal="true"
            android:layout_marginTop="16dp"
            android:textSize="16sp"
            android:text="@{viewModel.trackTitle}"/>

    </RelativeLayout>

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigation_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:background="#A9A9A9"
        android:fitsSystemWindows="false"
        app:headerLayout="@layout/playlist_menu_header"
        app:menu="@menu/playlist_menu">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <Button
                android:id="@+id/add_playlist_button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentBottom="true"
                android:layout_centerHorizontal="true"
                android:layout_marginBottom="16dp"
                android:text="Add Playlist" />

            <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/playlist_recycler_view"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="48dp" />
        </RelativeLayout>

    </com.google.android.material.navigation.NavigationView>

</androidx.drawerlayout.widget.DrawerLayout>
</layout>
playlist_item.xml
<?xml version="1.0" encoding="utf-8"?>

```

```

<layout xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:android="http://schemas.android.com/apk/res/android">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:padding="16dp">

        <TextView
            android:id="@+id/playlistTitle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text=""
            android:textColor="@color/black"
            android:textSize="18sp" />
        <TextView
            android:id="@+id/playlistTitle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text=""
            android:textColor="@color/black"
            android:textSize="18sp" />

        <ImageButton
            android:id="@+id/deleteButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_delete"
            android:contentDescription="@string/delete_playlist"

            <ImageButton
                android:id="@+id/deleteButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/ic_delete"
                android:contentDescription="@string/delete_playlist"

            android:background="?attr/selectableItemBackgroundBorderless"
            android:padding="8dp" />
        </LinearLayout>
    </layout>
playlist_menu_header.xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

```

Изм.	Лист	№ докум.	Подпись	Дата

```
android:background="#A9A9A9">
```

```
<androidx.appcompat.widget.Toolbar  
    android:id="@+id/toolbar3"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"  
    android:layout_alignParentEnd="true"  
    android:background="#E64A19"  
    android:minHeight="?attr/actionBarSize"  
    android:theme="?attr/actionBarTheme" />
```

```
<TextView  
    android:id="@+id/textView"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginTop="16dp"  
    android:text="@string/playlists"  
    android:textColor="@android:color/black"  
    android:textSize="20sp" />
```

```
</RelativeLayout>
```

					ПГУ 1.090301.03.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ В

(Обязательное)

Аудиоплеер для Android

Презентация

					ПГУ 1.090301.03.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		



Аудиоплеер для ОС Android

Автор работы: студент группы 20ВВП1 Беляков А. М.

Научный руководитель: к.т.н. доцент кафедры ВТ Слепцов Н. В.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист

97

Актуальность проекта

В настоящее время мобильные телефоны и планшеты занимают важную роль в жизни человека. Различные устройства на системе Android предоставляют широкий спектр возможностей для пользователя. Довольно востребованным классом приложений являются различные аудио плееры, так как мало кто из людей хочет покупать и иметь отдельный гаджет, когда всегда под рукой есть смартфон.

Важность создания приложения для проигрывания аудио файлов обусловлена постоянным развитием мобильных технологий и популярностью различных мультимедийных приложения среди пользователей мобильных устройств.

Изм.	Лист	№ докум.	Подпись	Дата

Цель работы

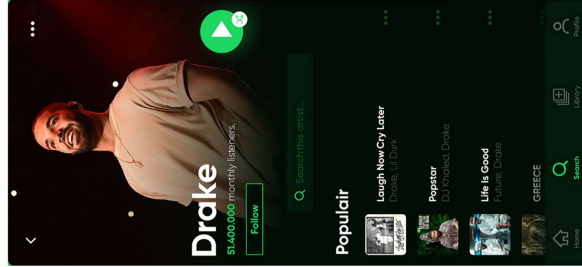
Целью дипломной работы является разработка аудио плеера для операционной системы Android, который будет отвечать современным требованиям пользователей и давать им доступ к комфортному инструменту для прослушивания аудио контента.

В приложении должен быть реализован следующий функционал:

- ◆ Создание плейлистов
- ◆ Управление воспроизводимой музыкой
- ◆ Сохранение состояния сессии

Изм.	Лист	№ докум.	Подпись	Дата

Обзор аналогов Spotify



Достоинства

- Обширная библиотека композиций
- Персональные рекомендации
- Понятный и удобный дизайн

Недостатки

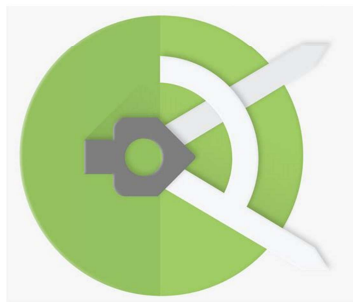
- Необходимость подписки для всех функций
- Не доступно на территории РФ

Изм.	Лист	№ докум.	Подпись	Дата

Стек используемых технологий



Kotlin

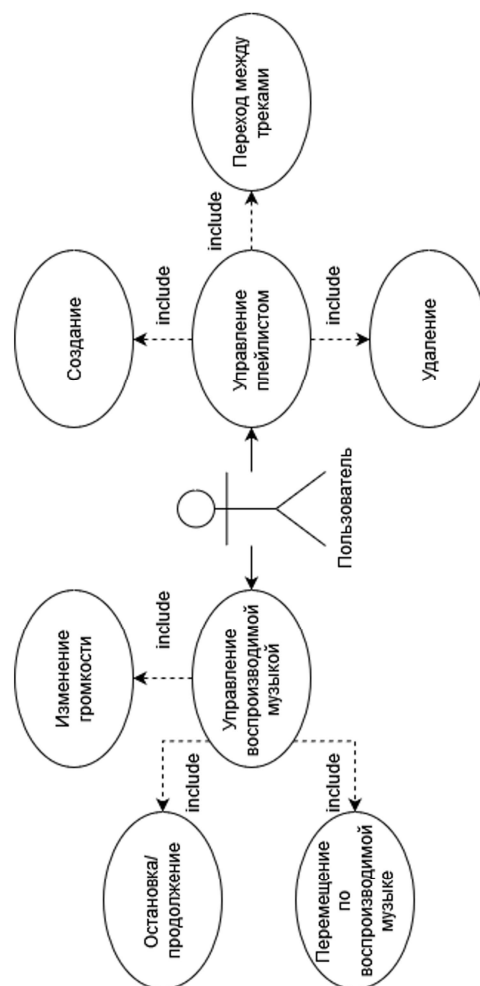


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ

Лист
101

UML-диаграмма прецедентов



Приложение “Shuffleit”



Основной экран приложения



Экран списка плейлистов



Экран добавления плейлиста



Изм.	Лист	№ докум.	Подпись	Дата

Заключение

- ◆ Результатом выполнения данной выпускной квалификационной работы можно считать успешно разработанное мобильное приложение «Shuffleit», полностью соответствующее заданию.
- ◆ В ходе разработки приложения были получены навыки создания программ под ОС Android, работы с механизмами View и DataBinding, изучена архитектура MVVM.
- ◆ Результаты тестирования показали, что приложение полностью соответствует всем поставленным задачам, выполняет свою работу и имеет удобный пользовательский интерфейс.

Изм.	Лист	№ докум.	Подпись	Дата



Спасибо за внимание!



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.03.001 ПЗ