МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА — Российский технологический университет»

Институт: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дисциплина: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КУРСОВАЯ РАБОТА**

на тему:

«Мобильное приложение Playlist Maker (плейлисты и треки)»

Выполнил(а): студент(ка) группы \_\_\_\_\_\_\_\_\_\_\_\_ Альбахтин Илья \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил(а): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва — 2026

# СОДЕРЖАНИЕ

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящем отчёте применяются следующие термины с соответствующими определениями.

Плейлист — пользовательская подборка музыкальных треков, объединённых общей темой или назначением.

Трек — единица аудиоконтента (композиция), имеющая метаданные: название, исполнителя, длительность и др.

Избранное — список треков, помеченных пользователем как понравившиеся для быстрого доступа.

История поиска — набор последних поисковых запросов пользователя для ускорения повторных поисков.

Репозиторий — слой приложения, предоставляющий единый API для работы с данными из разных источников (сеть/БД/хранилище настроек).

Состояние экрана — набор данных, описывающий текущий UI (например: «поиск», «успех», «ошибка»), используемый для реактивного обновления интерфейса.

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчёте применяются следующие сокращения и обозначения.

API — Application Programming Interface

DAO — Data Access Object

DFD (Data Flow Diagram) — диаграмма потоков данных

IDEF0 (Integrated DEFinition) — методология функционального моделирования

JSON — JavaScript Object Notation

MVVM — Model–View–ViewModel

REST — Representational State Transfer

SDK — Software Development Kit

SQL — Structured Query Language

UI — User Interface

UX — User eXperience

Содержание

[СОДЕРЖАНИЕ 2](#_Toc220013734)

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 3](#_Toc220013735)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 4](#_Toc220013736)

[ВВЕДЕНИЕ 6](#_Toc220013737)

[1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 7](#_Toc220013738)

[1.1 Описание предметной области 7](#_Toc220013739)

[1.2 Анализ существующих аналогов 8](#_Toc220013740)

[1.2.1 Spotify 8](#_Toc220013741)

[1.2.2 Яндекс Музыка 14](#_Toc220013742)

[1.3 Техническое задание 20](#_Toc220013743)

[1.3.1 Определение пользовательских требований 20](#_Toc220013744)

[1.3.2 Определение функциональных требований 20](#_Toc220013745)

[2 ПРОЕКТНАЯ ЧАСТЬ 26](#_Toc220013746)

[2.1 Описание процесса в нотации IDEF0 26](#_Toc220013747)

[2.2 Описание процесса в нотации DFD 28](#_Toc220013748)

[2.3 Описание и обоснование выбора программного обеспечения 29](#_Toc220013749)

[2.4 Архитектура программной системы 30](#_Toc220013750)

[3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ 32](#_Toc220013751)

[3.1 Описание моделей и структур данных 32](#_Toc220013752)

[3.2 Тестирование программного продукта 33](#_Toc220013753)

[3.3 Полный листинг (структура проекта и ключевые интерфейсы) 49](#_Toc220013754)

[ЗАКЛЮЧЕНИЕ 68](#_Toc220013755)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 69](#_Toc220013756)

# ВВЕДЕНИЕ

Целью данной курсовой работы является разработка прототипа мобильного приложения для платформы Android,

предназначенного для поиска музыкальных треков и управления пользовательскими плейлистами и избранным.

Приложение «Playlist Maker» предоставляет интерфейс для поиска треков по названию/исполнителю, просмотра

карточки трека с расширенными метаданными, добавления треков в избранное, а также создания и управления плейлистами.

Актуальность темы обусловлена ростом объёма цифрового музыкального контента и потребностью пользователей быстро

находить нужные композиции и организовывать их в удобные подборки. Мобильное приложение позволяет решать эту задачу

непосредственно на устройстве пользователя, обеспечивая быстрый доступ к избранным трекам и созданным плейлистам.

В рамках разработки приложения реализованы следующие основные задачи:

* разработка пользовательского интерфейса на Jetpack Compose с использованием Material 3;
* реализация навигации между экранами (главный экран, поиск, карточка трека, плейлисты, избранное, настройки);
* поиск треков через внешний REST API (iTunes Search API) с обработкой состояний «поиск», «успех», «ошибка», «пусто»;
* хранение данных в локальной базе данных Room (плейлисты, треки, отметка «избранное», принадлежность к плейлисту);
* сохранение истории поиска в локальном хранилище DataStore (до 10 последних запросов);
* реализация дополнительных действий в настройках: поделиться приложением, написать в поддержку, открыть пользовательское соглашение, а также переключение режима темы (системная/светлая/тёмная).

Объектом исследования выступает мобильное приложение для управления музыкальными подборками.

Предметом исследования являются методы проектирования Android-приложений, организация данных (сетевые DTO, доменные модели, сущности БД),

а также взаимодействие UI-слоя с локальными и удалёнными источниками данных.

В исследовательской части курсовой работы рассматривается предметная область и анализируются существующие аналоги.

В проектной части описываются процессы и архитектура приложения, приводятся диаграммы IDEF0 и DFD (в текстовом описании).

В технологической части рассматриваются основные модели данных, структура проекта и приводится перечень ключевых интерфейсов и сценариев тестирования.

# 1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

## 1.1 Описание предметной области

Предметная область данной курсовой работы связана с организацией и управлением музыкальным контентом на мобильных устройствах.

Пользователь взаимодействует с каталогом треков, формирует персональные подборки (плейлисты) и отмечает избранные композиции.

В рамках предметной области рассматриваются:

* поиск треков по текстовому запросу (название, исполнитель);
* просмотр карточки трека с дополнительными метаданными (обложка, год, альбом, длительность);
* хранение пользовательских данных локально (списки плейлистов и треков);
* ведение истории поиска для повторного использования запросов;
* управление избранными треками (добавление/удаление из избранного);
* управление плейлистами (создание, удаление, просмотр деталей плейлиста, перенос/слияние плейлистов);
* обеспечение удобной навигации и адаптивного UI в соответствии с рекомендациями Android.

Поскольку поиск треков требует актуальных данных, приложение использует внешний сервис (iTunes Search API).

Пользовательские действия, связанные с избранным и плейлистами, сохраняются локально, что обеспечивает доступность данных даже без подключения к сети.

## 1.2 Анализ существующих аналогов

Для формирования требований к приложению «Playlist Maker» был выполнен анализ популярных музыкальных сервисов и приложений,

предоставляющих пользователям функциональность поиска и создания плейлистов.

Ниже приведены примеры аналогов и их ключевые особенности.

### 1.2.1 Spotify

Описание:

Spotify — международный музыкальный сервис потокового прослушивания с развитой системой рекомендаций и плейлистов.

Пользователь может искать треки и артистов, сохранять музыку в медиатеку и создавать плейлисты.

Достоинства:

* быстрый поиск по большому каталогу;
* развитая система рекомендаций;
* удобное управление плейлистами и совместное редактирование;
* кроссплатформенность.

Недостатки:

* часть функциональности доступна только по подписке;
* зависимость от стабильного интернет-соединения для потокового прослушивания;
* сложность интерфейса из-за большого количества возможностей.

Покажем далее на рисунках интерфейс основных страниц приложения.

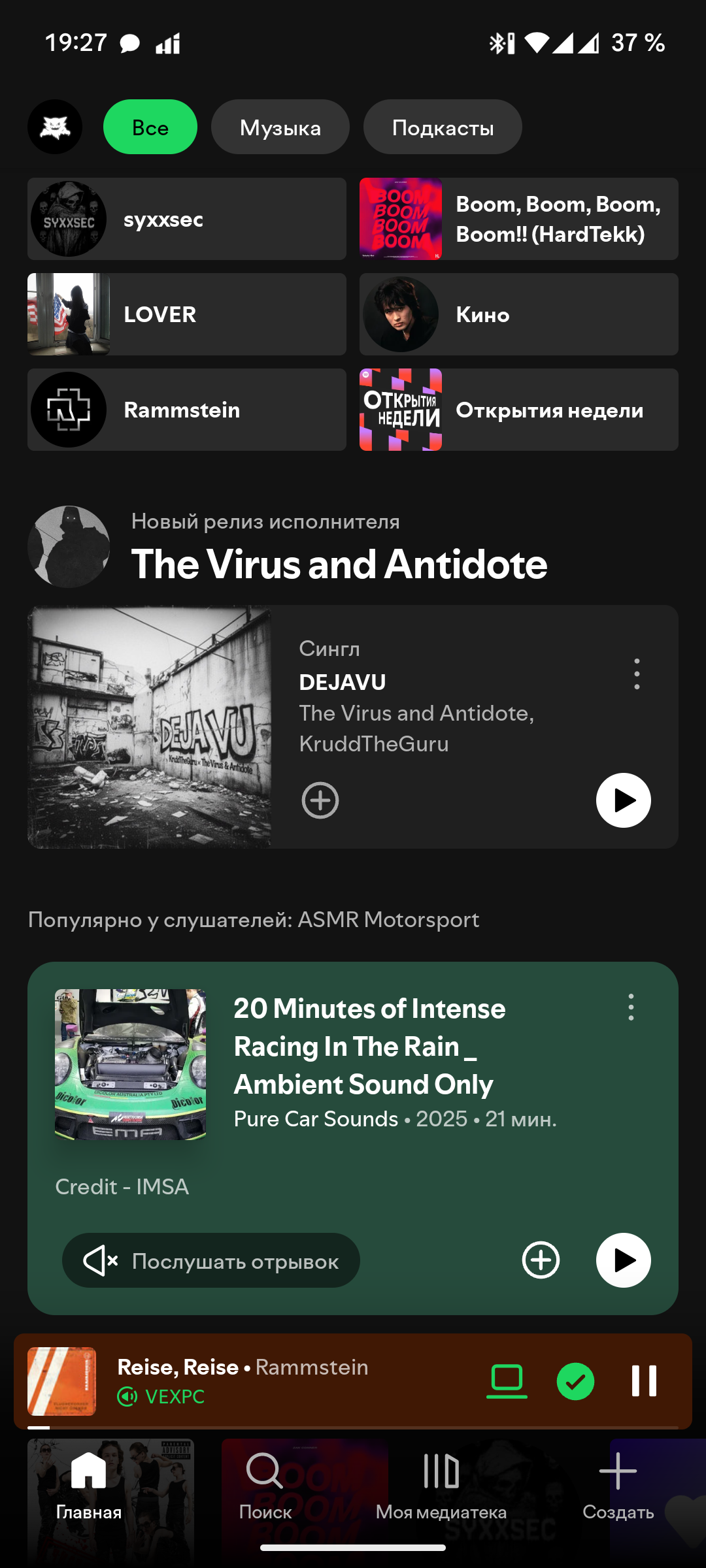


Рисунок 1 – Spotify: пример экрана №1

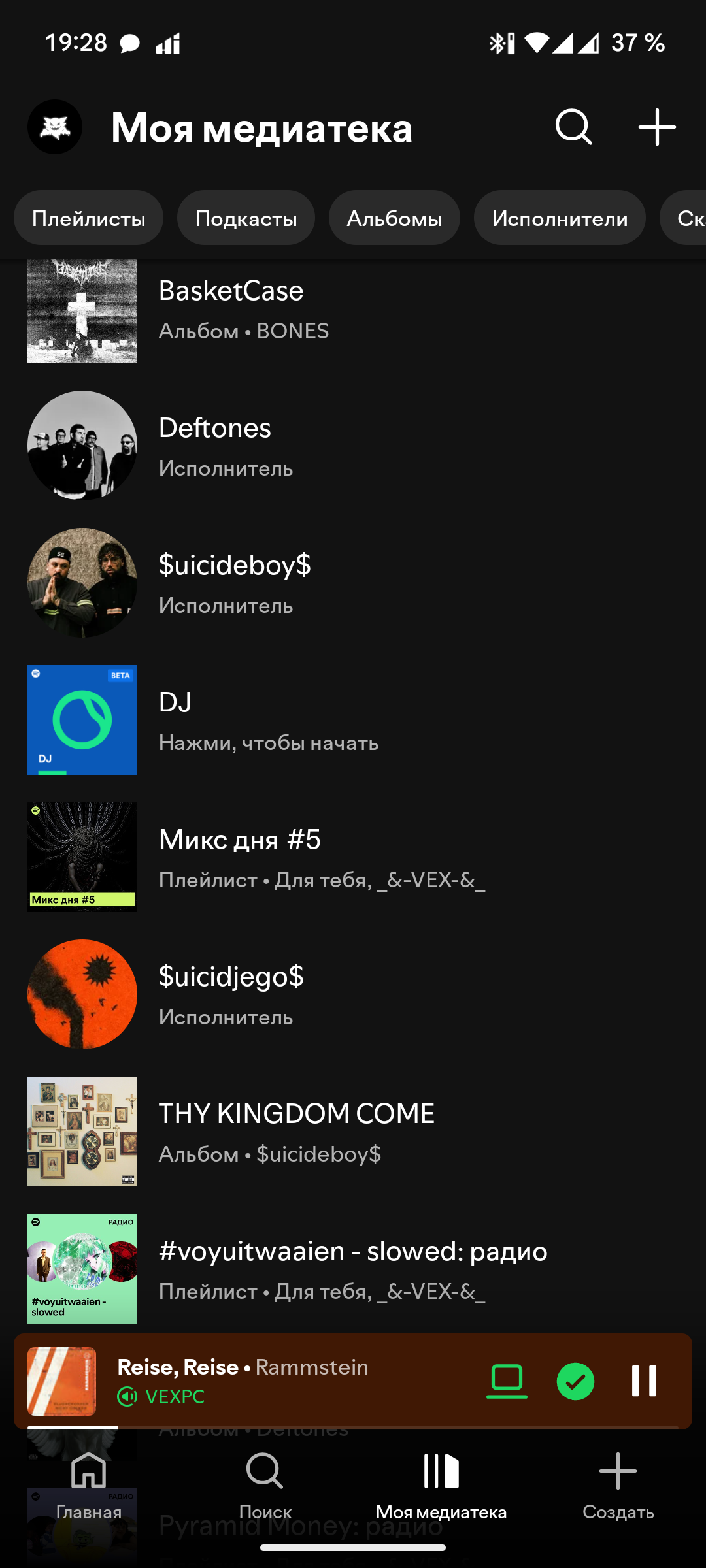


Рисунок 2 – Spotify: пример экрана №2

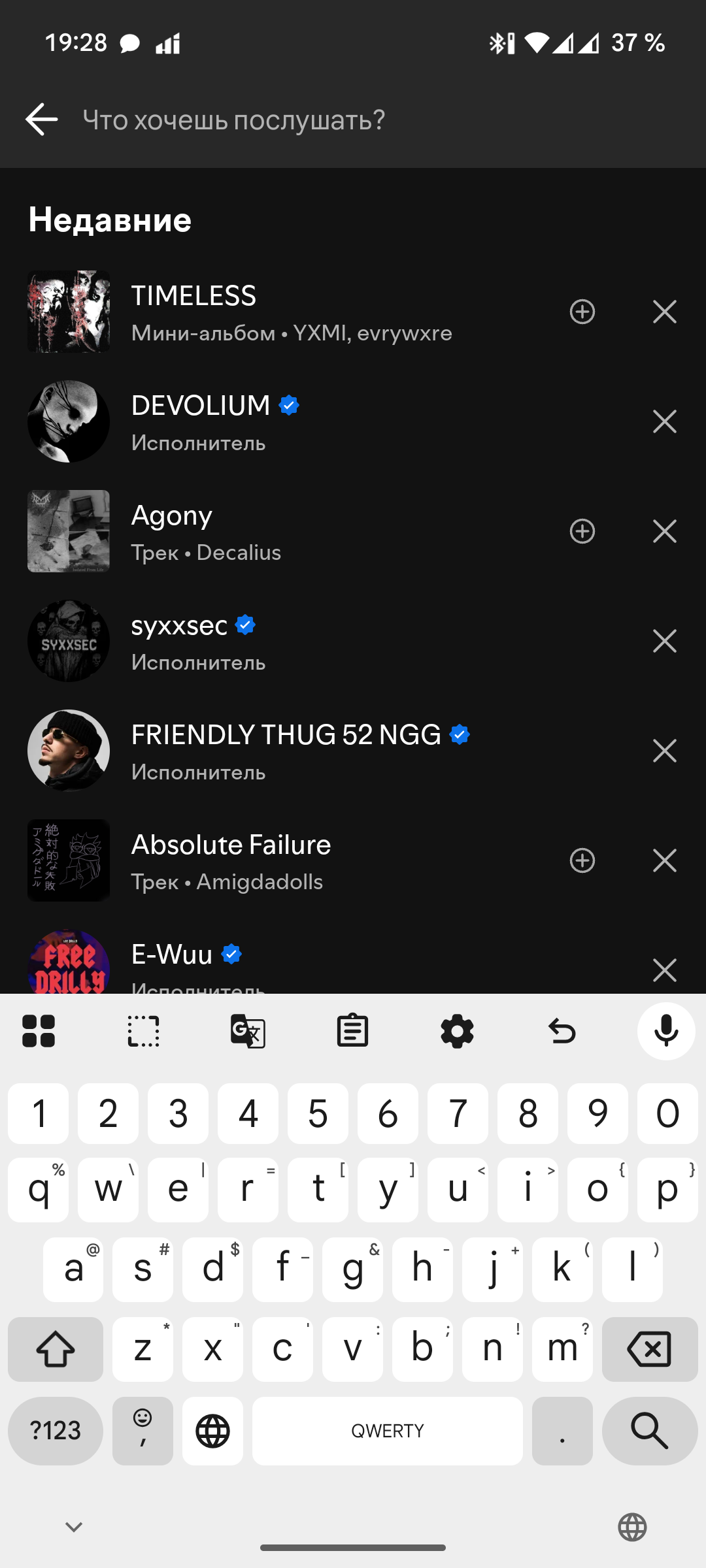


Рисунок 3 – Spotify: пример экрана №3

Отзывы пользователей Spotify

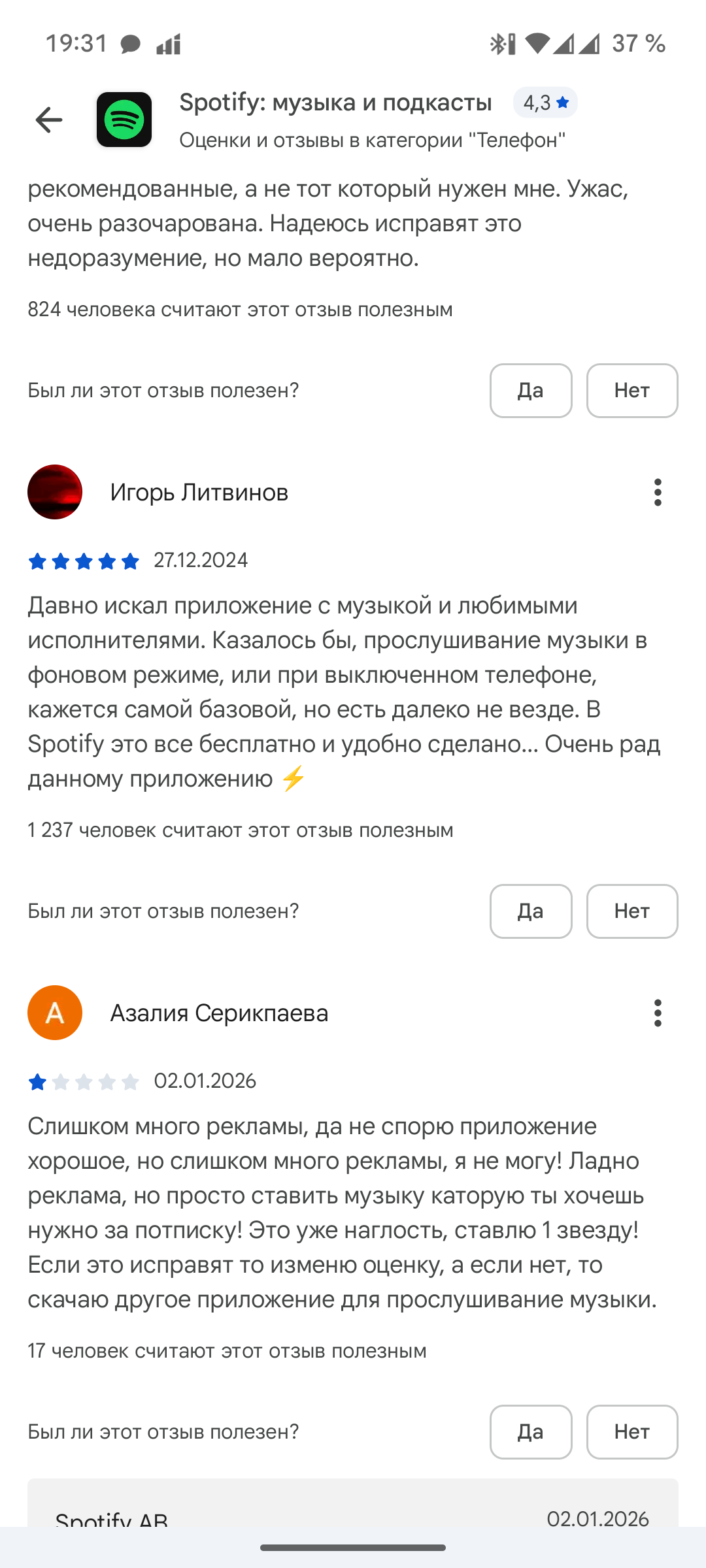


Рисунок 4 – Spotify: отзыв пользователя №1

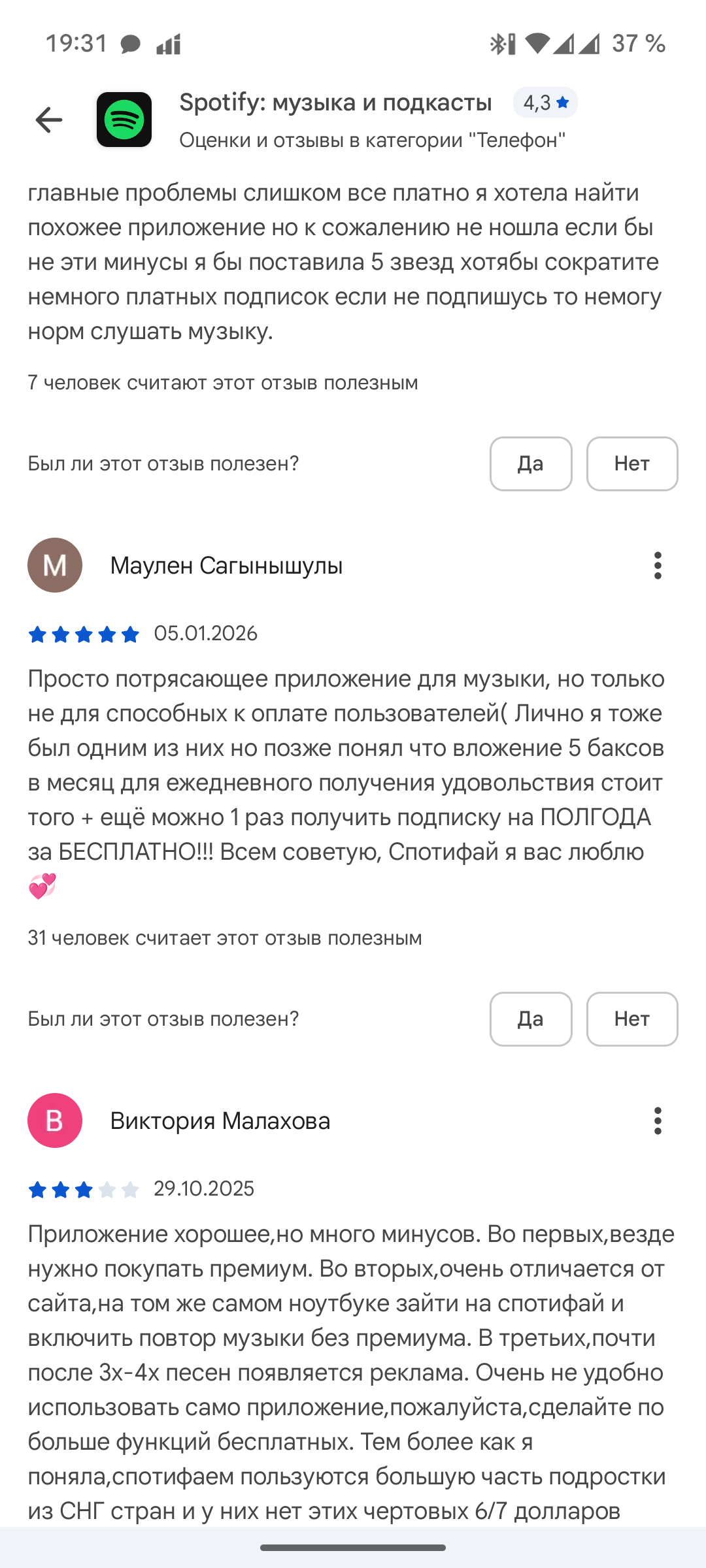


Рисунок 5 – Spotify: отзыв пользователя №2

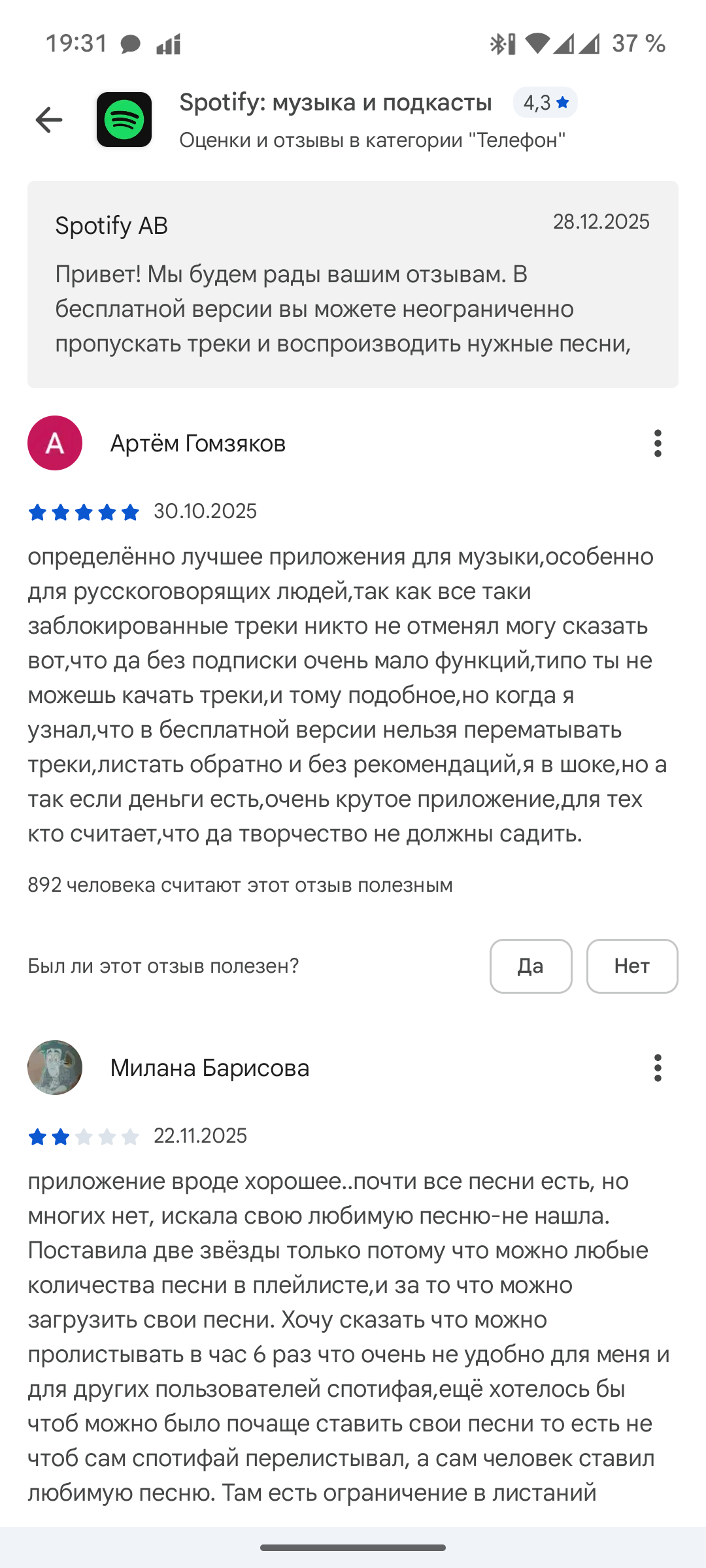


Рисунок 6 – Spotify: отзыв пользователя №3

### 1.2.2 Яндекс Музыка

Описание:

Яндекс Музыка — российский сервис для прослушивания музыки с акцентом на рекомендации и подборки.

В приложении доступны поиск, избранное и плейлисты.

Достоинства:

* качественная локализация и интеграция с экосистемой сервисов;
* персональные подборки и «Моя волна»;
* удобный интерфейс на мобильных устройствах.

Недостатки:

* часть функций доступна только с подпиской;
* высокий объём данных при активном использовании онлайн-режима;
* ограниченная применимость для учебных проектов из-за закрытых API.

Покажем далее на рисунках интерфейс основных страниц приложения.

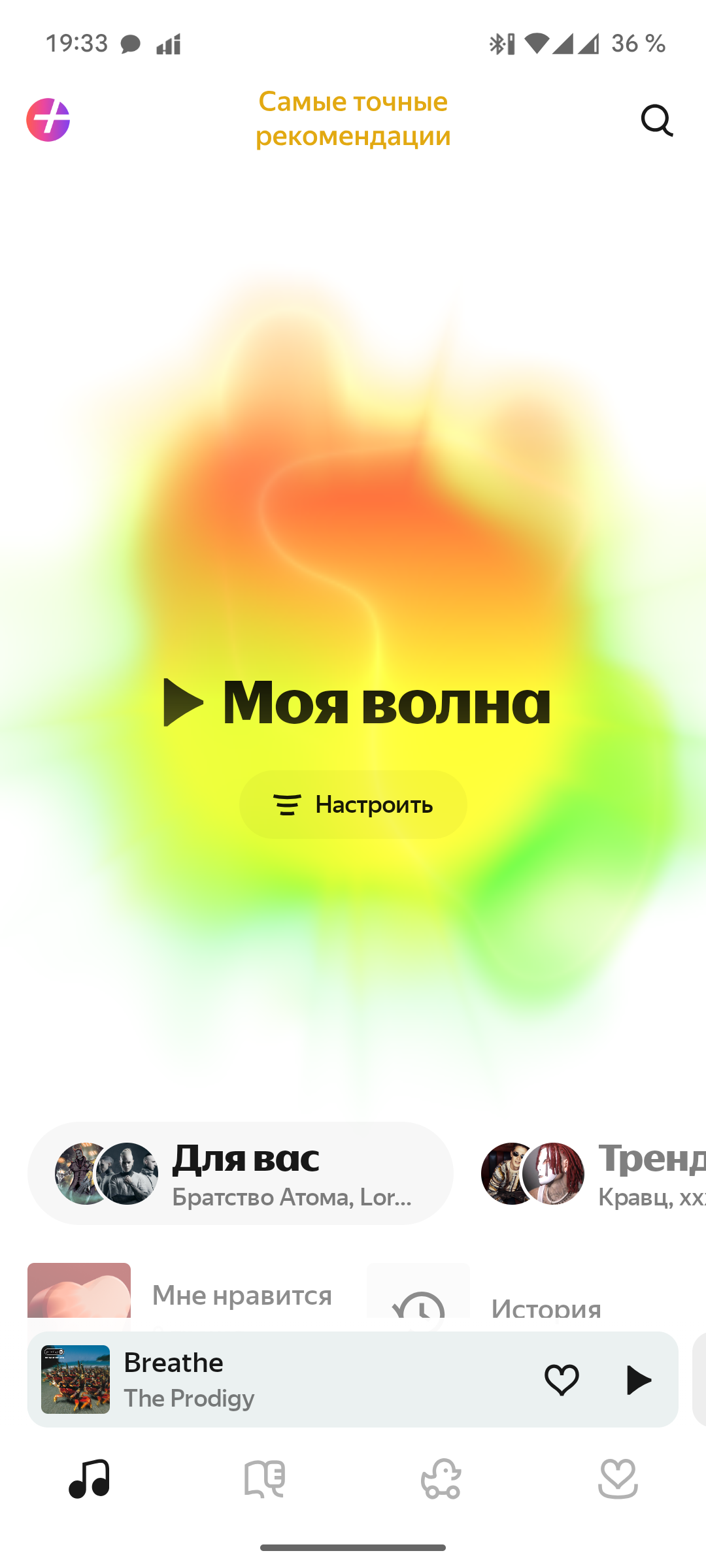


Рисунок 7 – Яндекс Музыка: пример экрана №1

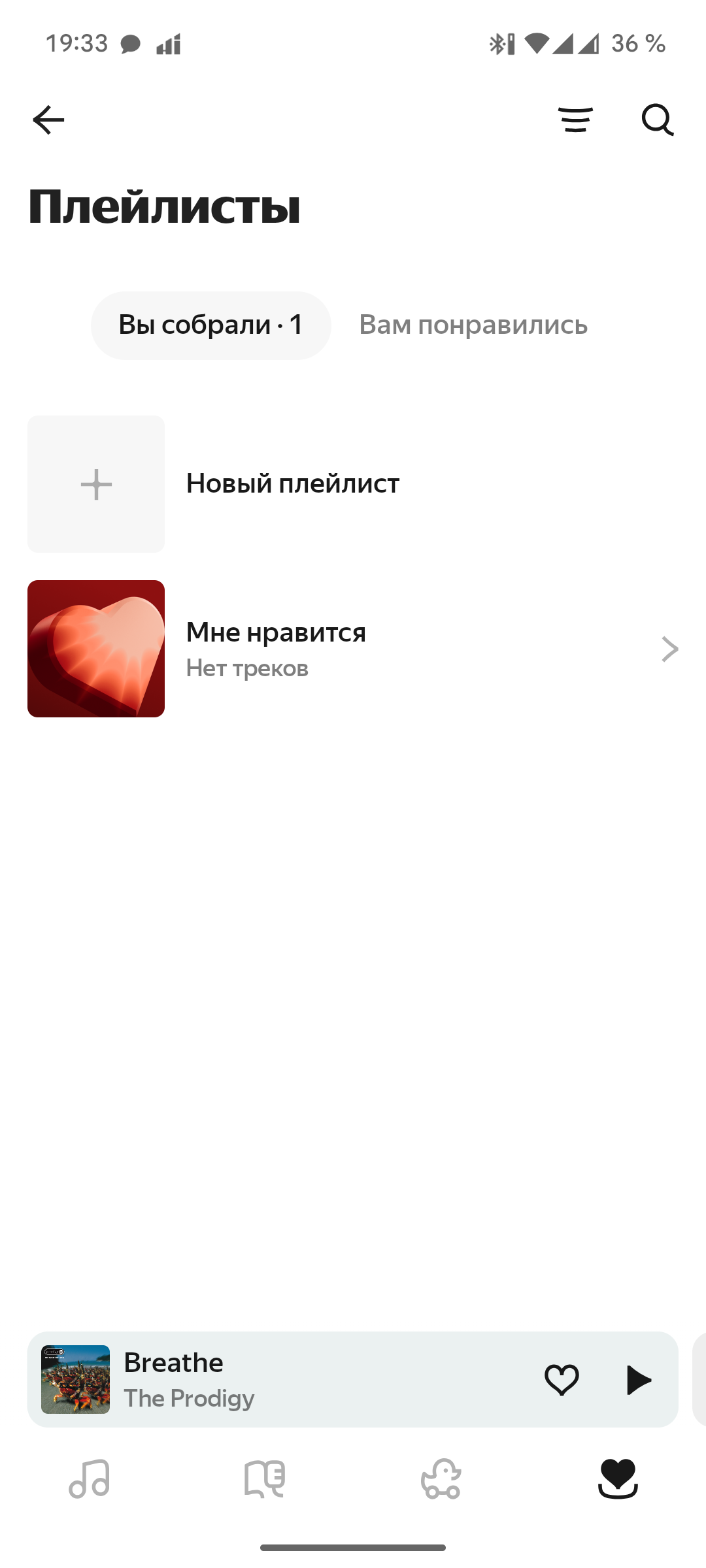
****

Рисунок 8 – Яндекс Музыка: пример экрана №2

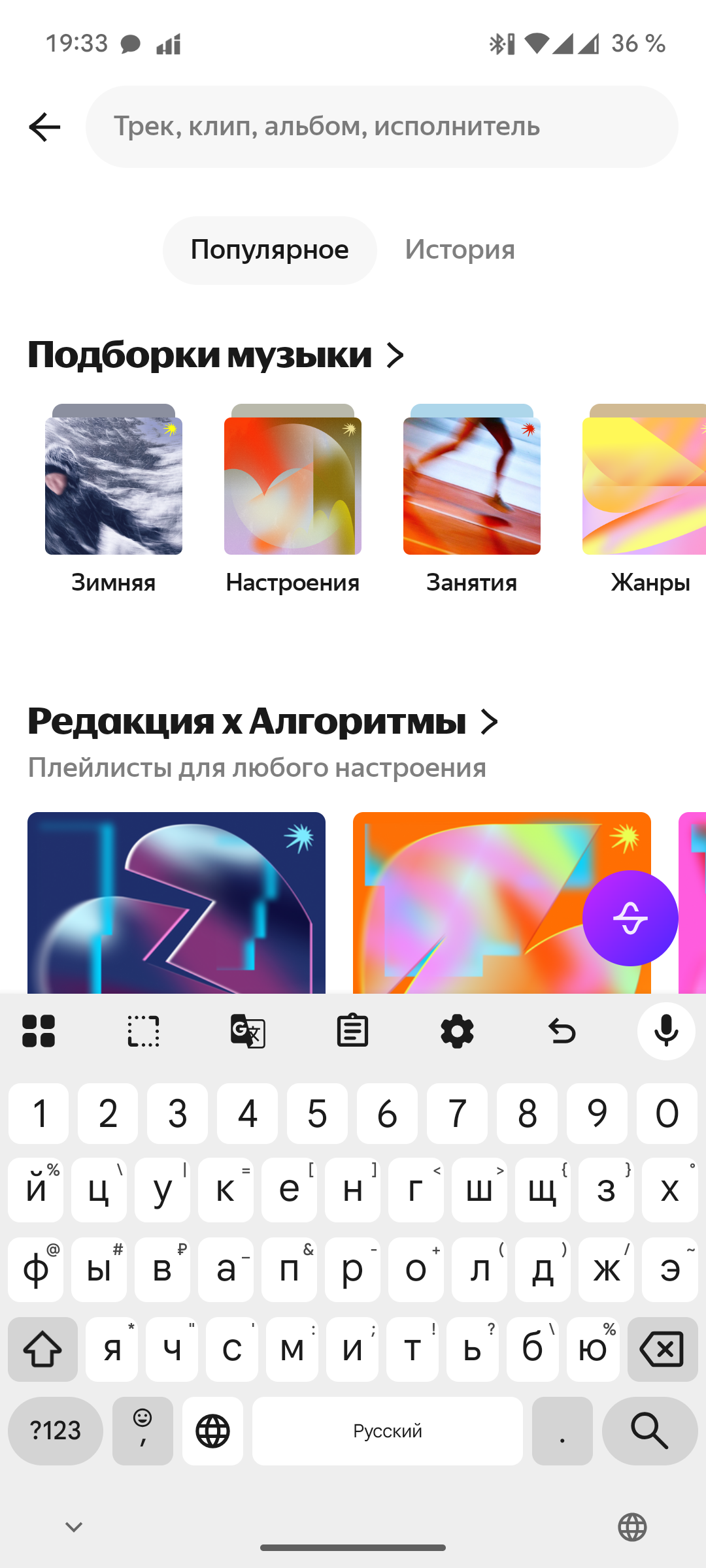
****

Рисунок 9 – Яндекс Музыка: пример экрана №3

Отзывы пользователей Яндекс Музыка

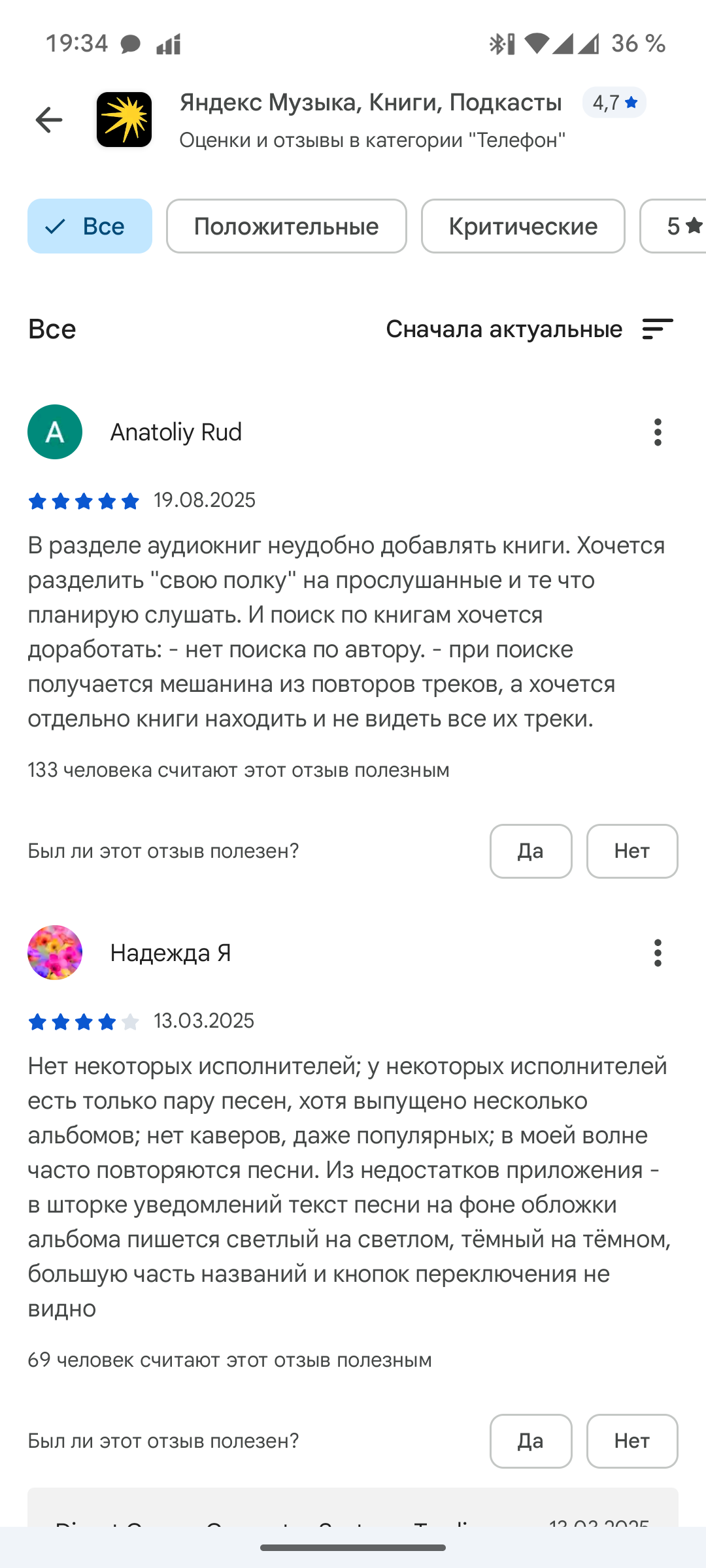


Рисунок 10 – Яндекс Музыка: отзыв пользователя №1

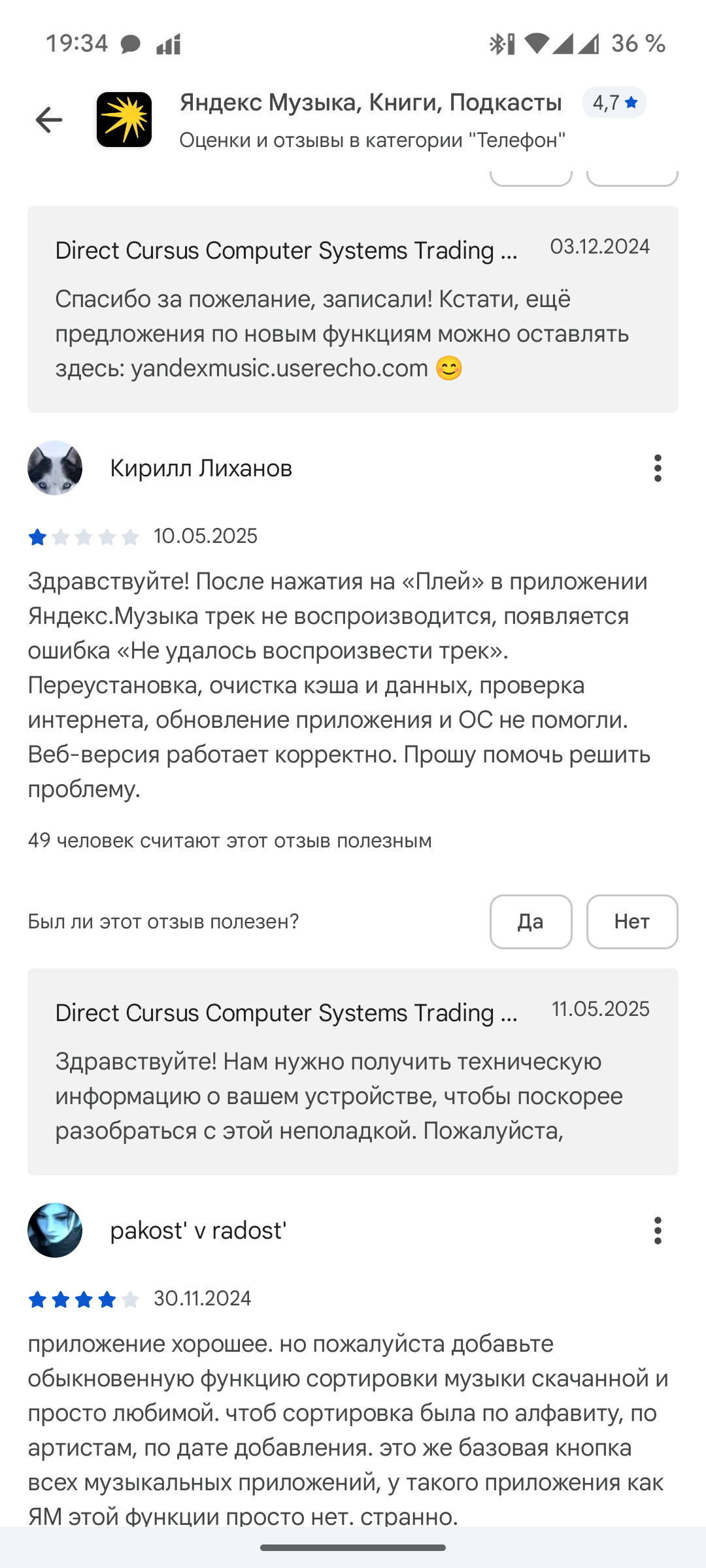
****

Рисунок 11 – Яндекс Музыка: отзыв пользователя №2

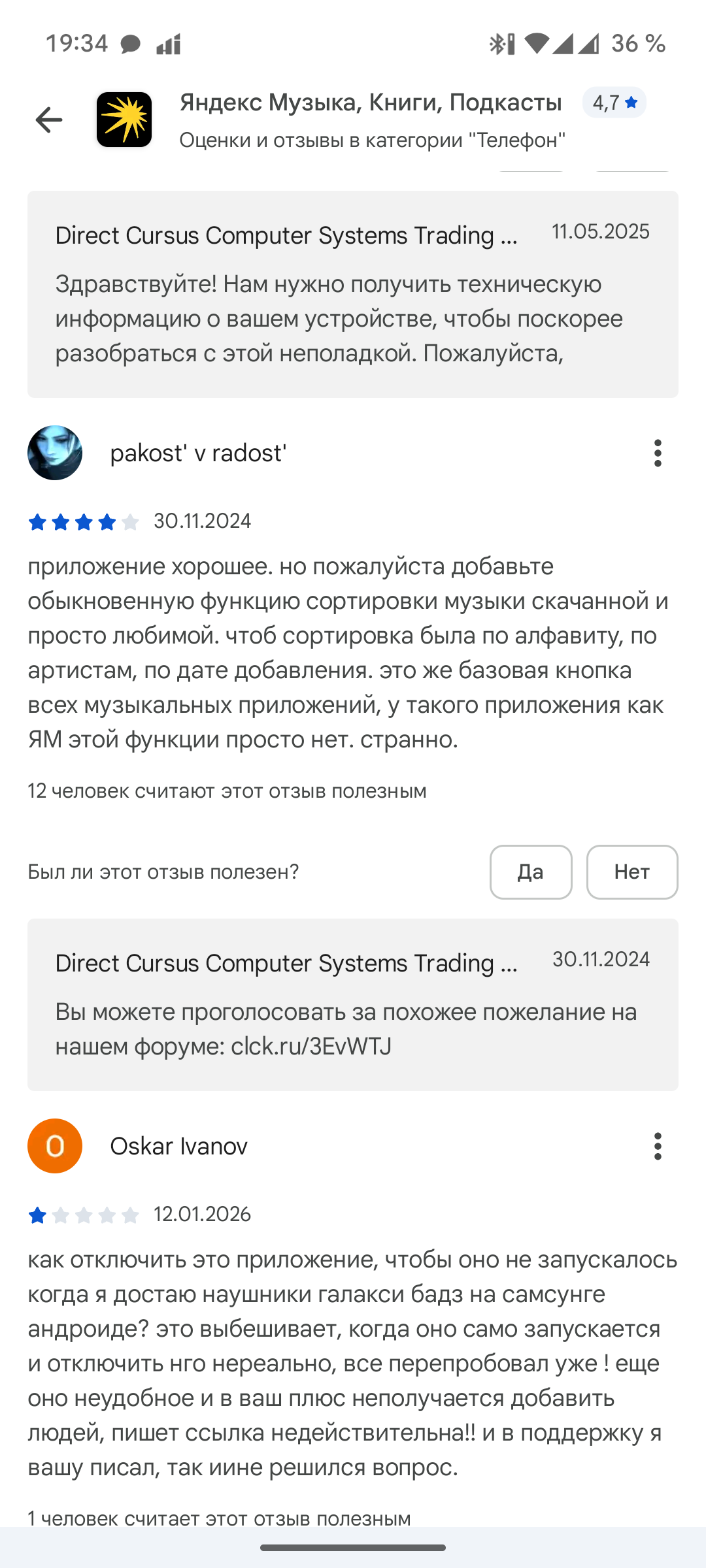
****

Рисунок 12 – Яндекс Музыка: отзыв пользователя №3

Вывод по аналогам:

Аналоги подтверждают важность функций поиска, избранного и плейлистов. Для учебного проекта целесообразно реализовать

ключевые сценарии управления музыкой, сохраняя простоту интерфейса и применяя стандартный стек Android-разработки.

Соединим полученные данные в таблицу сравнения (таблица 1).

Таблица 1 – Таблица сравнения существующих аналогов

|  |  |  |  |
| --- | --- | --- | --- |
| Приложение | Основные функции | Достоинства | Недостатки |
| Spotify | Поиск, плейлисты, рекомендации, медиатека | Большой каталог, рекомендации, кроссплатформенность | Часть функций по подписке, сложный интерфейс |
| Яндекс Музыка | Поиск, плейлисты, избранное, рекомендации | Локализация, удобство, интеграция с сервисами | Часть функций по подписке, закрытость API |

## 1.3 Техническое задание

### 1.3.1 Определение пользовательских требований

Пользователь приложения должен иметь возможность:

* перейти с главного экрана к разделам «Поиск», «Плейлисты», «Избранное», «Настройки»;
* выполнить поиск треков по текстовому запросу;
* видеть состояния результата поиска: первоначальное (подсказка), загрузка, успех, пустой результат, ошибка;
* выбрать трек из списка результатов и открыть карточку трека;
* добавить трек в избранное и удалить его из избранного;
* добавить выбранный трек в существующий плейлист;
* просмотреть список избранных треков;
* просмотреть список плейлистов;
* создать новый плейлист с названием, описанием и обложкой (выбор изображения);
* открыть карточку плейлиста и увидеть список треков в нём;
* удалить плейлист;
* удалить трек из плейлиста;
* объединить плейлисты (перенести треки из одного плейлиста в другой);
* посмотреть историю поиска и очистить историю;
* поделиться приложением через системное меню «Поделиться»;
* написать письмо в поддержку, используя почтовый клиент на устройстве;
* открыть пользовательское соглашение в браузере;
* изменить режим темы приложения (системная/светлая/тёмная).

### 1.3.2 Определение функциональных требований

Функциональные требования к системе включают следующие компоненты.

А) Поиск треков

* Ввод поискового запроса пользователем.
* Отправка сетевого запроса к iTunes Search API (GET /search) с параметрами term и entity.
* Преобразование ответа (JSON) в DTO и дальнейшее отображение доменных моделей в UI.
* Сохранение поискового запроса в историю (до 10 элементов).
* Возможность повторить последний запрос при ошибке («Обновить»).

Б) Карточка трека

* Отображение обложки трека и метаданных (название, исполнитель, год, альбом, длительность).
* Переключение статуса «Избранное» (добавить/убрать) с сохранением в локальную БД.
* Добавление трека в выбранный плейлист.

В) Избранное

* Отображение списка избранных треков (favorite = 1) из локальной БД.
* Возможность удалить трек из избранного.

Г) Плейлисты

* Создание плейлиста (name, description, coverImageUri) и сохранение в Room.
* Просмотр списка плейлистов и количества треков в каждом.
* Просмотр карточки плейлиста и списка треков, принадлежащих плейлисту.
* Удаление плейлиста и очистка/перенос его треков.
* Объединение плейлистов (перенос треков из sourceId в targetId и удаление sourceId).

Д) История поиска

* Хранение истории запросов в DataStore Preferences.
* Ограничение длины истории до 10 элементов.
* Очистка истории по действию пользователя.

Е) Настройки

* Системный шаринг (Intent ACTION\_SEND) с текстом о приложении.
* Письмо в поддержку (Intent ACTION\_SENDTO / mailto) с заданной темой и шаблоном текста.
* Открытие пользовательского соглашения по URL: https://practicum.yandex.ru/legal/student/
* Переключение режима темы (ThemeMode: SYSTEM/LIGHT/DARK).

Ключевые нефункциональные требования:

* поддержка устройств с Android 9 (API 28) и выше;
* корректная обработка ошибок сети и пустых результатов;
* сохранность пользовательских данных (плейлисты/избранное/история) между запусками приложения;
* отзывчивость интерфейса за счёт асинхронной работы с сетью и БД (корутины, Flow/StateFlow).

|  |  |  |  |
| --- | --- | --- | --- |
| **Стейкхолдер** | **Интерес/ценность** | | **Влияние на проект** |
| Пользователь приложения | Быстрый поиск и удобные плейлисты/избранное | | Высокое (формирует требования и UX) |
| Преподаватель/заказчик (университет) | Соответствие методичке и качественная защита | | Высокое (критерии оценки) |
| Разработчик (студент) | Получение результата и опыта, сдача проекта | | Высокое (реализация и сроки) |
| Поставщик данных (iTunes Search API) | Предоставление корректного API/доступности сервиса | | Среднее (влияет на качество поиска) |
| Платформа Android / магазин приложений | Требования к безопасности и совместимости | | Среднее (ограничения платформы) |
| **Параметр** | | **Описание** | |
| Контекст | | Ищет музыку для учёбы и отдыха, создаёт плейлисты «для фона». | |
| Цель | | Быстро найти трек и сохранить в плейлист | |
| Боли (проблемы) | | Много шагов для добавления, непонятные состояния загрузки | |
| Сценарии | | поиск → просмотр карточки → добавить в избранное/плейлист | |
| **Параметр** | | **Описание** | |
| Контекст | | Собирает энергичные треки для тренировок, ценит минимализм. | |
| Цель | | Быстро собрать подборку и открыть избранное | |
| Боли (проблемы) | | Сложные меню, отвлекающие элементы | |
| Сценарии | | поиск → просмотр карточки → добавить в избранное/плейлист | |
| **Параметр** | | **Описание** | |
| Контекст | | Систематизирует любимых исполнителей, отмечает избранные треки. | |
| Цель | | Удобно управлять избранным и плейлистами | |
| Боли (проблемы) | | Потеря данных, отсутствие истории поиска | |
| Сценарии | | поиск → просмотр карточки → добавить в избранное/плейлист | |
| **ID** | **User story** | | **Критерии приёмки** | |
| US-01 | Как пользователь, я хочу видеть главный экран с разделами, чтобы быстро перейти к нужной функции. | | На главном экране есть кнопки/пункты меню; переходы работают. | |
| US-02 | Как пользователь, я хочу искать треки по запросу, чтобы находить нужную музыку. | | После ввода запроса отображается список результатов. | |
| US-03 | Как пользователь, я хочу видеть состояния «загрузка/ошибка/пусто», чтобы понимать результат поиска. | | Отображаются соответствующие экраны/сообщения. | |
| US-04 | Как пользователь, я хочу открыть карточку трека, чтобы увидеть подробную информацию. | | Открывается экран деталей с метаданными. | |
| US-05 | Как пользователь, я хочу добавить трек в избранное, чтобы вернуться к нему позже. | | Трек появляется в списке избранного. | |
| US-06 | Как пользователь, я хочу удалить трек из избранного, чтобы управлять списком. | | Трек исчезает из избранного. | |
| US-07 | Как пользователь, я хочу создать плейлист, чтобы группировать треки по настроению. | | Плейлист создаётся и отображается в списке. | |
| US-08 | Как пользователь, я хочу добавить трек в плейлист, чтобы собрать подборку. | | Трек отображается в выбранном плейлисте. | |
| US-09 | Как пользователь, я хочу видеть историю поиска, чтобы быстро повторять запросы. | | История отображается и ограничена по размеру. | |
| US-10 | Как пользователь, я хочу очистить историю поиска, чтобы удалить приватные запросы. | | История очищается по действию пользователя. | |
| US-11 | Как пользователь, я хочу объединить плейлисты, чтобы не дублировать подборки. | | Треки переносятся, исходный плейлист удаляется. | |
| US-12 | Как пользователь, я хочу изменить тему, чтобы использовать приложение комфортно. | | Тема меняется и сохраняется. | |
| **МЕСТО ДЛЯ СКРИНШОТА/ДИАГРАММЫ (вставьте изображение)** | | | |
| **Итерация** | **Цель** | | **Результат** |
| Sprint 1 | Исследование и требования | | Анализ аналогов, стейкхолдеры, user stories, NFR |
| Sprint 2 | Прототипирование | | Прототип экранов в Figma, согласование UX |
| Sprint 3 | Реализация | | Поиск, карточка трека, избранное, плейлисты |
| Sprint 4 | Доведение качества | | Обработка ошибок, история, настройки, эвристический анализ |

# 2 ПРОЕКТНАЯ ЧАСТЬ

## 2.1 Описание процесса в нотации IDEF0

Ниже приводится текстовое описание функциональной модели (IDEF0) для приложения «Playlist Maker».

Контекстная диаграмма A-0: «Управление треками и плейлистами»

Входы (Input):

* поисковый запрос пользователя;
* данные трека, выбранного пользователем;
* данные нового плейлиста (название, описание, обложка);
* команды пользователя (добавить в избранное, добавить в плейлист, удалить, объединить и т. п.).

Выходы (Output):

* список найденных треков;
* карточка трека;
* список плейлистов и карточка плейлиста;
* список избранных треков;
* обновлённые данные в локальных хранилищах (Room, DataStore);
* переходы между экранами приложения.

Управление (Control):

* требования к UX и UI (Material Design);
* ограничения платформы Android и правила работы с разрешениями;
* доступность сети и ограничения внешнего API;
* логика ограничения истории поиска (не более 10 элементов).

Механизмы (Mechanism):

* мобильное устройство Android;
* стек разработки: Kotlin, Jetpack Compose, ViewModel, Coroutines, Navigation Compose;
* сетевые библиотеки Retrofit/OkHttp/Gson;
* локальная БД Room;
* DataStore Preferences для истории поиска.

Декомпозиция A0:

A1 «Поиск треков» — обработка ввода, сетевой запрос, отображение результата, сохранение истории.

A2 «Управление избранным» — добавление/удаление отметки favorite и отображение списка избранного.

A3 «Управление плейлистами» — создание/удаление/просмотр плейлистов, добавление и удаление треков, объединение плейлистов.

A4 «Настройки приложения» — шаринг, поддержка, соглашение, режим темы.

## 2.2 Описание процесса в нотации DFD

Ниже приведено текстовое описание потоков данных (DFD) в приложении.

Внешние сущности:

E1 Пользователь — вводит запросы и управляет плейлистами/избранным.

E2 iTunes Search API — возвращает результаты поиска (JSON).

Хранилища данных:

D1 Room Database — хранит таблицы tracks и playlists.

D2 DataStore Preferences — хранит историю поисковых запросов.

Основные процессы:

P1 Поиск треков:

* Пользователь вводит запрос → P1 отправляет запрос в E2 → получает ответ → преобразует в список треков → отображает в UI.
* Запрос сохраняется в D2 (история).

P2 Управление избранным:

* Пользователь отмечает трек как избранный → P2 обновляет поле favorite в D1.
* Экран «Избранное» получает данные из D1 (favorite = 1) и отображает список.

P3 Управление плейлистами:

* Пользователь создаёт плейлист → P3 записывает данные в D1 (таблица playlists).
* Пользователь добавляет трек в плейлист → P3 обновляет поле playlistId у записи tracks в D1.
* Пользователь открывает плейлист → P3 читает данные playlists и связанные tracks из D1.
* Пользователь объединяет плейлисты → P3 переносит треки (moveTracksBetweenPlaylists) и удаляет исходный плейлист.

P4 Настройки:

* Пользователь выбирает действие (поделиться/написать/соглашение/тема) → P4 формирует системные Intent или обновляет состояние темы.

## 2.3 Описание и обоснование выбора программного обеспечения

Для разработки приложения выбран следующий набор инструментов и библиотек.

Язык и среда разработки:

* Kotlin — основной язык разработки Android-приложений, обеспечивает безопасность типов, лаконичность и хорошую интеграцию с Jetpack.
* Android Studio — основная среда разработки, отладка, эмулятор, профилирование.

UI и архитектура:

* Jetpack Compose + Material 3 — декларативный UI, ускоряет разработку и упрощает поддержку экранов.
* AndroidX Navigation Compose — навигация на основе маршрутов (route) и NavHost.
* ViewModel — хранение состояния и логики экранов, переживает изменения конфигурации.
* Kotlin Coroutines + Flow/StateFlow — асинхронная работа с сетью/БД и реактивные обновления UI.

Сеть:

* Retrofit 2 — создание API-клиента и описание REST-методов.
* OkHttp + HttpLoggingInterceptor — HTTP-клиент и логирование запросов (удобно для отладки).
* Gson — парсинг JSON-ответов iTunes Search API в DTO.

Локальное хранение:

* Room — ORM над SQLite, типобезопасные DAO и автоматическая генерация SQL.
* DataStore Preferences — современная замена SharedPreferences для хранения простой информации (история поиска).

## 2.4 Архитектура программной системы

Архитектура приложения построена по принципам разделения ответственности и включает слои:

1) Presentation (UI)

* Jetpack Compose-экраны: MainScreen, SearchScreen, DetailScreen, PlaylistsScreen, PlaylistDetailScreen, FavoritesScreen, SettingsScreen.
* Навигация реализована через маршруты (AppScreen) и NavHostController.
* UI подписывается на StateFlow/Flow из ViewModel и отображает состояние.

2) ViewModel

* SearchViewModel — управляет поиском, историей запросов, выбранным треком, добавлением в избранное и в плейлист. Использует SearchState (Initial/Searching/Success/Fail).
* PlaylistsViewModel — управляет списком плейлистов, избранными треками, созданием/удалением плейлистов, добавлением/удалением треков из плейлистов и объединением плейлистов.

3) Domain

* Модели: Track, Playlist.
* Интерфейсы репозиториев: TracksRepository, PlaylistsRepository, SearchHistoryRepository. Domain-слой определяет «что нужно сделать», не привязываясь к конкретным источникам данных.

4) Data

* Network: ITunesApiService (Retrofit) + DTO (TrackDto, TracksSearchResponse).
* Database: AppDatabase (Room), сущности TrackEntity и PlaylistEntity, DAO (TrackDao, PlaylistDao).
* Preferences: SearchHistoryPreferences (DataStore), ограничение истории до 10 элементов.
* Реализации репозиториев: TracksRepositoryImpl, PlaylistsRepositoryImpl, SearchHistoryRepositoryImpl.

Инициализация зависимостей:

* Используется объект Creator (service locator), который создаёт Retrofit/OkHttp, Room Database и репозитории, после чего передаёт их во ViewModel (через фабрики).

Преимущества выбранного подхода:

* повышение тестопригодности и модульности;
* упрощение поддержки и расширения функциональности;
* минимизация связности UI с источниками данных.

# 3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

## 3.1 Описание моделей и структур данных

В приложении выделены следующие основные модели данных.

Сетевые модели (DTO):

* TrackDto — модель ответа API (JSON), содержит поля trackId, trackName, artistName, trackTimeMillis, artworkUrl100, collectionName, releaseDate, previewUrl и др.
* TracksSearchResponse — модель ответа поиска, содержит resultCount и список результатов (results: List<TrackDto>).

Доменные модели:

* Track — модель трека, используемая в UI и Domain: id, trackName, artistName, trackTime (в формате строки), coverResId (опционально), artworkUrl (опционально), year (опционально), album (опционально), playlistId, favorite.
* Playlist — модель плейлиста: id, name, description, coverImageUri (опционально), tracks (List<Track>).

Модели базы данных (Room Entities):

* TrackEntity (таблица tracks): id (PK), trackName, artistName, trackTime, coverResId, artworkUrl, year, album, playlistId, favorite.
* PlaylistEntity (таблица playlists): id (PK AUTOINCREMENT), name, description, coverImageUri.

Схема БД (фрагмент, генерируется Room):

* tracks: CREATE TABLE tracks ( id INTEGER PRIMARY KEY NOT NULL, trackName TEXT NOT NULL, artistName TEXT NOT NULL, trackTime TEXT NOT NULL, coverResId INTEGER, artworkUrl TEXT, year INTEGER, album TEXT, playlistId INTEGER NOT NULL, favorite INTEGER NOT NULL );
* playlists: CREATE TABLE playlists ( id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, name TEXT NOT NULL, description TEXT NOT NULL, coverImageUri TEXT );

DAO-операции:

* TrackDao: - получение всех треков и избранных через Flow; - добавление/обновление трека; - обновление favorite; - привязка трека к плейлисту (playlistId), очистка плейлиста и перенос треков между плейлистами.
* PlaylistDao: - получение списка плейлистов через Flow; - добавление плейлиста; - удаление плейлиста по id.

DataStore (история поиска):

* SearchHistoryPreferences хранит строку с запросами, разделёнными запятой, ограничивая количество элементов до 10.
* История предоставляется в виде Flow<List<String>>.

## 3.2 Тестирование программного продукта

Тестирование проводилось методом ручного функционального тестирования на эмуляторе Android и/или устройстве.

Ниже приведены основные тест-кейсы.

Тест 1. Переходы по главному меню

Шаги: открыть приложение → нажать «Поиск», «Плейлисты», «Избранное», «Настройки».

Ожидаемый результат: выполняется корректный переход на соответствующие экраны.

Тест 2. Поиск треков (успешный)

Шаги: открыть «Поиск» → ввести валидный запрос (например, «Muse») → дождаться результата.

Ожидаемый результат: отображается список треков, статус SearchState = Success.

Тест 3. Поиск треков (пустой результат)

Шаги: ввести запрос, по которому нет результатов.

Ожидаемый результат: отображается сообщение «Ничего не найдено».

Тест 4. Поиск треков (ошибка)

Шаги: отключить интернет → выполнить поиск.

Ожидаемый результат: отображается состояние ошибки (например, «Ошибка сервера») и кнопка «Обновить».

Тест 5. История поиска

Шаги: выполнить несколько поисков → открыть экран поиска.

Ожидаемый результат: отображается блок «История поиска» со списком последних запросов (не более 10).

Тест 6. Очистка истории поиска

Шаги: нажать «Очистить» в истории поиска.

Ожидаемый результат: история очищается, список становится пустым.

Тест 7. Карточка трека

Шаги: открыть трек из результатов → проверить поля (название, артист, год, альбом, длительность).

Ожидаемый результат: карточка отображается корректно; при отсутствии данных показываются корректные значения/плейсхолдеры.

Тест 8. Добавление/удаление из избранного

Шаги: на карточке трека нажать «В избранное» → перейти в «Избранное» → убедиться, что трек появился → убрать из избранного.

Ожидаемый результат: статус favorite сохраняется в БД; список избранного обновляется.

Тест 9. Создание плейлиста

Шаги: открыть «Плейлисты» → добавить плейлист → указать название, описание, выбрать изображение → создать.

Ожидаемый результат: плейлист появляется в списке; данные сохраняются в БД.

Тест 10. Добавление трека в плейлист и удаление из плейлиста

Шаги: открыть карточку трека → «В плейлист» → выбрать плейлист → открыть плейлист → удалить трек из плейлиста.

Ожидаемый результат: track.playlistId обновляется; список треков плейлиста отображается корректно.

Тест 11. Объединение плейлистов

Шаги: создать два плейлиста → добавить треки в один → выполнить объединение с другим.

Ожидаемый результат: треки переносятся в целевой плейлист, исходный плейлист удаляется; отображается сообщение об успешном объединении.

Тест 12. Настройки: соглашение, шаринг, поддержка

Шаги: открыть «Настройки» → «Поделиться приложением» → проверить системный share sheet; «Написать в поддержку» → проверить создание письма с темой и текстом; «Пользовательское соглашение» → проверить открытие браузера по ссылке.

Ожидаемый результат: системные Intent формируются корректно.

По результатам тестирования основные сценарии работают корректно, ошибки сети обрабатываются, данные сохраняются между запусками.

Результаты тестирования представлены на рисунках ниже.



Рисунок 13 – Главный экран приложения

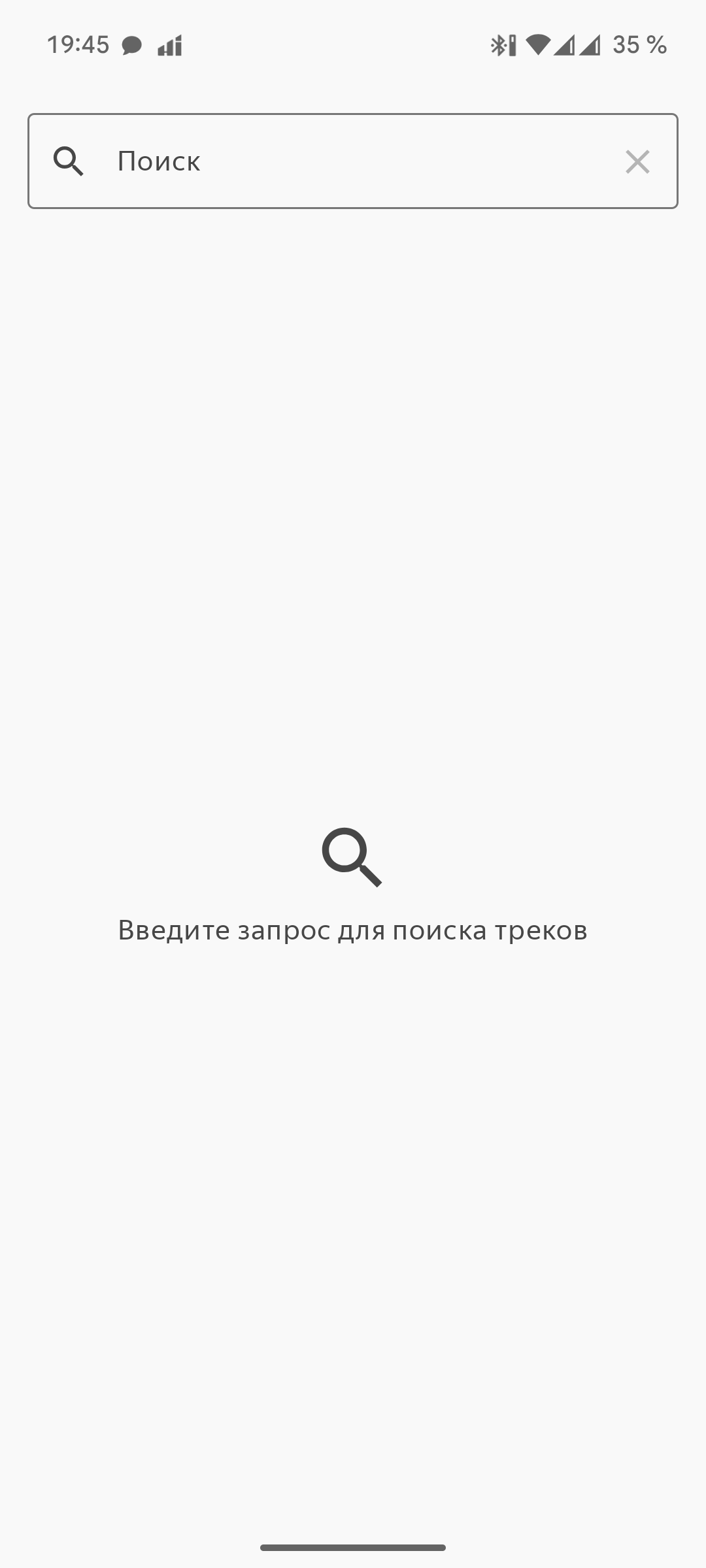
****

Рисунок 14 – Экран поиска: начальное состояние

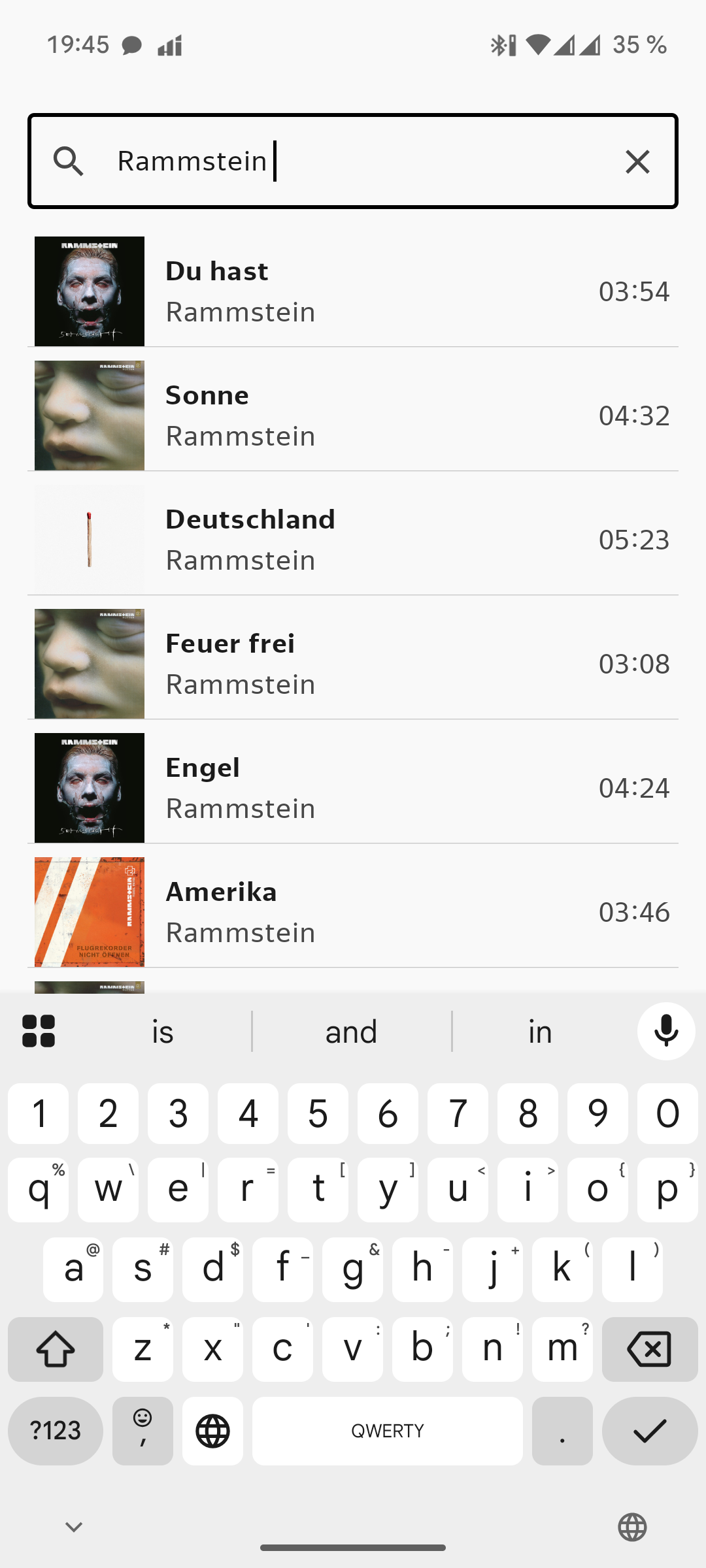
****

Рисунок 15 – Экран поиска: результаты поиска

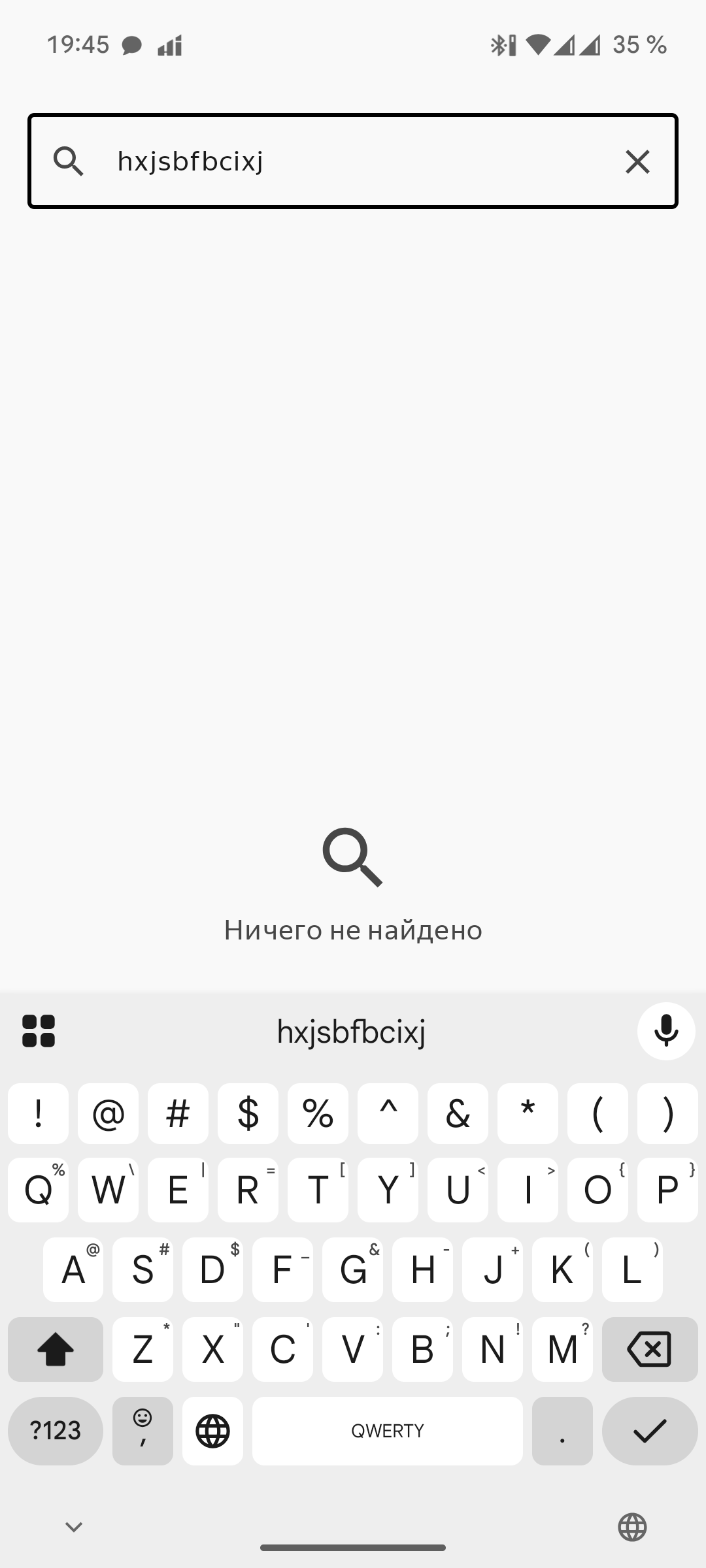
****

Рисунок 16 – Экран поиска: состояние ошибки/пустой результат

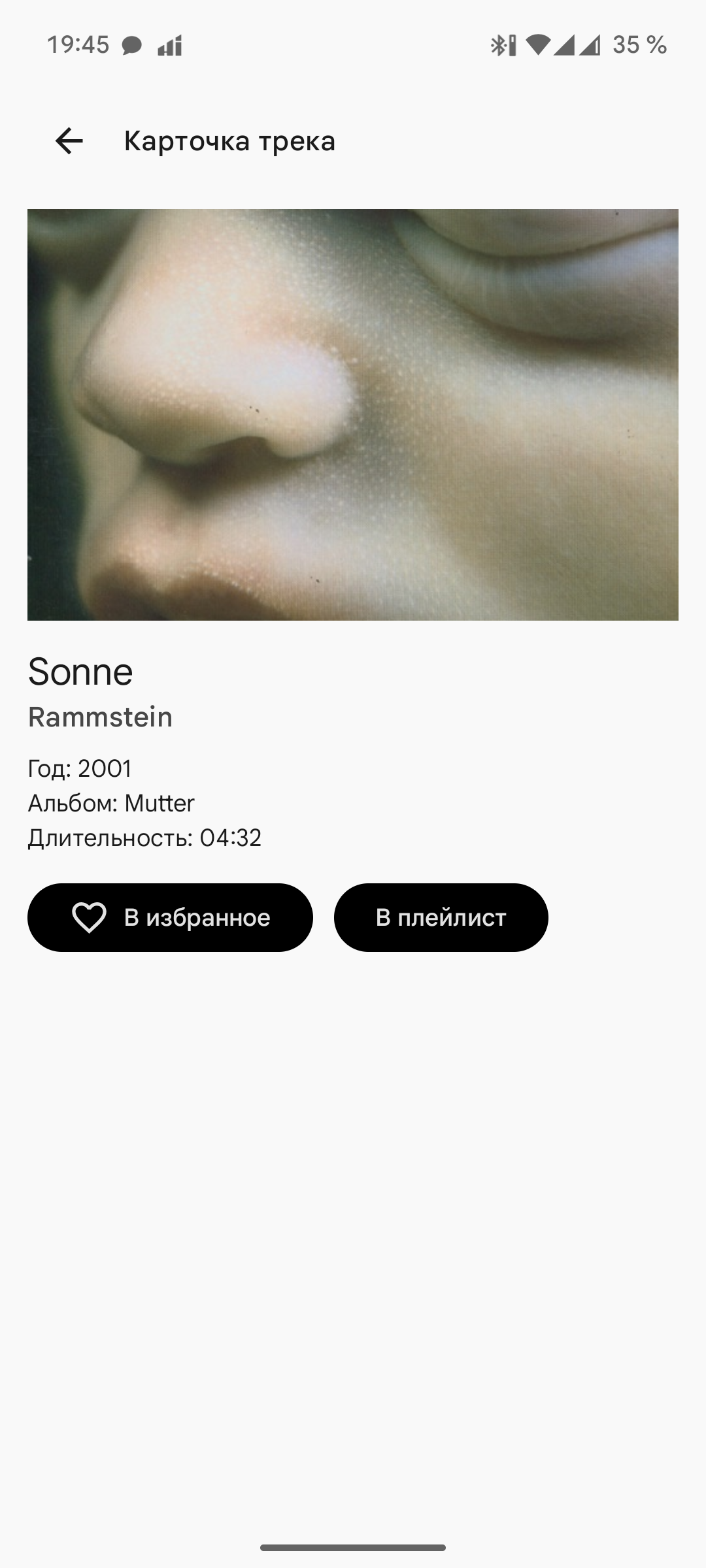
****

Рисунок 17 – Карточка трека

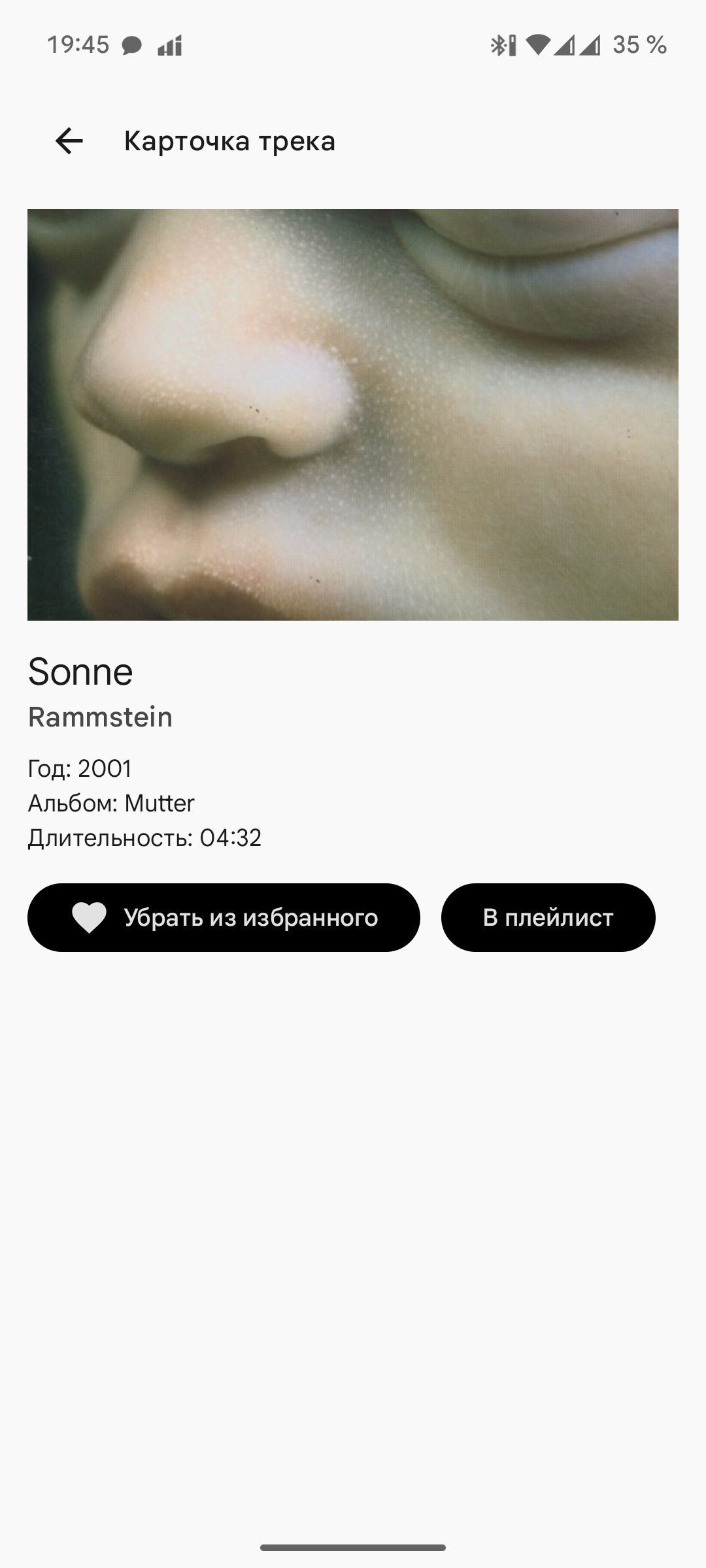
****

Рисунок 18 – Действие «Добавить в избранное» (до/после)

****

Рисунок 19 – Экран «Избранное»

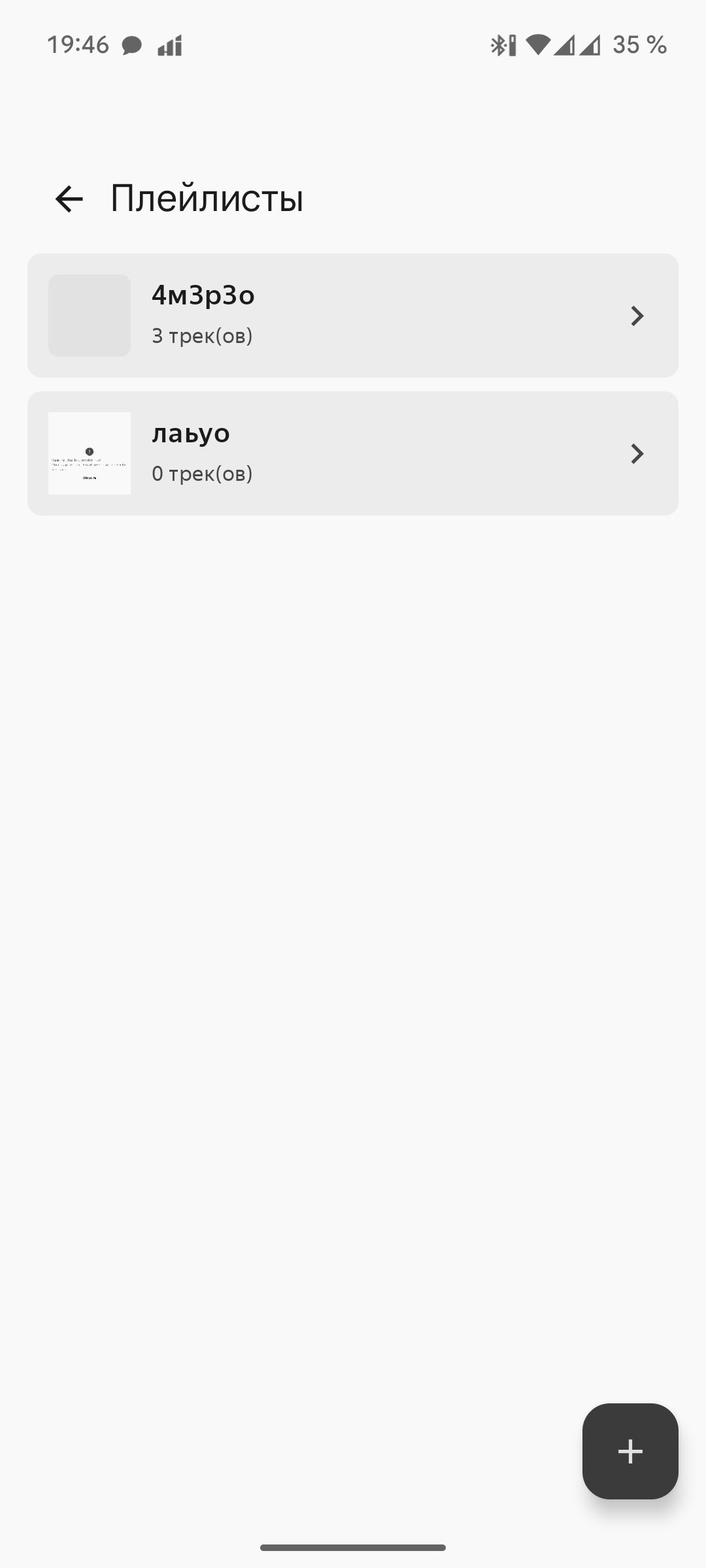
****

Рисунок 20 – Экран «Плейлисты»: список

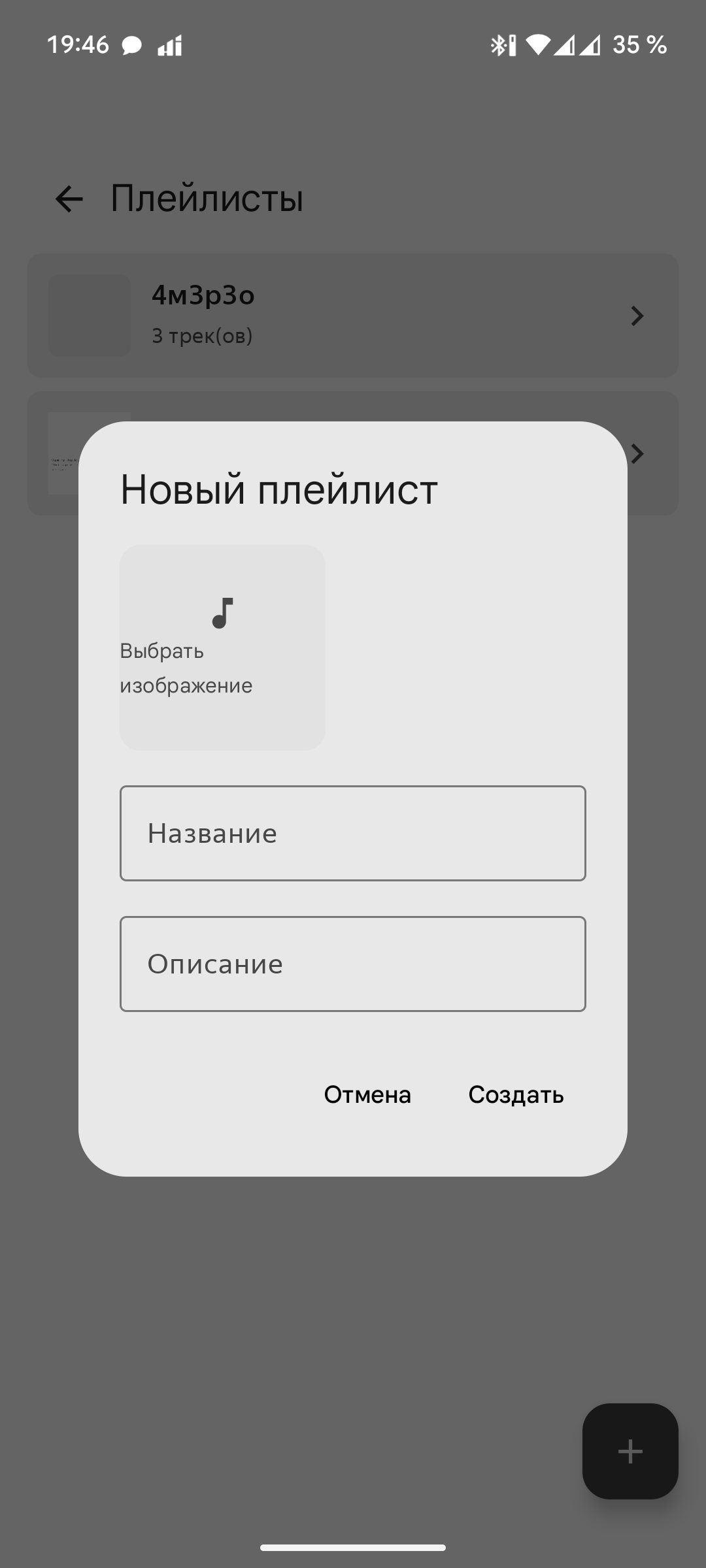
****

Рисунок 21 – Создание плейлиста (диалог/экран)

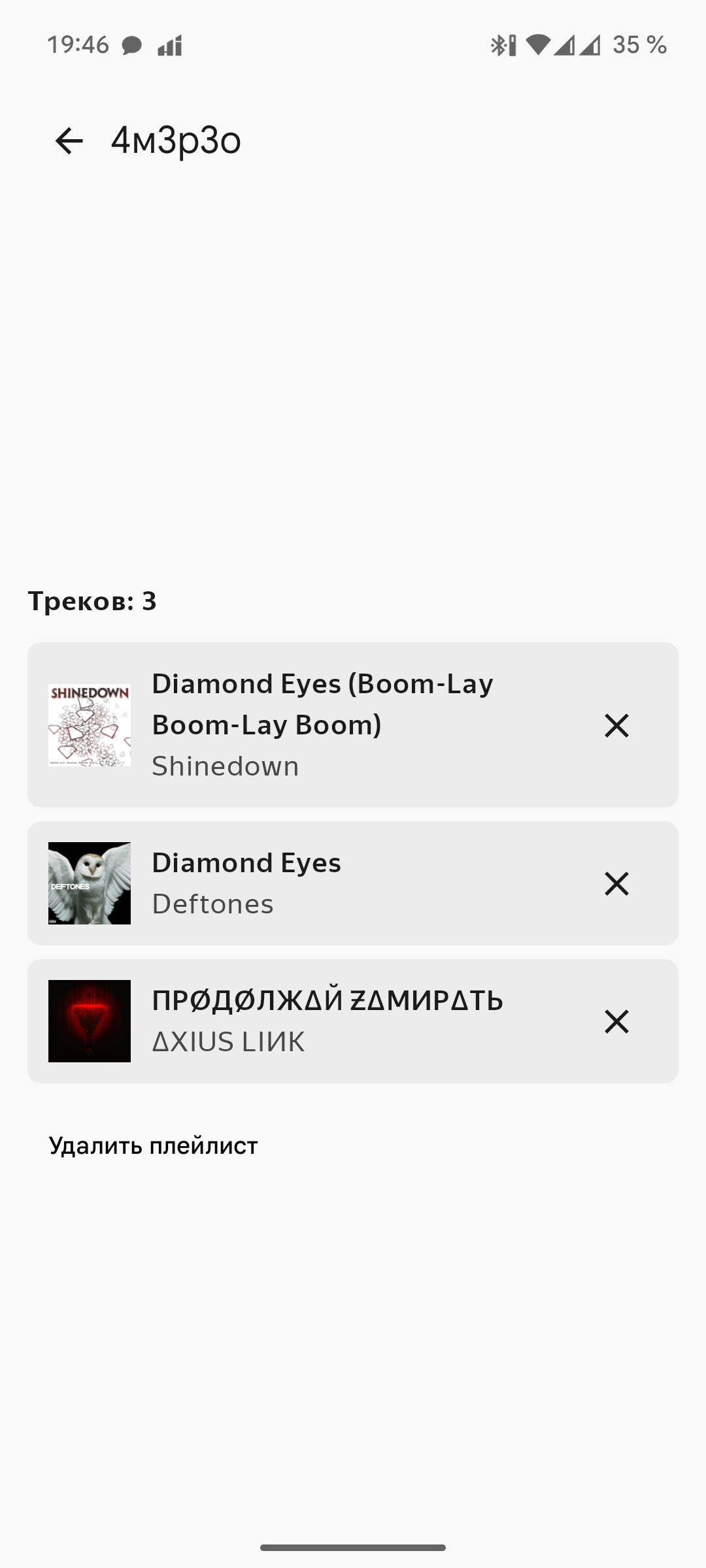
****

Рисунок 22 – Экран плейлиста: детали и список треков

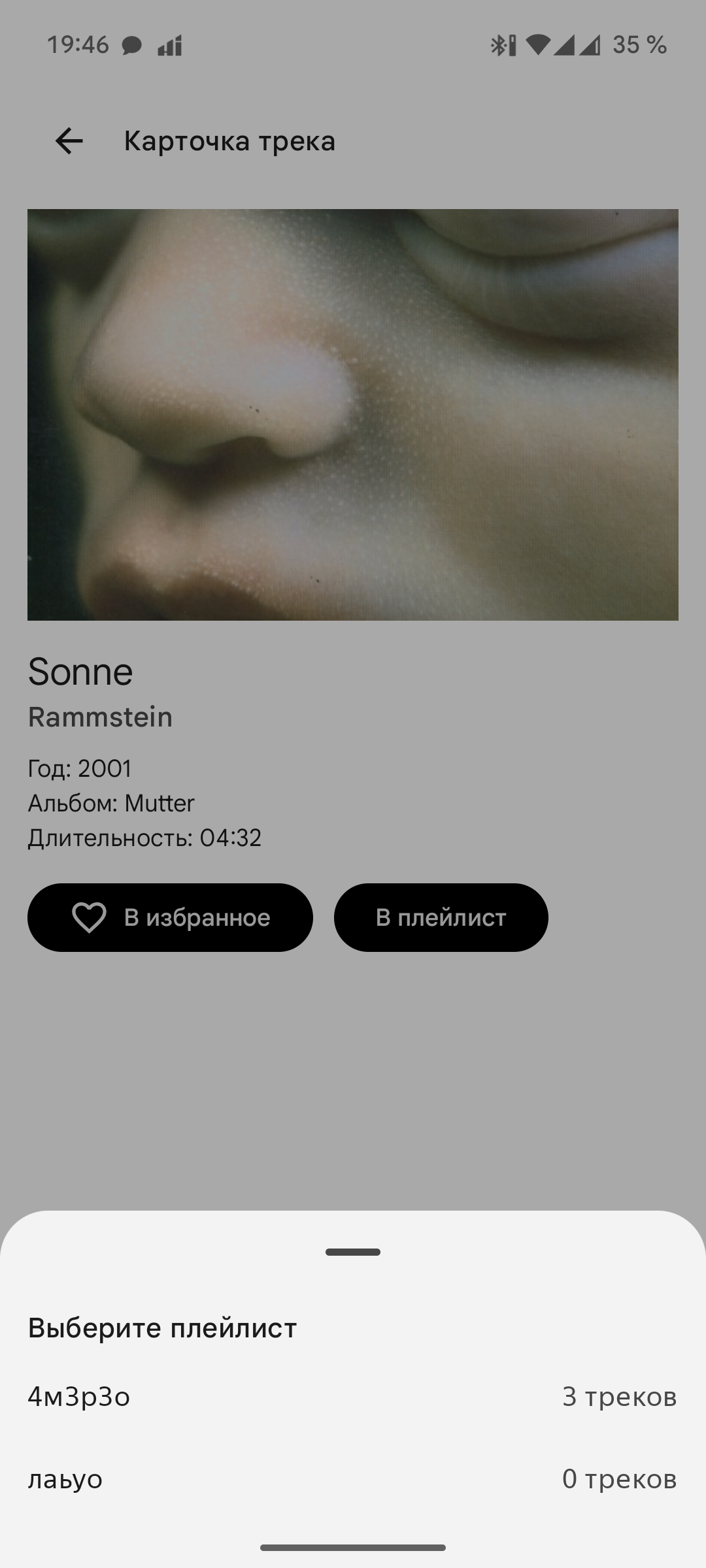
****

Рисунок 23 – Добавление трека в плейлист (выбор плейлиста)

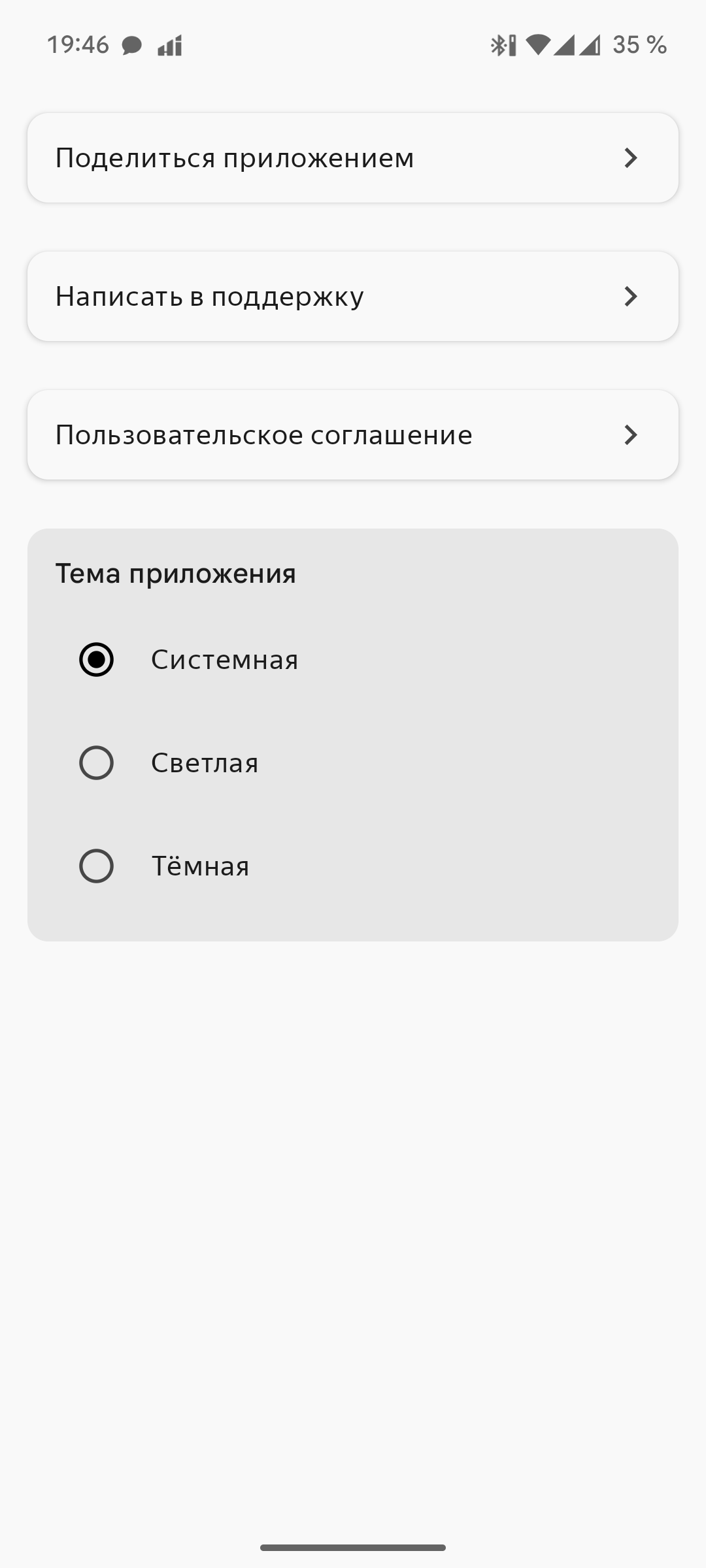
****

Рисунок 24 – Экран «Настройки/профиль»: тема, шаринг, поддержка, соглашение

## 3.3 Полный листинг (структура проекта и ключевые интерфейсы)

Ниже приведена структура ключевых пакетов приложения и перечень основных классов/интерфейсов (кратко).

Пакеты и назначение:

* com.example.playlist\_maker\_android\_albahtinilya - MainActivity, навигационный хост и Compose-экраны приложения - AppScreen (маршруты навигации)
* com.example.playlist\_maker\_android\_albahtinilya.creator - Creator (инициализация Retrofit/OkHttp, Room и репозиториев)
* com.example.playlist\_maker\_android\_albahtinilya.domain.model - Track, Playlist
* com.example.playlist\_maker\_android\_albahtinilya.domain.repository - TracksRepository, PlaylistsRepository, SearchHistoryRepository
* com.example.playlist\_maker\_android\_albahtinilya.data.dto - TrackDto, TracksSearchResponse
* com.example.playlist\_maker\_android\_albahtinilya.data.network - ITunesApiService
* com.example.playlist\_maker\_android\_albahtinilya.data.db - AppDatabase, TrackEntity, PlaylistEntity, TrackDao, PlaylistDao
* com.example.playlist\_maker\_android\_albahtinilya.data.preferences - SearchHistoryPreferences (DataStore)
* com.example.playlist\_maker\_android\_albahtinilya.data.repository - TracksRepositoryImpl, PlaylistsRepositoryImpl, SearchHistoryRepositoryImpl
* com.example.playlist\_maker\_android\_albahtinilya.ui.search - SearchViewModel, SearchState, DetailScreen
* com.example.playlist\_maker\_android\_albahtinilya.ui.playlists - PlaylistsViewModel
* com.example.playlist\_maker\_android\_albahtinilya.ui.theme - ThemeMode, типографика (Yandex Sans), тема Compose

Ключевые интерфейсы (выдержка по методам):

* ITunesApiService: - search(term: String, entity: String): TracksSearchResponse
* TracksRepository: - searchTracks(expression: String): List<Track> - getFavoriteTracks(): Flow<List<Track>> - insertTrackToPlaylist(track: Track, playlistId: Long) - updateTrackFavoriteStatus(track: Track, isFavorite: Boolean)
* PlaylistsRepository: - getAllPlaylists(): Flow<List<Playlist>> - addNewPlaylist(name: String, description: String, coverImageUri: String?) - mergePlaylists(sourceId: Long, targetId: Long)
* SearchHistoryRepository: - history: Flow<List<String>> - addQuery(query: String) - clearHistory()

Ниже приведены листинги ключевых файлов и фрагментов кода приложения из проекта D:\GitHub\MIREA\playlist-maker-android-AlbahtinIlya. Часть экранов реализована как Composable-функции внутри файла MainActivity.kt.

**Листинг 1 – MainActivity.kt (фрагмент: MainActivity/PlaylistApp)**

|  |
| --- |
| package com.example.playlist\_maker\_android\_albahtinilya  import android.Manifest  import android.content.Intent  import android.content.pm.PackageManager  import android.net.Uri  import android.os.Bundle  import androidx.activity.ComponentActivity  import androidx.activity.compose.rememberLauncherForActivityResult  import androidx.activity.compose.setContent  import androidx.activity.enableEdgeToEdge  import androidx.activity.result.contract.ActivityResultContracts  import androidx.compose.foundation.isSystemInDarkTheme  import androidx.compose.runtime.SideEffect  import androidx.compose.foundation.Image  import androidx.compose.foundation.background  import androidx.compose.foundation.clickable  import androidx.compose.foundation.combinedClickable  import androidx.compose.foundation.layout.Arrangement  import androidx.compose.foundation.layout.Box  import androidx.compose.foundation.layout.Column  import androidx.compose.foundation.layout.Row  import androidx.compose.foundation.layout.Spacer  import androidx.compose.foundation.layout.fillMaxSize  import androidx.compose.foundation.layout.fillMaxWidth  import androidx.compose.foundation.layout.height  import androidx.compose.foundation.layout.padding  import androidx.compose.foundation.layout.size  import androidx.compose.foundation.layout.statusBarsPadding  import androidx.compose.foundation.layout.width  import androidx.compose.foundation.lazy.LazyColumn  import androidx.compose.foundation.lazy.items  import androidx.compose.foundation.shape.RoundedCornerShape  import androidx.compose.ui.draw.clip  import androidx.compose.material.icons.Icons  import androidx.compose.material.icons.automirrored.filled.ArrowBack  import androidx.compose.material.icons.automirrored.filled.KeyboardArrowRight  import androidx.compose.material.icons.filled.Add  import androidx.compose.material.icons.filled.Clear  import androidx.compose.material.icons.filled.Favorite  import androidx.compose.material.icons.filled.FavoriteBorder  import androidx.compose.material.icons.filled.Error  import androidx.compose.material.icons.filled.History  import androidx.compose.material.icons.filled.Menu  import androidx.compose.material.icons.filled.MusicNote  import androidx.compose.material.icons.filled.Search  import androidx.compose.material.icons.filled.Settings  import androidx.compose.material3.AlertDialog  import androidx.compose.material3.CircularProgressIndicator  import androidx.compose.material3.Divider  import androidx.compose.material3.FloatingActionButton  import androidx.compose.material3.HorizontalDivider  import androidx.compose.material3.Icon  import androidx.compose.material3.IconButton  import androidx.compose.material3.MaterialTheme  import androidx.compose.material3.ModalBottomSheet  import androidx.compose.material3.OutlinedTextField  import androidx.compose.material3.RadioButton  import androidx.compose.material3.Scaffold  import androidx.compose.material3.Surface  import androidx.compose.material3.Text  import androidx.compose.material3.TextButton  import androidx.compose.material3.ExperimentalMaterial3Api  import androidx.compose.runtime.Composable  import androidx.compose.runtime.collectAsState  import androidx.compose.runtime.getValue  import androidx.compose.runtime.mutableStateOf  import androidx.compose.runtime.remember  import androidx.compose.runtime.saveable.rememberSaveable  import androidx.compose.runtime.setValue  import androidx.compose.ui.Alignment  import androidx.compose.ui.Modifier  import androidx.compose.ui.graphics.vector.ImageVector  import androidx.compose.ui.layout.ContentScale  import androidx.compose.ui.platform.LocalContext  import androidx.compose.ui.platform.LocalView  import androidx.compose.ui.res.painterResource  import androidx.compose.ui.res.stringResource  import androidx.compose.ui.text.font.FontWeight  import androidx.compose.ui.text.style.TextOverflow  import androidx.compose.ui.tooling.preview.Preview  import androidx.compose.ui.unit.dp  import androidx.compose.ui.unit.sp  import androidx.core.view.WindowCompat  import androidx.core.content.ContextCompat  import com.example.playlist\_maker\_android\_albahtinilya.creator.Creator  import androidx.lifecycle.viewmodel.compose.viewModel  import androidx.navigation.NavHostController  import androidx.navigation.NavType  import androidx.navigation.compose.NavHost  import androidx.navigation.compose.composable  import androidx.navigation.compose.rememberNavController  import androidx.navigation.navArgument  import com.example.playlist\_maker\_android\_albahtinilya.R  import com.example.playlist\_maker\_android\_albahtinilya.domain.model.Playlist  import com.example.playlist\_maker\_android\_albahtinilya.domain.model.Track  import com.example.playlist\_maker\_android\_albahtinilya.ui.playlists.PlaylistsViewModel  import com.example.playlist\_maker\_android\_albahtinilya.ui.search.DetailScreen  import com.example.playlist\_maker\_android\_albahtinilya.ui.search.SearchState  import com.example.playlist\_maker\_android\_albahtinilya.ui.search.SearchViewModel  import com.example.playlist\_maker\_android\_albahtinilya.ui.theme.PlaylistmakerandroidAlbahtinIlyaTheme  import com.example.playlist\_maker\_android\_albahtinilya.ui.theme.ThemeMode  import coil.compose.AsyncImage  class MainActivity : ComponentActivity() {  override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  enableEdgeToEdge()  Creator.init(applicationContext)  setContent {  PlaylistApp()  }  }  }  @Composable  private fun PlaylistApp() {  var themeMode by rememberSaveable { mutableStateOf(ThemeMode.SYSTEM) }  val view = LocalView.current  val useDark = when (themeMode) {  ThemeMode.SYSTEM -> isSystemInDarkTheme()  ThemeMode.LIGHT -> false  ThemeMode.DARK -> true  }  PlaylistmakerandroidAlbahtinIlyaTheme(themeMode = themeMode) {  SideEffect {  val window = (view.context as? ComponentActivity)?.window  if (window != null) {  WindowCompat.getInsetsController(window, view).isAppearanceLightStatusBars = !useDark  }  }  PlaylistHost(  themeMode = themeMode,  onThemeChange = { themeMode = it }  )  }  } |

**Листинг 2 – MainActivity.kt (фрагмент: AppScreen/PlaylistHost)**

|  |
| --- |
| private enum class AppScreen(val route: String) {  MAIN("main"),  SEARCH("search"),  DETAIL("detail"),  SETTINGS("settings"),  PLAYLISTS("playlists"),  PLAYLIST\_DETAIL("playlistDetail/{id}"),  FAVORITES("favorites");  companion object {  fun playlistDetailRoute(id: Long) = "playlistDetail/$id"  }  }  @Composable  private fun PlaylistHost(  navController: NavHostController = rememberNavController(),  themeMode: ThemeMode,  onThemeChange: (ThemeMode) -> Unit  ) {  NavHost(  navController = navController,  startDestination = AppScreen.MAIN.route  ) {  composable(AppScreen.MAIN.route) {  MainScreen(  onSearch = { navController.navigate(AppScreen.SEARCH.route) },  onPlaylists = { navController.navigate(AppScreen.PLAYLISTS.route) },  onFavorites = { navController.navigate(AppScreen.FAVORITES.route) },  onSettings = { navController.navigate(AppScreen.SETTINGS.route) }  )  }  composable(AppScreen.SEARCH.route) { backStackEntry ->  val mainEntry = remember(backStackEntry) { navController.getBackStackEntry(AppScreen.MAIN.route) }  val vm: SearchViewModel =  viewModel(factory = SearchViewModel.factory(), viewModelStoreOwner = mainEntry)  SearchScreen(  viewModel = vm,  onTrackClick = { track ->  vm.selectTrack(track)  navController.navigate(AppScreen.DETAIL.route)  }  )  }  composable(AppScreen.DETAIL.route) { backStackEntry ->  val mainEntry = remember(backStackEntry) { navController.getBackStackEntry(AppScreen.MAIN.route) }  val searchVm: SearchViewModel = viewModel(  factory = SearchViewModel.factory(),  viewModelStoreOwner = mainEntry  )  val playlistsVm: PlaylistsViewModel = viewModel(factory = PlaylistsViewModel.factory())  DetailScreen(  viewModel = searchVm,  playlistsViewModel = playlistsVm,  onBack = { navController.popBackStack() }  )  }  composable(AppScreen.SETTINGS.route) {  SettingsScreen(  themeMode = themeMode,  onThemeChange = onThemeChange  )  }  composable(AppScreen.PLAYLISTS.route) {  val vm: PlaylistsViewModel = viewModel(factory = PlaylistsViewModel.factory())  PlaylistsScreen(  viewModel = vm,  onBack = { navController.popBackStack() },  onPlaylistClick = { id -> navController.navigate(AppScreen.playlistDetailRoute(id)) }  )  }  composable(  route = AppScreen.PLAYLIST\_DETAIL.route,  arguments = listOf(navArgument("id") { type = NavType.LongType })  ) { entry ->  val id = entry.arguments?.getLong("id") ?: 0  val vm: PlaylistsViewModel = viewModel(factory = PlaylistsViewModel.factory())  val mainEntry = remember(entry) { navController.getBackStackEntry(AppScreen.MAIN.route) }  val searchVm: SearchViewModel =  viewModel(factory = SearchViewModel.factory(), viewModelStoreOwner = mainEntry)  PlaylistDetailScreen(  playlistId = id,  viewModel = vm,  onBack = { navController.popBackStack() },  onTrackClick = { track ->  searchVm.selectTrack(track)  navController.navigate(AppScreen.DETAIL.route)  }  )  }  composable(AppScreen.FAVORITES.route) { backStackEntry ->  val mainEntry = remember(backStackEntry) { navController.getBackStackEntry(AppScreen.MAIN.route) }  val playlistsVm: PlaylistsViewModel = viewModel(factory = PlaylistsViewModel.factory())  val searchVm: SearchViewModel =  viewModel(factory = SearchViewModel.factory(), viewModelStoreOwner = mainEntry)  FavoritesScreen(  viewModel = playlistsVm,  onBack = { navController.popBackStack() },  onTrackClick = { track ->  searchVm.selectTrack(track)  navController.navigate(AppScreen.DETAIL.route)  }  )  }  }  } |

**Листинг 3 – MainActivity.kt (фрагмент: SearchScreen)**

|  |
| --- |
| private fun SearchScreen(viewModel: SearchViewModel, onTrackClick: (Track) -> Unit) {  var query by rememberSaveable { mutableStateOf("") }  val placeholder = stringResource(id = R.string.search\_placeholder)  val state by viewModel.searchState.collectAsState()  val history by viewModel.history.collectAsState()  val colors = MaterialTheme.colorScheme  Surface(modifier = Modifier.fillMaxSize(), color = colors.background) {  Column(  modifier = Modifier  .fillMaxSize()  .statusBarsPadding()  .padding(horizontal = 16.dp, vertical = 24.dp),  verticalArrangement = Arrangement.Top,  horizontalAlignment = Alignment.CenterHorizontally  ) {  OutlinedTextField(  value = query,  onValueChange = {  query = it  viewModel.searchTracks(it)  },  modifier = Modifier.fillMaxWidth(),  placeholder = { Text(text = placeholder) },  leadingIcon = {  IconButton(onClick = { viewModel.searchTracks(query) }) {  Icon(  imageVector = Icons.Filled.Search,  contentDescription = stringResource(R.string.content\_desc\_search)  )  }  },  trailingIcon = {  IconButton(  onClick = {  if (query.isNotEmpty()) {  query = ""  viewModel.clearSearch()  }  },  enabled = query.isNotEmpty()  ) {  Icon(  imageVector = Icons.Filled.Clear,  contentDescription = stringResource(R.string.content\_desc\_clear)  )  }  },  singleLine = true  )  Spacer(modifier = Modifier.height(16.dp))  when (val screenState = state) {  is SearchState.Initial -> {  if (history.isEmpty()) {  PlaceholderBox(  icon = Icons.Filled.Search,  message = stringResource(R.string.search\_initial\_message)  )  } else {  Column(  modifier = Modifier.fillMaxSize(),  verticalArrangement = Arrangement.Top  ) {  Row(  modifier = Modifier.fillMaxWidth(),  verticalAlignment = Alignment.CenterVertically  ) {  Text(  text = stringResource(R.string.search\_history\_title),  style = MaterialTheme.typography.titleMedium,  modifier = Modifier.weight(1f)  )  TextButton(onClick = { viewModel.clearHistory() }) {  Text(text = stringResource(R.string.search\_history\_clear))  }  }  Spacer(modifier = Modifier.height(8.dp))  LazyColumn(verticalArrangement = Arrangement.spacedBy(8.dp)) {  items(history) { item ->  HistoryItem(text = item, onClick = {  query = item  viewModel.searchTracks(item)  })  }  }  }  }  }  is SearchState.Searching -> {  Box(  modifier = Modifier.fillMaxSize(),  contentAlignment = Alignment.Center  ) {  CircularProgressIndicator()  }  }  is SearchState.Success -> {  val tracks = screenState.tracks  if (tracks.isEmpty()) {  PlaceholderBox(  icon = Icons.Filled.Search,  message = stringResource(R.string.search\_empty\_message)  )  } else {  LazyColumn(  modifier = Modifier.fillMaxSize(),  verticalArrangement = Arrangement.spacedBy(8.dp)  ) {  items(tracks) { track ->  TrackListItem(  track = track,  onClick = { onTrackClick(track) }  )  HorizontalDivider(thickness = 0.5.dp, color = colors.outlineVariant)  }  }  }  }  is SearchState.Fail -> {  PlaceholderBox(  icon = Icons.Filled.Error,  message = if (screenState.message.isBlank()) {  stringResource(R.string.search\_error\_placeholder)  } else {  stringResource(R.string.search\_error, screenState.message)  },  actionLabel = stringResource(R.string.search\_retry),  onAction = { viewModel.retryLastQuery() }  )  }  }  }  }  }  @Composable  private fun TrackListItem(track: Track, onClick: () -> Unit) {  val colors = MaterialTheme.colorScheme  Row(  modifier = Modifier  .fillMaxWidth()  .clickable { onClick() },  verticalAlignment = Alignment.CenterVertically,  horizontalArrangement = Arrangement.spacedBy(12.dp)  ) {  AsyncImage(  model = track.artworkUrl ?: track.coverResId ?: R.drawable.ic\_music,  contentDescription = stringResource(R.string.content\_desc\_track\_cover, track.trackName),  contentScale = ContentScale.Crop,  placeholder = painterResource(id = R.drawable.ic\_music),  error = painterResource(id = R.drawable.ic\_music),  modifier = Modifier  .padding(start = 4.dp)  .size(64.dp)  .background(colors.surfaceVariant, RoundedCornerShape(6.dp))  )  Column(  modifier = Modifier.weight(1f),  horizontalAlignment = Alignment.Start  ) {  Text(  text = track.trackName,  fontWeight = FontWeight.Bold,  maxLines = 1,  overflow = TextOverflow.Ellipsis  )  Text(  text = track.artistName,  maxLines = 1,  overflow = TextOverflow.Ellipsis,  color = colors.onSurfaceVariant  )  }  Text(  text = track.trackTime,  modifier = Modifier.padding(end = 4.dp),  color = colors.onSurfaceVariant  )  }  }  @Composable  private fun PlaceholderBox(  icon: ImageVector,  message: String,  actionLabel: String? = null,  onAction: (() -> Unit)? = null  ) {  val colors = MaterialTheme.colorScheme  Box(  modifier = Modifier.fillMaxSize(),  contentAlignment = Alignment.Center  ) {  Column(  horizontalAlignment = Alignment.CenterHorizontally,  verticalArrangement = Arrangement.spacedBy(8.dp)  ) {  Icon(imageVector = icon, contentDescription = null, tint = colors.onSurfaceVariant, modifier = Modifier.size(48.dp))  Text(text = message, color = colors.onSurfaceVariant, style = MaterialTheme.typography.bodyLarge)  if (actionLabel != null && onAction != null) {  TextButton(onClick = onAction) {  Text(text = actionLabel)  }  }  }  }  }  @Composable  private fun HistoryItem(text: String, onClick: () -> Unit) {  val colors = MaterialTheme.colorScheme  Surface(  shape = RoundedCornerShape(8.dp),  color = colors.surface,  tonalElevation = 1.dp,  shadowElevation = 0.dp,  modifier = Modifier  .fillMaxWidth()  .clickable { onClick() }  ) {  Row(  modifier = Modifier  .fillMaxWidth()  .padding(horizontal = 12.dp, vertical = 10.dp),  verticalAlignment = Alignment.CenterVertically  ) {  Icon(  imageVector = Icons.Filled.History,  contentDescription = null,  tint = colors.onSurfaceVariant  )  Spacer(modifier = Modifier.width(12.dp))  Text(text = text, color = colors.onSurface)  }  }  } |

**Листинг 4 – MainActivity.kt (фрагмент: PlaylistDetailScreen)**

|  |
| --- |
| private fun PlaylistDetailScreen(  playlistId: Long,  viewModel: PlaylistsViewModel,  onBack: () -> Unit,  onTrackClick: (Track) -> Unit  ) {  val colors = MaterialTheme.colorScheme  val playlistState by viewModel.getPlaylist(playlistId).collectAsState(initial = null)  Surface(color = colors.background) {  Column(  modifier = Modifier  .fillMaxSize()  .statusBarsPadding()  .padding(16.dp)  ) {  Row(  modifier = Modifier.fillMaxWidth(),  verticalAlignment = Alignment.CenterVertically  ) {  IconButton(onClick = onBack) {  Icon(  imageVector = Icons.AutoMirrored.Filled.ArrowBack,  contentDescription = stringResource(R.string.content\_desc\_back)  )  }  Text(  text = playlistState?.name ?: stringResource(R.string.playlist\_header),  style = MaterialTheme.typography.titleLarge  )  }  playlistState?.let { playlist ->  Box(  modifier = Modifier  .fillMaxWidth()  .height(200.dp)  .padding(vertical = 12.dp)  ) {  if (playlist.coverImageUri != null) {  AsyncImage(  model = playlist.coverImageUri,  contentDescription = stringResource(R.string.playlist\_cover),  contentScale = ContentScale.Crop,  modifier = Modifier  .fillMaxSize()  .clip(RoundedCornerShape(12.dp))  )  } else {  Surface(  modifier = Modifier.fillMaxSize(),  shape = RoundedCornerShape(12.dp),  color = colors.surfaceVariant  ) {  Box(contentAlignment = Alignment.Center) {  Icon(  imageVector = Icons.Filled.MusicNote,  contentDescription = null,  tint = colors.onSurfaceVariant  )  }  }  }  }  Text(  text = playlist.description,  color = colors.onSurfaceVariant,  modifier = Modifier.padding(vertical = 4.dp)  )  Text(  text = stringResource(R.string.playlist\_tracks\_label, playlist.tracks.size),  fontWeight = FontWeight.Bold  )  Spacer(modifier = Modifier.height(12.dp))  LazyColumn(verticalArrangement = Arrangement.spacedBy(8.dp)) {  items(playlist.tracks) { track ->  Surface(  shape = RoundedCornerShape(8.dp),  tonalElevation = 1.dp,  shadowElevation = 0.dp,  color = colors.surface  ) {  Row(  modifier = Modifier  .fillMaxWidth()  .padding(12.dp)  .clickable { onTrackClick(track) },  verticalAlignment = Alignment.CenterVertically,  horizontalArrangement = Arrangement.spacedBy(12.dp)  ) {  AsyncImage(  model = track.artworkUrl ?: track.coverResId ?: R.drawable.ic\_music,  contentDescription = stringResource(R.string.content\_desc\_track\_cover, track.trackName),  contentScale = ContentScale.Crop,  placeholder = painterResource(id = R.drawable.ic\_music),  error = painterResource(id = R.drawable.ic\_music),  modifier = Modifier  .size(48.dp)  .background(colors.surfaceVariant, RoundedCornerShape(6.dp))  )  Column(modifier = Modifier.weight(1f)) {  Text(track.trackName, fontWeight = FontWeight.Medium)  Text(track.artistName, color = colors.onSurfaceVariant)  }  IconButton(onClick = { viewModel.deleteTrackFromPlaylist(track) }) {  Icon(  imageVector = Icons.Filled.Clear,  contentDescription = stringResource(R.string.content\_desc\_remove\_from\_playlist)  )  }  }  }  }  }  Spacer(modifier = Modifier.height(12.dp))  TextButton(onClick = {  viewModel.deletePlaylist(playlist.id)  onBack()  }) { Text(stringResource(R.string.playlist\_delete)) }  } ?: run {  Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {  Text(stringResource(R.string.playlist\_not\_found))  }  }  }  }  } |

**Листинг 5 – ui/search/DetailScreen.kt (экран деталей трека)**

|  |
| --- |
| package com.example.playlist\_maker\_android\_albahtinilya.ui.search  import androidx.compose.foundation.clickable  import androidx.compose.foundation.layout.Arrangement  import androidx.compose.foundation.layout.Column  import androidx.compose.foundation.layout.Row  import androidx.compose.foundation.layout.Spacer  import androidx.compose.foundation.layout.fillMaxSize  import androidx.compose.foundation.layout.fillMaxWidth  import androidx.compose.foundation.layout.height  import androidx.compose.foundation.layout.padding  import androidx.compose.foundation.layout.size  import androidx.compose.foundation.layout.statusBarsPadding  import androidx.compose.foundation.layout.width  import androidx.compose.foundation.rememberScrollState  import androidx.compose.foundation.verticalScroll  import androidx.compose.material.icons.Icons  import androidx.compose.material.icons.filled.ArrowBack  import androidx.compose.material.icons.filled.Favorite  import androidx.compose.material.icons.filled.FavoriteBorder  import androidx.compose.material3.Button  import androidx.compose.material3.ExperimentalMaterial3Api  import androidx.compose.material3.Icon  import androidx.compose.material3.IconButton  import androidx.compose.material3.MaterialTheme  import androidx.compose.material3.ModalBottomSheet  import androidx.compose.material3.Surface  import androidx.compose.material3.Text  import androidx.compose.material3.rememberModalBottomSheetState  import androidx.compose.runtime.Composable  import androidx.compose.runtime.collectAsState  import androidx.compose.runtime.getValue  import androidx.compose.runtime.mutableStateOf  import androidx.compose.runtime.saveable.rememberSaveable  import androidx.compose.runtime.setValue  import androidx.compose.ui.Alignment  import androidx.compose.ui.Modifier  import androidx.compose.ui.layout.ContentScale  import androidx.compose.ui.res.painterResource  import androidx.compose.ui.res.stringResource  import androidx.compose.ui.unit.dp  import com.example.playlist\_maker\_android\_albahtinilya.R  import com.example.playlist\_maker\_android\_albahtinilya.ui.playlists.PlaylistsViewModel  import coil.compose.AsyncImage  @OptIn(ExperimentalMaterial3Api::class)  @Composable  fun DetailScreen(  viewModel: SearchViewModel,  playlistsViewModel: PlaylistsViewModel,  onBack: () -> Unit  ) {  val colors = MaterialTheme.colorScheme  val track = viewModel.selectedTrack.collectAsState().value  val playlists by playlistsViewModel.playlists.collectAsState(initial = emptyList())  var showPlaylists by rememberSaveable { mutableStateOf(false) }  val sheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)  if (track == null) {  Text(text = stringResource(R.string.detail\_track\_not\_selected), color = colors.onSurfaceVariant)  return  }  if (showPlaylists) {  ModalBottomSheet(  onDismissRequest = { showPlaylists = false },  sheetState = sheetState  ) {  Column(  modifier = Modifier  .fillMaxWidth()  .padding(horizontal = 16.dp, vertical = 8.dp),  verticalArrangement = Arrangement.spacedBy(8.dp)  ) {  Text(  text = stringResource(R.string.detail\_choose\_playlist),  style = MaterialTheme.typography.titleMedium,  color = colors.onSurface  )  playlists.forEach { playlist ->  Row(  modifier = Modifier  .fillMaxWidth()  .clickable {  playlistsViewModel.insertTrackToPlaylist(track, playlist.id)  showPlaylists = false  }  .padding(vertical = 8.dp),  verticalAlignment = Alignment.CenterVertically,  horizontalArrangement = Arrangement.spacedBy(12.dp)  ) {  Text(text = playlist.name, modifier = Modifier.weight(1f))  Text(  text = stringResource(R.string.detail\_tracks\_in\_playlist, playlist.tracks.size),  color = colors.onSurfaceVariant  )  }  }  }  }  }  Surface(color = colors.background) {  Column(  modifier = Modifier  .fillMaxSize()  .statusBarsPadding()  .padding(16.dp)  .verticalScroll(rememberScrollState()),  verticalArrangement = Arrangement.Top  ) {  Row(  modifier = Modifier.fillMaxWidth(),  verticalAlignment = Alignment.CenterVertically  ) {  IconButton(onClick = onBack) {  Icon(  imageVector = Icons.Filled.ArrowBack,  contentDescription = stringResource(R.string.content\_desc\_back)  )  }  Spacer(modifier = Modifier.width(8.dp))  Text(  text = stringResource(R.string.detail\_screen\_title),  style = MaterialTheme.typography.titleMedium  )  }  Spacer(modifier = Modifier.height(16.dp))  AsyncImage(  model = track.artworkUrl ?: track.coverResId ?: R.drawable.ic\_music,  contentDescription = stringResource(R.string.content\_desc\_track\_cover, track.trackName),  contentScale = ContentScale.Crop,  placeholder = painterResource(id = R.drawable.ic\_music),  error = painterResource(id = R.drawable.ic\_music),  modifier = Modifier  .fillMaxWidth()  .height(240.dp)  )  Spacer(modifier = Modifier.height(16.dp))  Text(text = track.trackName, style = MaterialTheme.typography.titleLarge)  Text(text = track.artistName, style = MaterialTheme.typography.titleMedium, color = colors.onSurfaceVariant)  Spacer(modifier = Modifier.height(8.dp))  Text(  text = stringResource(R.string.detail\_year, track.year?.toString().orEmpty()),  style = MaterialTheme.typography.bodyMedium  )  Text(  text = stringResource(R.string.detail\_album, track.album.orEmpty()),  style = MaterialTheme.typography.bodyMedium  )  Text(text = stringResource(R.string.detail\_duration, track.trackTime), style = MaterialTheme.typography.bodyMedium)  Spacer(modifier = Modifier.height(12.dp))  Row(horizontalArrangement = Arrangement.spacedBy(12.dp)) {  Button(onClick = { viewModel.toggleFavorite(track, !track.favorite) }) {  Icon(  imageVector = if (track.favorite) Icons.Filled.Favorite else Icons.Filled.FavoriteBorder,  contentDescription = stringResource(R.string.favorites\_title)  )  Spacer(modifier = Modifier.width(8.dp))  Text(  if (track.favorite) {  stringResource(R.string.detail\_from\_favorites)  } else {  stringResource(R.string.detail\_to\_favorites)  }  )  }  Button(onClick = { showPlaylists = true }) {  Text(stringResource(R.string.detail\_add\_to\_playlist))  }  }  }  }  } |

**Листинг 6 – data/db/AppDatabase.kt**

|  |
| --- |
| package com.example.playlist\_maker\_android\_albahtinilya.data.db  import androidx.room.Database  import androidx.room.RoomDatabase  @Database(entities = [TrackEntity::class, PlaylistEntity::class], version = 2, exportSchema = false)  abstract class AppDatabase : RoomDatabase() {  abstract fun trackDao(): TrackDao  abstract fun playlistDao(): PlaylistDao  } |

**Листинг 7 – data/db/TrackEntity.kt**

|  |
| --- |
| package com.example.playlist\_maker\_android\_albahtinilya.data.db  import androidx.room.Entity  import androidx.room.PrimaryKey  @Entity(tableName = "tracks")  data class TrackEntity(  @PrimaryKey val id: Long,  val trackName: String,  val artistName: String,  val trackTime: String,  val coverResId: Int? = null,  val artworkUrl: String? = null,  val year: Int? = null,  val album: String? = null,  val playlistId: Long = 0,  val favorite: Boolean = false  ) |

**Листинг 8 – data/db/PlaylistEntity.kt**

|  |
| --- |
| package com.example.playlist\_maker\_android\_albahtinilya.data.db  import androidx.room.Entity  import androidx.room.PrimaryKey  @Entity(tableName = "playlists")  data class PlaylistEntity(  @PrimaryKey(autoGenerate = true) val id: Long = 0,  val name: String,  val description: String,  val coverImageUri: String? = null  ) |

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы разработан прототип Android-приложения «Playlist Maker» для поиска треков и управления плейлистами.

В приложении реализованы основные пользовательские сценарии: поиск треков через iTunes Search API, просмотр карточки трека,

управление избранными треками, создание и управление плейлистами, история поиска, а также экран настроек с дополнительными действиями.

Использование Jetpack Compose, ViewModel и Kotlin Coroutines обеспечило реактивный и отзывчивый интерфейс,

а применение Room и DataStore — сохранность пользовательских данных между запусками приложения.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Android Developers. Jetpack Compose: https://developer.android.com/jetpack/compose

2. Android Developers. Guide to app architecture (MVVM): https://developer.android.com/topic/architecture

3. Android Developers. ViewModel: https://developer.android.com/topic/libraries/architecture/viewmodel

4. Android Developers. Room Persistence Library: https://developer.android.com/training/data-storage/room

5. Android Developers. DataStore: https://developer.android.com/topic/libraries/architecture/datastore

6. Android Developers. Navigation Compose: https://developer.android.com/jetpack/compose/navigation

7. Kotlin Coroutines: https://kotlinlang.org/docs/coroutines-overview.html

8. Retrofit 2: https://square.github.io/retrofit/

9. OkHttp: https://square.github.io/okhttp/

10. iTunes Search API (Apple): https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/iTuneSearchAPI/

11. Material Design 3: https://m3.material.io/