

**Министерство науки и высшего образования Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ»**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ ДЛЯ ФИЛЬТРАЦИИ  
МУЗЫКАЛЬНЫХ ФАЙЛОВ НА ОСНОВЕ АНАЛИЗА ТЕМПА  
КОМПОЗИЦИЙ**

Автор *Токарева Ульяна Романовна* \_\_\_\_\_

Направление подготовки (специальность) 09.03.04 «Программная  
инженерия»

Квалификация *Бакалавр*

Руководитель *Тропченко А. Ю., проф., д.т.н.* \_\_\_\_\_

**К защите допустить**

Руководитель ОП *Алиев Т. И., проф., д.т.н.* \_\_\_\_\_

«\_\_\_\_\_» 20\_\_\_\_ г.

Санкт-Петербург, 2019 г.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	6
1. Аналитический обзор.....	9
1.1. Теоретическое обоснование.....	9
1.2. Существующие решения.....	10
1.3. Постановка цели и задач.....	16
2. Подготовительный этап.....	18
2.1. Общие требования.....	18
2.2. Сценарии использования приложения .....	18
2.3. Платформа Android .....	21
2.3.1. Общие сведения.....	21
2.3.2. Программирование под Android.....	23
2.4. Выбор инструментов .....	28
2.5. Выводы.....	31
3. Исследование методов и средств определения параметра ВРМ .....	33
3.1. Общие сведения о структуре аудиофайлов .....	33
3.2. Обзор методов определения ВРМ.....	37
3.2.1. Энергетические пики.....	38
3.2.2. Автокорреляционная функция.....	39
3.3. Выводы.....	42
4. Разработка мобильного приложения .....	44
4.1. Архитектура приложения .....	44
4.2. Клиентская часть .....	48
4.2.1. Выбор библиотек.....	50
4.2.2. UML диаграммы .....	51
4.2.3. Особенности реализации.....	57
4.3. Серверная часть .....	59
4.3.1. Выбор инструментария.....	59
4.3.2. Особенности реализации.....	60
4.4. Интерфейс приложения.....	61
4.5. Тестирование и верификация .....	61
ЗАКЛЮЧЕНИЕ.....	63
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	64
ПРИЛОЖЕНИЕ А. Жизненные циклы компонентов Android-приложения ..	66

ПРИЛОЖЕНИЕ Б. Реализация MP3-декодера .....	68
ПРИЛОЖЕНИЕ В. Реализация серверной части .....	71

## ВВЕДЕНИЕ

В последнее время нарастает интерес к здоровому образу жизни, поддержанию хорошей физической формы и занятиям спортом. Все большее количество людей включаются как в специально организованные программы физической активности, так и в самостоятельные тренировки. Среди разных видов спорта бег на выносливость пользуется большой популярностью и у спортсменов, и у обычновенных людей.

Данные спортивной науки и практики тренировок свидетельствуют о том, что повышение качества движения напрямую связано с овладением техникой бега на длительные дистанции, поэтому значительное количество программ спортивной подготовки включает в себя кроссовый бег. Умение поддерживать заданный темп — навык, который позволяет достичь результата на любой дистанции.

Задача поддержания постоянной запланированной скорости часто оказывается достаточно сложной для начинающих бегунов: первые минуты забега проходят в слишком высоком темпе, который не соответствует физической подготовке спортсмена, затем скорость неуклонно падает, что приводит к излишней усталости и травмам. В связи с этим проблема поддержания темпа становится для спортсмена одной из ключевых.

На сегодня существует несколько вариантов ее решения, так, бегуны могут использовать спортивные часы, отслеживающие местоположение с помощью датчика GPS, метроном, музыку с ярко выраженными битами и т.п.. У таких решений есть ряд ограничений: спортивные часы являются дорогостоящим аксессуаром и потому могут быть недоступны для начинающего спортсмена, также они отвлекают внимание спортсмена; метроном оказывается недостаточно удобным в эксплуатации вне спортивного зала; случайное музыкальное сопровождение часто критикуется, поскольку аудиозаписи с различной частотой ударов вынуждают спортсмена постоянно менять свой темп, подстраиваясь под музыку.

Вместе с тем, именно музыка вызывает наибольший эмоциональный отклик у человека, повышает настроение, вызывает желание продолжить тренировку даже при появлении признаков утомления. Поэтому спортсмены прибегают к преодолению существующего недостатка через фильтрацию музыкальных произведений как исходя из личных приоритетов по жанру, так и отталкиваясь от собственных физических возможностей поддержания темпа. Во втором случае важным становится такой параметр музыкального произведения, как BPM (beat per

minute) — количеству ударов в минуту. На сегодняшний день разработаны настольные программы и веб-приложения, которые позволяют узнать BPM определенной композиции, также существуют сервисы, которые подбирают пользователю музыку на основе введенного темпа. Это помогает спортсмену решить поставленную задачу, однако вызывает ряд затруднений: с одной стороны, возникают дополнительные временные затраты на использование настольных программ, на скачивание предложенных сторонним сервисом мелодий, а с другой — зачастую невозможно отобрать мелодии по приоритету жанров, направлений, интересов пользователя.

На наш взгляд, назрела необходимость разработки мобильной программы, которая позволит пользователю по заданным параметрам (темп бега или прогулки) получить набор подходящих композиций из содержания памяти мобильного телефона и запустить данный плейлист.

В связи с этим целью работы является создание мобильного приложения, обеспечивающего спортсменов синхронной музыкальной поддержкой во время тренировки.

Предметную область исследования составляют алгоритмы определения BPM как база, необходимая для фильтрации аудиозаписей по заданному темпу, и разработка приложений для мобильных устройств.

### **Задачи исследования:**

- обзор существующих алгоритмов определения темпа музыкальных композиций;
- исследование вопроса разработки приложений для мобильной платформы Android;
- реализация мобильного приложения, осуществляющего фильтрацию музыкальных файлов на основе анализа темпа композиций.

**Степень теоретической разработанности вопроса** В русскоязычной литературе описание алгоритмов определения темпа встречается достаточно редко.

**Степень прикладной разработанности вопроса** Одна из основных задач реализуемой программы — определение темпа музыкальной композиции — существует уже довольно долго и решается разными алгоритмами. На рынке представлено большое количество web-сервисов и настольных программ, которые опреде-

ляют темп конкретной аудиозаписи, однако для решения поставленной нами задачи они не подходят: пользователю необходимо установить соединение своего мобильного телефона с компьютером, перенести имеющиеся файлы с аудиозаписями на носитель информации компьютера, обработать каждую композицию и самостоятельно составить необходимый для себя плейлист.

**При разработке были использованы:**

- интегрированная среда разработки (IDE) AndroidStudio;
- платформа Android как одна из самых распространенных на рынке мобильных устройств;
- язык программирования Java как официальный язык разработки для платформы Android.

## ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР

### 1.1. Теоретическое обоснование

Интенсивные спортивные занятия помогают человеку поддерживать хорошую физическую форму и обладать положительным настроением. Современные люди выбирают самые разные виды физической нагрузки, в числе наиболее привлекательных спортивных занятий оказывается беговая тренировка и для ее оптимизации спортсмены используют множество разных средств, одним из которых является музыка.

Исследования показывают, что музыка оказывает целый ряд полезных эффектов в области спорта: положительный эмоциональный отклик, улучшение физических показателей и большую физиологическую эффективность [13].

Большое количество исследований влияния музыки на физическую работоспособность было проведено в Университете Брунеля в Англии доцентом кафедры спортивной психологии доктором Костасом Карагеоргис. В одном из исследований [10] изучалось влияние прослушивания музыки на выносливость и психоэмоциональное состояние профессиональных атлетов при выполнении упражнений на беговой дорожке. Выносливость спортсменов была на 18,1% и 19,7% выше при исполнении синхронной мотивированной и нейтральной музыки, соответственно, по сравнению с ее отсутствием. Была установлена закономерность сохранения повышенного настроения у атлетов после тренировок под музыку. Еще один важным результатом данного исследования выступает фиксация факта снижения концентрации молочной кислоты в организме при музыкальном сопровождении тренировочного процесса. Это является крайне важным выводом, поскольку ее накопление в организме во время тренировок и соревновательной деятельности существенно снижает работоспособность и результативность спортивных достижений особенно в циклических видах спорта.

Доктор Костас Карагеоргис установил еще одну закономерность, связанную с характером музыки и его влиянием на продуктивность спортивных занятий: мотивирующая музыка обычно отличается быстрым темпом (больше 120 ударов в минуту), сильным ритмом, вдохновляющей лирикой и возвышенной гармонической структурой. Эти признаки в совокупности позволяют увеличить энергичность и стимулировать общую физическую активность спортсмена. Нейтральная музыка не имеет этих характеристик, но не считается демотивирующей, поскольку даже при ее звучании степень эффективности упражнений повышается. Уче-

ным было также отмечено, что мотивационная составляющая музыки не настолько важна, как способность музыки синхронизироваться с движениями спортсмена.

В аналогичном исследовании [9] получены сходные результаты, что время до физиологического истощения во время бега на беговой дорожке значительно больше при тренировке с синхронной мотивирующей музыкой, чем без нее.

В ходе выполнения другого независимого исследования [12] были получены комментарии респондентов о значимости характеристик музыкальных композиций в контексте спорта. Значимость ритма была обозначена всеми участниками. Типичным ответом участников звучит следующим образом: «В действительности не имеет значения, какая именно музыка играет, если у нее хороший ритм; если она ритмичная, то подходит для тренировки». Респонденты высказали идею, что человек естественным образом склонен замечать ритм и следовать ему.

Скорость или темп музыки были упомянуты 85% участников исследования. Как правило, немного увеличенный относительно спокойного темп оценивался как мотивирующим во время упражнений. респонденты упоминают, что смена темпа с медленного на быстрый приводит к значительному улучшению в усилиях и скорости движений. Поэтому было предложено, чтобы в идеале музыкальный темп совпадал с темпом движения.

Таким образом, музыкальное сопровождение с постоянным ритмом, соответствующим ритму активности, способствует повышению эффективности тренировки. Благодаря музыкальному сопровождению тренировочного процесса спортсмены достигают больших результатов.

На сегодня имеется несколько решений, позволяющих фильтровать музыкальные коллекции на основе анализа BPM. Рассмотрим существующие решения подробнее.

## **1.2. Существующие решения**

<https://jog.fm> jog.fm представляет собой web-сервис для подбора музыкальных композиций для бега, велосипедной езды и прогулки. На настоящий момент это одно из самых популярных решений, рекомендации по его использованию часто встречаются на тематических порталах. Сервис предлагает ввести пользователю свой темп прогулки или бега, BPM или каденс, после чего отбираются наиболее подходящие для введенного критерия композиции.

Найденные результаты можно отсортировать по популярности, темпу, алфавиту или дате добавления. Также есть возможность регистрации, благодаря которой понравившиеся композиции пользователь может отметить для быстрого дальнейшего доступа. Предполагается, что найденные аудиозаписи можно прослушать на самом портале или на стороннем ресурсе (spotify.com), но на момент написания данной работы этот функционал недоступен из-за нестабильной работы сайта. Также отсутствует возможность скачать композиции, соответственно, пользователь не имеет прямой возможности воспользоваться данным сервисом в полной мере, так как остается необходимость получать доступ к коллекциям прибегая к сторонним инструментам.

Интерфейс сервиса представлен на изображениях ниже (рис. 1, 2 и 3).

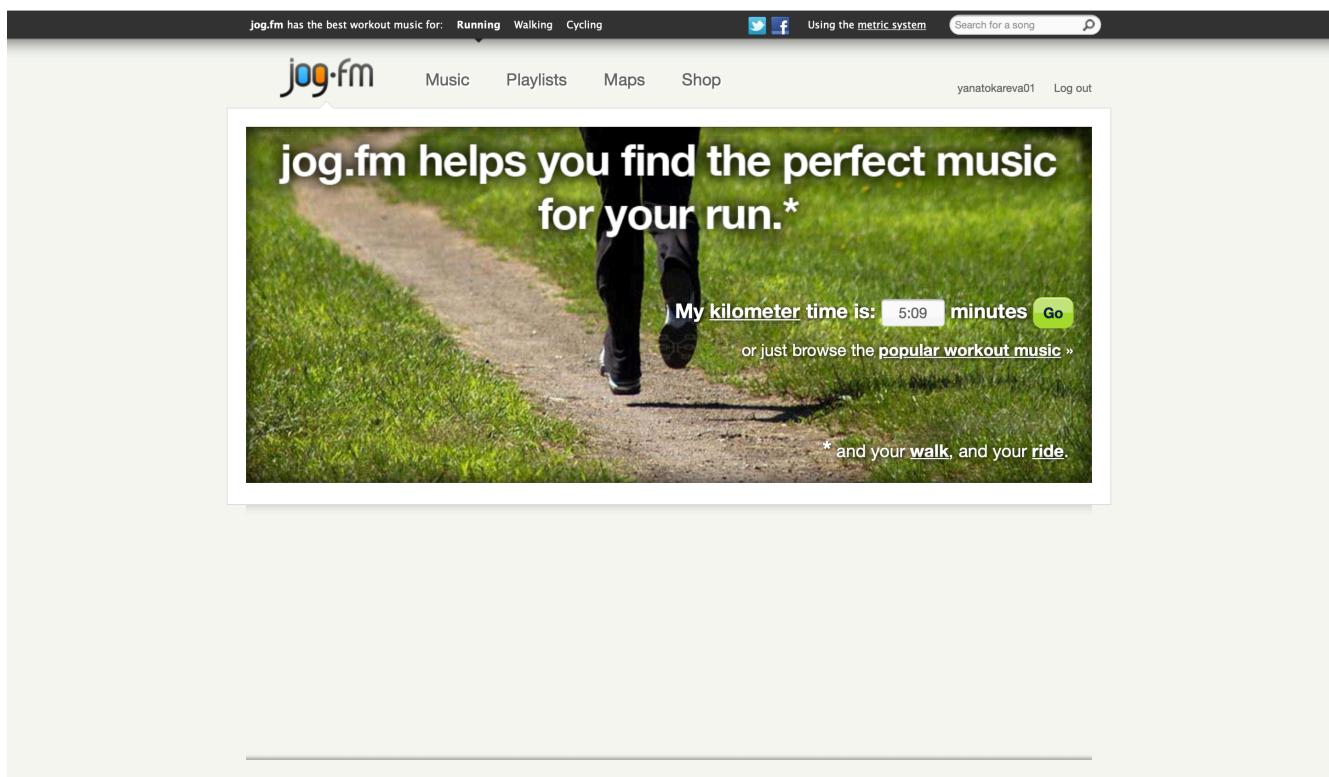


Рисунок 1 – Главная страница

<https://songbpm.com> songbpm.com — простой и малофункциональный сайт от создателей jog.fm, позволяющий определить BPM отдельной композиции. Данный сервис не направлен на использование спортсменами, его функционал сильно ограничен: пользователю предлагается осуществить поиск по названию, артисту и узнать BPM найденных композиций. Музыку нельзя скачать, из нее нельзя сделать плейлист, но можно прослушать на сторонних платформах — spotify и

The screenshot shows the jog.fm website interface. At the top, there's a navigation bar with links for Music, Playlists, Maps, and Shop. On the right, it shows the user's name 'yanatokareva01' and a Log out link. Below the navigation, a sidebar on the left contains a 'Add a song' button, sorting options (Hot, Most Added, BPM, Newest, A to Z), a 'Top Charts' section, a 'Narrow by genre' section, and a 'Search' section with filters for BPM (167, 182) and Pace (5:09). A tip at the bottom of the sidebar says: 'To add a song to a playlist, just click the + icon.' The main content area is titled 'Running songs' and displays a list of songs with their artists, titles, genres, explicit status, purchase links (iTunes, AmazonMP3), and metrics (best for a kilometer, BPM). The first few songs listed are Bastille's 'The Anchor', Lenny Kravitz's 'American Woman', John Anderson's 'Chicken Truck', Eminem's 'Lose Yourself (Soundtrack Version)', Panic! At The Disco's 'High Hopes', Drake's 'In My Feelings', Depeche Mode's 'A Photograph of You', and Pharrell Williams' 'Happy (from "Despicable Me 2")'.

Рисунок 2 – Поиск по композициям

This screenshot shows the jog.fm website again, but this time the search results are for 'Running playlists'. The layout is similar to Figure 2, with the same top navigation and sidebar. The main content area is titled 'Running playlists' and lists various running playlists created by users. Each entry includes the playlist title, artist, duration, number of songs, and sharing options (Facebook, Twitter, Spotify). The playlists listed include 'RunCaseyRun', 'Carebear968', 'Iaevuslevus', 'k8t', 'burntcamper', 'Jenajo', 'lafutura', and 'kate8927'. The sidebar on the left shows the user has selected the 'Running' category under 'My Playlists' and 'Narrow by exercise'.

Рисунок 3 – Поиск по плейлистам

amazon. Интерфейс прост и понятен, присутствует только одно поле ввода, нет ничего лишнего, отсутствует регистрация и реклама. Данный сайт предназначен только для тех случаев, когда нужно узнать, подходит ли какая-то конкретная ком-

позиция для бега; для формирования плейлистов сервис не приспособлен. Интерфейс songbpm.com представлен на рисунке 4.

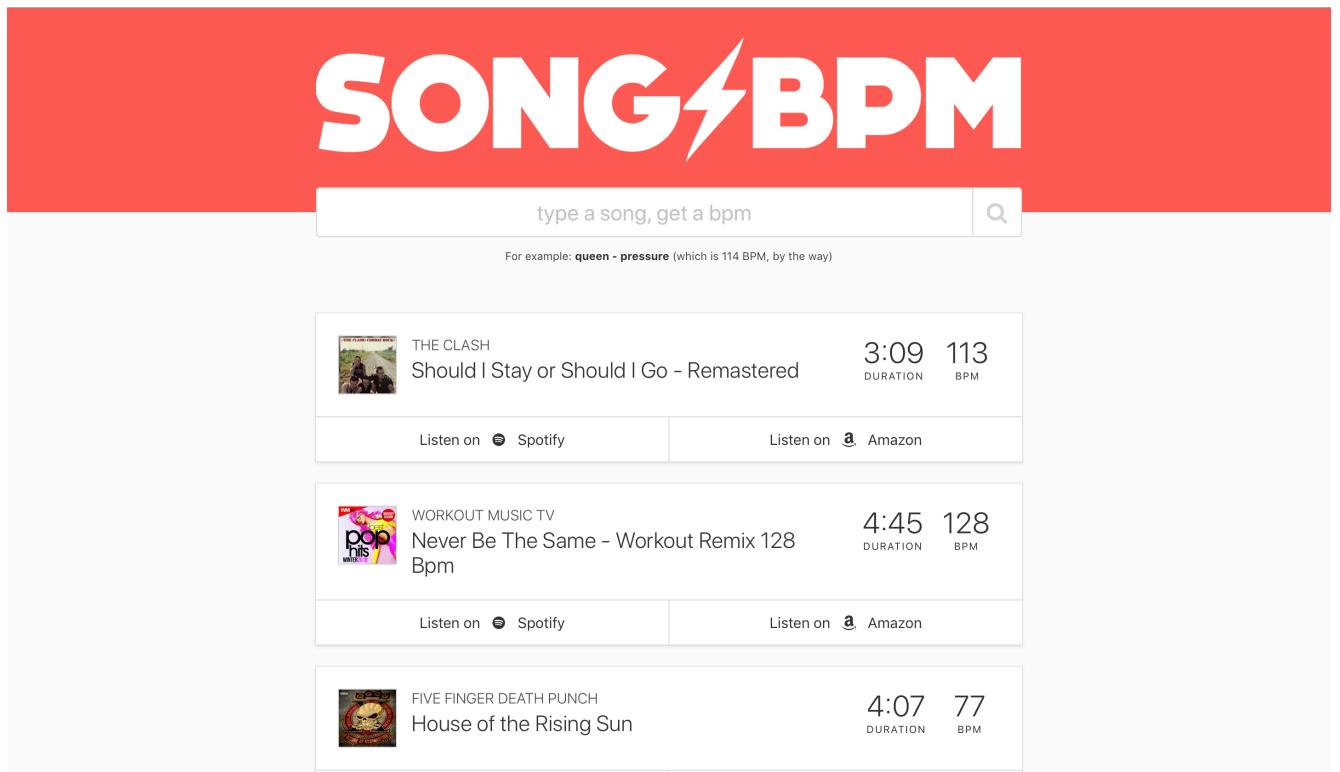


Рисунок 4 – Интерфейс songbpm.com

**RockMyRun** RockMyRun — мобильное приложение для платформ Android и iOS, предоставляющее готовые музыкальные миксы с известным BPM. Авторы приложения заявляют, что миксы созданы специально для спорта лучшими музыкантами.

Приложение включает в себя большое количество функций, например, подбор по жанру, длительности, BPM с возможностью добавления композиций в избранное для дальнейшего быстрого доступа. Функциональность не ограничивается бегом, есть возможность найти музыку для прогулок и других упражнений. Однако воспроизведение музыки возможно только в режиме «онлайн», что требует постоянного Интернет-соединения. Во время воспроизведения часто музыку прерывает аудиореклама, для отключения которой необходима оплата, так как приложение лишь условно бесплатное. В платной версии присутствуют дополнительные функции (возможность изменять темп музыки в соответствии с введенным BPM, либо через синхронизацию с темпом спортсмена) и отсутствует реклама. Среди существенных недостатков приложения — наличие только англоязычного

интерфейса, нестабильная работа приложения, частые сбои и ошибки. Интерфейс приложения представлен на рисунке 5.

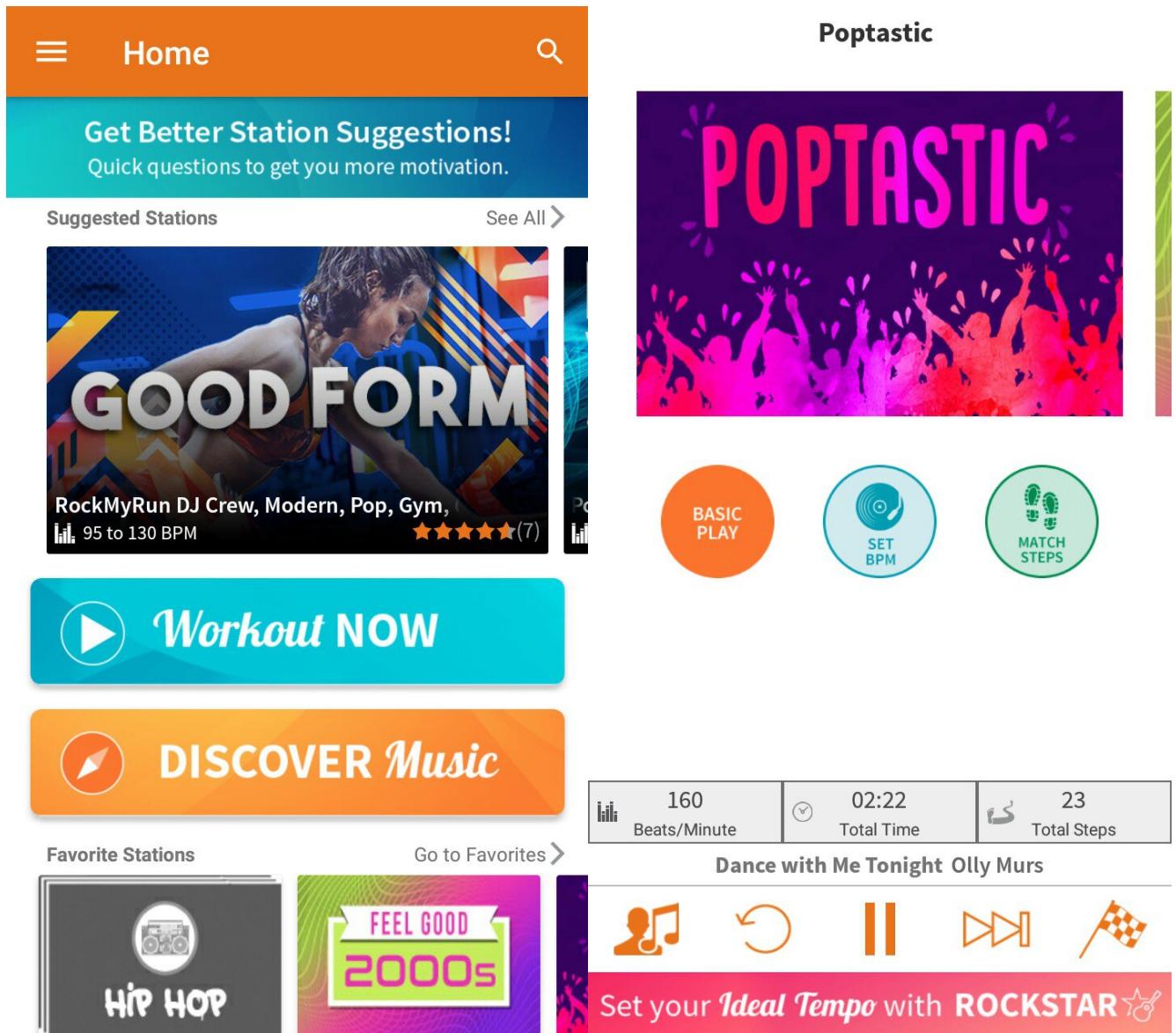


Рисунок 5 – Интерфейс RockMyRun

**Hit Your Run** Hit Your Run — мобильное приложение для платформы Android, которое повторяет функционал RockMyRun, но делает его более элегантным и простым в использовании. Пользователю предлагается выбрать уже готовый набор музыки для прослушивания опираясь на значение BPM. Пользователи отмечают хорошее качество подобранных композиций в плейлистиках. Наборы сгруппированы по темпу (медленная, средняя и быстрая скорость), а также по жанру. Прослушивание музыки доступно «онлайн», но данные кэшируются, что позволяет запускать повторное воспроизведение без подключения к сети Интернет. Кэш-

память легко очищается из приложения, что важно, учитывая небольшой объем памяти мобильных телефонов.

У приложения есть несколько существенных недостатков. Одним из них является полное отсутствие контроля проигрываемой музыки: песни нельзя переключать, ставить на паузу, пропускать и т.п. Для того, чтобы приостановить воспроизведение, что часто необходимо во время тренировки, нужно либо заново найти проигрываемый плейлист в приложении, либо остановить само приложение. Количество плейлистов для воспроизведения сильно ограничено и нет возможности просмотреть их содержимое, т.е. пользователь, запуская какой-либо плейлист, опирается на его изображение, название и BPM. При запуске плейлиста на экране появляется реклама. Так же как и у всех ранее рассмотренных решений в приложении отсутствует русский язык. Интерфейс приложения представлен на рисунке 6.

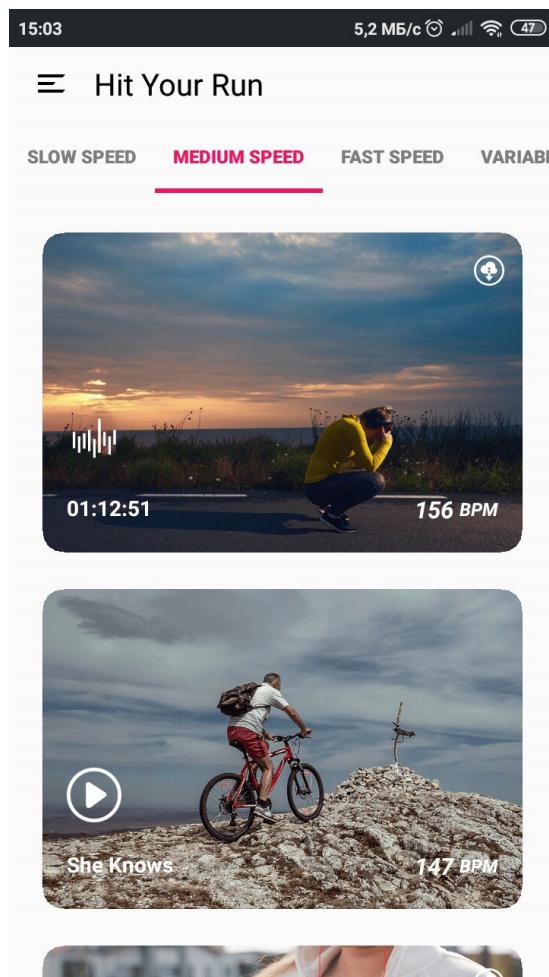


Рисунок 6 – Интерфейс Hit Your Run

Итак, рассмотрено четыре существующих решения для подбора музыкального сопровождения для выполнения спортивных упражнений: два web-сервиса

и два мобильных приложения. Все решения реализуют схожий функционал, кроме <https://songbpm.com>, где нет возможности подобрать музыку на основе BPM, а есть лишь возможность узнать BPM заранее известной композиции. Сравнение сервисов приведено в таблице 1.

Таблица 1 – Сравнение существующих решений

	<a href="https://jog.fm">https://jog.fm</a>	<a href="https://songbpm.com">https://songbpm.com</a>	RockMyRun	Hit Your Run
Платформа	web	web	Android, iOS	Android
Является бесплатным	Да	Да	Нет	Нет
Прослушивание музыки	Да	Нет	Да	Да
Поддержка русского языка	Нет	Нет	Нет	Нет
Использование собственной коллекции	Да	Да	Нет	Нет

### 1.3. Постановка цели и задач

У всех рассмотренных решений есть общая черта: либо авторами предлагаются уже готовые плейлисты, либо пользователю необходимо обрабатывать каждую желаемую композицию отдельно (определять BPM, скачивать на стороннем ресурсе, переносить на устройство, которое будет использоваться для воспроизведения во время занятия спортом, добавлять в свой плейлист). Выполнение всех этих действий трудоемко, так как необходимо совершать большое количество дополнительной работы.

В то же время известно, что в качестве устройств воспроизведения музыкальных композиций современные спортсмены используют портативные устройства, такие как мобильные телефоны и музыкальные плееры. Однако наиболее универсальным гаджетом на сегодня является переносное средство связи, атрибут повседневной жизни — мобильный телефон.

Агентства We Are Social и Hootsuite провели исследование [11] и установили, что на данный момент пользователями мобильных телефонов являются 5 миллиардов людей, большинство из которых используют смартфоны. Множество спортсменов берут с собой телефон на пробежку для того, чтобы использовать возможности музыки для оптимизации физических усилий. Известно, что

в смартфонах люди хранят собственные музыкальные коллекции, включающие наиболее предпочтаемые музыкальные композиции, отличающиеся жанром, манерой исполнения, стилем звучания. Данные музыкальные коллекции используются для прослушивания в общественном транспорте, на прогулках, на отдыхе, на работе и т.д..

Соответственно, хранящаяся музыка часто разнообразна, поэтому ее использование для бега не всегда возможно. Исходя из этого, возможность современных мобильных устройств самостоятельно, с помощью установленного программного обеспечения или через сеть Интернет, обрабатывать и отбирать музыкальные композиции с учетом характеристики ВРМ позволила бы существенно помочь людям получать более высокие результаты от тренировочного процесса.

На наш взгляд, назрела необходимость разработки мобильной программы, которая позволит пользователю по заданным параметрам (желаемый темп бега или прогулки) получить набор подходящих композиций из локальной коллекции мобильного телефона и воспроизвести полученный плейлист.

В связи с этим целью работы является создание мобильного приложения, обеспечивающего спортсменов синхронной музыкальной поддержкой во время тренировки.

Предметную область исследования составляют алгоритмы определения ВРМ как база, необходимая для фильтрации аудиозаписей по заданному темпу или каденсу и разработка приложения для мобильных устройств.

### **Задачи исследования:**

- обзор существующих алгоритмов определения темпа музыкальных композиций;
- исследование вопроса разработки приложений для мобильной платформы Android;
- реализация мобильного приложения, осуществляющего фильтрацию музыкальных файлов на основе анализа темпа композиций.

## ГЛАВА 2. ПОДГОТОВИТЕЛЬНЫЙ ЭТАП

В данной главе рассмотрены исходные данные для разработки мобильного приложения и сформулированы требования, на основании которых разработаны сценарии пользования приложением и выбрана инструментальная база.

### 2.1. Общие требования

В результате аналитического обзора было установлено, что в настоящее время нет приложений для мобильных платформ, в частности Android, позволяющих вычислять темп музыкальных файлов и создавать отфильтрованные по темпу списки воспроизведения. Для устранения указанного пробела предлагается разработать мобильное приложение для платформы Android, удовлетворяющее следующим требованиям:

- а) Приложение должно предоставлять список всех аудиофайлов на устройстве;
- б) Приложение должно определять BPM как отдельной композиции, так и всех сразу;
- в) В приложении должна присутствовать возможность ввода темпа спортсмена;
- г) После ввода темпа приложение должно формировать и сохранять плейлист с музыкальными композициями, отфильтрованными по параметру BPM;
- д) В приложении должна присутствовать возможность удалить созданный плейлист;
- е) В приложении должна присутствовать возможность открыть плейлист для воспроизведения.

### 2.2. Сценарии использования приложения

Исходя из описанных выше требований, разработаем и опишем сценарии использования приложения. Диаграмма сценариев использования представлена на рисунке 7.

**Просмотр главного экрана** Под главным экраном подразумевается экран, на который пользователь попадает при запуске приложения. На нем располагается форма создания нового плейлиста, недавно созданные плейлисты и другая полезная информация. При открытии данного экрана система будет показывать диалог,

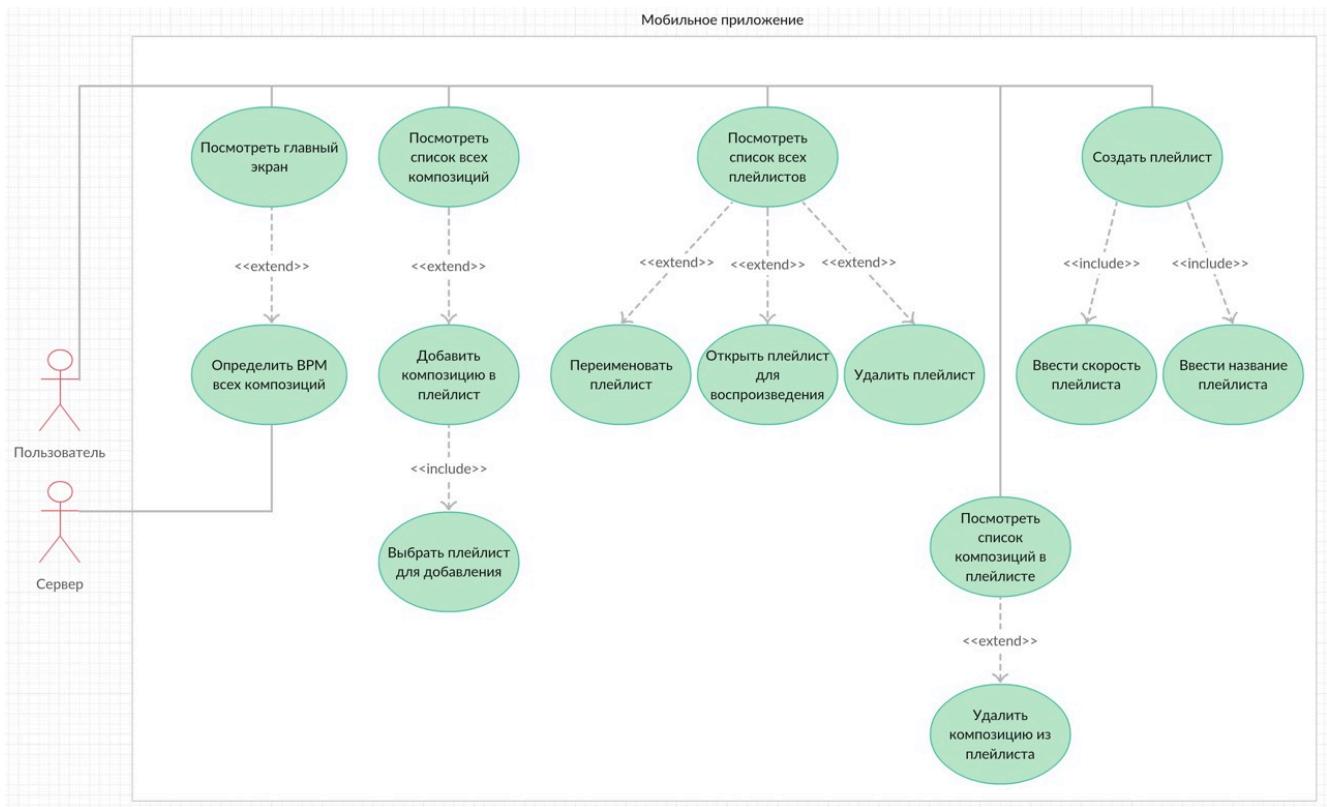


Рисунок 7 – Use-case диаграмма

оповещающий пользователя о количестве музыкальных файлов без определенного BPM и предлагающий определить их. Диалог можно закрыть, при этом он должен содержать компонент checkbox, при нажатии на который диалог больше не будет отображаться при каждом последующем открытии главного экрана.

Так как данный прецедент является точкой входа в мобильное приложение, отобразим его в виде диаграммы деятельности на рис. 8

**Создание плейлиста** Данный прецедент является одним из ключевых прецедентов разрабатываемого мобильного приложения. Его поток начинается с главного экрана, где пользователь вводит желаемое время бега и нажимает на кнопку, которая ведет к созданию плейлиста. Далее, на экране отобразится диалог, предлагающий пользователю ввести название создаваемого плейлиста. После создания плейлиста пользователь увидит экран, связанный с прецедентом «Просмотр списка композиций плейлиста».

**Просмотр списка композиций плейлиста** Прецедент просмотр плейлиста позволяет пользователю увидеть экран, на котором отображаются все музыкальные композиции плейлиста и его название. Дополнительные возможности: удаление

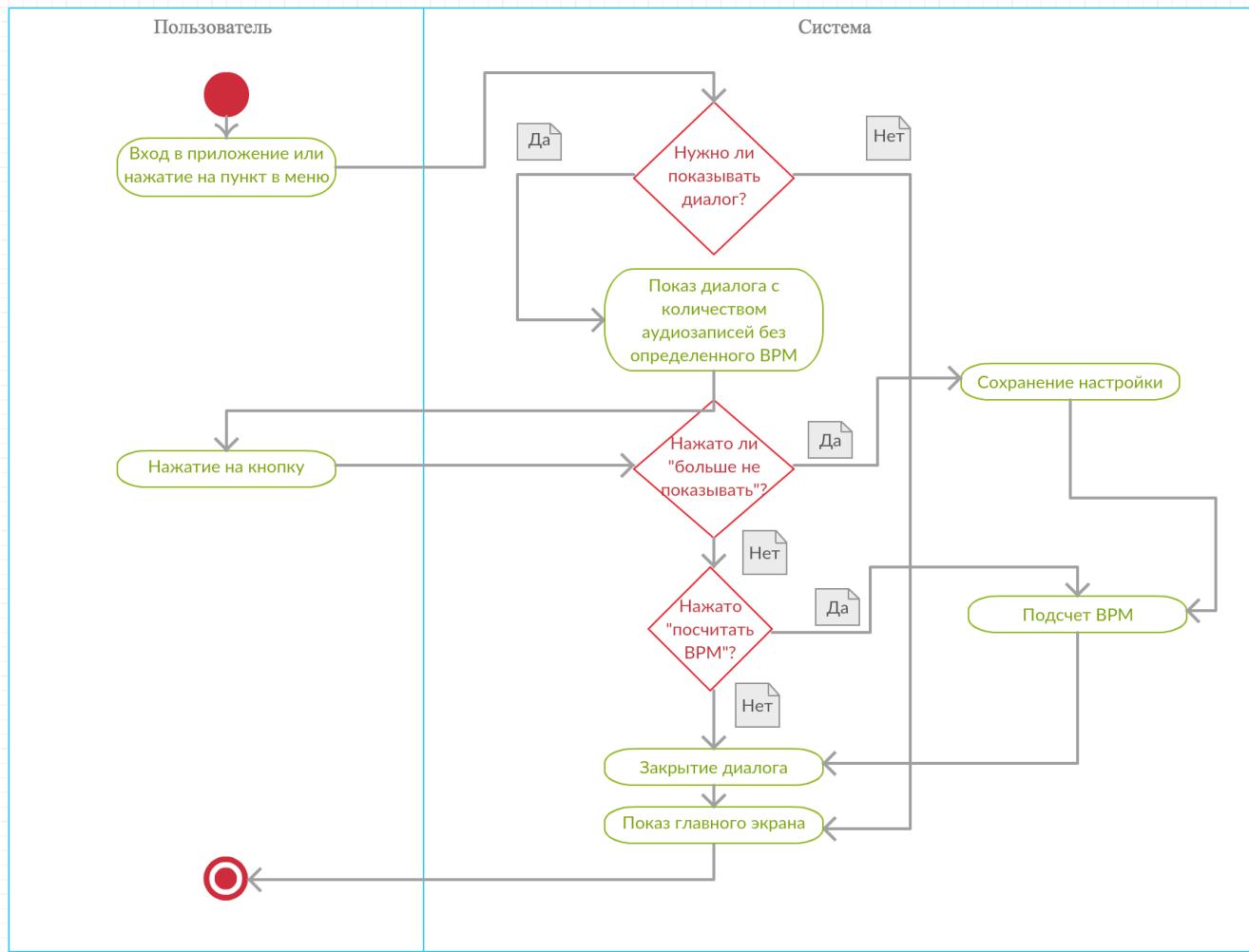


Рисунок 8 – Диаграмма деятельности ”Просмотр главного экрана”

композицию из плейлиста; меню, с помощью которого можно удалить плейлист или открыть его для прослушивания.

**Просмотр списка всех музыкальных композиций** Одной из возможностей пользователя будет являться просмотр всей локальной музыкальной библиотеки. Композиции будут представлены в виде списка. Каждый элемент списка будет содержать обложку альбома, название, имя артиста и ВРМ композиции. Также, элемент списка будет содержать кнопку, при нажатии на которую будет отображаться выпадающее меню, позволяющее определить ВРМ композиции.

**Просмотр списка всех плейлистов** Помимо просмотра библиотеки, в приложении будет присутствовать экран, отображающий список всех плейлистов пользователя. Каждый элемент списка будет содержать обложку, название, ВРМ и скорость, для которой данный ВРМ предназначен. Пользователь будет иметь возмож-

ность переименовать или удалить плейлист с помощью дополнительного меню, которое будет отображаться в виде выпадающего списка при нажатии на кнопку. При нажатии на сам элемент списка, будет открыт экран, связанный с прецедентом «Просмотр списка композиций плейлиста»

Очевидно, что для успешной реализации проекта необходимо исследовать особенности выбранной мобильной платформы для обоснованного выбора методов и средств решения поставленных задач. Рассмотрим мобильную платформу Android подробнее.

## 2.3. Платформа Android

### 2.3.1. Общие сведения

Android – мобильная платформа, разработанная корпорацией Google [6]. Основана на модифицированном ядре Linux и других свободных проектах. Ориентирована в первую очередь на мобильные устройства с сенсорным управлением. В общем виде структура платформы представлена на рис. 9. Рассмотрим каждый из уровней подробнее.

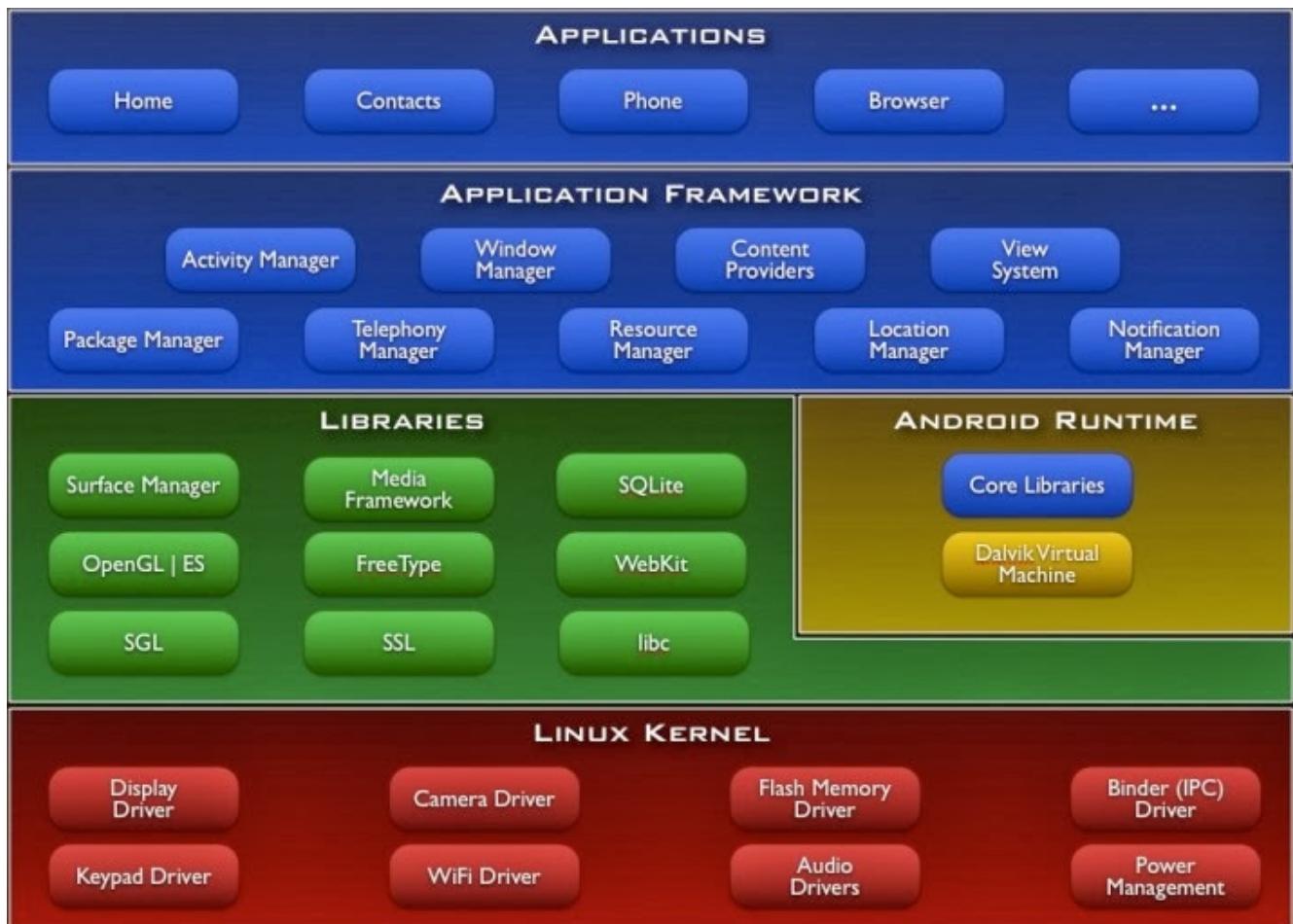


Рисунок 9 – Архитектура Android

**Kernel layer** На нижнем уровне расположено модифицированное ядро Linux, реализующее базовые функциональные возможности устройств. Однако ни пользователи, ни разработчики не взаимодействуют с данным уровнем напрямую.

Ядро включает в себя следующие компоненты и функции:

- планировщик процессов (Process Scheduling);
- управление памятью (Memory Management Programs);
- реализация сетевого стека (Network Stack);
- настройки безопасности (Security Settings);
- управление питанием (Power Management Software);
- драйверы устройств (Hardware Drivers).

Для повышения уровня абстракции при работе с широким спектром устройством использована прослойка Hardware Abstraction Layer (HAL), позволяющая разработчикам взаимодействовать с оборудованием независимо от типа устройств.

**Native Libraries Layer (NLL)** Так как Android нацелен в первую очередь на рынок мобильных устройств с низкой производительной и малым объемом оперативной памяти, то все ресурсоемкие функции были вынесены на отдельный уровень – Native Libraries Layer, где библиотеки реализованы на языках C/C++ и оптимизированы для конкретной платформы. На данном уровне реализованы следующие библиотеки:

- libc;
- libm;
- SGL (2D Library);
- WebKit;
- FreeType;
- OpenGL|ES (3D Library);
- SQLite;
- Media Framework (запись и воспроизведение различных аудио и видео файлов);
- Surface Manager (отрисовка окон на экране).

**Android Runtime** На одной горизонтали с Native Libraries находится Android Runtime (среда выполнения приложений), состоящая из набора библиотек ядра Java (Core) и виртуальной машины Dalvik.

Виртуальной машине Dalvik присущи следующие свойства:

- использует регистры;
- оптимизирована для работы с малым объемом оперативной памяти;
- изначально спроектирована с возможностью запуска множества экземпляров ВМ одновременно;
- использует средства системы для изоляции процессов, управления памятью и поддержки многопоточности.

Каждое приложение запускается в собственном экземпляре виртуальной машины Dalvik, поэтому все процессы изолированы от ОС и друг от друга.

**Application Framework Layer** Application Framework Layer (Java API Framework) включает в себя набор Java-библиотек, предоставляющих доступ пользовательским приложениям к нижестоящим уровням посредством абстракций. Данный уровень включает в себя следующие компоненты:

- Activity Manager (управление жизненным циклом приложения);
- Content Providers (обмен информацией между приложениями);
- Telephony Manager (управление голосовыми вызовами);
- Location Manager (навигация);
- Resource Manager (управление различными ресурсами).

**Application Layer** Application layer является вершиной представленной структуры. Взаимодействие пользователя с мобильной платформой чаще всего ограничивается именно этим уровнем. В него входят такие базовые функции как звонки, веб-серфинг, менеджер контактов и т. д.

### 2.3.2. Программирование под Android

**Манифест приложения** В корневой папке каждого приложения должен находиться файл `AndroidManifest.xml`. В файле манифеста содержится важная информация о приложении, которая требуется системе Android. Только получив эту информацию, система может выполнить какой-либо код приложения. Среди прочего файл манифеста выполняет следующие действия:

- задает имя пакета Java для приложения, являющееся уникальным идентификатором;
- описывает компоненты приложения — операции, службы, приемники широковещательных сообщений и поставщиков контента, из которых состоит приложение;
- содержит имена классов, которые реализуют каждый компонент, и публикует их возможности (указывает, например, какие сообщения Intent они могут принимать). На основании этих деклараций система Android может определить, из каких компонентов состоит приложение и при каких условиях их можно запускать;
- определяет в каких процессах будут размещаться компоненты приложения;
- объявляет, какие разрешения должны быть выданы приложению, чтобы приложение могло получить доступ к защищенным частям API-интерфейса и взаимодействовать с другими приложениями;
- объявляет разрешения, требуемые для взаимодействия с компонентами данного приложения;
- содержит список классов Instrumentation, которые при выполнении приложения предоставляют сведения о профиле и прочую информацию. Эти объявления присутствуют в файле манифеста только во время разработки и отладки приложения и удаляются перед его публикацией;
- объявляет минимальный уровень API-интерфейса Android, требуемый приложению;
- содержит список библиотек, с которыми должно быть связано приложение.

**Activity** Класс Activity — важнейший класс Android API. Способ запуска Activity является фундаментальной частью модели приложений для данной платформы. В отличие от парадигм программирования, в которых приложения запускаются методом main(), система Android инициирует код в экземпляре Activity, вызывая конкретные методы, которые соответствуют определенным этапам его жизненного цикла.

Мобильные приложения отличаются от десктопных тем, что взаимодействие с пользователем не всегда начинается в одном и том же месте. Например, при открытии приложения почты на экране отобразится список писем. Однако, если при использовании другого приложения, для социальной сети, происходит

запуск почтового приложения, вы можете перейти сразу непосредственно к экрану написания письма.

Класс Activity предназначен для облегчения реализации этой парадигмы. Когда одно приложение вызывает другое,зывающее приложение вызывает Activity в другом приложении, а не приложение как единое целое. Таким образом, Activity служит точкой входа для взаимодействия приложения с пользователем.

Activity предоставляет окно, в котором приложение рисует свой пользовательский интерфейс. Это окно обычно заполняет экран, но может быть меньше экрана и плавать поверх других окон. Как правило, одна реализация класса Activity представляет один экран в приложении.

Большинство приложений содержат несколько экранов, что означает, что они содержат несколько Activity. Как правило, одно Activity в приложении указывается в качестве основного, которое является первым экраном, появляющимся при запуске приложения пользователем. Каждое Activity может затем запустить другое Activity для выполнения различных действий.

Чтобы использовать Activity в приложении, необходимо зарегистрировать информацию о них в манифесте приложения и надлежащим образом управлять жизненными циклами действий.

Жизненный цикл класса Activity изображен на рис. А.1

**Фрагменты** Организация приложения на основе нескольких Activity не всегда может быть оптимальной. Мир ОС Android довольно сильно фрагментирован и состоит из многих устройств. И если для мобильных аппаратов с небольшими экранами взаимодействие между разными Activity выглядит довольно неплохо, то на больших экранах — планшетах, телевизорах окна Activity смотрелись бы неорганично из-за большого размера экрана. Собственно поэтому и появилась концепция фрагментов. Пример использования фрагментов представлен на рис. 10.

Фрагмент существует в контексте Activity и имеет свой жизненный цикл, вне Activity обособлено он существовать не может. Каждое Activity может иметь несколько фрагментов.

Каждый класс фрагмента наследуется от базового класса Fragment и имеет свой жизненный цикл, состоящий из этапов, отображенных на рис. А.2

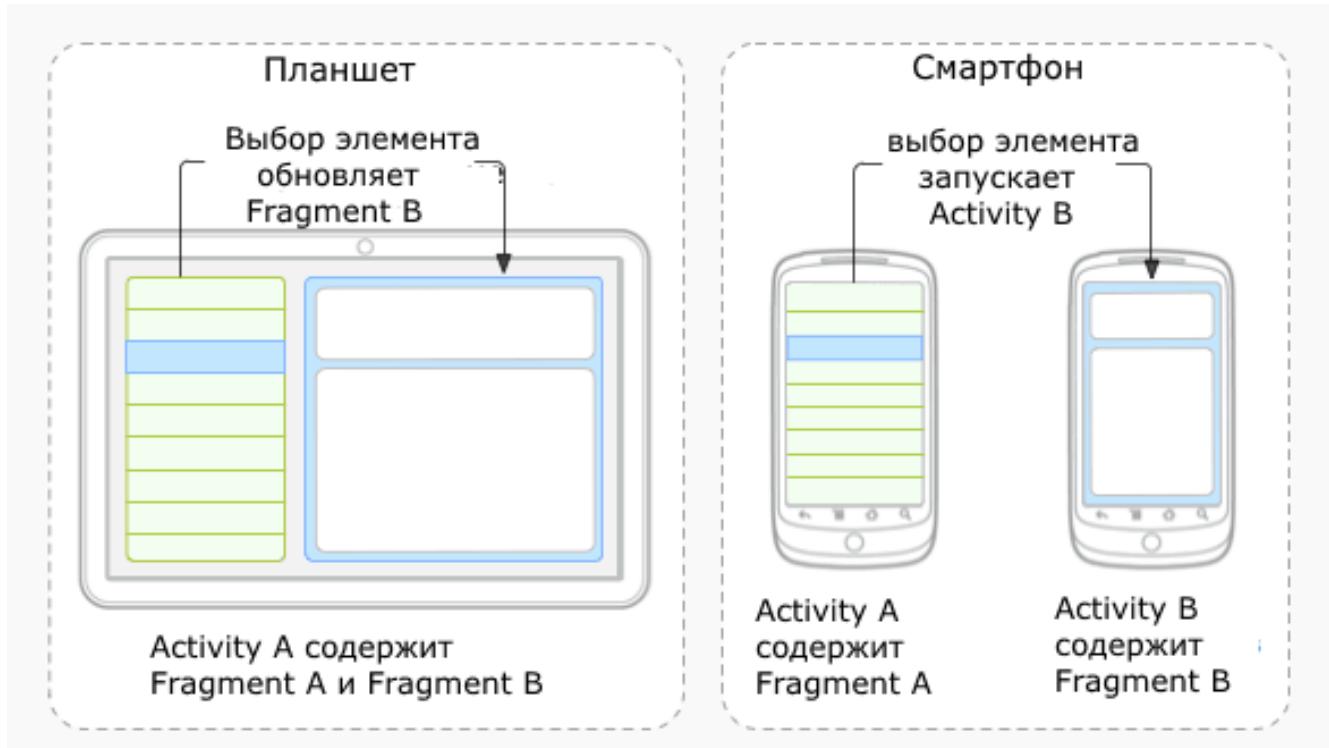


Рисунок 10 – Использование фрагментов

**Intents** Intent — это объект обмена сообщениями, который используется для коммуникации между приложениями или внутри него. Приложение может как отправлять, так и получать объекты типа Intent, которые в основном используются в трех ситуациях:

— **Для запуска Activity**

Для запуска нового экземпляра Activity необходимо передать объект Intent методу `startActivity()`. Объект Intent описывает операцию, которую требуется запустить, а также содержит все остальные необходимые данные;

— **Для запуска службы**

Службу можно запустить для выполнения однократного действия (например, чтобы загрузить файл), передав объект Intent методу `startService()`. Объект Intent описывает службу, которую требуется запустить, а также содержит все остальные необходимые данные;

— **Для рассылки широковещательных сообщений**

Широковещательное сообщение — сообщение, которое может принять любое приложение. Система выдает различные широковещательные сообщения о системных событиях, например, когда система загружается или устройство начинает заряжаться. Для выдачи широковещательных сооб-

щений другим приложениям необходимо передать объект Intent методу sendBroadcast(), sendOrderedBroadcast() или sendStickyBroadcast().

Существует два типа объектов Intent — явные и неявные. Явные объекты Intent указывают компонент, который требуется запустить, по имени (полное имя класса) и обычно используются для запуска компонента из собственного приложения, поскольку разработчику известно имя класса операции или службы, которую необходимо запустить. Например, можно запустить новую операцию в ответ на действие пользователя или запустить службу, чтобы загрузить файл в фоновом режиме.

Неявные объекты Intent не содержат имени конкретного компонента. Вместо этого они в целом объявляют действие, которое требуется выполнить, что дает возможность компоненту из другого приложения обработать этот запрос. Например, если требуется показать пользователю место на карте, то с помощью неявного объекта Intent можно запросить, чтобы это сделало другое приложение, в котором такая возможность предусмотрена. Пример работы неявного объекта Intent изображен на рисунке 11.

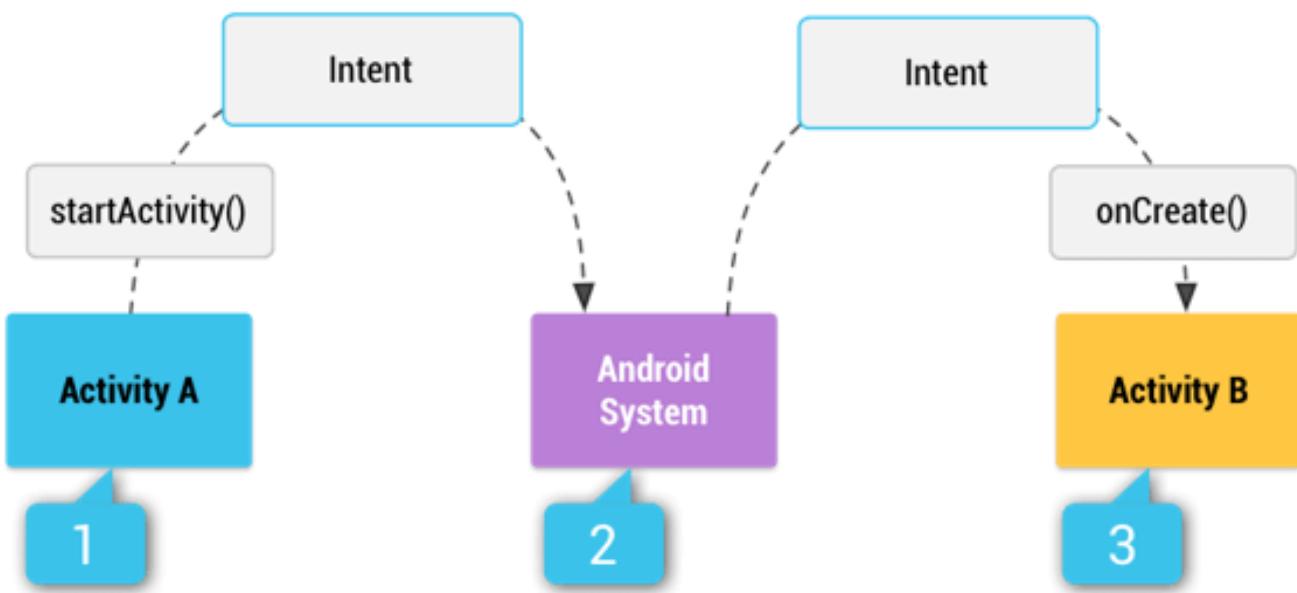


Рисунок 11 – Неявный объект Intent

**Службы** Service или служба является компонентом приложения, который может выполнять длительные операции в фоновом режиме и не содержит пользовательского интерфейса. Другой компонент приложения может запустить службу, которая продолжит работу в фоновом режиме даже в том случае, когда пользователь

перейдет в другое приложение. Кроме того, компонент может привязаться к службе для взаимодействия с ней и даже выполнять межпроцессное взаимодействие (IPC). Например, служба может обрабатывать сетевые транзакции, воспроизводить музыку, выполнять ввод-вывод файла или взаимодействовать с поставщиком контента, и все это в фоновом режиме.

Фактически служба может принимать две формы – запущенная или привязанная.

Служба является «запущенной», когда компонент приложения (например, activity) запускает ее вызовом `startService()`. После запуска служба может работать в фоновом режиме в течение неограниченного времени, даже если уничтожен компонент, который ее запустил. Обычно запущенная служба выполняет одну операцию и не возвращает результатов вызывающему компоненту. Например, она может загружать или выгружать файл по сети. Когда операция выполнена, служба должна остановиться самостоятельно.

Служба является «привязанной», когда компонент приложения привязывается к ней вызовом `bindService()`. Привязанная служба предлагает интерфейс клиент-сервер, который позволяет компонентам взаимодействовать со службой, отправлять запросы, получать результаты и даже делать это между разными процессами посредством межпроцессного взаимодействия (IPC). Привязанная служба работает только пока к ней привязан другой компонент приложения. К службе могут быть привязаны несколько компонентов одновременно, но когда все они отменяют привязку, служба уничтожается.

## 2.4. Выбор инструментов

**Версия API Android** Уровень API Andriod — целое число, однозначно идентифицирующее версию API фреймворка, предлагаемого платформой Android. API Framework — интерфейс, который разработчики могут использовать для взаимодействия с системой. API фреймворка состоит из:

- основного набора пакетов и классов;
- набора XML элементов и атрибутов для файла манифеста и файлов ресурсов;
- набора разрешений, которые приложение может запрашивать;
- набора интентов.

В конфигурационном файле `AndroidManifest.xml` необходимо указать минимальный уровень API, который будет поддерживать приложение, уровень API,

для которого приложение предназначено и максимальный уровень API. По официальной статистике, на май 2019 года большинство устройств используют API 23, а самая высокая версия API — 28 [8]. Соответственно, в качестве минимальной версии выберем 23, а целевой — 28.

**Язык программирования** На сегодня язык Java является одним из самых распространенных и популярных языков программирования [14]. Первая версия языка появилась еще в 1996 году. Текущей версией является Java 12, которая вышла в марте 2019 года. Java активно применяется для создания программного обеспечения для целого ряда устройств: обычных ПК, планшетов, смартфонов и мобильных телефонов и даже бытовой техники. Также Java — официальный язык разработки для Android.

Java является объектно-ориентированным языком, поддерживающим статическую типизацию, полиморфизм, наследование [7]. Ключевой особенностью данного языка является то, что его код является не чисто компилируемым как, например, у языков C/C++, а транслируемым в специальный байт-код. Полученный байт-код передается в управление виртуальной машине Java — JVM (Java Virtual Machine).

Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах — Windows, Linux, Mac OS и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код. Среди недостатков такого подхода — не самая высокая производительность.

Kotlin — это относительно молодой язык от российской компании JetBrains. Появился он в 2011 году. На конференции Google I/O 2017 команда разработчиков Android сообщила, что Kotlin получил официальную поддержку для разработки Android-приложений.

Как и Java, C и C++, Kotlin — это статически типизированный язык. Он поддерживает как объектно-ориентированное, так и процедурное программирование. Также компилируется в JavaScript, и в исполняемый код ряда платформ через инфраструктуру LLVM. Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция. Язык полно-

стью совместим с Java, что позволяет Java-разработчикам постепенно перейти к его использованию; в частности, в Android язык встраивается с помощью Gradle, что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком. Язык разрабатывается с 2010 года, представлен общественности в июле 2011.

## Среда разработки

**IntelliJ IDEA** Интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Среда доступна в двух редакциях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0. В данной среде реализована полная поддержка Java SE, Kotlin, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В редакции Ultimate Edition, доступной под коммерческой лицензией, реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

Продукты JetBrains содержат в себе множество функций, упрощающих процесс программирования — автоматическое именование переменных и закрытие скобок, автоматическое создание шаблонных классов. Это позволяет сэкономить большое количество времени.

**Android Studio** Интегрированная среда разработки для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

Android Studio (см. рис. 12) это официальное средство разработки Android, которое рекомендует Google. Данная среда отличается хорошей документированностью. Android Studio основана на программном обеспечении IntelliJ IDEA от компании JetBrains. Данная среда разработки доступна для Windows, OS X и Linux. В Android Studio множество встроенных функций и возможностей, таких как:

- встроенный редактор макетов пользовательских интерфейсов;
- анализатор APK;
- быстрый эмулятор;
- инструменты профилирования в реальном времени;

- встроенная сборка приложений, основанная на gradle;
- рефакторинг кода;
- статический анализатор кода.

Таким образом, AndroidStudio обладает всеми преимуществами IntelliJ IDEA, но при этом адаптирован для разработки под Android. Поэтому остановимся на данном продукте.

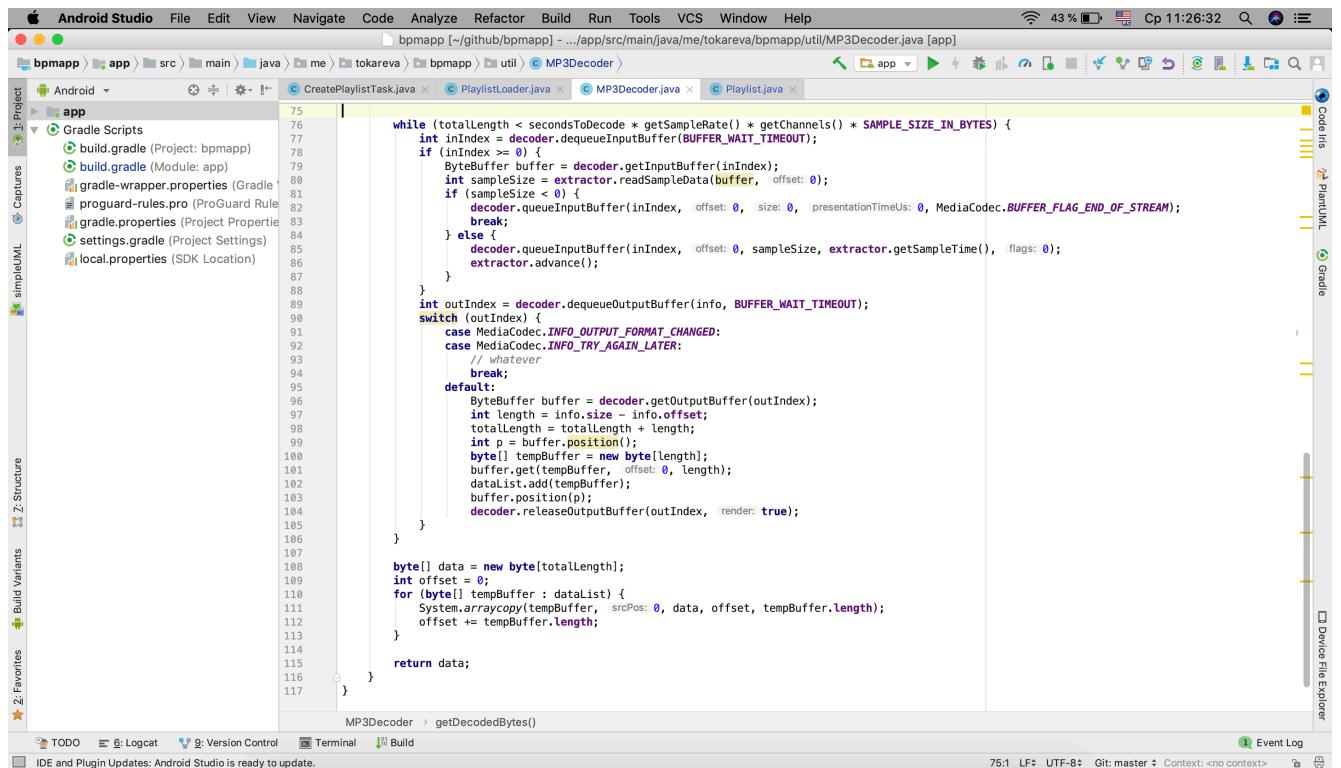


Рисунок 12 – Интерфейс Android Studio

**Система сборки** Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven. Gradle был разработан для расширяемых многопроектных сборок, и поддерживает инкрементальные сборки, определяя, какие компоненты дерева сборки не изменились и какие задачи, зависимые от этих частей, не требуют перезапуска.

Отметим, что Gradle интегрирован в AndroidStudio.

## 2.5. Выводы

В данном разделе были сформулированы основные требования к приложению, разработаны сценарии использования, подробно исследована мобильная платформа Android и выбрана инструментальная база.

Однако разработка рассматриваемого приложения невозможна без детального исследования вопроса работы с аудиофайлами, а также вопроса определения параметра BPM композиции. Рассмотрим эти вопросы подробнее.

## ГЛАВА 3. ИССЛЕДОВАНИЕ МЕТОДОВ И СРЕДСТВ ОПРЕДЕЛЕНИЯ ПАРАМЕТРА ВРМ

### 3.1. Общие сведения о структуре аудиофайлов

Известно, что звук представляет собой механические колебаний в некоторой упругой среде (например, воздухе), которые аналитически могут быть представлены как сигнал сложной формы (см. рис. 13).

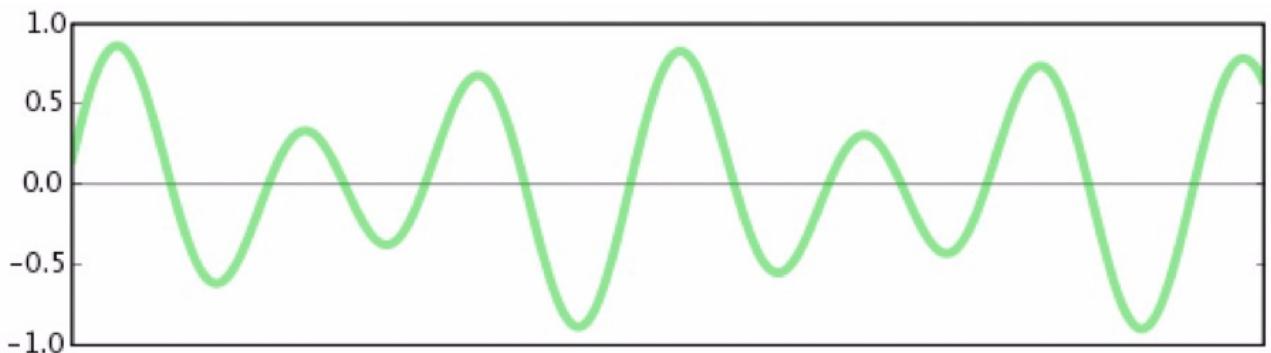


Рисунок 13 – Графическое представление аудио сигнала

Аналоговые носители аудиоинформации (например, магнитная кассета) несут информацию об исходном сигнале посредством непрерывного изменения некоторой физической величины (для кассеты – намагниченности ленты). Аналоговая запись позволяет сохранить мельчайшие детали, но подвержена множеству источников зашумления: погрешности изготовления носителя, неточность записывающего устройства и проигрывателя. Очевидно, что при копировании записи уровень шума возрастает. Посредством сложной математической обработки шум может быть минимизирован, но не устранен полностью.

В настоящее время аудиофайлы сохраняются и передаются в оцифрованном виде. Цифровые записи не меняют свое качество со временем и могут быть скопированы без каких-либо дополнительных помех. Кроме того, к оцифрованным аудиозаписям могут быть применены некоторые эффекты, например, усиление определенных частот или изменение высоты тона.

**Оцифровка аудио-сигналов** Оцифровка сигнала производится в два этапа: дискретизация и квантование [5]. На первом этапе — этапе дискретизации — сохраняются амплитуды сигнала через некоторые равные промежутки времени, определяемые частотой дискретизации (см. рис. 14). На втором этапе — этапе квантования — амплитуды представляются целыми числами, принадлежащими некоторым

ром конечному диапазону возможных значений, определяемых глубиной кодирования. Рассмотрим ограничивающие параметры подробнее.

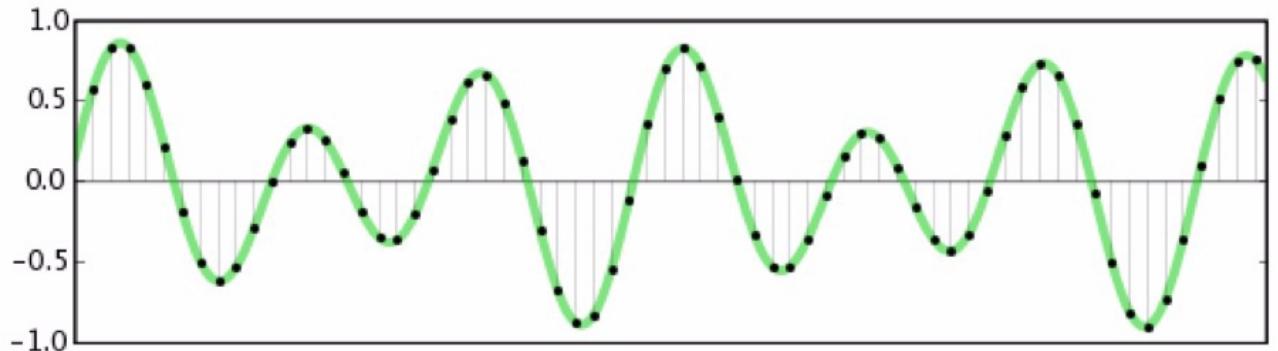


Рисунок 14 – Графическое представление аудио сигнала

**Частота дискретизации** Частота дискретизации определяет число отсчетов, зафиксированных в единицу времени. Очевидно, что с увеличением частоты дискретизации повышается качество восстановленного сигнала (расширяется частотный диапазон). В случае, когда частота дискретизации превышает максимальную частотную составляющую исходного сигнала в два и более раза, аналоговая форма сигнала может быть восстановлена с высокой точностью [2]. Частоты, превышающие половину частоты дискретизации, теряются и не могут быть восстановлены. Поэтому перед дискретизацией необходимо пропустить исходный сигнал через фильтр нижних частот с границей, равной половине частоты дискретизации. Исключение данного этапа может привести к искажению сигнала при восстановлении, называемому наложением.

Таким образом, половина частоты дискретизации представляет верхний частотный предел исходного сигнала, называемый частотой Найквиста.

Человеческое ухо чувствительно к звуковым сигналам с частотами приблизительно от 20 до 20000 Гц. Звуки за пределами этого диапазона не слышны. Поэтому частота дискретизации 40 000 Гц является абсолютным минимумом, необходимым для воспроизведения всего спектра слышимых звуков.

Частота дискретизации, используемая в аудио CD, составляет 44,1 кГц, но человеческая речь понятна при значительно больших ограничениях: в телефонии, например, передаются частоты только до 4 кГц. Наибольшее распространение получили следующие частоты дискретизации: 8 кГц, 16 кГц, 22,05 кГц, 44,1 кГц, 48 кГц, 96 кГц и 192 кГц.

На рис. 15 графически представлена процедура дискретизации при различных частотах дискретизации (слева – низкая, справа – повышенная).

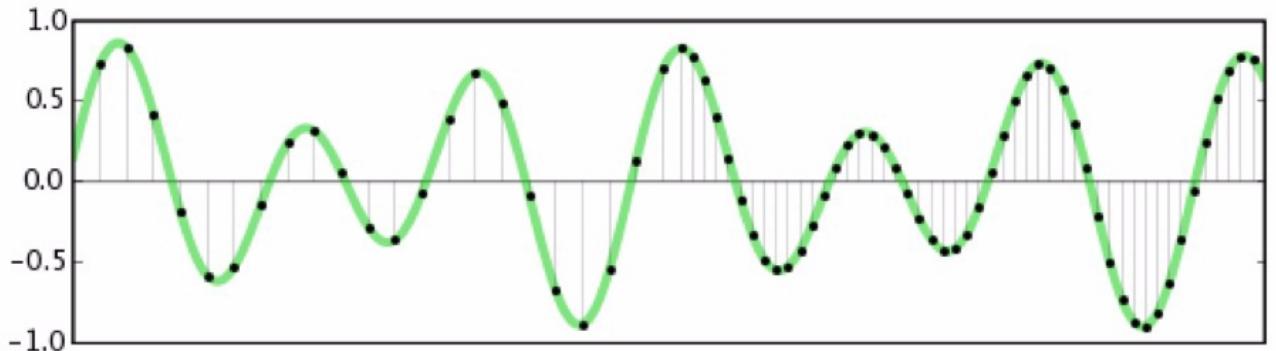


Рисунок 15 – Процедура дискретизации

**Глубина кодирования** Другой мерой качества звука является глубина кодирования, которая определяется разрядностью диапазона возможных значений. Чем выше разрядность (на один отсчет выделяется больше бит информации), тем точнее представление каждой выборки. Увеличение разрядности приводит к расширению динамического диапазона аудиозаписи, иными словами, к увеличению разницы в громкости между самыми громкими и самыми мягкими звуками, которые могут быть представлены.

Динамический диапазон измеряется в децибелах (дБ). Человеческое ухо может воспринимать звуки с динамическим диапазоном до 90 дБ. Однако всегда, когда это возможно, рекомендуется оцифровывать аудиозаписи с динамическим диапазоном, превышающим 90 дБ, для обеспечения максимальной точности. Распространенные типовые форматы и соответствующий им динамический диапазон представлены ниже:

- 8-битное целое число: 48 дБ;
- 16-разрядное целое число: 96 дБ;
- 24-разрядное целое число: 145 дБ.

Однако имеется физическое ограничение к выбору динамического диапазона, обусловленное возможностями аппаратуры в части входного и выходного преобразований. На практике компромиссом в выборе динамического диапазона выступает 16-битная разрядность квантования. Именно этот формат использован в большинстве компьютерных аудиофайлов.

На рис. 16 представлен процесс квантования при различных разрядностях.

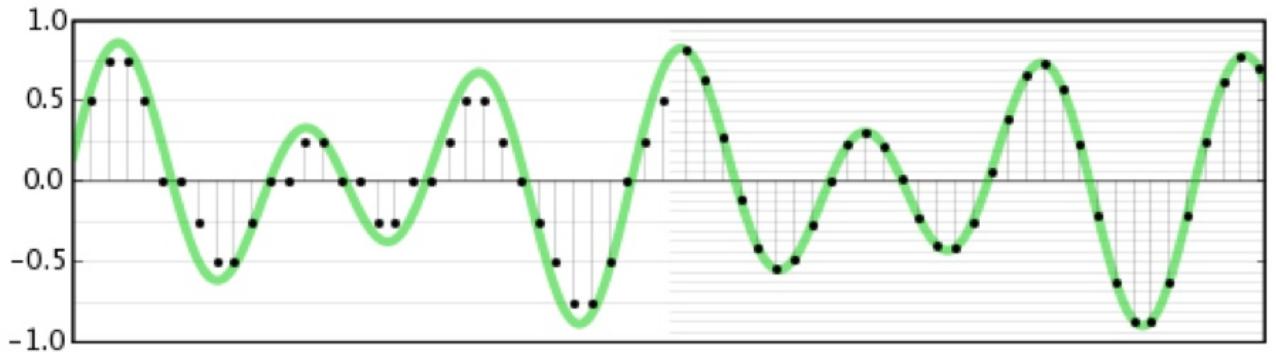


Рисунок 16 – Квантование

Подытожим, качество цифровой аудиозаписи в значительной степени зависит от двух факторов: частоты дискретизации и глубины кодирования. Увеличение частоты дискретизации или глубины кодирования в каждом отсчете повышает качество записи, но также увеличивает объем пространства, используемого аудиофайлами на компьютере или диске.

**Размер аудио файлов** Оцифрованное аудиофайлы большие: несжатый двухканальный аудиофайл с частотой дискретизации 44,1 кГц и глубиной кодирования 16 бит продолжительностью 1 минуту занимает около 10 Мб:

$$2 \cdot 44100 \cdot 16 \cdot 60 \approx 10 \quad (1)$$

Поэтому возникла потребность в сжатии аудиофайлов. Рассмотрим основные форматы файлов.

**Pulse Code Modulation** Первым форматом для хранения аудиосигналов выступила группа файлов, основанных на технологии пульсового кодирования (Pulse Code Modulation, PCM). Каждая цифра в таких файлах описывает один отсчет при оцифровке. К этой группе относятся форматы файлов: WAV, AIFF. Данные форматы точно передают звук, но занимают много места.

Однако с этими файлами удобно работать, так как данные хранятся в незакодированном виде. Поэтому для обработки сжатых аудиофайлов первым шагом проводится их декодирование в формат PCM.

В таблице 2 представлена структура кадра PCM для двухканального аудиофайла с частотой дискретизации 44,1 кГц и разрядностью квантования 16 бит.

Таблица 2 – Структура PCM-кадра

Байт 1	Байт 2	Байт 3	Байт 4
Мл. байт отсчета для левого канала	Ст. байт отсчета для левого канала	Мл. байт отсчета для правого канала	Мл. байт отсчета для правого канала

**Форматы со сжатием** Ранние форматы для сокращения размеров использовали логарифмическую шкалу записи отсчетов. Как известно из математики, логарифмическая шкала позволяет более компактно записывать большие числа (т.е. использовать меньше бит) при этом хорошо отражать динамику описываемой кривой звуковой волны. Современные форматы используют изысканные алгоритмы сжатия, основанные на психологических исследованиях. При разработке таких форматов учитывалась специфика восприятия звука человеком: не все частоты воспринимаются человеком даже в слышимом диапазоне, поэтому их можно исключить из записи. К таким форматам относятся MP3 (MPEG I, layer 3), OggVorbis, и WMA (Windows MediaAudio).

Первым форматом, позволяющим эффективно уменьшать объем аудиофайлов, является стандарт MP3 (MPEGI, Layer 3). MP3 — это метод сжатия с потерями, позволяющий значительно уменьшить размер цифрового аудиофайла с небольшим влиянием на качество. Коэффициент сжатия достигает десяти раз. MP3 исключает из аудиофайла компоненты, к которым человеческий слух не чувствителен. Алгоритм основывается на психоакустической модели восприятия звуков человеком.

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

### 3.2. Обзор методов определения BPM

Для нахождения темпа музыки программным способом необходимо понять, как мы определяем темп на слух и что такое бит.

Имитация физических явлений, подчиняющимся известным математическим законам, в ряде приближений всегда выполнима. Но если говорить о более абстрактных понятиях, таких как чувства, мы столкнемся с трудностями, т.к.

чувства не описать такими законами. Самые простые вещи, которые человек может почувствовать, часто самые трудные для математического описания. Определение темпа музыки следует этой закономерности: обнаружение музыкального темпа не представляется сложным для человека или животного, однако, это всего лишь чувство. Чувство, из-за которого человек начинает двигаться в такт, хлопать в ладоши, постукивать ногами. В данной главе рассматриваются способы, с помощью которых машина может определить темп так же, как это делает человек, который полагается на свои ощущения. Существует достаточное количество алгоритмов, которым удалось получить хорошие результаты в этом деле и мы рассмотрим несколько из них.

В общем случае под темпом музыки следует понимать частоту следования периодических энергетических всплесков (т.е. битов) в сигнале. Звук, который мы слышим, содержит определенную энергию, очевидно, что чем больше энергии передает звук, тем громче он будет казаться. Но звук будет восприниматься как бит, только если его энергия значительно превосходит энергию смежных участков аудио-сигнала. Например, в монотонном сигнале с большими всплесками энергии распознавание битов является простой задачей, но если звук разнообразный и громкий сам по себе, то решение задачи значительно усложняется. Таким образом, в простейшем случае можно заключить, что бит — это всплеск энергии в окрестности, а частота следования битов и есть темп. Данное заключение приведет нас к простейшей модели алгоритма.

### 3.2.1. Энергетические пики

В простейшем случае выделение бита может быть выполнено путем сравнения мгновенной мощности  $P$  сигнала со средней мощностью  $\bar{P}$  на временном интервале  $T_i$ . При превышении разницы между мгновенной и средней мощностью на пороговую величину  $\Delta P$  следует зафиксировать наличие бита. Рассмотрим данный метод подробнее.

Для обработки сигнала с целью выделения бита удобнее перейти от амплитудных значений сигнала к его мгновенной мощности. Мгновенная мощность сигнала пропорциональна квадрату его амплитуды [2].

$$P(t) = S^2(t) \tag{2}$$

Из-за особенностей восприятия звука человеком, для расчетов допустимо разбить сигнал на отрезки длительностью 50 мс, что соответствует 2205 отсчетам при частоте дискретизации 44,1 кГц. Такие наборы из 2205 отсчетов будем называть окнами. Так, мгновенная мощность  $i$ -го окна рассчитывается по формуле:

$$P[i] = \sum_{t=i \times 2205}^{i \times 2205 + 2205 - 1} P(t) \quad (3)$$

Как было сказано ранее выделение бита происходит путем сравнения мгновенной мощности со средней мощностью. Для вычисления средней мощности будем использовать отрезок сигнала длительностью 1 сек, т.е. 20 окон. Так, средняя мощность  $i$ -й секунды рассчитывается по формуле:

$$P[i] = \frac{1}{20} \sum_{t=i \times 20}^{i \times 20 + 20 - 1} P[t] \quad (4)$$

Для оптимизации расчетов, будем хранить вектор, содержащий 20 последних рассчитанных мгновенных мощностей сигнала:

$$\text{historyBuffer} = (P_0, P_1, \dots, P_{18}, P_{20}). \quad (5)$$

Итак, обобщенный алгоритм представлен в листинге 1:

Данный метод отличается высоким быстродействием, но имеет низкую точность, недостаточную для разрабатываемого приложения. Рассмотрим альтернативные способы.

### 3.2.2. Автокорреляционная функция

Эффективным по критерию точности способом определения ВРМ выступает использование автокорреляционной функции.

В общем случае под автокорреляцией понимается математический инструмент, позволяющий определить схожесть сигнала самого с собой, смещенным во времени. Данный аппарат позволяет выделить повторяющиеся сегменты в исходном сигнале. Рассмотрим предложенный математический аппарат подробнее.

Согласно определению автокорреляция определяет меру сходства между сигналом  $f(t)$  и его копией, имеющей произвольную задержку на время  $\tau$  [2]. Чем больше перекрытие сигнала  $f(t)$  его копией, тем больше значение функции авто-

Листинг 1 – Определение BPM с помощью мощности сигнала

```

function BPM(s)
    bits  $\leftarrow$  0
    historyBuffer  $\leftarrow$  array(20)
    windowLength  $\leftarrow$  2205

    windowsCount  $\leftarrow \frac{\text{length}(s)}{\text{windowLength}}$ 

    for i = 0 to windowsCount do
        momentPower  $\leftarrow P(s, i)$ 
        averagePower  $\leftarrow P(\text{historyBuffer})$ 

        historyBuffer.push(momentPower)

        if momentPower  $> C \cdot \text{averagePower}$  then
            bits ++
        end if
    end for

    duration  $\leftarrow \frac{\text{length}(s)}{44100}$ 
    bpm  $\leftarrow \frac{1}{60} \cdot \frac{\text{bits}}{\text{duration}}$ 

    return bpm
end function

```

корреляции. Для функции  $f(t)$ , принимающей только действительные значения, функция автокорреляции может быть записана как

$$R(\tau) = \int_{-\infty}^{+\infty} f(t) \cdot f(t - \tau) dt. \quad (6)$$

Однако при работе с оцифрованным звуком приходится иметь дело с дискретным сигналом. Выражение 6 для дискретного сигнала  $f[t]$  принимает следующий вид:

$$R[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot f[m - n] \quad (7)$$

Таким образом, путем смещения сигнала во времени становится возможным выделить задержку, при которой обеспечивается максимальная схожесть сигналов. Для большинства музыкальных записей максимум автокорреляции приходится на моменты наложения звуков инструментальных ударов, задающих темп композиции. Следовательно, BPM может быть вычислен через задержку сигнала относительно самого себя, обеспечивающую максимум корреляции:

$$BPM = \frac{F_s \cdot 60}{n} \quad (8)$$

где  $F_s$  – частота дискретизации,  $n$  – задержка сигнала в семплах, обеспечивающая максимум автокорреляционной функции.

На рисунке 17 представлен пример работы автокорреляционной функции, полученный автором настоящей работы, для небольшого участка музыкальной композиции:

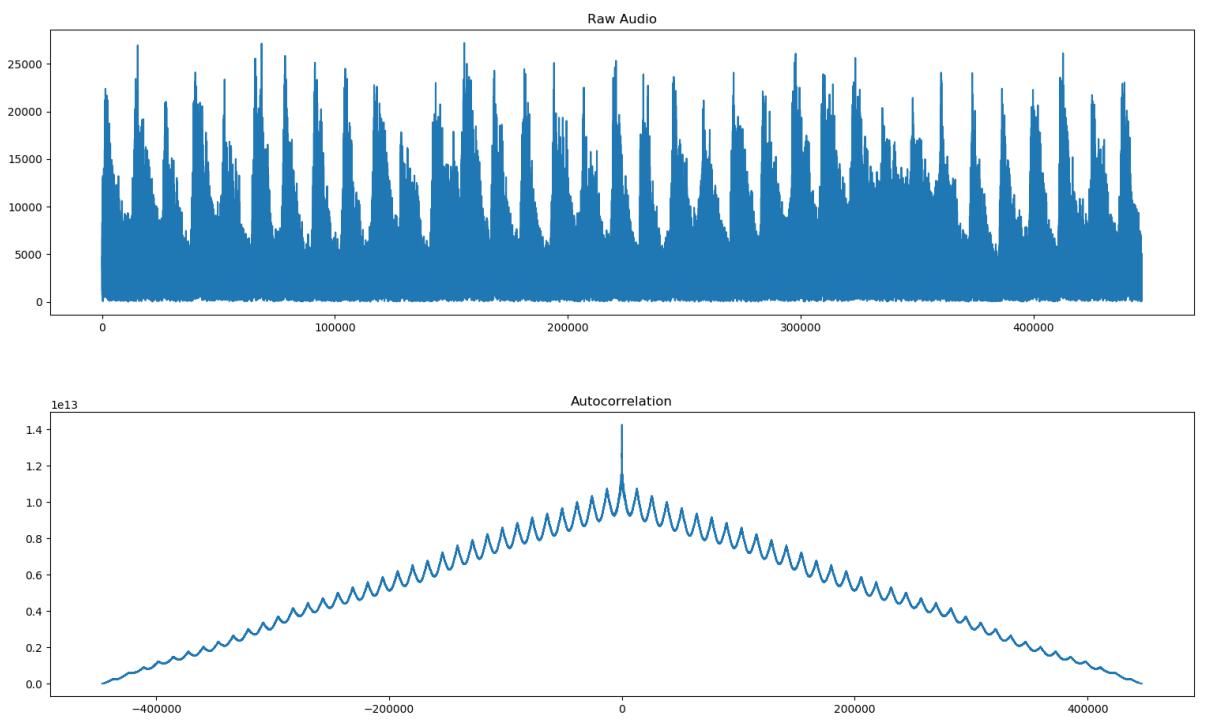


Рисунок 17 – Исходный сигнал и его автокорреляция

Общий алгоритм вычисления BPM с использованием автокорреляционной функции представлен в листинге 2.

Листинг 2 – Вычисление BPM с использованием автокорреляционной функции

```

function BPM(s)
    autocorr = autocorrelation(s)           ▷ Вычисляем автокорреляцию
    lag = findFirstPeak(autocorr) ▷ Находим задержку сигнала, при которой
        автокорреляция максимальна

     $BPM = \frac{F_s \cdot 60}{lag}$            ▷ Вычисляем BPM по формуле 8

    return BPM
end function

```

**Уменьшение сложности алгоритма** Вычисление автокорреляционной функции для отрывка аудиофайла по формуле 7 является алгоритмом квадратичного времени  $O(n^2)$ , однако оно может быть сведено к алгоритму линейнологарифмического времени  $O(n \log n)$ .

Метод уменьшения сложности вычисления основан на теореме Хинчина — Колмогорова, утверждающей, что автокорреляционная функция сигнала есть фурье-образ его спектральной плотности мощности [2]. Поскольку для дискретных сигналов для вычисления их спектров существует алгоритм быстрого преобразования Фурье, имеющий порядок сложности  $O(n \log n)$ , то имеется возможность ускорить вычисление автокорреляционной функции за счет вычисления спектра сигнала, затем его мощности (квадрата модуля) и затем обратного фурье-преобразования. Ненормированная автокорреляционная функция может быть вычислена с помощью алгоритма, представленного в листинге 3.

Листинг 3 – Вычисление автокорреляции с использованием теоремы Хинчина-Колмогорова

```

function autocorrelation(X(t))
     $F_r(f) = FFT(X(t))$            ▷ FFT - быстрое преобразование Фурье
     $S(f) = F_r(f) \cdot F_r^*(f)$       ▷ * - комплексно-сопряженное
     $R(\tau) = IFFT(S(f))$           ▷ IFFT - обратное преобразование Фурье

    return R( $\tau$ )
end function

```

### 3.3. Выводы

В разделе рассмотрена структура аудиофайлов и способы их декодирования. Исследованы несколько методов определения параметра BPM.

Простейший способ, основанный на нахождении темпа через разницу во времени между энергетическими пиками, отличается простотой и высоким быстродействием, но имеет низкую точность — недостаточную для разрабатываемого приложения.

Альтернативный способ, основанный на использовании автокорреляционной функции, точнее, особенно, на «сложных» композициях; однако имеет относительно высокую сложность вычисления. Как было рассмотрено в предыдущем разделе, мобильные устройства не отличаются быстродействием. Поэтому при разработке мобильного приложения необходимо учитывать это ограничение.

## ГЛАВА 4. РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ

### 4.1. Архитектура приложения

Одним из самых важных этапов в разработке любого ПО является этап проектирования архитектуры. Грамотно спроектированная архитектура упрощает добавление или изменение функционала, позволяет сделать процесс поиска ошибок менее трудозатратным, уменьшает уровень сложности проекта и минимизирует затраты на его поддержку.

Итак, в состав разрабатываемого приложения входит сложное математическое вычисление: обработка музыкального сигнала и определение его BPM. Вычислительные ресурсы современных мобильных телефонов ограничены для выполнения поставленной нами задачи: приложение, выполняющее такую работу на аппаратной платформе смартфона будет работать слишком медленно, что будет сказываться негативно на опыте пользователя. В качестве решения предлагается использовать архитектуру клиент-сервер, которая заняла прочное место среди распределенных вычислений, для эффективного распределения вычислительных задач и делегирования части работы на стороннюю аппаратную платформу, предлагающую большую производительность, по сравнению с мобильным телефоном. Рассмотрим архитектуру клиент-сервер подробнее.

**Общие сведения** Клиент-серверная архитектура предполагает распределение прикладных задач между различными узлами для достижения большей эффективности (уменьшение времени отклика, повышение надежности и стабильности системы, экономия ресурсов там, где это необходимо). Узлы, входящие в данную систему, не являются равноправными: уровень представления данных находится на машине пользователя, т.е., клиенте, а уровень вычислений и управления данными на узле, называемым сервером. На рис. 18 приведена общая иллюстрация данной модели вычислений.

**Клиент** Клиентом принято называть объект, осуществляющий запрос к серверу на выполнение какой-либо операции или предоставление какой-либо информации. Как правило, клиентом является рабочая станция, с которой пользователь ведет активное взаимодействие. Соответственно, на стороне клиента большое значение имеет уровень представления. Как правило, в большинстве систем, работающих по модели клиент-сервер, графическому интерфейсу пользователя

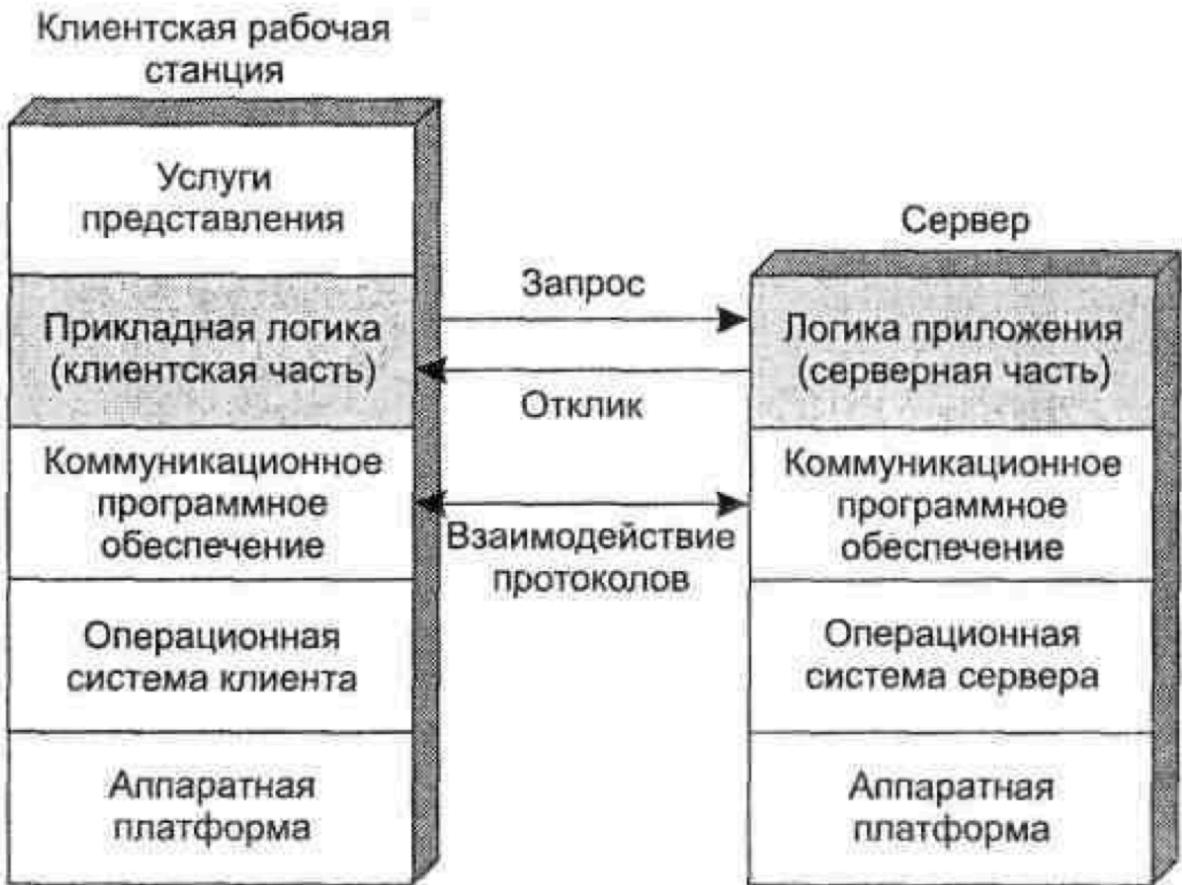


Рисунок 18 – Общий вид архитектуры клиент-сервер

(Graphical User Interface, GUI) уделяется серьезное внимание. Такой интерфейс должен быть простым, удобным, должен обладать быстрым откликом и одновременно являться мощным и гибким.

**Сервер** Сервером называют объект, предоставляющий клиентам информацию и набор услуг. Как правило, сервером является высокопроизводительное приложение, обеспечивающее одновременный доступ нескольких клиентов к одному и тому же функционалу (в таком случае говорят о многопользовательском режиме).

**Взаимодействие между узлами** Помимо серверов и клиентов в окружение архитектуры клиент-сервер входит сеть. Описываемая модель по определению является распределенной, соответственно, взаимодействие между узлами осуществляется через общую локальную, глобальную или составную сеть по принципу запрос-ответ. Данный принцип предполагает отправку запросов клиентским ПО и возврат ответов на запросы серверным ПО.

Базовым программным обеспечением и на клиенте, и на сервере является операционная система. Очевидно, что на обоих узлах аппаратные платформы и ОС могут отличаться, но такие различия не имеют значения при использовании общих коммуникационных протоколов между узлами. Примером такого протокола является протокол HTTP, он лежит в основе обмена данными в сети Интернет [4]. Клиенты и серверы взаимодействуют, обмениваясь индивидуальными сообщениями (а не потоком данных). Сообщения, отправленные клиентом, называются запросами, а сообщения, отправленные сервером, называются ответами.

Каждое сообщение состоит из трех частей, которые следуют друг за другом:

- a) Стартовая строка, определяющая тип сообщения и различающаяся для запроса и ответа. Стартовая строка запроса содержит метод, URI целевого ресурса (в котором клиент может передавать параметры выполнения запроса после символа "?"), версию HTTP протокола, а ответа — версию протокола, код состояния и пояснение. Пример стартовой строки:

— Запрос:

```
GET /path?param1=value1&param2=value2 HTTP/1.1
```

```
Host: ru.wikipedia.org
```

— Ответ:

```
HTTP/1.1 200 OK
```

- б) Заголовки, являющиеся разделенными двоеточием парами ключ-значение.

Пример заголовков:

```
Server: Apache/2.2.11 (Win32) PHP/5.3.0
```

```
Last-Modified: Sat, 16 Jan 2010 21:16:42 GMT
```

```
Content-Type: text/plain; charset=windows-1251
```

```
Content-Language: ru
```

- в) Тело сообщения, использующееся для передачи объекта, связанного с запросом или ответом. Не является обязательным.

С помощью такой структуры сообщений клиент может общаться с сервером передавая данные в параметрах запроса (URI), заголовках и теле запроса. Конкретный способ передачи данных и методы кодирования определяются выбранной разработчиками стратегией.

**Концепции архитектуры** Существуют различные концепции построения архитектуры клиент-сервер:

- **Слабый клиент – мощный сервер** — вся обработка информации осуществляется целиком сервером. Сервер получает необработанные данные от клиента, выполняет работу и посыпает готовый результат, не требующий дополнительной обработки. Клиент только ведет диалог с пользователем: составляет запрос, отсылает запрос, принимает запрос и выводит информацию пользователю.
- **Сильный клиент** — часть обработки информации перепоручается клиенту.

**Применение архитектуры** При проекции используемой архитектуры на разрабатываемую систему очевидно, что в качестве клиента будет выступать мобильное приложение (в дальнейшем клиент), предоставляющее пользователю графический интерфейс, а в качестве сервера — приложение серверного типа (в дальнейшем сервер), вычисляющее значение BPM музыкальных композиций. Сервер будет находиться на удаленной машине и будет способен принимать запросы от множества клиентов одновременно.

Сетевое соединение между узлами является узким местом в применяемой архитектуре. Для того чтобы уменьшить нагрузку на канал связи и в том числе на сервер, будем использовать стратегию сильного клиента. Так, нет необходимости вычислять BPM аудиозаписи используя весь сигнал, достаточно выполнить обработку ограниченного фрагмента, например, 10 секунд. Также, учитывая особенности построения музыкальных композиций необходимо пропустить 10-15 секунд от начала звучания (вводный фрагмент). Эти задачи будем решать в рамках мобильного приложения, это позволит уменьшить объем передаваемых по сети данных.

Учитывая специфику протокола HTTP, данные аудиосигнала будем передавать в теле запроса, используя метод POST. Помимо аудиосигнала, для вычисления BPM серверу необходимо знать как минимум два дополнительных параметра: частоту дискретизации и количество аудио каналов. Будем передавать эти параметры в URI в качестве параметров запроса *sample\_rate* и *channels\_count*. Ответ сервера будет содержать код статуса 200 в случае успеха или 400 в случае обработанной ошибки. В теле будет содержаться единственное число — вычисленное значение BPM.

Итак, мы рассмотрели глобальную архитектуру разрабатываемой системы, теперь рассмотрим клиент и сервер отдельно и подробнее.

## 4.2. Клиентская часть

В предыдущем разделе было подробно рассмотрено взаимодействие между клиентом и сервером. Теперь рассмотрим архитектуру клиента отдельно.

Для построения чистой архитектуры необходимо выделить основные компоненты, входящие в состав приложения. На рис. 19 представлены эти компоненты.

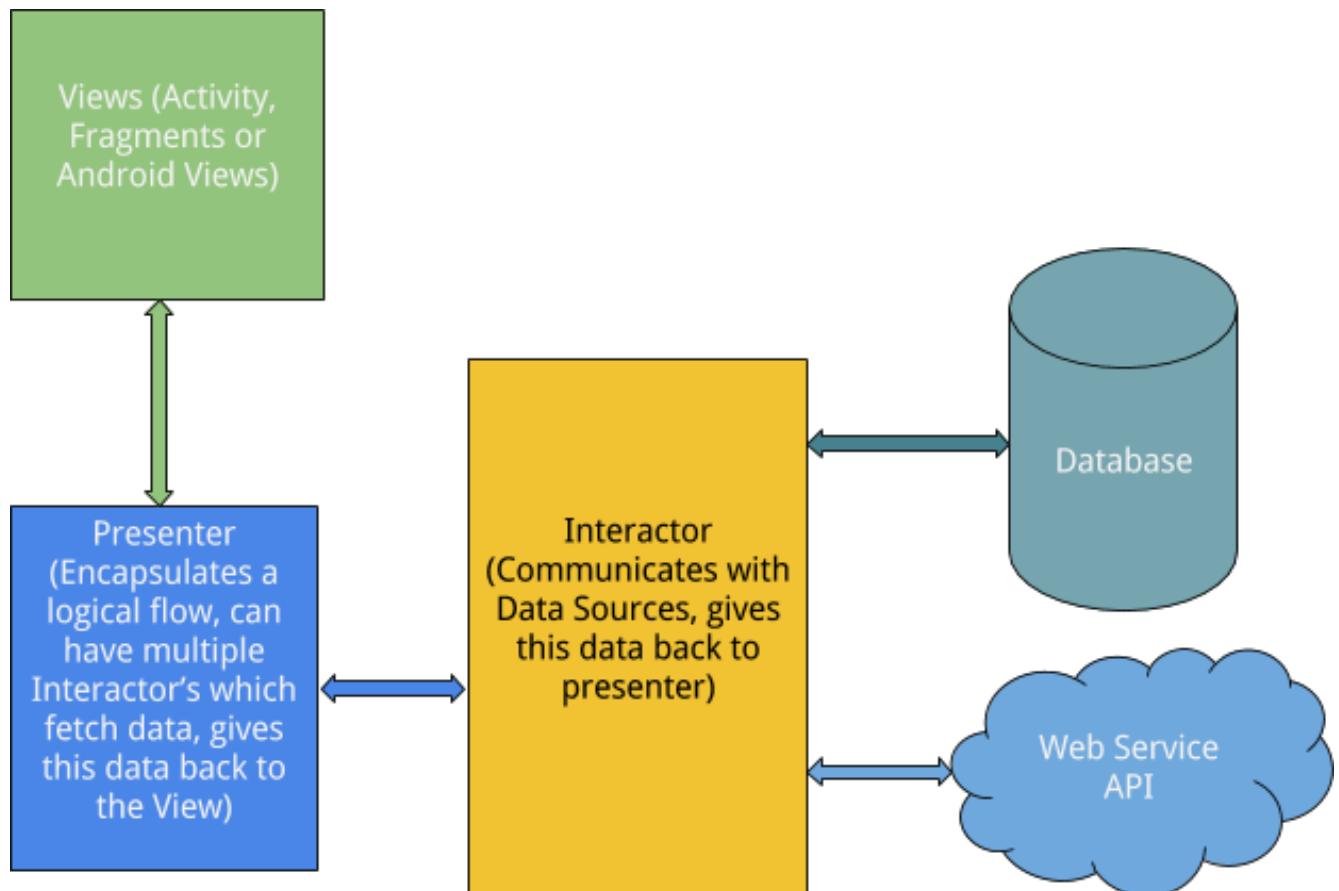


Рисунок 19 – Общий вид компонентов мобильного приложения

**Views** На данном уровне происходит взаимодействие мобильного приложения с пользователем.

**Presenter** Слой, который инкапсулирует в себе бизнес-логику. Является промежуточным слоем между слоем Interactor и Views.

**Interactor** На данном уровне происходит взаимодействие с источниками данных — хранилищем мобильного телефона, базой данных.

**Services** Службы, запущенные в фоновом режиме выполняют длительные задачи без взаимодействия с интерфейсом пользователя.

Важной частью разрабатываемого мобильного приложения является вычисление BPM. Эта задача состоит из нескольких этапов:

- а) Получение списка аудиозаписей
- б) Декодирование mp3
- в) Выделение фрагмента для обработки
- г) Отправка запроса с фрагментом на сервер
- д) Обработка результатов

Программируя такую логику важно не допустить ряд специфичных ошибок. Рассмотрим эти ошибки подробнее.

**Работа с сетью в UI потоке** Задача UI потока заключается в отображении графического интерфейса на экране. Выполнение в этом потоке длительных операций приводит к ситуации, когда приложение не отвечает или работает слишком медленно. Частично эта проблема решена самой платформой Android с помощью ограничений: при попытке сетевого взаимодействия будет инициировано исключение. Чтобы предотвратить такое поведение, для выполнения длительных задач необходимо пользоваться следующими инструментами:

- AsyncTask
- Thread
- Handler

**Неаккуратное использование объекта Context** Context это базовый абстрактный класс, имеющий доступ к различным ресурсам и реализующий широкий спектр функционала, например создание новых Activity, отправку широковещательных сообщений, получение объектов Intent, доступ к файловой системе и т.п.. От класса Context унаследованы такие базовые классы как Activity, Service и Application.

Сохранение объекта Context в статической переменной и передача его в другие объекты ведет к утечкам памяти, так как ссылка на Activity предоставившую данный объект продолжает существовать и сборщик мусора не может освободить память, когда жизненный цикл Activity завершен.

Другая проблема с объектом Context связана с его использованием тогда, когда Activity уже перестала существовать, например, в методе onPostExecute() класса AsyncTask. Для избежания исключения NullPointerException рекомендуется использовать класс WeakReference для создания слабой ссылки.

**Отсутствие кэширования данных** Сетевое взаимодействие производит полезную информацию, которую необходимо сохранять для экономии пользовательского трафика путем избегания повторных запросов. В Android данные можно сохранять различными способами, например, в хранилище SharedPreferences или в базе данных SQLite.

Рассмотрев данные ошибки, можно описать процесс работы модуля, осуществляющий вычисление BPM. Данный модуль будет являться службой (Service), так как именно служба идеально подходит для выполнения длительных и регулярных операций в фоне. При запуске службы будет создан отдельный поток, в котором будет происходить последовательное декодирование музыкальных файлов и отправка их на сервер для вычисления BPM. Полученный от сервера результат будет записываться в базе данных в формате ключ-значение, где в качестве ключа будет выступать имя файла, а в качестве значения — вычисленный BPM. Прогресс процесса будет отображаться в уведомлении.

#### 4.2.1. Выбор библиотек

На текущий момент развито движение открытого исходного кода и существует громадное количество готовых библиотек, содержащих код для решения самых разносторонних задач, поэтому нет необходимости разрабатывать модуль, который уже был написан и протестирован другими людьми. Таким образом, для разработки мобильного приложения стоит вопрос выбора библиотек, которые будут использоваться. Для Android-проекта зависимости описываются в файле build.gradle. Рассмотрим основные из них.

- Nammu — библиотека, позволяющая упростить работу с разрешениями.
- Retrofit — известная библиотека от компании Square предоставляющая типобезопасный HTTP-клиент для Android и Java. Отличается простотой использования.
- Universal Image Loader — библиотека для асинхронной загрузки изображений, кэширования и отображения. Позволяет загружать изображения из ин-

тернета и из файловой системы, обладает широким спектром настроек. Самая популярная Android-библиотека на GitHub.

- CircleImageView — отображает изображение в круглом View.
- Realm — ORM (Object-Relational Mapping) база данных для мобильных платформ на базе iOS и Android. Realm отличается от других решений высокой скоростью работы, простотой в использовании, а так же «живыми» объектами, то есть, при изменении данных в базе, обновятся и объекты в коде, которые ссылаются на эти данные.

#### 4.2.2. UML диаграммы

В ходе разработки мобильного приложения были созданы пакеты классов, изображенные на рис. 20.

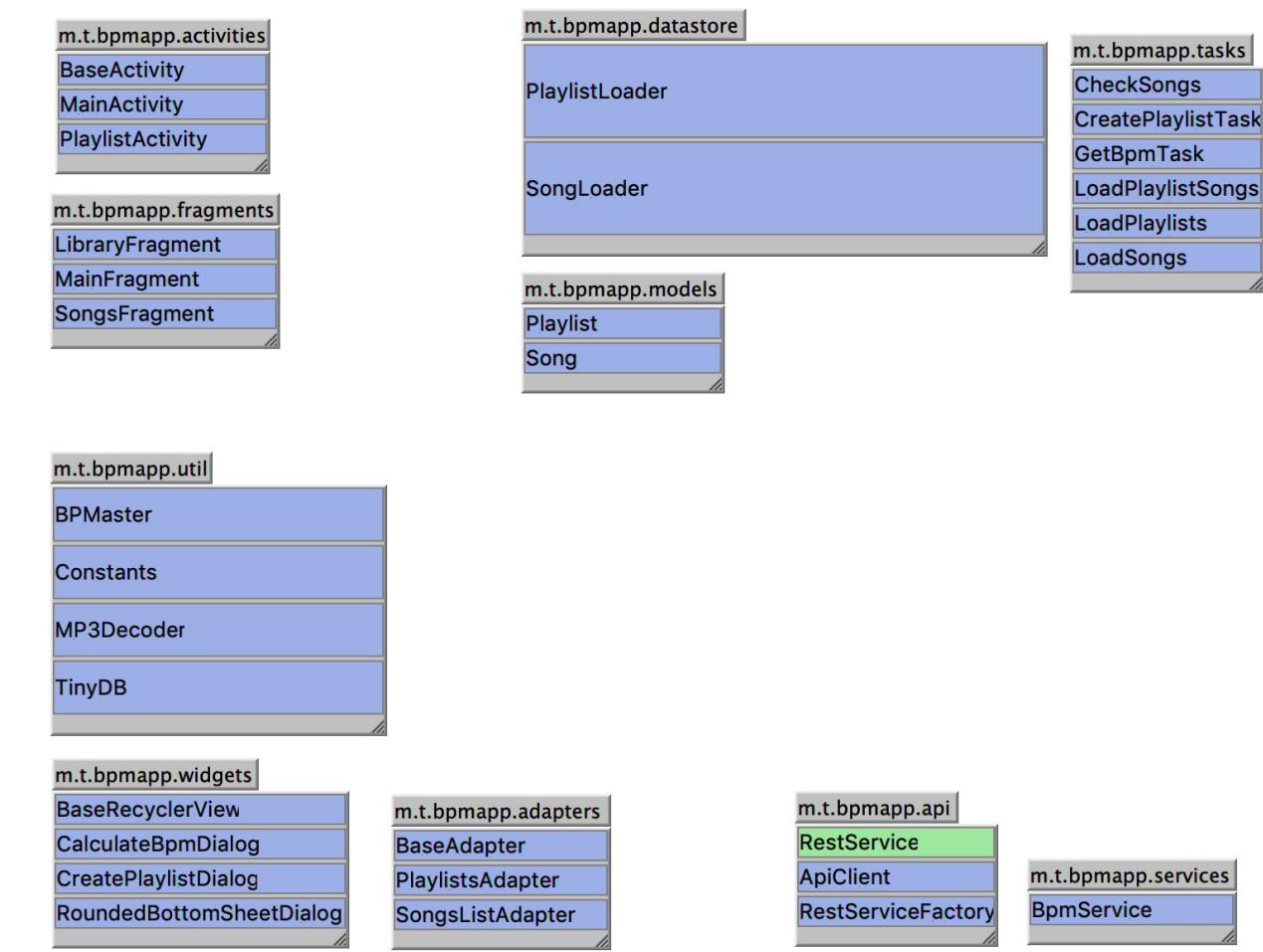


Рисунок 20 – Диаграмма пакетов

Рассмотрим подробнее пакеты и классы, в них содержащиеся.

- bpmap.activities** Данный пакет содержит в себе все классы типа Activity.
- BaseActivity унаследован от класса AppCompatActivity (часть Android SDK) и является базовым классом для всех Activity, реализованных в мобильном приложении.
  - MainActivity является точкой входа в приложение, данный класс выполняет проверку разрешений и отрисовку бокового меню, которое используется для навигации между основными экранами.
  - PlaylistActivity используется для отображения конкретного плейлиста.

Диаграмма классов пакета bpmap.activities представлена на рис. 21.

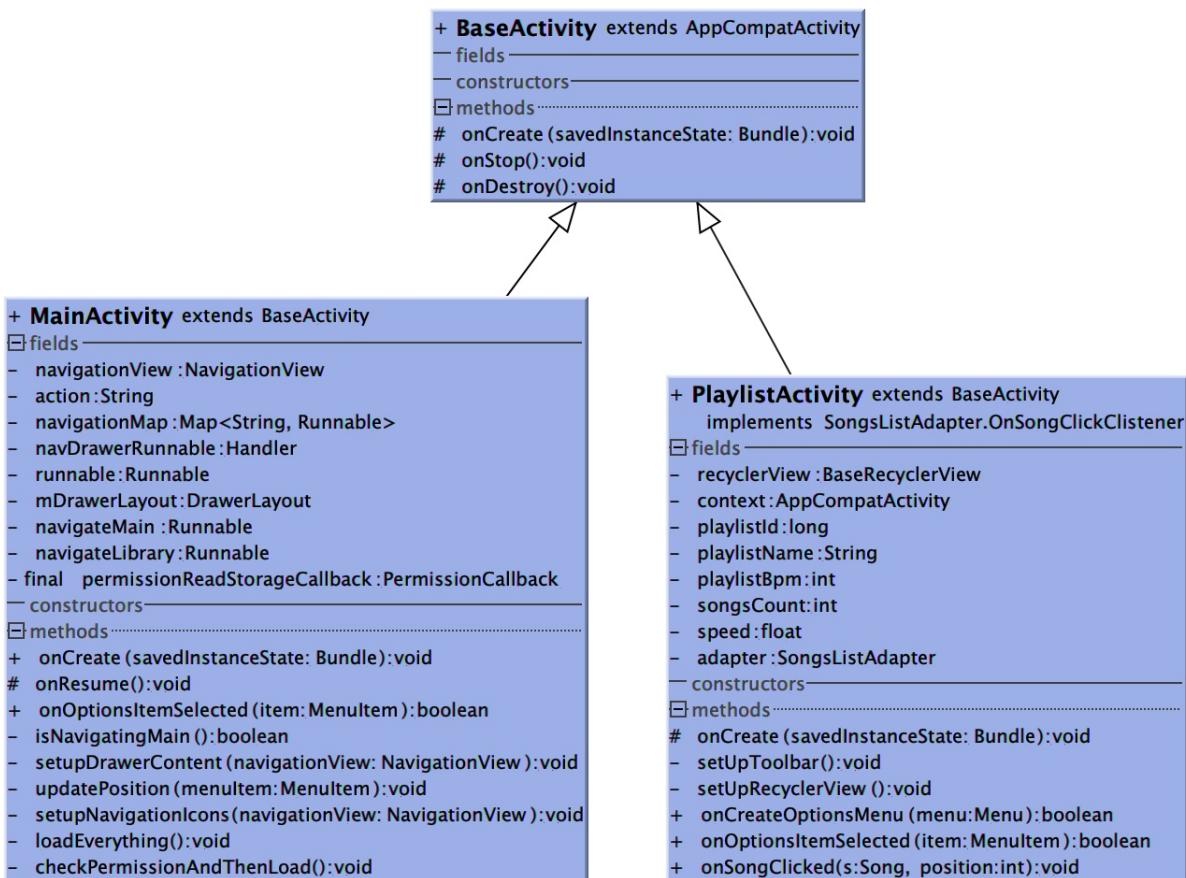


Рисунок 21 – Диаграмма классов пакета bpmap.activities

- bpmap.fragments** Пакет bpmap.fragments содержит в себе все классы типа Fragment.

- MainFragment отображает главный экран приложения, на который попадает пользователь, когда открывает приложение. На данной странице содержит-

ся форма для создания плейлиста, где пользователь выбирает желаемую скорость бега, а также полезную информацию, например, последние созданные плейлисты.

- LibraryFragment используется для отрисовки вкладок на экране библиотеки. Данный фрагмент упрощает навигацию для фрагментов SongsFragment, PlaylistsFragment.
- SongsFragment является частью фрагмента LibraryFragment и используется для отображения информации о всех музыкальных композициях в мобильном телефоне.
- PlaylistsFragment так же является частью фрагмента LibraryFragment и отображает все плейлисты пользователя.

На рис. 22 изображена диаграмма классов данного пакета.

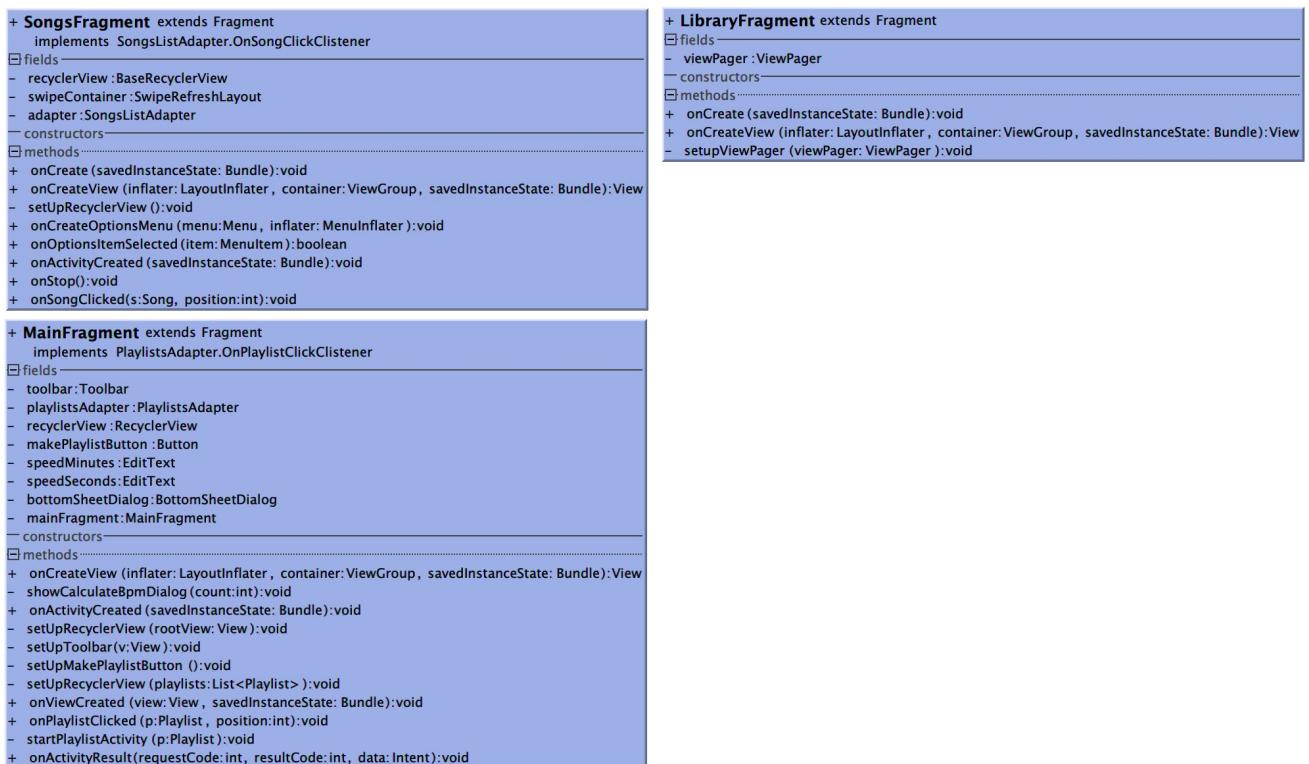


Рисунок 22 – Диаграмма классов пакета bpmapp.fragments

**bpmapp.datastore и bpmapp.models** Пакет bpmapp.datastore содержит классы, которые общаются с файловой системой и базой данных получая и сохраняя информацию о песнях и плейлистиах. В своих методах классы из пакета bpmapp.datastore оперируют классами Song и Playlist из пакета bpmapp.models.

Классы Song и Playlist являются POJO (Plain Old Java Object) и наследуются от класса RealmObject, что позволяет использовать базу данных Realm используя ORM (Object Relation Mapping) подход.

- SongLoader:
  - возвращает массив, состоящий из всех музыкальных композиций файловой системы, каждая из которых содержит заполненное поле bpm, которое хранится в базе данных Realm;
  - сохраняет пару имя файла-bpm в базе данных Realm;
  - возвращает URI обложки альбома.
- PlaylistLoader:
  - возвращает массив, состоящий из всех музыкальных плейлистов файловой системы, каждый из которых содержит заполненное поле bpm, которое хранится в базе данных Realm;
  - сохраняет пару имя плейлиста-bpm в базе данных Realm;
  - возвращает URI обложки альбома для первой; песни с обложкой из плейлиста;
  - создает новый плейлист;
  - добавляет композицию в плейлист;
  - удаляет плейлист;
  - возвращает массив, содержащий все музыкальные композиции, входящие в состав конкретного плейлиста.
- Song — представляет музыкальную композицию, содержит в себе такую информацию как название композиции, исполнителя, длительность, BPM, имя файла, название альбома.
- Playlist — представляет плейлист, содержит имя плейлиста, BPM, количество композиций в плейлисте, путь к файлу обложки, скорость.

На рис. 23 изображена диаграмма классов пакетов `bpmapp.datastore` и `bpmapp.models`.

**bpmapp.widgets** Для создания UI-компонентов в системе Android необходимо реализовывать собственные классы, которые наследуются от View. В разрабатываемом приложении используется собственная реализация диалогов пользователя, для этого пакет `bpmapp.widgets` содержит следующие классы:



Рисунок 23 – Диаграмма классов пакетов bpmapp.datastore и bpmapp.models

- `RoundedBottomSheetDialog` — базовый класс для всех диалогов приложения, который использует собственный стиль для создания закругленных углов;
- `CalculateBpmDialog` — диалог типа `RoundedBottomSheetDialog`. Данные диалог отображается при старте приложения, сообщая пользователю о присутствии в телефоне аудиозаписей без определенного BPM и предлагает начать расчет BPM прямо сейчас. Также данный диалог позволяет пользователю закрыть его и больше не показывать;
- `CreatePlaylistDialog` — диалог типа `RoundedBottomSheetDialog`, в котором пользователю предлагается ввести название плейлиста при его создании.

**bpmapp.adapters** Для отображения списков в разрабатываемом мобильном приложении используется компонент `RecyclerView`, который по-умолчанию использует шаблон проектирования `ViewHolder`. Данный шаблон подразумевает создание специального класса, наследника `RecyclerView.Adapter`. `Adapter` содержит данные списка, связывает их с графическими компонентами списка с помощью

приватного класса ItemHolder и обрабатывает пользовательское взаимодействие с элементами списка.

- PlaylistsAdapter используется для компонента RecyclerView, отображающего список плейлистов.
- SongsListAdapter используется для компонента RecyclerView, отображающего список музыкальных файлов.

**brmapp.api и brmapp.services** Пакет brmapp.services содержит все службы мобильного приложения, а brmapp.api включает в себя все классы, использующиеся для сетевого взаимодействия. Рассмотрим классы, которые входят в состав данных пакетов:

- BpmService — служба, которая в фоновом режиме выполняет декодирование и обрезку аудиозаписей, отправку декодированных данных на сервер для определения BPM и сохранение результатов в базу данных. Для общения с сервером данный класс использует класс ApiClient из пакета brmapp.api;
- ApiClient является singleton-классом содержащий методы взаимодействия с сервером. В качестве клиента используется класс RestServiceFactory;
- RestServiceFactory содержит в себе единственный метод create(), принимающий интерфейс взаимодействия (например, RestService) и возвращающий инициализированный объект Retrofit с уже настроенным временем timeout, обработчиком JSON и базовым URI сервера;
- RestService является интерфейсом, в котором с помощью аннотаций библиотеки Retrofit описывается формат взаимодействия с сервером — метод запроса, параметры запроса, тело сообщения, возвращаемый результат.

**brmapp.tasks** Многие обязательные к выполнению задачи являются слишком трудоемкими, для размещения в UI-потоке. Как было сказано в пред. разделе, такие задачи нужно выносить в отдельный поток. Пакет brmapp.tasks содержит классы типа AsyncTask для асинхронного выполнения следующих действий:

- LoadSongsTask — загружает все аудиозаписи;
- LoadPlaylistSongsTask — загружает все аудиозаписи конкретного плейлиста;
- LoadPlaylistsTask — загружает все плейлисты;

- CreatePlaylistTask — создает плейлист;
- CheckBpm — определяет, у скольких аудиозаписей BPM еще не был определен.

**brtapp.util** Пакет brtapp.util содержит остальные необходимые для работы приложения классы:

- Constants — используется для хранения различных строковых констант;
- MP3Decoder — класс, выполняющий декодирование файлов формата mp3 с помощью классов MediaCodec и MediaExtractor;
- BPMaster — класс, объединяющий в себе всю логику вычисления темпа музыкальной композиции.

#### 4.2.3. Особенности реализации

**Декодирование** Начиная с версии API 16 (Android 4.1), был представлен класс MediaCodec, обеспечивающий прямой доступа к медиакодекам (компонентам кодера/декодера). MediaCodec является частью низкоуровневой инфраструктуры Android для работы с мультимедиа и обычно используется с вместе с такими классами как MediaExtractor, MediaSync, MediaMuxer, MediaCrypto, MediaDrm, Image, Surface и AudioTrack.

В общем случае MediaCodec получает данные для декодирования, обрабатывает их и генерирует результат. Обработка происходит в асинхронном режиме с использованием набора входных и выходных буферов.

Рабочий процесс выглядит следующим образом: клиент запрашивает у кодека пустой буфер для последующего заполнения закодированными данными; после заполнения клиент передает буфер на обработку в кодек. Кодек декодирует эти данные и сохраняет в одном из своих выходных буферов. Клиент запрашивает у кодера декодированные данные, считывает их и возвращает пустой буфер обратно в кодек. Рабочий процесс схематично изображен на рисунке 24.

Кодек может работать с тремя видами данных: сжатыми, необработанными аудиоданными и необработанными видеоданными.

При работе с аудиоданными выходные буфера кодека заполняются PCM-фреймами, представляющими собой чередующуюся последовательность семплов для каждого канала. Семплы представляется в 16-битном формате.

В течение жизненного цикла (см. рис. 25) кодек пребывает в одном из трех состояний: остановлен, выполняется или освобожден. При создании экземпля-

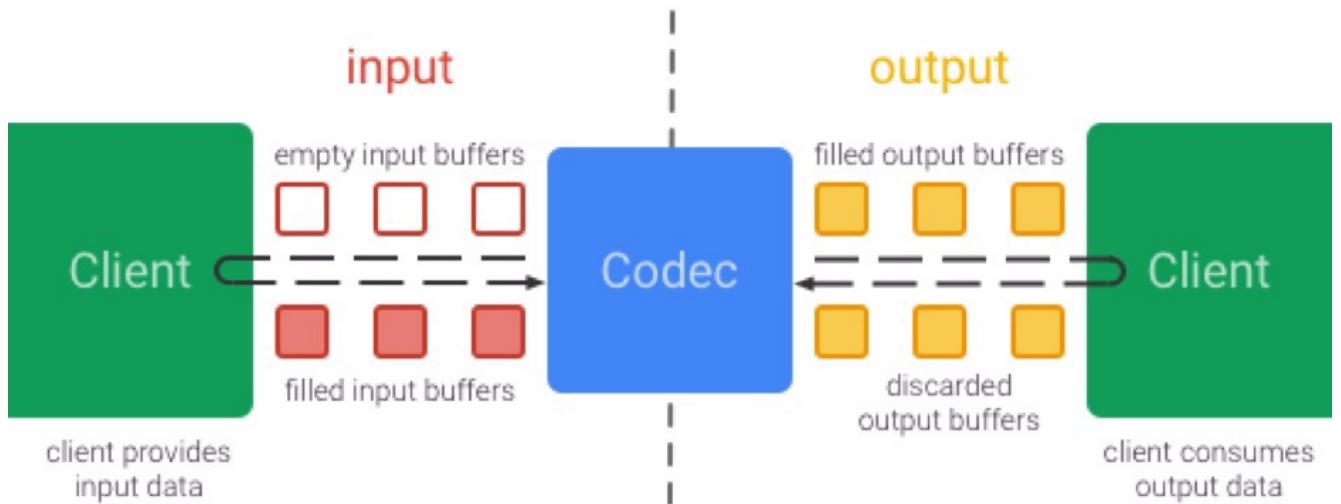


Рисунок 24 – Процесс работы MediaCodec

ра кодек находится в неинициализированном состоянии, поэтому изначально его нужно настроить посредством вызова метода `configure()`. После настройки кодек переходит в состояние `configured` и ожидает запуска через вызов метода `start()`. Запущенный кодек характеризуется состоянием `executed`.

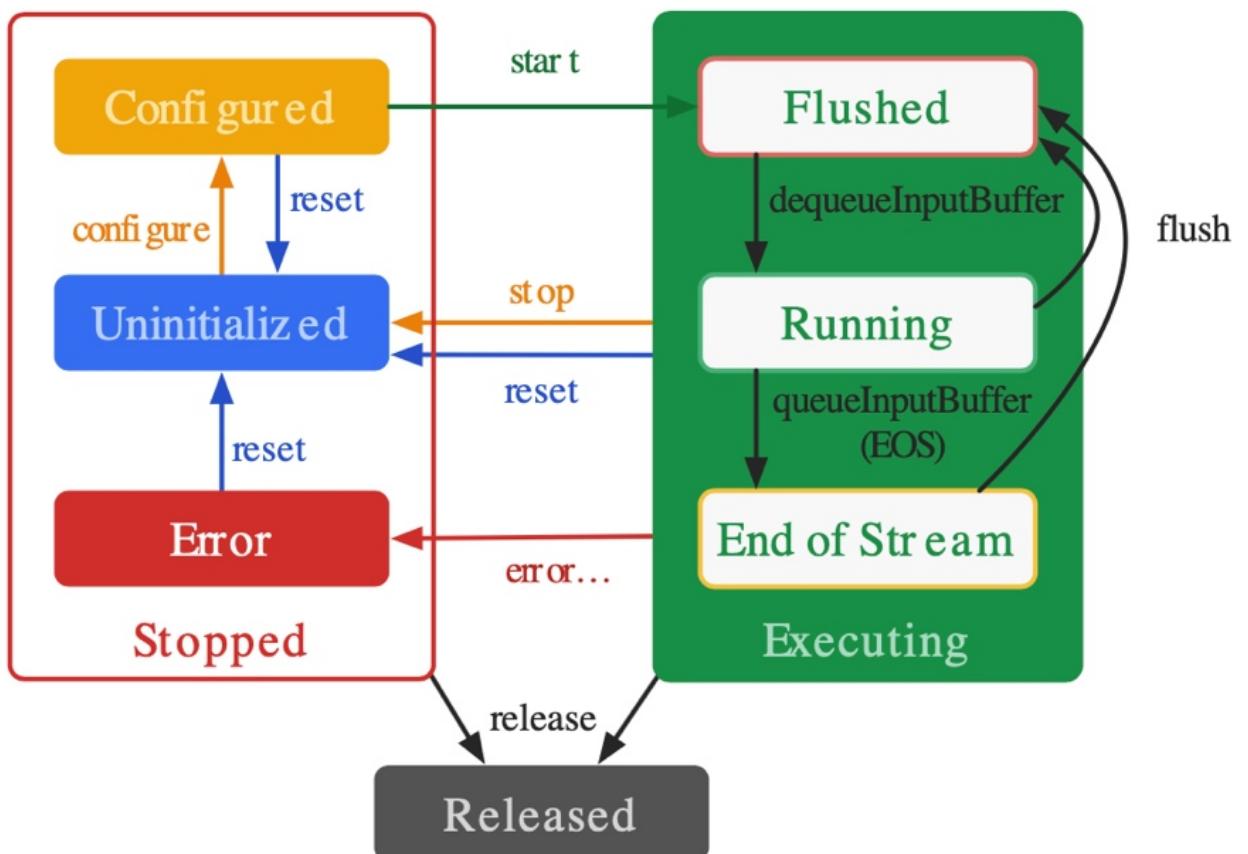


Рисунок 25 – Жизненный цикл MediaCodec

После выполнения указанных операций становится возможной манипуляция с очередями буферов, рассмотренная ранее.

После инициализации и запуска кодека ни входные, ни выходные буфера клиенту не доступны. Для отправки закодированных данных у кодека нужно запросить буфер через вызов метода `dequeueInputBuffer()`, заполнить его данными и отправить обратно, используя метод `queueInputBuffer()`. Кодек, в свою очередь, возвращает выходной буфер с обработанными данными в ответ на вызов метода `dequeueOutputBuffer()`. После обработки выходного буфера его необходимо вернуть обратно кодеку, вызвав метод `releaseOutputBuffer()`.

По завершению декодирования нужно вызвать метод `stop()`, возвращающий кодек в неинициализированное состояние, из которого он снова может быть настроен и использован. Для завершения работы с кодеком вызывается метод `release()`.

Исходный код реализации класса `MP3Decoder`, в котором происходит декодирование MP3-файлов с помощью класса `MediaCodec` находится в приложении Б.

### 4.3. Серверная часть

Второй частью выбранной архитектуры выступает сервер, вычисляющий ВРМ. Данный компонент не хранит никакие данные, не реализует авторизацию пользователей, не осуществляет кэширование. Рассмотрим серверную часть подробнее.

#### 4.3.1. Выбор инструментария

**Язык программирования** Python — это интерпретируемый объектно-ориентированный язык, он поддерживается практически во всех операционных системах и большинство его модулей распространяется свободно [1, 3]. Python используется для разработки приложений с графическим интерфейсом, для работы с базами данных, для создания web-сайтов и многое другое. Язык программирования Python отличается лапидарным синтаксисом и является одним из наиболее подходящих для программирования математических вычислений.

**Фреймворк для web-разработки** Для упрощения разработки web-приложений, воспользуемся готовым решением для Python — Flask.

Flask — микрофреймворк для создания web-приложений. Данный фреймворк относится к категории минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности. Данная особенность ускоряет разработку и минимизирует количество ошибок. Пример простого Flask-сервера представлен в листинге 4.

#### Листинг 4 – Простой Flask-сервер

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

**Модули для математических вычислений** Как было сказано раньше, Python хорошо подходит для математических вычислений, одна из причин — наличие большого количества модулей, подходящих для научных расчетов. В списке представлены некоторые из них:

- NumPy — добавляет в Python поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокогоуровневых (и очень быстрых) математических функций для операций с этими массивами;
- SciPy — библиотека, основанная на NumPy и предназначенная для выполнения научных и инженерных расчётов;
- Matplotlib — библиотека для визуализации данных двумерной графикой.

#### 4.3.2. Особенности реализации

Разработанный web-сервер состоит из нескольких функций. С помощью фреймворка Flask была разработана функция calculate(), которая запускается при получении запроса от клиента. После изъятия параметров запроса (sampleRate и channelsCount), вызывается функция bpmaster(), вычисляющая BPM музыкальной композиции, данные которой находятся в теле запроса. Данная функция реализует алгоритм, описанный в главе 3.2. Код разработанного сервера приведен в приложении B.

#### 4.4. Интерфейс приложения

Интерфейс обеспечивает комфортное взаимодействие пользователя с системой и положительный опыт, поэтому этапу разработки интерфейса часто уделяется большое количество внимания.

На рисунке 26 изображен разработанный главный экран, на который пользователь попадает при запуске приложения.

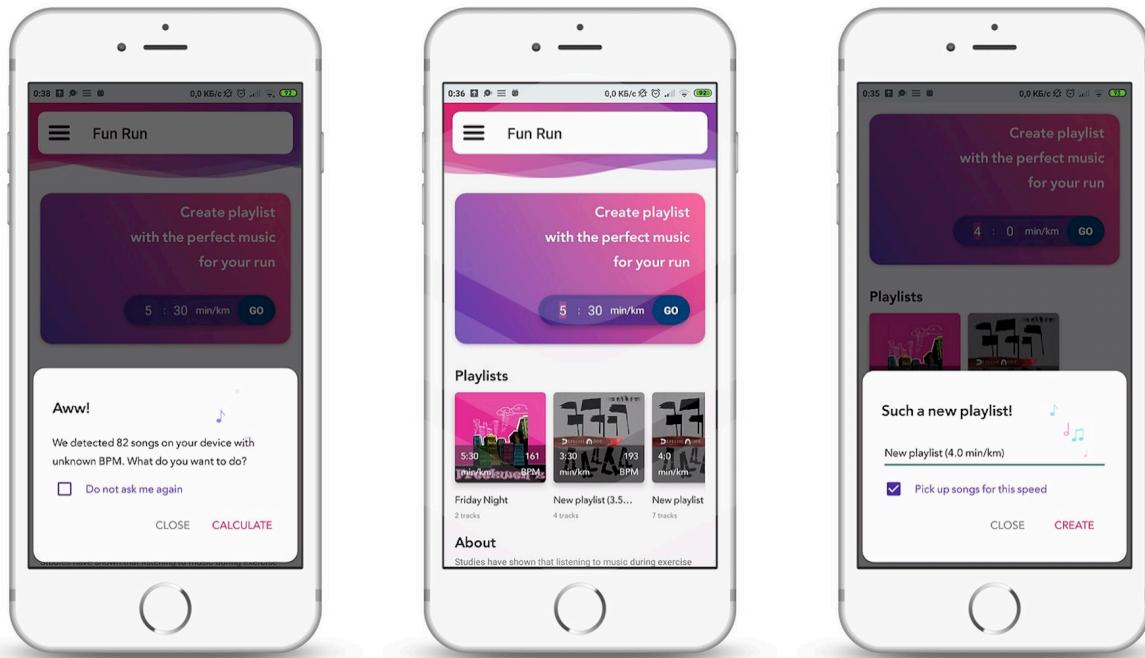


Рисунок 26 – Интерфейс разработанного приложения

На рисунке 27 изображен экран библиотеки, который представляет из себя два списка (плейлистов и композиций) между которыми можно легко перемещаться, а также экран просмотра плейлиста.

Интерфейс приложения был разработан с использованием рекомендаций к дизайну мобильных приложений и с учетом современных трендов.

#### 4.5. Тестирование и верификация

Важным этапом разработки является тестирование системы. Ключевая функция разрабатываемого приложения — вычисление BPM — должна работать с наименьшим количеством ошибок и неточностей. Для тестирования данной функции возьмем за истину данные ранее рассмотренного сервиса <https://songbpm.com>. В таблице 3 представлены результаты тестирования:

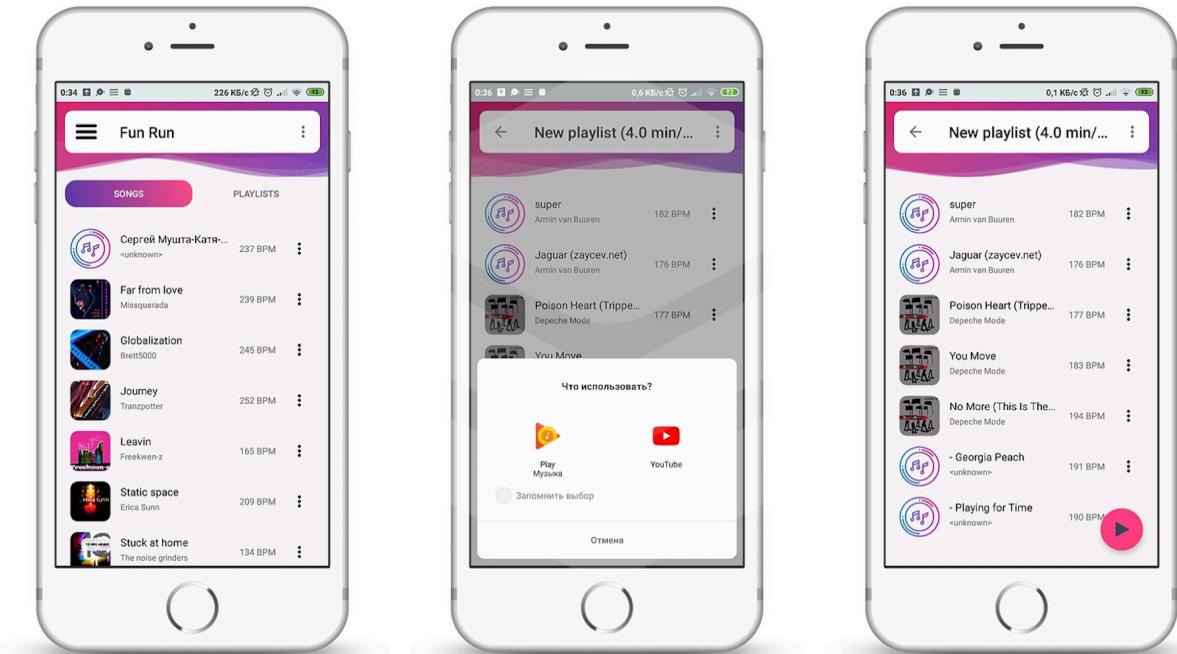


Рисунок 27 – Интерфейс разработанного приложения

Таблица 3 – Сравнение результатов тестирования

<i>N</i>	Композиция	<a href="https://songbpm.com">https://songbpm.com</a>	Приложение
1	Thirty Seconds To Mars - This Is War	160	158
2	Panic! At The Disco - High Hopes	82	162
3	Frank Sinatra - Jingle Bells	87	169
4	Ундервуд - ракеты на марс	158	158

В строках 2 и 3 результаты тестирования отличаются, но являются кратными. Такой результат означает более точное распознавание битов разработанным приложением.

Таким образом, предложенный способ вычисления BPM обеспечивает высокую точность, что подтверждается сравнением результатов со сторонними сервисами.

## ЗАКЛЮЧЕНИЕ

В предложенной работе решена актуальная задача — разработка мобильного приложения, позволяющего формировать плейлисты с отфильтрованными музыкальными композициями по темпу, необходимого для повышения эффективности тренировок в условиях роста заинтересованности в активном образе жизни.

При разработке был выполнен аналитический обзор существующих решений с выделением имеющихся в них недостатков. На основании наработанных материалов были сформулированы требования к разрабатываемому приложению.

В качестве ядра приложения — метода определения BPM — используется способ, основанный на использовании автокорреляционной функции и обладающий высокой точностью. Результаты работы адаптированного метода верифицированы путем сравнения с результатами вычислений альтернативных сервисов (<https://songbpm.com>).

Для обхода низкой производительности мобильных устройств использована клиент-серверная архитектура. В настоящее время наблюдается активное развитие мобильных сетей передачи данных (например, внедрение таких технологий как LTE и, в ближайшей перспективе, 5G), поэтому ограничения, накладываемые данной моделью в аспекте интернет-трафика следует считать несущественными.

Таким образом, на данный момент предложенное приложение не имеет аналогов и является перспективным принципиально новым решением.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

- 1 *Мартелли А., Рейвенскрофт А., Холден С.* Python. Справочник. Полное описание языка. — Диалектика, 2018. — 896 с.
- 2 Цифровая обработка сигналов: Учебное пособие по дисциплине «Цифровая обработка сигналов» / Ю. Н. Матвеев [и др.]. — Санкт-Петербург : СПбНИУ ИТМО, 2013. — 166 с.
- 3 *Саммерфилд М.* Программирование на Python 3. Подробное руководство. — Символ-Плюс, 2009. — 608 с.
- 4 *Таненбаум Э., Уэзеролл Д.* Компьютерные сети. — 5-е изд. — СПб : Питер, 2012. — 916 с.
- 5 *Умняшкин С. В.* Основы теории цифровой обработки сигналов: Учебное пособие. — 4-е изд. — Москва : Техносфера, 2018. — 528 с.
- 6 *Уолд А., Дейтел Х., Дейтел П.* Android для разработчиков. — Питер, 2016. — 512 с.
- 7 *Шилдт Г.* Java. Полное руководство. — Диалектика, 2018. — 1488 с.
- 8 *Android Developers.* Distribution dashboard [Электронный ресурс] / Google. — 2019. — URL: <https://developer.android.com/about/dashboards>.
- 9 *Bood R. J. [et al.]*. The Power of Auditory-Motor Synchronization in Sports: Enhancing Running Performance by Coupling Cadence with the Right Beats // PLOS ONE. — 2013. — Aug. — Vol. 8, no. 8. — P. 1–8. — DOI: 10.1371/journal.pone.0070758.
- 10 Effects of synchronous music on treadmill running among elite triathletes / P. Terry [et al.] // Journal of science and medicine in sport / Sports Medicine Australia. — 2011. — July. — Vol. 15. — P. 52–70. — DOI: 10.1016/j.jsams.2011.06.003.
- 11 *Kemp S.* Digital 2019: global Internet use accelerates [Электронный ресурс]. — 2019. — URL: <https://wearesocial.com/blog/2019/01/digital-2019-global-internet-use-accelerates>.
- 12 *Priest D., Karageorghis C.* Characteristics and effects of motivational music in sport and exercise // European Physical Education Review. — 2008. — Jan. — Vol. 14. — P. 351–371.

- 13 The Heat Is On: Effects of Synchronous Music on Psychophysiological Parameters and Running Performance in Hot and Humid Conditions / L. Nikol [et al.] // Frontiers in Psychology. — 2018. — Vol. 9. — P. 1114. — ISSN 1664-1078. — DOI: 10.3389/fpsyg.2018.01114.
- 14 TIOBE Index for May 2019 [Электронный ресурс] / TIOBE Software BV. — 2019. — URL: <https://www.tiobe.com/tiobe-index/>.

## ПРИЛОЖЕНИЕ А. ЖИЗНЕННЫЕ ЦИКЛЫ КОМПОНЕНТОВ ANDROID-ПРИЛОЖЕНИЯ

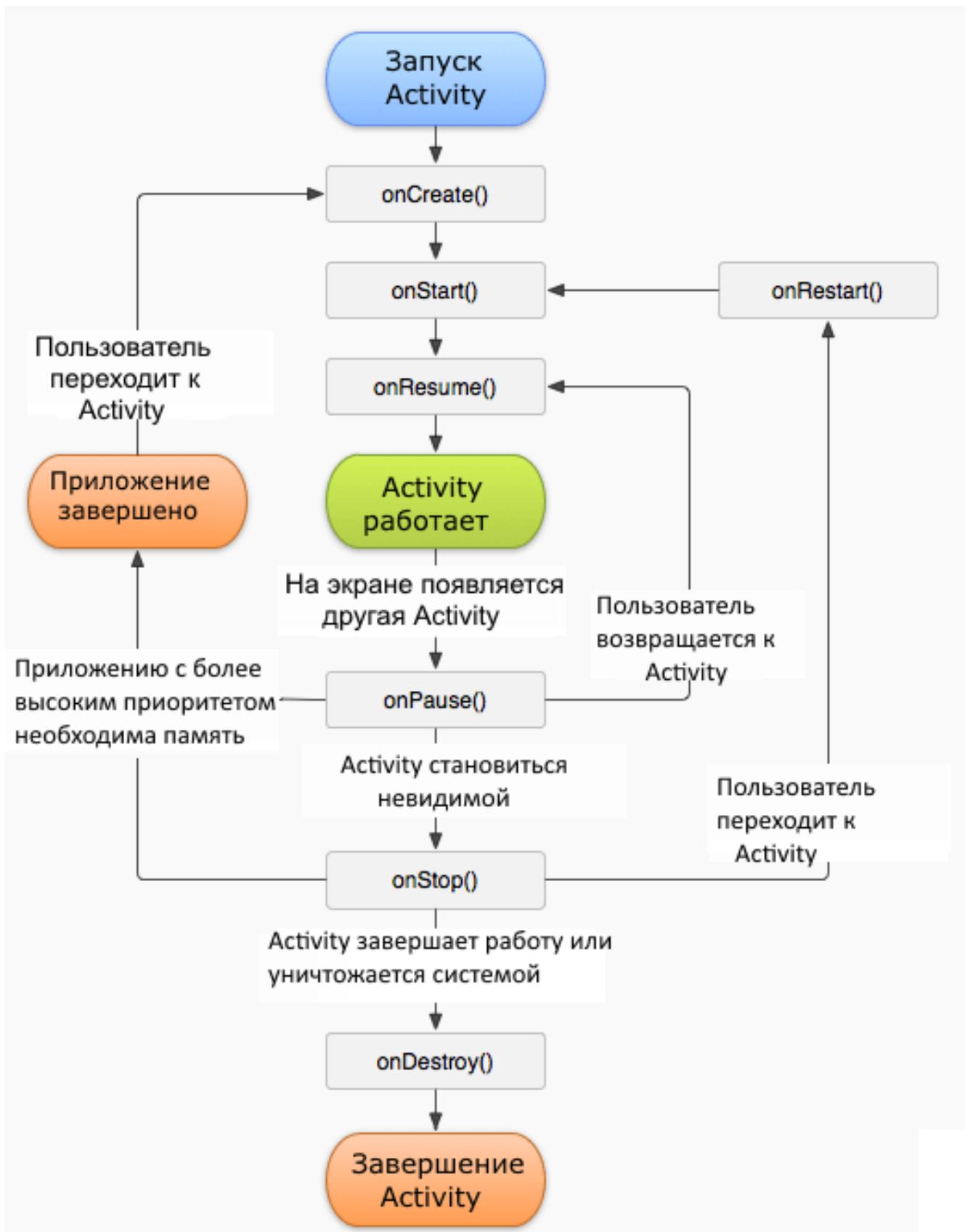


Рисунок А.1 – Жизненный цикл Activity



Рисунок А.2 – Жизненный цикл фрагмента

## ПРИЛОЖЕНИЕ Б. РЕАЛИЗАЦИЯ МР3-ДЕКОДЕРА

Листинг Б.1 – Реализация MP3-декодера

```

public class MP3Decoder {
    private final int SAMPLE_SIZE_IN_BYTES = 2;
    private final int MICROSECONDS_IN_SECONDS = 1000000;
    private final int BUFFER_WAIT_TIMEOUT = 5000;

    private MediaFormat format;
    private MediaExtractor extractor;
    private MediaCodec decoder;

    /**
     * @param filename audio file to process
     * @param secondsToDecode how many seconds to process
     * @param secondsToSkip how many seconds to skip
     * @return array of PCM data
     * @throws IOException
     */
    public byte[] decode(String filename, int secondsToDecode, int
        secondsToSkip) throws IOException {
        init(filename);
        byte[] bytes = getDecodedBytes(secondsToDecode, secondsToSkip
            );
        release();

        return bytes;
    }

    private void init(String filename) throws IOException {
        extractor = new MediaExtractor();
        extractor.setDataSource(filename);
        extractor.selectTrack(0);

        format = extractor.getTrackFormat(0);
        decoder = MediaCodec.createDecoderByType(format.getString(
            MediaFormat.KEY_MIME));
        decoder.configure(format, null, null, 0);
        decoder.start();
    }

    private void release() {

```

```

        decoder.stop();
        decoder.release();
        extractor.release();
    }

    public int getSampleRate() {
        return format.getInteger(MediaFormat.KEY_SAMPLE_RATE);
    }

    public int getChannels() {
        return format.getInteger(MediaFormat.KEY_CHANNEL_COUNT);
    }

    /**
     * Decodes raw MP3 to byte array
     * @param secondsToDecode how many seconds to process
     * @param secondsToSkip how many seconds to skip
     * @return array of decoded bytes, splited by channels [channel1,
     *         channel2, channel1, channel2]
     */
    private byte[] getDecodedBytes(int secondsToDecode, int
secondsToSkip) {
        extractor.seekTo(MICROSECONDS_IN_SECONDS * secondsToSkip,
MediaExtractor.SEEK_TO_CLOSEST_SYNC);

        MediaCodec.BufferInfo info = new MediaCodec.BufferInfo();
        ArrayList<byte[]> dataList = new ArrayList<>();
        int totalLength = 0;

        while (totalLength < secondsToDecode * getSampleRate() *
getChannels() * SAMPLE_SIZE_IN_BYTES) {
            int inIndex = decoder.dequeueInputBuffer(
                BUFFER_WAIT_TIMEOUT);
            if (inIndex >= 0) {
                ByteBuffer buffer = decoder.getInputBuffer(inIndex);
                int sampleSize = extractor.readSampleData(buffer, 0);
                if (sampleSize < 0) {
                    decoder.queueInputBuffer(inIndex, 0, 0, 0,
                        MediaCodec.BUFFER_FLAG_END_OF_STREAM);
                    break;
                } else {

```

```

        decoder.queueInputBuffer(inIndex, 0, sampleSize,
            extractor.getSampleTime(), 0);
        extractor.advance();
    }
}

int outIndex = decoder.dequeueOutputBuffer(info,
    BUFFER_WAIT_TIMEOUT);
switch (outIndex) {
    case MediaCodec.INFO_OUTPUT_FORMAT_CHANGED:
    case MediaCodec.INFO_TRY_AGAIN_LATER:
        // whatever
        break;
    default:
        ByteBuffer buffer = decoder.getOutputBuffer(
            outIndex);
        int length = info.size - info.offset;
        totalLength = totalLength + length;
        int p = buffer.position();
        byte[] tempBuffer = new byte[length];
        buffer.get(tempBuffer, 0, length);
        dataList.add(tempBuffer);
        buffer.position(p);
        decoder.releaseOutputBuffer(outIndex, true);
    }
}

byte[] data = new byte[totalLength];
int offset = 0;
for (byte[] tempBuffer : dataList) {
    System.arraycopy(tempBuffer, 0, data, offset, tempBuffer.
        length);
    offset += tempBuffer.length;
}

return data;
}
}

```

## ПРИЛОЖЕНИЕ В. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ

Листинг В.1 – Реализация серверной части

```

import scipy
import math
from scipy.signal import find_peaks
import numpy as np

from flask import Flask
from flask import request

app = Flask(__name__)

@app.route('/', methods=['POST', 'GET'])
def calculate():
    sampleRate = int(request.args.get('sampleRate'))
    channels = int(request.args.get('channels'))

    result = str(bpmaster(request.data, sampleRate, channels))

    return result


def bpmaster(string, sample_rate, channels):
    samples = np \
        .frombuffer(string, dtype=np.dtype([('re', np.int16), ('im', np.int16)])) \
        .view(np.int16).astype(np.float32).view(np.complex64)

    rms_samples = list(map(lambda x: abs(x), samples))

    next_power_of_two = math.ceil(math.log(len(rms_samples), 2))

    deficit = int(math.pow(2, next_power_of_two) - len(rms_samples))
    arr = np.concatenate((np.zeros(deficit), rms_samples))

    A = np.fft.fft(arr)
    S = A * np.conj(A)
    C = np.fft.ifft(S)

    positive_autocorrelation = C[0:5 * sample_rate]

```

```
peaks = find_peaks(positive_autocorrelation, distance=5000)
result = [x for x in list(map(lambda x: int(60 * sample_rate / x),
, peaks[0]))) if x < 300]

if (len(result) != 0):
    return result[0]

return -1

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```