МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

КУРСОВАЯ РАБОТА

по дисциплине «Проектирование и разработка мобильных приложений»

Тема курсовой работы: «Приложение для воспроизведения медиатеки»

Студент группы ИКБО-22-23	Жулин Антон Александрович	(подпись)
Руководитель курсовой работы	Степанов Павел Валериевич	(подпись)
Работа представлена к защите	«16» Марта 2026 г.	
Допущен к защите	«16» Марта 2026 г.	

Москва 2026 г.

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

Утверждаю
И.О.Заведующего кафедрой МОСИТ
_____ Головин С.А.
«16» марта 2026 г.

ЗАДАНИЕ

на выполнение курсовой работы

по дисциплине «Проектирование и разработка мобильных приложений»

Студент Жулин Антон Александрович Группа ИКБО-22-23

Тема «Приложение медиаплеер для Android»

Исходные данные: Разрабатываемый прототип мобильного приложения должен предоставлять возможности полностью интерактивной системы с понятным дружественным UI и обеспечивать необходимую функциональность, которая формируется в зависмости от заданной темы и предметной области изучаемых вопросов.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:
Установка и настройка мобильной ОС с применением виртуальных сред. Установка и настройка IDE для мобильной разработки. Установка и настройка эмуляторов мобильного девайса.
Анализ предметной области разрабатываемой программной системы, сбор и анализ требований, составление ТЗ согласно ГОСТ 19.201-78. Реализация ролевой модели безопасности.
Изучение жизненного цикла мобильных программ и их компонентов, а также создание мобильного программного комплекса с применением языков программирования высокого уровня согласно теме курсовой работы. Реализация в создаваемом программном комплексе визуальных элементов, сервисов, методов хранения данных.
Тестирование и отладка кода созданного программного продукта, контрольные примеры работы.
Возможно портирование написанной программной системы на внешних хостах в сети Интернет.
Отчет по курсовой работе в виде пояснительной записки.

Срок представления к защите курсовой работы: **до** «30» мая 2026 г.

Задание на курсовую работу выдал _____ Степанов П.В.
Подпись руководителя *(ФИО руководителя)*
«16» марта 2026 г.

Задание на курсовую работу получил _____ Жулин А. А.
Подпись обучающегося *(ФИО обучающегося)*
«16» марта 2026 г.

СОДЕРЖАНИЕ

● ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ

API (Application Programming Interface): Интерфейс программирования приложений – набор готовых классов, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. Используется, например, при работе с Android API для доступа к медиафайлам.

CRUD (Create, Read, Update, Delete): Акроним, обозначающий базовый набор операций (Создание, Чтение, Обновление, Удаление) для работы с данными в персистентных хранилищах, таких как базы данных.

DAO (Data Access Object): Объект доступа к данным – паттерн проектирования, предоставляющий абстрактный интерфейс к какой-либо базе данных или другому механизму персистентности. В проекте используется для взаимодействия с базой данных Room.

DFD (Data Flow Diagram): Диаграмма потоков данных – графическая нотация, используемая для описания потоков информации и их преобразований в системе.

IDE (Integrated Development Environment): Интегрированная среда разработки – комплекс программных средств, используемый программистами для разработки программного обеспечения, например, Android Studio.

IDEF0 (Integration Definition for Function Modeling): Нотация для функционального моделирования, используемая для описания основных функций системы и потоков данных между ними.

Jetpack Compose: Современный декларативный инструментарий пользовательского интерфейса Android для создания нативных UI.

Kotlin: Статически типизированный язык программирования, официально поддерживаемый для разработки Android-приложений.

Material 3: Актуальная версия системы дизайна от Google, предоставляющая компоненты и рекомендации для создания современных пользовательских интерфейсов.

MediaStore: Системный сервис Android, который индексирует медиафайлы (аудио, видео, изображения) на устройстве и предоставляет стандартизированный доступ к их метаданным.

MVVM (Model-View-ViewModel): Архитектурный паттерн, используемый для разделения логики представления данных (View) от бизнес-логики и данных (Model) с помощью ViewModel. Применяется в проекте с использованием классов ViewModel.

Room: Библиотека из состава Android Jetpack, предоставляющая слой абстракции над SQLite для более удобной и безопасной работы с локальными базами данных.

SQLite: Компактная встраиваемая реляционная база данных, широко используемая в мобильных приложениях для локального хранения данных.

UI (User Interface): Пользовательский интерфейс – средства, при помощи которых пользователь взаимодействует с программой или устройством.

UML (Unified Modeling Language): Унифицированный язык моделирования – графический язык для спецификации, визуализации, проектирования и документирования программных систем.

URI (Uniform Resource Identifier): Унифицированный идентификатор ресурса – строка символов, идентифицирующая ресурс в сети (например, URL) или в файловой системе устройства.

UX (User Experience): Пользовательский опыт – совокупность впечатлений, эмоций и удобства, которые пользователь получает от взаимодействия с продуктом или системой.

ViewModel: Класс из состава Android Jetpack, предназначенный для хранения и управления данными, связанными с пользовательским интерфейсом, с учетом жизненного цикла компонентов UI.

ExoPlayer (Media3): Медиаплеер уровня приложения для Android, разработанный Google, предоставляющий расширенные возможности для воспроизведения аудио и видео.

MediaSession (Media3): Компонент Android, позволяющий приложению взаимодействовать с медиаконтроллерами, отвечать на аппаратные кнопки управления воспроизведением, отображать информацию о медиа на экране блокировки и в уведомлениях.

ВВЕДЕНИЕ

Предметной областью данного проекта является разработка программного обеспечения для воспроизведения аудиофайлов — аудиоплеера "HulLayer" — на мобильных устройствах под управлением операционной системы Android. Аудиоплееры представляют собой востребованные приложения, основной функцией которых является декодирование и воспроизведение звуковых данных из различных форматов файлов (например, MP3, AAC, FLAC, WAV).

Целью данной курсовой работы является проектирование и разработка функционального прототипа аудиоплеера "HulLayer", ориентированного на воспроизведение локальных аудиофайлов, хранящихся на устройстве пользователя. Актуальность проекта обусловлена стремлением предоставить пользователям простой и современный пользовательский интерфейс, созданный с использованием актуальных технологий Android-разработки, таких как Jetpack Compose и Material 3. Приложение ориентировано на современные версии операционной системы Android (версии 12 и выше), что позволяет использовать новейшие API для улучшения пользовательского опыта и безопасности.

Разрабатываемый аудиоплеер " HulLayer " должен обеспечивать ключевые функции, ожидаемые от подобных приложений: эффективное управление медиатекой, включая сканирование аудиофайлов и извлечение метаданных; надежное управление процессом воспроизведения аудио; возможность управления очередью треков; а также поддержку фонового воспроизведения музыки с отображением системных уведомлений и элементов управления.

1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

1.1 Описание предметной области

Предметной областью данного проекта является разработка программного обеспечения для воспроизведения аудиофайлов на мобильных устройствах под управлением операционной системы Android. Аудиоплееры представляют собой приложения, основной функцией которых является декодирование и воспроизведение звуковых данных из различных форматов файлов (например, MP3, AAC, FLAC, WAV). Ключевые аспекты предметной области включают:

Управление медиатекой: Сканирование локального хранилища устройства для обнаружения аудиофайлов, извлечение метаданных (название трека, исполнитель, альбом, обложка, длительность), каталогизация и предоставление пользователю удобного интерфейса для навигации по музыкальной коллекции. Проект " HulLayer " нацелен на воспроизведение локальных аудиофайлов, хранящихся на устройстве пользователя.

Функции воспроизведения: Обеспечение стандартных операций управления воспроизведением, таких как старт, пауза, остановка, переход к следующему/предыдущему треку, перемотка, регулировка громкости.

Управление очередью и плейлистами: Возможность создания и управления списками воспроизведения (плейлистами), формирование очереди проигрывания треков, режимы случайного воспроизведения (shuffle) и повтора (repeat).

Пользовательский интерфейс: Предоставление интуитивно понятного и эстетически приятного интерфейса для взаимодействия с функциями плеера, отображения информации о треке и обложек альбомов. Проект " HulLayer " использует Jetpack Compose и Material 3 для современного UI/UX.

Фоновое воспроизведение и уведомления: Способность продолжать воспроизведение музыки, когда приложение неактивно (свернуто или экран выключен), и предоставление системных уведомлений с элементами управления воспроизведением.

Работа с системными ресурсами: Корректное управление аудиофокусом, обработка событий подключения/отключения наушников, взаимодействие с другими мультимедийными приложениями.

Проект " HulLayer " ориентирован на платформу Android 12 и выше, что позволяет использовать современные API для доступа к медиафайлам (READ_MEDIA_AUDIO) и другие возможности системы, направленные на улучшение пользовательского опыта и безопасности.

1.2 Анализ существующих аналогов

AIMP for Android представляет собой мобильную версию популярного аудиопроигрывателя для Windows, который славится своим качеством звука и широким набором функций, обеспечивая поддержку большого количества аудиоформатов, включая loseless варианты, такие как FLAC, APE, WV, а также популярные форматы MP3, AAC, OGG и Opus. Встроенный многополосный эквалайзер с предустановками и возможностью ручной настройки позволяет гибко управлять звучанием, а развитое управление плейлистами поддерживает форматы M3U, M3U8, XSPF, PLS и CUE sheet, включая "умные" плейлисты, синхронизирующиеся с папками. Пользователи также могут редактировать теги аудиофайлов, слушать интернет-радио, настраивать вывод звука через OpenSL, AudioTrack или AAudio, пользоваться таймером сна, будильником и поддержкой Android Auto, а интерфейс кастомизировать с помощью сменных тем оформления. К сильным сторонам AIMP относятся высокое качество воспроизведения звука, обширная поддержка форматов, гибкие настройки эквалайзера, удобная работа с плейлистами и тегами, при этом приложение является бесплатным и не содержит рекламы; однако некоторым пользователям интерфейс может показаться перегруженным или менее современным, сканирование больших медиатек может занимать время, а навигация по ним не всегда интуитивно понятна.

VLC for Android, являющийся кроссплатформенным медиаплеером с открытым исходным кодом, известен своей способностью воспроизводить

практически любые медиаформаты, позволяя проигрывать большинство локальных видео и аудио файлов, сетевые потоки, включая адаптивную потоковую передачу, и DVD ISO, и поддерживая все популярные аудио и видео форматы, такие как MKV, MP4, AVI, MOV, Ogg, FLAC, TS, M2TS, Wv, AAC, без необходимости загрузки дополнительных кодеков. VLC предлагает полную аудиотеку с поиском, эквалайзером и фильтрами, управление плейлистами, поддержку многодорожечного аудио и субтитров, а удобство использования повышается за счет управления жестами для громкости, яркости и перемотки, наличия виджета для управления аудио, поддержки управления с гарнитуры и отображения обложек. Главными преимуществами VLC являются его исключительная всеядность форматов, бесплатность, отсутствие рекламы и шпионских модулей, открытый исходный код, а также мощные возможности по работе с видео и сетевыми потоками; с другой стороны, пользовательский интерфейс и управление плейлистами могут показаться некоторым пользователям менее интуитивными или "неуклюжими", обновление медиатеки может занимать продолжительное время, и, несмотря на то что VLC является мощным видеоплеером, его аудиофункции могут быть не такими глубоко проработанными, как у специализированных аудиоплееров.

1.3 Выбор инструментальных средств моделирования

Для моделирования функциональности и структуры разрабатываемого аудиоплеера " HulLayer " были выбраны следующие подходы и инструментальные средства:

Функциональное моделирование (IDEF0):

Нотация: IDEF0 (Integration Definition for Function Modeling) была выбрана для описания основных функций системы и потоков данных (информации, управления) между ними. Эта нотация позволяет наглядно представить систему как совокупность взаимосвязанных функциональных блоков, что полезно для понимания общей архитектуры и логики работы приложения. Пользователем

были предоставлены диаграммы в этой нотации (например, файл 1.xml и изображение image_72c655.jpg).

Инструмент: Для создания и редактирования IDEF0 диаграмм использовался онлайн-инструмент diagrams.net (ранее известный как draw.io). Этот инструмент является бесплатным, кроссплатформенным, поддерживает экспорт в различные форматы (включая XML, как в файле 1.xml) и предоставляет необходимые графические примитивы для построения IDEF0 диаграмм.

Структурное и поведенческое моделирование (с элементами UML):

Нотация: Хотя строгая UML-диаграмма не была явно указана как основная, для описания взаимодействия компонентов и структуры классов были использованы диаграммы, близкие по своей сути к компонентным диаграммам UML или диаграммам классов с акцентом на взаимодействие. Пример такой диаграммы представлен в файле image_66ece9.jpg. Такие диаграммы помогают визуализировать основные модули системы (например, PlayerService, PlayerViewModel, DatabaseManager), их ключевые операции и потоки данных/управления между ними. В некоторых случаях, описание таких диаграмм может производиться в терминах DFD (Data Flow Diagrams) для акцентирования на потоках данных.

Инструмент: Диаграммы такого типа также могут быть созданы с помощью diagrams.net, которое поддерживает широкий набор элементов UML и других нотаций для моделирования программных систем.

Моделирование данных (концептуальное):

Нотация: для описания структуры хранимых данных (информации о треках) использовалось текстовое описание сущностей и их атрибутов, как это представлено в схеме базы данных Room для TrackInfo в README.md. Это соответствует концептуальному уровню моделирования данных.

Инструмент: Описание схемы данных может быть выполнено в любом текстовом редакторе или интегрировано в документацию проекта.

Обоснование выбора:

IDEF0 была выбрана для верхнеуровневого функционального анализа, так как она хорошо подходит для декомпозиции системы на функции и определения их взаимосвязей с точки зрения входов, выходов, управления и механизмов.

Diagrams.net выбран как универсальный, доступный и гибкий инструмент, позволяющий создавать различные типы диаграмм, необходимые для моделирования как функциональных, так и структурных аспектов системы.

Использование диаграмм компонентов/классов (или их аналогов) помогает детализировать структуру приложения и взаимодействие его частей, что важно при объектно-ориентированном подходе к разработке, используемом в проекте " HulLayer " (Kotlin, классы ViewModel, Service и т.д.).

Текстовое описание схемы данных для Room является простым и эффективным способом задокументировать структуру базы данных на ранних этапах и для последующей реализации.

Такой набор инструментальных средств и нотаций позволяет комплексно подойти к моделированию системы " HulLayer ", охватывая как ее функциональные аспекты, так и внутреннюю структуру и потоки данных.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Определение пользовательских требований

Для разрабатываемого аудиоплеера " HulLayer " определена одна основная роль пользователя, взаимодействующего с системой:

Пользователь — человек, который использует приложение для прослушивания аудиофайлов, хранящихся на его устройстве.

На основе этой роли были составлены следующие пользовательские требования:

Пользователь (основная и единственная роль):

- просматривать список локальных аудиофайлов (треков), доступных на устройстве;

- получать информацию о каждом треке (название, исполнитель, альбом, обложка, длительность);

- воспроизводить выбранные аудиофайлы;

- управлять процессом воспроизведения (воспроизведение, пауза, переход к следующему/предыдущему треку, перемотка);

- видеть элементы управления воспроизведением и информацию о текущем треке во время фонового воспроизведения через системное уведомление;

- отмечать треки как "избранные" для быстрого доступа или формирования списка предпочтений;

- управлять очередью воспроизведения;

- просматривать простой и современный пользовательский интерфейс, соответствующий актуальным версиям Android.

- предоставлять необходимые разрешения для доступа к аудиофайлам и отображения уведомлений для корректной работы приложения.

2.2 Определение функциональных требований

На основе пользовательских требований были определены следующие функциональные требования для аудиоплеера " HulLayer ":

Система должна запрашивать у пользователя разрешение на доступ к аудиофайлам на устройстве (READ_MEDIA_AUDIO) при первом запуске или если разрешение не предоставлено.

Система должна запрашивать у пользователя разрешение на отправку уведомлений (POST_NOTIFICATIONS) для отображения элементов управления воспроизведением.

Система должна сканировать MediaStore устройства для обнаружения локальных аудиофайлов и извлекать их метаданные (название, исполнитель, альбом, длительность, URI файла, URI обложки).

Система должна сохранять (кэшировать) информацию об обнаруженных аудиофайлах в локальной базе данных (Room) для быстрого доступа.

Система должна отображать список доступных аудио треков пользователю.

Система должна предоставлять пользователю возможность выбрать трек для воспроизведения.

Система должна обеспечивать основные функции управления воспроизведением:

Воспроизведение/Пауза.

Переход к следующему треку.

Переход к предыдущему треку.

Перемотка текущего трека (изменение позиции воспроизведения).

Система должна отображать информацию о текущем воспроизводимом треке: название, исполнитель, альбом, обложка альбома, текущая позиция и общая длительность.

Система должна поддерживать фоновое воспроизведение аудио, когда приложение свернуто или экран устройства выключен.

Система должна отображать уведомление с элементами управления воспроизведением (Play/Pause, Next, Previous) и информацией о текущем треке во время фонового воспроизведения.

Система должна позволять пользователю отмечать треки как "избранные" и снимать эту отметку. Статус "избранное" должен сохраняться в базе данных.

Система должна корректно обрабатывать события, связанные с аудиофокусом (например, приостанавливать воспроизведение при поступлении звонка).

Система должна корректно обрабатывать подключение/отключение наушников или других аудиоустройств (например, приостанавливать воспроизведение при отключении наушников).

Система должна предоставлять приветственный экран для новых пользователей.

Система должна обеспечивать навигацию между основными экранами приложения (например, экран плеера, экран списка треков, экран настроек – если они будут реализованы).

2.3 Описание процесса в нотации IDEF0

На вход системы поступают два основных элемента. Во-первых, это "Local Audio Files" (Локальные аудиофайлы) – сами аудиофайлы, которые хранятся на устройстве пользователя и предназначены для воспроизведения. Во-вторых, это "User Interactions" (Взаимодействия с пользователем), включающие команды, такие как Play (Воспроизведение), Pause (Пауза), Skip (Пропустить) и Seek (Перемотка), которые пользователь передает для управления процессом воспроизведения.

Управление выполнением основной функции осуществляется двумя факторами. "Android Permissions (READ_MEDIA_AUDIO)" (Разрешения Android (READ_MEDIA_AUDIO)) представляют собой разрешения от операционной системы Android, необходимые для чтения аудиофайлов и доступа к медиаконтенту. "Android System Lifecycle Events" (События

жизненного цикла системы Android) – это системные события, такие как сворачивание приложения или входящий звонок, которые могут оказывать влияние на работу плеера.

В результате своей работы система производит два типа выходов. "Audio Playback" (Воспроизведение аудио) – это непосредственно звуковой поток, воспроизводимый приложением. "UI Updates & Notifications" (Обновления интерфейса и уведомления) включают визуальные изменения в пользовательском интерфейсе, например, отображение информации о текущем треке и прогрессе воспроизведения, а также системные уведомления, позволяющие управлять плеером из панели уведомлений.

Диаграмма представляет собой декомпозицию основной функции "Play Local Audio Files" (A0) системы " HulLayer Audio Player" на пять ключевых подфункций. Она детализирует, как система инициализируется, обрабатывает медиафайлы, управляет воспроизведением, взаимодействует с пользователем и обрабатывает фоновые задачи.

Первая подфункция, A1 "Initialize System & Request Permissions" (Инициализация системы и запрос разрешений), запускается при старте системы ("System Startup"). Она запрашивает необходимые разрешения Android ("Android Permissions") и, в случае успеха, передает сигнал "Permission Granted" (Разрешение получено) следующей функции.

Подфункция A2 "Discover & Cache Media Files" (Обнаружение и кэширование медиафайлов) активируется после получения разрешения от A1. Она использует "Local Audio Files" (Локальные аудиофайлы) в качестве входных данных, а "MediaStore API" и "Room Database" как механизмы для обнаружения файлов и сохранения информации о них. Результатом её работы является "Media Queue" (Очередь медиафайлов), которая передается для воспроизведения.

Центральной является подфункция A3 "Manage Audio Playback" (Управление воспроизведением аудио). Она получает "Media Queue" от A2, а также "Playback Commands" (Команды воспроизведения) и "Playback Status" (Статус воспроизведения) от A4. Для своей работы она использует "ExoPlayer"

как механизм и управляется "Android Services". На выходе A3 предоставляет "Audio Playback" (Аудиовоспроизведение) и "Player State" (Состояние плеера).

Подфункция A4 "Handle UI Interactions" (Обработка взаимодействий с пользовательским интерфейсом) отвечает за взаимодействие с пользователем ("User Interactions"). Используя "ExoPlayer & Media3 Framework" и "Jetpack Compose" как механизмы, она генерирует "Playback Commands" и "Playback Status" для A3, а также "UI Updates" (Обновления интерфейса) для A5.

Наконец, подфункция A5 "Manage Background Service & Notifications" (Управление фоновой службой и уведомлениями) получает "Player State" от A3 и "UI Updates" от A4. Она управляется событиями жизненного цикла системы ("System Lifecycle") и использует "MediaSessionService" как механизм для отображения "Notifications" (Уведомлений) пользователю.

2.4 Описание процесса в нотации DFD

Диаграмма в нотации DFD (рисунок 3) отображает течение информации в пределах процесса, где для отображения входных и выходных данных используются стрелки, процессы отображаются прямоугольниками (или в данном случае, компоненты системы интерпретируются как процессы), хранилища данных — прямоугольниками без правой стороны (или сущностями, содержащими данные, такими как база данных), а внешние объекты, взаимодействующие с системой, — прямоугольниками.

Представленная диаграмма иллюстрирует потоки данных в приложении для воспроизведения аудио: процесс начинается с компонента MediaResolve, который получает "audio files" из внешнего источника Android MediaStore и преобразует их в "track metadata", направляя данные для сохранения ("store tracks") в DatabaseManager, который, в свою очередь, взаимодействует с хранилищем TrackDatabase (Room) для сохранения и извлечения "track list". DatabaseManager передает "track list" компоненту QueueManager, который формирует "media queue" для PlayerService.

Взаимодействие с пользователем (User) осуществляется через PlayerScreen, который передает "UI events" и "interactions" в PlayerViewModel; PlayerViewModel обрабатывает эти события, передает "control commands" в PlayerService и получает от него "playback state", а также может инициировать "updateFavoriteStatus" в DatabaseManager. PlayerService управляет воспроизведением, используя "media queue", взаимодействует с MediaSession State для отражения "playback state" и отправляет "media notification" во внешнюю систему уведомлений (System Notification).

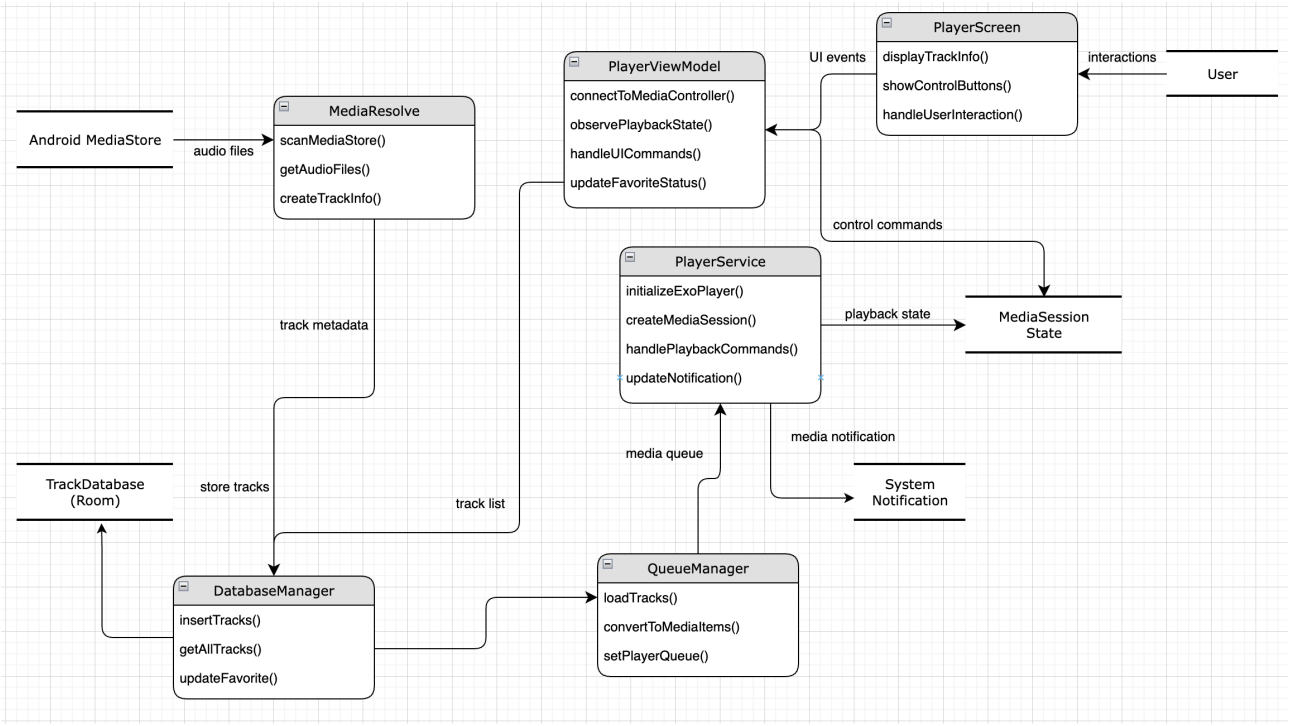


Рисунок 3 — Декомпозиция процесса «Использование приложения» в нотации DFD

2.5 Описание используемых средств разработки

Для разработки мобильного аудиоплеера " HulLayer " были использованы следующие основные инструментальные средства:

Android Studio

Android Studio является официальной интегрированной средой разработки (IDE) для создания приложений под операционную систему Android. Она разработана компанией Google и основана на IntelliJ IDEA от JetBrains. Для проекта " HulLayer " Android Studio используется как основное средство для

написания кода на языке Kotlin, проектирования пользовательского интерфейса с помощью Jetpack Compose, сборки, отладки и тестирования приложения. Ключевые возможности Android Studio, используемые в проекте:

Редактор кода с поддержкой Kotlin, автодополнением, рефакторингом и статическим анализом, интегрированная система сборки Gradle, позволяющая управлять зависимостями и конфигурациями сборки, инструменты для работы с Jetpack Compose, включая предварительный просмотр composable-функций в реальном времени, отладчик для пошагового выполнения кода и анализа состояния приложения, эмуляторы устройств Android для тестирования на различных конфигурациях и версиях ОС, интеграция с системой контроля версий Git.

GitHub Desktop

GitHub Desktop — это графический клиент для системы контроля версий Git, разработанный GitHub. Он упрощает взаимодействие с репозиториями, размещенными на GitHub или других Git-хостингах. В рамках проекта " HulLayer " GitHub Desktop используется для:

Управления локальными и удаленными Git-репозиториями.

Выполнения основных Git-операций, таких как коммит (фиксация изменений), создание веток, слияние веток (merge), отправка изменений на удаленный сервер (push) и получение обновлений (pull), визуализации истории изменений и сравнения версий файлов, упрощения совместной работы над проектом.

Использование Android Studio обеспечивает всестороннюю поддержку специфики разработки под Android, в то время как GitHub Desktop предоставляет удобный графический интерфейс для управления версиями исходного кода проекта " HulLayer ".

2.6 Архитектура программной системы

Архитектура программной системы "HulLayer" разработана в соответствии с официальными рекомендациями Google LLC по созданию

современных, масштабируемых и удобных в сопровождении Android-приложений. Эта архитектура основана на четком разделении ответственностей между слоями, как концептуально представлено на рисунке 4, и включает в себя слой пользовательского интерфейса (UI Layer), доменный слой (Domain Layer) и слой данных (Data Layer).

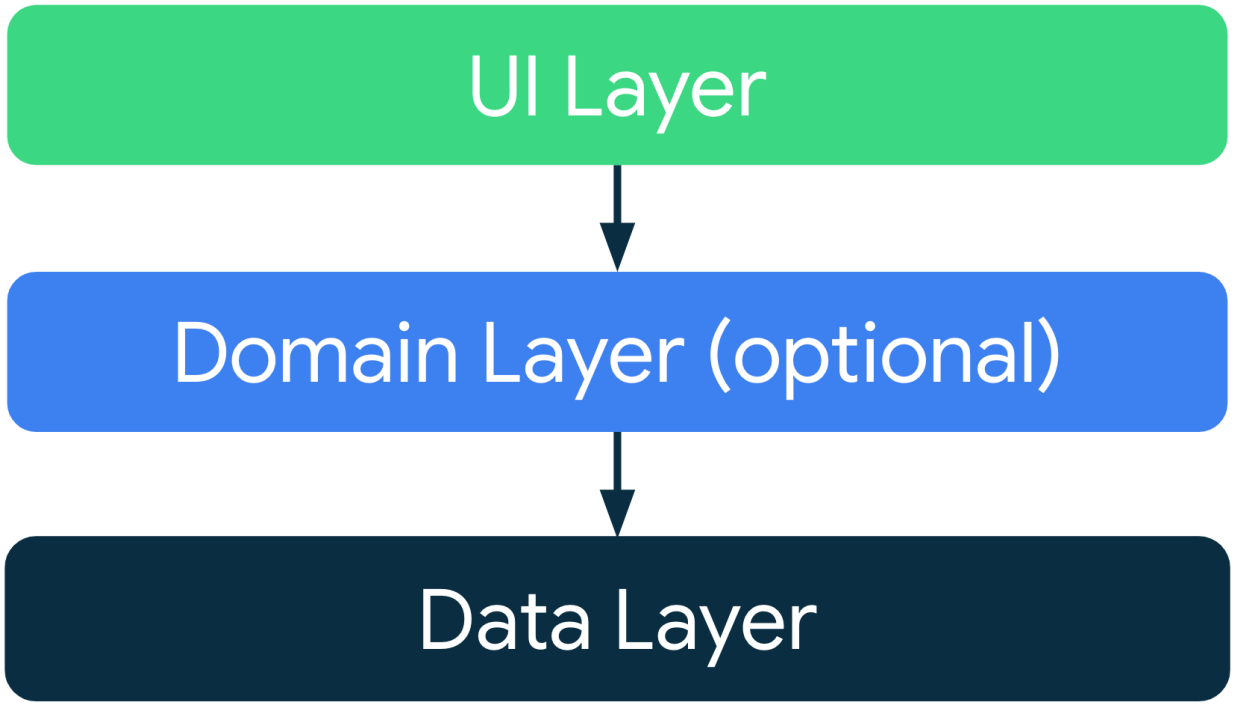


Рисунок 4 – Архитектура приложеня

Слой пользовательского интерфейса (UI Layer) отвечает за отображение данных на экране и обработку взаимодействия с пользователем. В проекте " HulLayer " этот слой реализован с использованием Jetpack Compose для декларативного построения пользовательского интерфейса. Ключевыми компонентами этого слоя являются MainActivity, Composable-функции экранов (например, PlayerScreen, WelcomeScreen) и переиспользуемые UI-элементы. Важной частью UI-слоя также являются классы ViewModel, такие как PlayerViewModel, которые управляют UI-состоянием (например, PlayerUiState), обрабатывают логику, связанную с отображением, и взаимодействуют с нижележащими слоями для получения данных и выполнения команд.

Доменный слой (Domain Layer), хотя и отмечен на схеме Google как опциональный, играет важную роль в " HulLayer ", инкапсулируя сложную

бизнес-логику, независимую от пользовательского интерфейса и деталей реализации слоя данных. Этот слой содержит основные правила и операции приложения. В " HulLayer " к нему относятся PlayerService, отвечающий за логику воспроизведения аудио и управление медиасессией, а также вспомогательные классы-менеджеры, такие как QueueManager (управление очередью воспроизведения) и PlayListManager. Использование доменного слоя способствует лучшей организации кода и его повторному использованию.

Слой данных (Data Layer) отвечает за предоставление и управление всеми данными приложения. Он абстрагирует источники данных от остальной части приложения. В " HulLayer " этот слой включает:

Источники данных (Data Sources): Классы, ответственные за получение данных. Сюда относится MediaResolve, который взаимодействует с Android MediaStore для получения списка аудиофайлов с устройства. Для работы с локальной базой данных используется TrackDao.

Репозитории: Хотя явных классов с названием "Repository" может не быть, роль репозиториев выполняют такие классы, как DatabaseManager, который координирует операции с данными (например, кэширование метаданных треков в базе данных Room).

Модели данных: Класс-сущность TrackInfo, представляющий аудио трек.

База данных: Локальная база данных Room (TrackDatabase) для персистентного хранения информации о треках.

Взаимодействие между слоями обычно следует принципу однонаправленного потока данных: UI Layer передает события в ViewModel, ViewModel может обращаться к Domain Layer для выполнения бизнес-операций или к Data Layer (через репозитории) для получения/сохранения данных. Обновленные данные и состояния передаются обратно в UI Layer через наблюдаемые объекты (например, StateFlow в PlayerViewModel), на которые подписан UI. Такая архитектура, рекомендованная Google LLC, способствует созданию надежных, легко тестируемых и поддерживаемых приложений.

На рисунке 5 показана архитектура приложения.

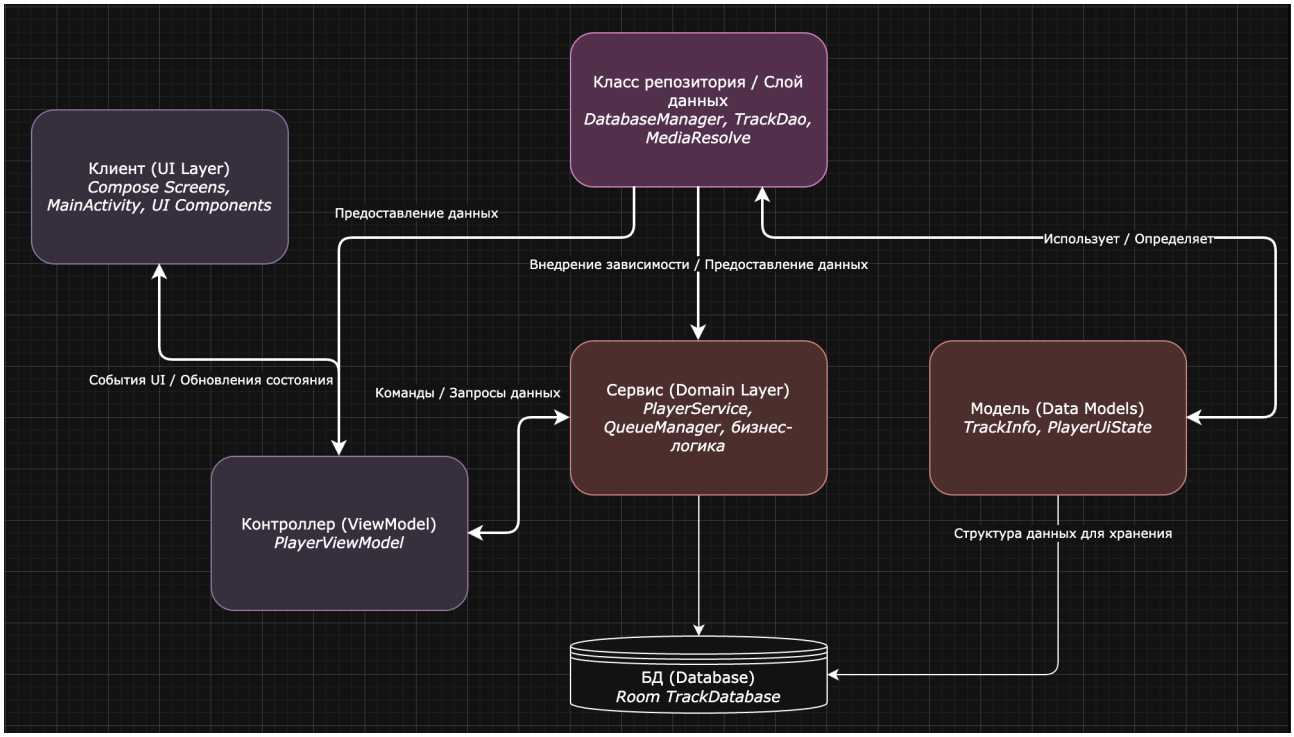


Рисунок 5 — Архитектура приложения

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Описание моделей и структур данных

В приложении " HulLayer " реализована следующая основная модель данных для представления аудио трека, которая хранится в локальной базе данных SQLite с использованием библиотеки Room:

Сущность TrackInfo

Данная сущность (@Entity) представляет собой отдельный аудио трек и хранится в таблице с именем trackList. Она обладает следующими параметрами:

id: Уникальный идентификатор трека.

Тип: Int (Целочисленный).

Ограничения: Первичный ключ (@PrimaryKey), значение генерируется автоматически (autoGenerate = true).

title: Название трека.

Тип: String (Строка).

artist: Имя исполнителя.

Тип: String (Строка).

album: Название альбома.

Тип: String (Строка).

duration: Длительность трека.

Тип: Long (Длинное целое), обычно в миллисекундах.

uri: URI (или путь к файлу), указывающий на расположение аудиофайла на устройстве.

Тип: String (Строка).

artUri: URI (или путь), указывающий на расположение обложки альбома для трека.

Тип: String (Строка).

```kotlin
package com.kirsegisan.hullayer.data.values

30 Usages
data class TrackInfo(
    val id: Long,
    val title: String,
    val artist: String,
    val album: String,
    val duration: Long,
    val uri: String?,
    val cover: String = "none",
) {
    1 Usage
    companion object {
        1 Usage
        val EMPTY = TrackInfo(
            id = -1,
            title = "",
            artist = "",
            album = "",
            duration = 0L,
            uri = "",
        )
    }
}
```

3.2 Описание работы приложения

3.2.1 Классы взаимодействия с данными в локальной базе данных (Room)

Для обеспечения персистентного хранения информации о музыкальных треках на устройстве пользователя в проекте " HulLayer " используется библиотека Room Persistence Library, входящая в состав Android Jetpack. Она предоставляет абстрактный слой над SQLite, упрощая работу с базой данных и обеспечивая проверку SQL-запросов на этапе компиляции. Основными компонентами для взаимодействия с базой данных являются класс самой базы данных, объекты доступа к данным (DAO) и классы-сущности.

Файл TrackDatabase.kt (рисунок 6) определяет основной класс базы данных приложения. Этот класс является абстрактным и наследуется от RoomDatabase.

С помощью аннотации @Database указываются все сущности (entities), которые будут храниться в базе данных (в данном случае, это только TrackInfo), а также версия базы данных. Версия важна для управления миграциями при изменении схемы БД. Класс TrackDatabase также предоставляет абстрактный метод для получения экземпляра TrackDao, через который осуществляются все операции с данными. В классе реализован паттерн Singleton (через companion object и свойство instance) для обеспечения того, чтобы в приложении существовал только один экземпляр базы данных, что предотвращает возможные конфликты доступа и избыточное использование ресурсов. Это соответствует общепринятой практике работы с Room, как описано в примере документа в разделе о клиентской части (пункт "Классы взаимодействия с данными в базе данных").

Файл TrackDao.kt (рисунок 7) представляет собой интерфейс Data Access Object (DAO). DAO — это основной компонент Room, который отвечает за определение методов для доступа к базе данных. Вместо написания SQL-запросов вручную, разработчик объявляет методы с аннотациями Room, такими как @Insert, @Update, @Delete и @Query.

Библиотека Room автоматически генерирует реализацию этих методов во время компиляции. В TrackDao определены методы для выполнения основных CRUD-операций (Create, Read, Update, Delete) над сущностью TrackInfo. Например, метод insertTrack() добавляет новый трек, updateTrack() обновляет существующий, getTracks() возвращает поток (Flow) со списком всех треков, а getTrackById() извлекает трек по его уникальному идентификатору. Использование Flow для методов запроса позволяет асинхронно получать обновления данных в UI при их изменении в базе. Данный подход аналогичен описанию DAO в вашем образце документа.

Файл DatabaseManager.kt (рисунок 8) выступает в роли управляющего компонента или фасада для работы с базой данных. Он инкапсулирует логику взаимодействия с TrackDao и предоставляет более высокоуровневые методы для других частей приложения, таких как сервисы или ViewModel. Например, DatabaseManager содержит методы для инициализации и заполнения базы данных треками, полученными от MediaResolve (метод fillDatabase()), получения списка треков (getInitialTrackList(), getDatabase()), обновления статуса "избранное" для трека (updateTrackFavoriteStatus()), а также удаления всех треков (deleteDatabase()). Операции, требующие длительного выполнения, такие как доступ к диску, выполняются асинхронно с использованием корутин Kotlin (CoroutineScope(Dispatchers.IO)), чтобы не блокировать основной поток приложения. Этот класс можно рассматривать как реализацию паттерна "Репозиторий" для источника данных, хранящегося в Room, что схоже с концепцией репозиториев, описанных в вашем образце.

3.2.2 Классы взаимодействия с системными данными (MediaStore)

Для обнаружения аудиофайлов, хранящихся непосредственно на устройстве пользователя, приложение " HulLayer " взаимодействует с системным сервисом Android — MediaStore. Этот компонент операционной системы индексирует медиафайлы (аудио, видео, изображения) и предоставляет стандартизированный способ доступа к их метаданным через ContentResolver. Взаимодействие с MediaStore в проекте инкапсулировано в классе MediaResolve.

MediaResolve.kt

Файл MediaResolve.kt (рисунок 8) содержит логику для запроса и извлечения информации об аудиофайлах из MediaStore Android.

Метод resolve(): Этот метод является ключевым в классе MediaResolve. Он выполняет запрос к MediaStore.Audio.Media.EXTERNAL_CONTENT_URI с

использованием ContentResolver, полученного из контекста приложения. В ходе выполнения запроса метод итерирует по курсору (Cursor), возвращенному ContentResolver, и для каждой найденной аудиозаписи извлекает необходимые метаданные.

К таким метаданным относятся:

- название трека (MediaStore.Audio.Media.TITLE),
- имя исполнителя (MediaStore.Audio.Media.ARTIST),
- название альбома (MediaStore.Audio.Media.ALBUM),
- длительность трека (MediaStore.Audio.Media.DURATION)
- URI файла (MediaStore.Audio.Media.DATA).

Извлеченная информация для каждого трека преобразуется в объект TrackInfo, который затем используется для заполнения локальной базы данных и отображения в пользовательском интерфейсе. Важно отметить, что для доступа к MediaStore приложению требуется соответствующее разрешение от пользователя (READ_MEDIA_AUDIO на Android 13+).

Рисунок 8 - Запрос и извлечение информации об аудиофайлах из MediaStore

3.2.3 Основная логика приложения (Domain Layer и Services)

Доменный слой в приложении " HulLayer " содержит основную бизнес-логику и сервисы, отвечающие за ключевые функции, такие как управление воспроизведением аудио и очередью треков. Этот слой спроектирован так, чтобы быть независимым от деталей пользовательского интерфейса и источников данных, что повышает модульность и тестируемость приложения.

Метод loadAndSetQueue() (рисунок 9) для загрузки очереди воспроизведения: Этот асинхронный метод отвечает за подготовку и установку очереди воспроизведения в ExoPlayer. Он использует QueueManager для

получения списка MediaItem и передает их плееру, после чего плеер подготавливается к воспроизведению.

```
package com.kirsegisan.hullayer.domain

import ...


2 Usages
@UnstableApi
class PlaybackNotificationProvider(context: Context) : DefaultMediaNotificationProvider(context) {
    override fun getMediaButtons(
        session: MediaSession,
        playerCommands: androidx.media3.common.Player.Commands,
        customLayout: ImmutableList<CommandButton>,
        showPauseButton: Boolean
    ): ImmutableList<CommandButton> {
        return super.getMediaButtons(
            session,
            playerCommands,
            mediaButtonPreferences = customLayout,
            showPauseButton
        )
    }
}
```

Ринсунок 9 – Загрузка очереди воспроизведения

Настройка уведомлений через PlayerNotificationManager (метод setupMediaStyleNotification()) (рисунок 10): для отображения уведомления о воспроизведении и управления плеером из него используется PlayerNotificationManager из библиотеки Media3. В этом методе конфигурируется адаптер для получения метаданных трека (название, исполнитель, обложка) и создается канал уведомлений. Сервис запускается в режиме foreground с этим уведомлением.

Обработка кастомных команд, например, ACTION_TOGGLE_FAVORITE в PlayerSessionCallback: Внутренний класс PlayerSessionCallback переопределяет метод onCustomCommand для обработки специфичных для приложения команд, отправленных медиаконтроллером. Например, команда ACTION_TOGGLE_FAVORITE инициирует обновление статуса "избранное" для трека в базе данных.

```kotlin
package com.kirsegisan.hullayer.domain

import ...

6 Usages
@UnstableApi
class PlaybackService : MediaLibraryService() {

    5 Usages
    private lateinit var player: ExoPlayer
    3 Usages
    private lateinit var session: MediaLibrarySession
    3 Usages
    private lateinit var trackRepository: TrackRepository
    2 Usages
    private lateinit var notificationProvider: PlaybackNotificationProvider

    2 Usages
    private val serviceJob = SupervisorJob()
    2 Usages
    private val serviceScope = CoroutineScope( context = Dispatchers.Main + serviceJob)
    3 Usages
    private lateinit var settingsRepository: SettingDataStoreRepository

    override fun onCreate() {
        super.onCreate()
        val container = (application as HulLayerApplication).container
        trackRepository = container.trackRepository
        settingsRepository = container.settingsRepository

        notificationProvider = PlaybackNotificationProvider( context = this@PlaybackService)
        setMediaNotificationProvider(notificationProvider)

        // Build Player
        player = ExoPlayer.Builder( context = this)
            .setAudioAttributes(AudioAttributes.DEFAULT, handleAudioFocus = true)
            .setHandleAudioBecomingNoisy(true)
            .build()

        player.addListener(object : Player.Listener {
            override fun onMediaItemTransition(mediaItem: MediaItem?, reason: Int) {
                saveCurrentState()
            }
        })

        // Build Session
        session = MediaLibrarySession
            .Builder( service = this, player, callback = LibrarySessionCallback())
            .setSessionActivity(getSingleTopActivity())
            .build()
    }
```

```
1 Usage
private fun getSingleTopActivity(): PendingIntent {
    return PendingIntent.getActivity(
        context = this,
        requestCode = 0,
        Intent( packageContext = this, cls = MainActivity::class.java),
        flags = PendingIntent.FLAG_IMMUTABLE or PendingIntent.FLAG_UPDATE_CURRENT
    )
}

override fun onGetSession(controllerInfo: MediaSession.ControllerInfo)
: MediaLibrarySession { return session }

override fun onDestroy() {
    session.release()
    player.release()
    serviceJob.cancel()
    super.onDestroy()
}

1 Usage
private fun saveCurrentState() {
    serviceScope.launch { this: CoroutineScope
        try {
            val prefs = settingsRepository.settingsPreferencesFlow.first()
            if (!prefs.isSavedState) return@launch
            val currentMediaId = player.currentMediaItem?.mediaId?.toIntOrNull() ?: return@launch
            val track = trackRepository.getTrackById( trackId = currentMediaId)
            settingsRepository.updateCurrentTrack( trackInfo = track)
        } catch (e: Exception) {
            Log.e( tag = "PlaybackService",  msg = "Failed to save state: ${e.message}")
        }
    }
}

1 Usage
private inner class LibrarySessionCallback : MediaLibrarySession.Callback {

    override fun onGetLibraryRoot(
        session: MediaLibrarySession,
        browser: MediaSession.ControllerInfo,
        params: LibraryParams?
    ): ListenableFuture<LibraryResult<MediaItem>> {
        val rootItem = MediaItem.Builder()
            .setMediaId(ROOT_ID)
            .setMediaMetadata(
                MediaMetadata.Builder()
                    .setTitle("Chirro Music")
                    .setIsBrowsable(true)
                    .setIsPlayable(false)
                    .build()
            )
            .build()
        return Futures.immediateFuture( value = LibraryResult.ofItem(rootItem, params))
    }

    override fun onGetChildren(
```

```
        override fun onGetChildren(
            session: MediaLibrarySession,
            browser: MediaSession.ControllerInfo,
            parentId: String,
            page: Int,
            pageSize: Int,
            params: LibraryParams?
        ): ListenableFuture<LibraryResult<ImmutableList<MediaItem>>> {
            if (parentId != ROOT_ID) {
                return Futures.immediateFuture(
                    value = LibraryResult.ofItemList(
                        items = ImmutableList.of(),
                        params
                    )
                )
            }

            // Using Guava Futures to bridge Coroutines -> ListenableFuture
            return Futures.submit(
                Callable {
                    importMediaItems(page, pageSize)
                }
            ) { command -> serviceScope.launch( context = Dispatchers.IO) { command.run() } }
        }

        1 Usage
        private fun importMediaItems(
            page: Int,
            pageSize: Int
        ): LibraryResult<ImmutableList<MediaItem>> {
            return try {
                val tracks = runBlocking { this: CoroutineScope
                    trackRepository.getTracksPaged(page, pageSize)
                }

                val mediaItems = tracks.map { it.toMediaItem() }
                LibraryResult.ofItemList(
                    items = ImmutableList.copyOf( elements = mediaItems),
                    params = null
                )
            } catch (_: Exception) {
                LibraryResult.ofError( errorCode = SessionError.ERROR_UNKNOWN)
            }
        }
    }

    2 Usages
    companion object {
        2 Usages
        private const val ROOT_ID = "root"
    }
}
```

Рисунок 10 – Отображения уведомления о воспроизведении и управления плеером

3.2.4 Классы предоставления данных

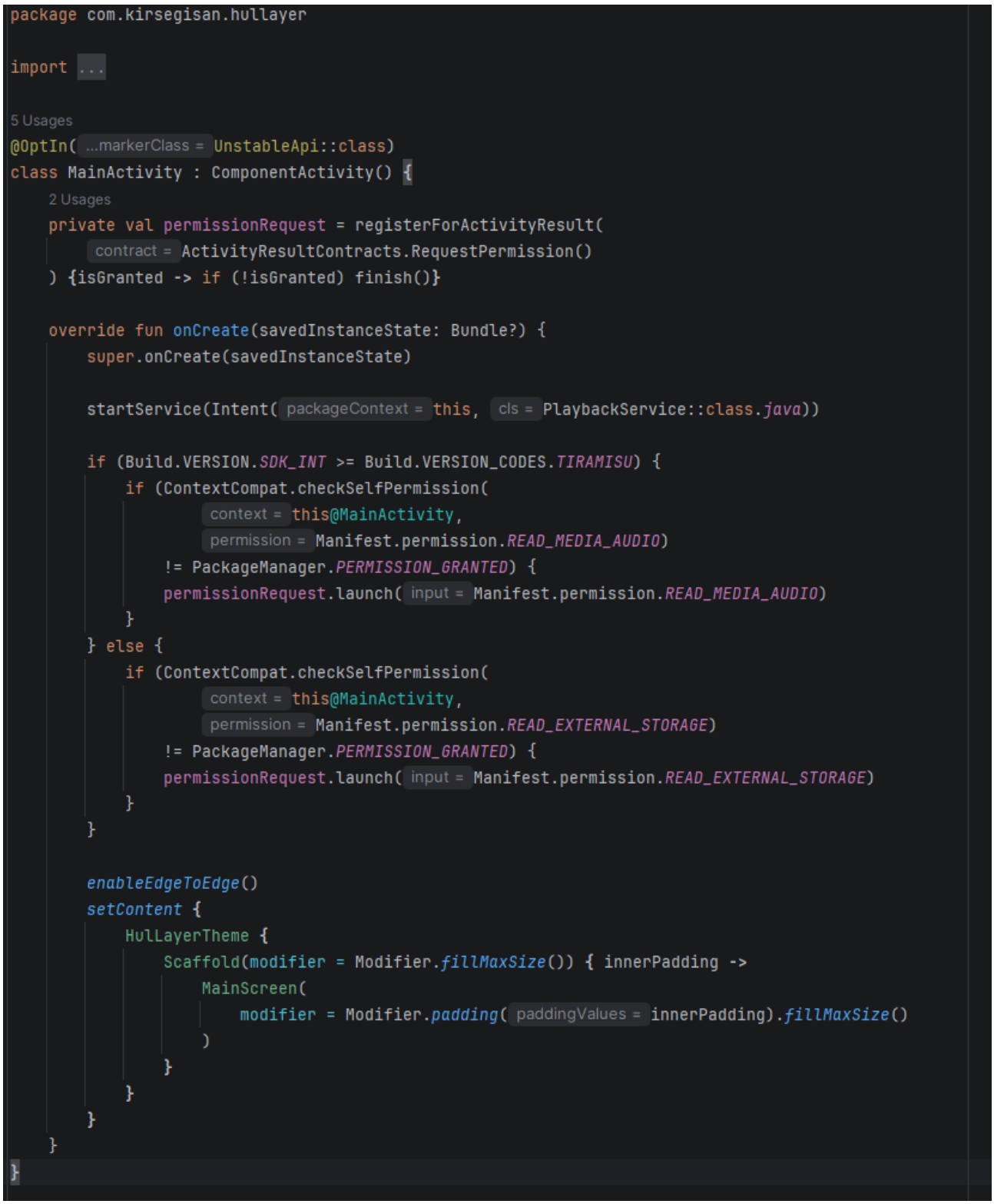
MainActivity.kt

Файл MainActivity.kt (рисунок 12) является основной точкой входа в приложение " HulLayer ".

Фрагменты onCreate() и методы запроса разрешений (requestPermissionLauncher, checkAudioPermissionAndInitialize()): В методе onCreate() происходит первоначальная настройка Activity, включая вызов enableEdgeToEdge() для поддержки отображения контента от края до края экрана и инициализацию пользовательского интерфейса с помощью setContent { HulLayer { MainScreen() } }.

Важной частью инициализации является проверка и запрос необходимых разрешений: POST_NOTIFICATIONS для отображения уведомлений о воспроизведении и READ_MEDIA_AUDIO для доступа к аудиофайлам на устройстве. Для этого используются ActivityResultContracts.RequestPermission() (экземпляры requestPermissionLauncher и requestNotificationPermissionLauncher), которые обрабатывают ответ пользователя на запрос разрешений.

При успешном получении разрешений вызывается метод initializeAppData(), который инициирует загрузку данных и запуск сервиса воспроизведения. Этот подход соответствует стандартной практике настройки главной Activity и управления разрешениями в Android-приложениях, как описано в образце (пункт о классах предоставления данных).



```
package com.kirsegisan.hullayer

import ...

5 Usages
@OptIn( ...markerClass = UnstableApi::class)
class MainActivity : ComponentActivity() {
    2 Usages
    private val permissionRequest = registerForActivityResult(
        contract = ActivityResultContracts.RequestPermission()
    ) {isGranted -> if (!isGranted) finish()}

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        startService(Intent( packageContext = this, cls = PlaybackService::class.java))

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            if (ContextCompat.checkSelfPermission(
                    context = this@MainActivity,
                    permission = Manifest.permission.READ_MEDIA_AUDIO)
                != PackageManager.PERMISSION_GRANTED) {
                permissionRequest.launch( input = Manifest.permission.READ_MEDIA_AUDIO)
            }
        } else {
            if (ContextCompat.checkSelfPermission(
                    context = this@MainActivity,
                    permission = Manifest.permission.READ_EXTERNAL_STORAGE)
                != PackageManager.PERMISSION_GRANTED) {
                permissionRequest.launch( input = Manifest.permission.READ_EXTERNAL_STORAGE)
            }
        }

        enableEdgeToEdge()
        setContent {
            HulLayerTheme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    MainScreen(
                        modifier = Modifier.padding( paddingValues = innerPadding).fillMaxSize()
                    )
                }
            }
        }
    }
}
```

Рисунок 12 – Настройка и инициализация пользовательского интерфейса

TrackInfoScreen.kt

Файл TrackInfoScreen.kt (рисунок 13) содержит Composable-функцию TrackInfoScreen(), предназначенную для отображения детальной информации о текущем воспроизводимом треке. Этот компонент является частью пользовательского интерфейса экрана плеера и отвечает за представление текстовых метаданных аудиозаписи.

```
package com.kirsegisan.hullayer.ui.layer

import ...


5 Usages
@Composable
fun TrackInfoView(
    modifier: Modifier = Modifier,
    trackInfo: TrackInfo
) {
    val name = trackInfo.title.ifEmpty { "name" }
    val artist = trackInfo.artist.ifEmpty { "artist" }
    val album = trackInfo.album.ifEmpty { "album" }
    val duration = trackInfo.duration

    Card { this: ColumnScope
        Row(
            modifier = modifier.padding( all = 8.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween
        ) { this: RowScope
            Column { this: ColumnScope
                Text(
                    text = name,
                    modifier = Modifier.padding(bottom = 4.dp),
                    style = MaterialTheme.typography.bodyLarge,
                    textAlign = TextAlign.Start,
                    softWrap = true,
                    overflow = TextOverflow.Ellipsis,
                    maxLines = 1,
                )

                Text(
                    text = "$artist - $album",
                    style = MaterialTheme.typography.bodyLarge,
                    textAlign = TextAlign.Start,
                    softWrap = true,
                    overflow = TextOverflow.Ellipsis,
                    maxLines = 1,
                    modifier = Modifier.padding(bottom = 4.dp)
                )
            }
        }
    }
}
```

Рисунок 13 - Отображения детальной информации о текущем воспроизводимом треке

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы был успешно спроектирован и разработан прототип мобильного аудиоплеера " HulLayer " для операционной системы Android. Основной целью являлось создание функционального приложения для воспроизведения локальных аудиофайлов, обладающего современным пользовательским интерфейсом и реализующего базовый набор возможностей, ожидаемых от программ такого класса.

Разработанный прототип аудиоплеера " HulLayer " демонстрирует успешную реализацию заявленных функциональных возможностей и соответствует поставленным в начале работы целям. Проект обладает потенциалом для дальнейшего развития, которое может включать добавление таких функций, как создание и управление плейлистами, реализация эквалайзера, расширенные возможности по сортировке и фильтрации медиатеки, а также дальнейшие улучшения пользовательского интерфейса и опыта взаимодействия.

● СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Остроух, А.В. Интеллектуальные информационные системы и технологии: монография / А.В. Остроух, А.Б. Николаев. — СПб: Лань, 2019. — 308 с.
2. Зубкова, Т.М. Технология разработки программного обеспечения: учебное пособие / Т.М. Зубкова. — СПб: Лань, 2019. — 324 с.
3. Smyth, Neil. Android Studio 4.1 Development Essentials – Kotlin Edition / Neil Smyth. – Cary (North Carolina (US)): Payload Media, Inc., 2020. – 1042 с.
4. Documentation for app developers [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/docs, свободный. Язык: Английский.
5. Руководство по языку Kotlin [Электронный ресурс]. –. – Режим доступа: https://kotlinlang.ru/docs, свободный. Язык: Русский.
6. App architecture - Compose [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/develop/ui/compose/architecture, свободный. Язык: Английский.
7. Guide to app architecture [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/topic/architecture, свободный. Язык: Английский.
8. Material Components Overview [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/design/ui/mobile/guides/components/material-overview, свободный. Язык: Английский.
9. ViewModel Overview [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/topic/libraries/architecture/viewmodel, свободный. Язык: Английский.

10. Services overview [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/develop/background-work/services, свободный. Язык: Английский.
11. URI Class Reference [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/reference/java/net/URI, свободный. Язык: Английский.
12. ViewModel and State in Compose Codelab [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/codelabs/basic-android-kotlin-compose-viewmodel-and-state#0, свободный. Язык: Английский.
13. Kotlin coroutines on Android [Электронный ресурс]. –. – Режим доступа: https://developer.android.com/kotlin/coroutines, свободный. Язык: Английский.
14. Build a simple playback app [Электронный ресурс]. . – Режим доступа: https://developer.android.com/media/implement/playback-app, свободный. Язык: Английский.
15. Composing suspending functions - Concurrent using async [Электронный ресурс]. –. – Режим доступа: https://kotlinlang.org/docs/composing-suspending-functions.html#concurrent-using-async, свободный. Язык: Английский.

● ПРИЛОЖЕНИЕ

Репозиторий разрабатываемого прототипа мобильного приложения находится по адресу: https://github.com/Kirsegisan/HulLayer