МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА:**

По дисциплине «Программирование Мобильных Систем»

Тема Программное средство **«**Музыкальная платформа»

**Исполнитель**

студент 3 курса группы 8 Гарабажиу М.А.

подпись, дата

**Руководитель**

ассистент Уласевич Н.И.

должность, ученая степень, ученое звание подпись, дата

Допущен(а) к защите \_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласевич Н.И.

подпись дата инициалы и фамилия

**Содержание**

[Введение 5](#_Toc166935915)

[1 Анализ требований к программному средству 6](#_Toc166935916)

[1.1 Аналитический обзор аналогов 6](#_Toc166935917)

[1.1.1 Spotify 6](#_Toc166935918)

[1.1.2. YouTube Music 9](#_Toc166935919)

[1.2 Разработка функциональных требований, определенных вариантов использования 12](#_Toc166935920)

[1.3 Вывод по разделу 13](#_Toc166935921)

[2 Проектирования программного средства 14](#_Toc166935922)

[2.1 Проектирование базы данных 14](#_Toc166935923)

[2.2 Проектирование ролей 16](#_Toc166935924)

[2.3 Проектирование навигации 16](#_Toc166935925)

[2.4 Спецификация функциональных требований 16](#_Toc166935926)

[2.5 Вывод по разделу 17](#_Toc166935927)

[3 Реализация приложения 18](#_Toc166935928)

[3.1 База данных и взаимодействие с ней 18](#_Toc166935929)

[3.2 Авторизация и регистрация 20](#_Toc166935930)

[3.3 Описание структуры проекта 20](#_Toc166935931)

[3.4 Вывод по разделу 22](#_Toc166935932)

[4 Руководство пользователя 23](#_Toc166935933)

[4.1 Вывод по разделу 28](#_Toc166935934)

[Заключение 29](#_Toc166935935)

[Список использованных источников 30](#_Toc166935936)

[Приложение A. Классы для работы с базой данных 31](#_Toc166935937)

# **Введение**

Во времена быстро развивающихся технологий стало актуальным использование вспомогательных сервисов для быстрого поиска информации. Область использования при этом может быть совершенно любой. Всё ограничивается лишь полетом фантазии человека и ёмкостью ресурсов сети Интернет, через которую можно получить доступ к всевозможным данным.

В связи с тем, что музыка является неотъемлемой частью жизни любого человека, было принято решение о создании музыкального сервиса. Эта же тема и была выбрана в качестве направленности курсового проекта, где необходимо было применить навыки владения технологиями frontend и backend программирования, различными их ответвлениями и интерпретациями для построения структуры проекта в мобильной версии.

Данная платформа упростит жизнь пользователю, так как станет возможным решение таких базовых задач, как:

1. Выполнение регистрации и авторизации.
2. Быстрый поиск конкретной музыки по названию или исполнителю.
3. Контроль над панелью воспроизводимой мелодии.
4. Создание своего плейлиста на основе понравившихся композиций.

Эти и другие функции предоставляются после авторизации пользователя.

Конечной целью курсового проекта является разработка программного средства, которое сможет на основе доступа к ресурсам предоставлять возможности по реализации.

Языком разработки выбран Java. Основываясь на принципах ООП будут проведены различные манипуляции над проектом. Для более систематизированной визуализации использована среда разработки Android Studio. В качестве средства проектирования базы данных используется SQLite.

# **Анализ требований к программному средству**

При создании программного продукта анализ прототипов является неотъемлемой частью при построении структуры проекта. При изучении сайтов-прототипов, основой которых служат их приложения, было выявлено несколько экземпляров схожих по функциональности. Приложения были протестированы на устройстве Galaxy A50 API 30, Android 10.

* 1. Аналитический обзор аналогов

1.1.1 Spotify

В глобальной сети Интернет существует большое количество проектов-аналогов, схожих друг на друга своей структурой и наполненностью. Можно рассмотреть ряд примеров, но везде будет заметна одна закономерность в построении каркаса. В связи с этим ниже рассматривается самый яркий представитель из всех существующих ныне музыкальных платформ – Spotify [1].

Рассмотрим первую страницу, которую видит пользователь при переходе на сайт. Можно заметить, что весь нужный пользователю функционал доступен с первых секунд посещения веб-сайта, что является хорошим способом по привлечению клиентов. На рисунке 1.1 можно ознакомиться с этим высказыванием визуально.

Если рассматривать каждую отдельную часть сайта, стоит отметить присутствие ссылки перехода на форму регистрации и возможность быстрого входа в ваш аккаунт, но при этом, даже если вы еще не зарегистрировались, система дает вам возможность использования платформы в режиме реального времени. Присутствие пункта справки также положительно раскрывает сайт со стороны удобства для пользователя. На отдельных подкатегориях музыкальной делимости пользователь может нажать на «Показать все» и найти то, что его интересует. Боковая панель оснащена самыми необходимыми пунктами, облегчающими пользование музыкальной платформой, предоставляя дополнительные возможности.

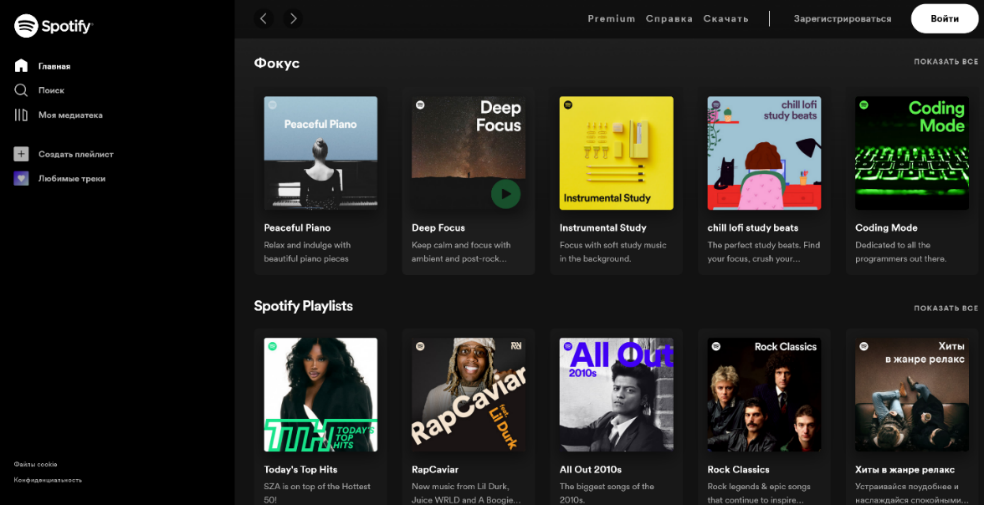


Рисунок 1.1 – Домашняя страница веб-сайта Spotify

Форма регистрации представлена на рисунке 1.2. Пользователь может внести свои данные и благодаря адресу электронной почты получить доступ к ресурсу. Если же аккаунт уже существует, то авторизация происходит с указанием только логина и пароля.

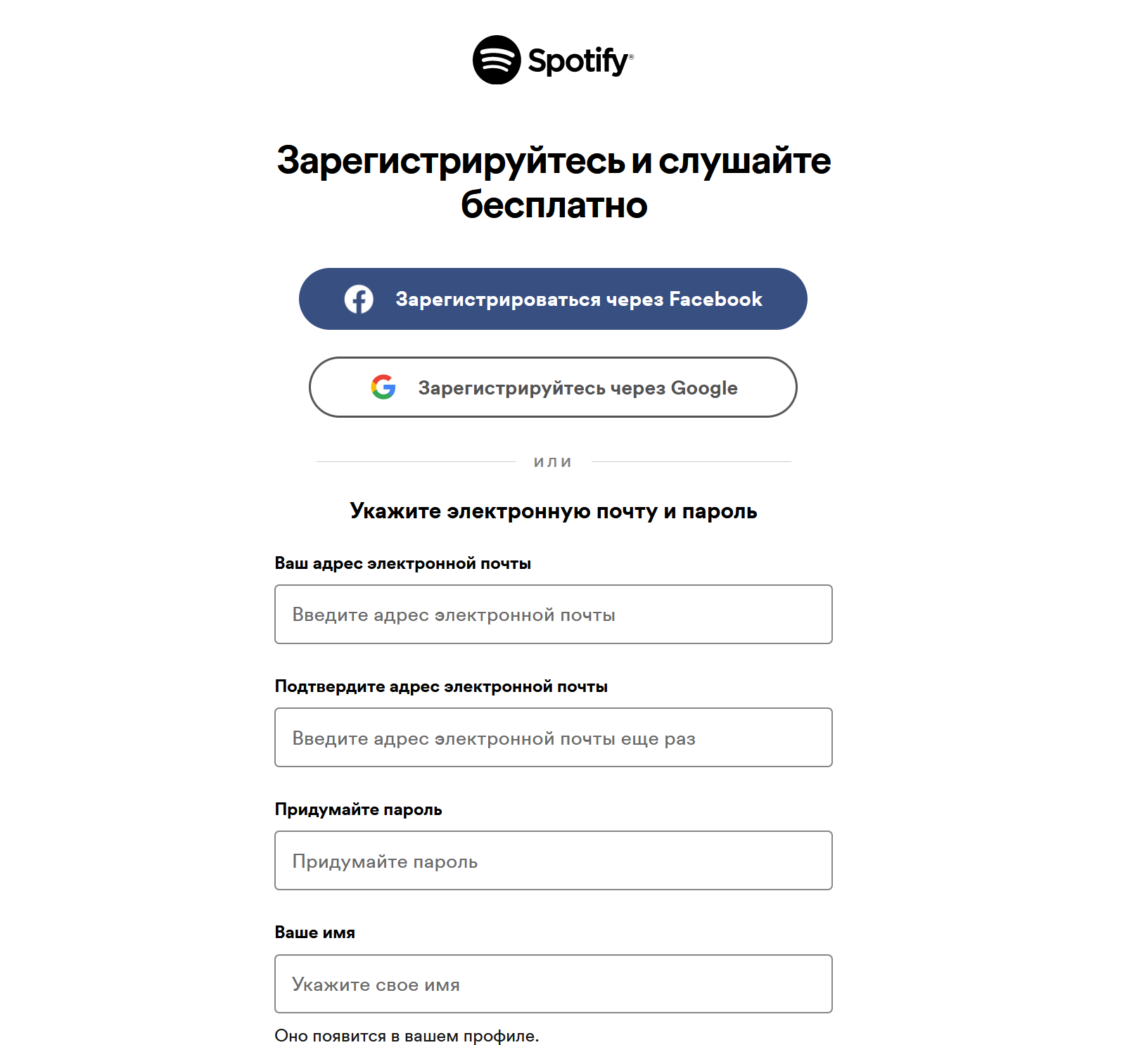
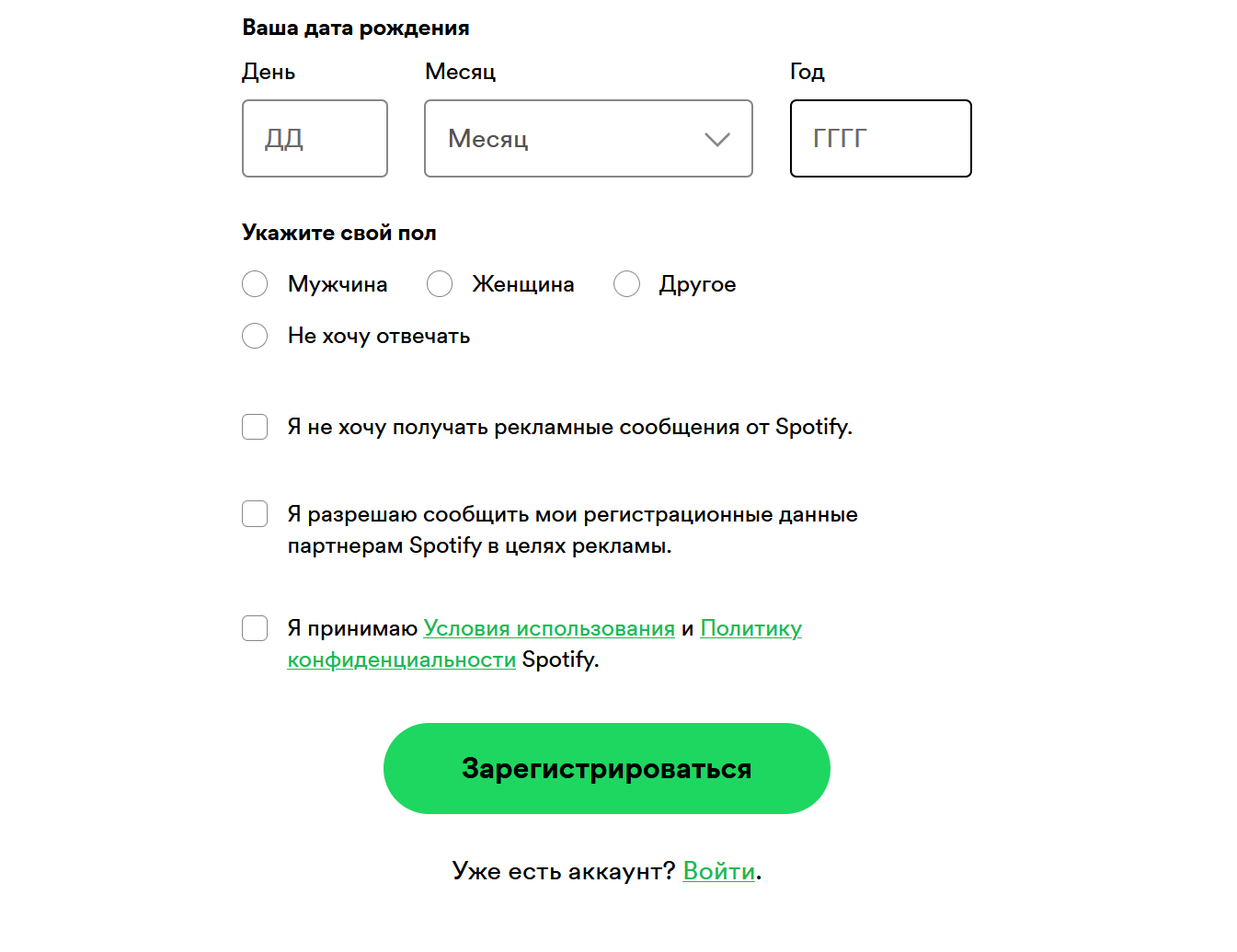
 

Рисунок 1.2 – Форма регистрации и авторизации пользователя

Протестируем работу функционала веб-сайта на конкретных примерах.

Функция добавления треков в избранные, а также создание и наполнение своего плейлиста значительно дополняет набор возможностей пользователей.

При нажатии на конкретный плейлист ниже отобразятся все композиции, входящие в него, описание в виде темы плейлиста, количество лайков, треков, и время, которое вы затратите на прослушивание. На рисунке 1.3 показано, как все эти детали размещены.

На рисунке 2.4 демонстрируется страница, появляющаяся при нажатии на панели навигации пункта «Поиск». Приятным бонусом стало появление не только панели поиска в верхней части экрана, но и предложения в основном поле по выбору музыки по предпочитаемому жанру, то есть, даже если вы сами не знаете, чего хотите, вы сможете в довольно короткие сроки найти что-то по вкусу.

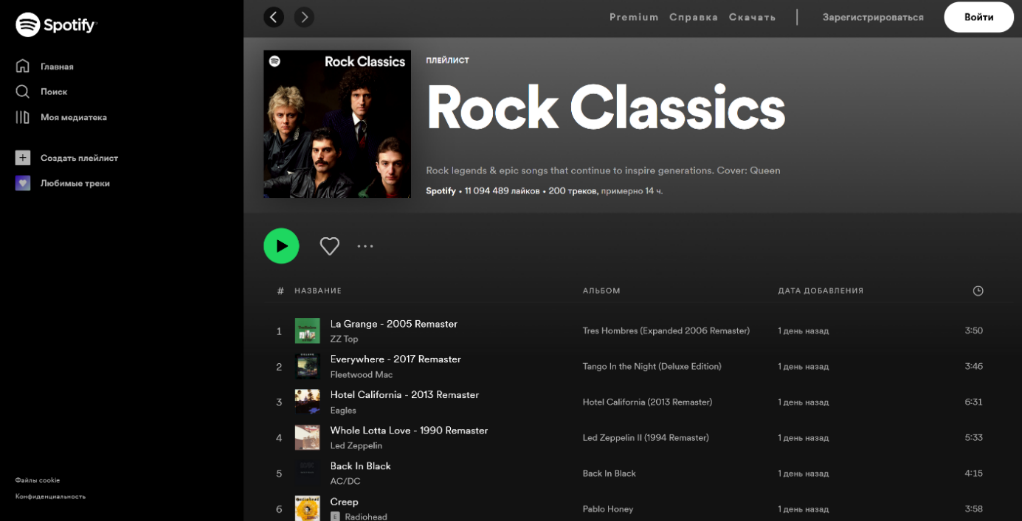


Рисунок 1.3 – Отображение наполненности плейлиста

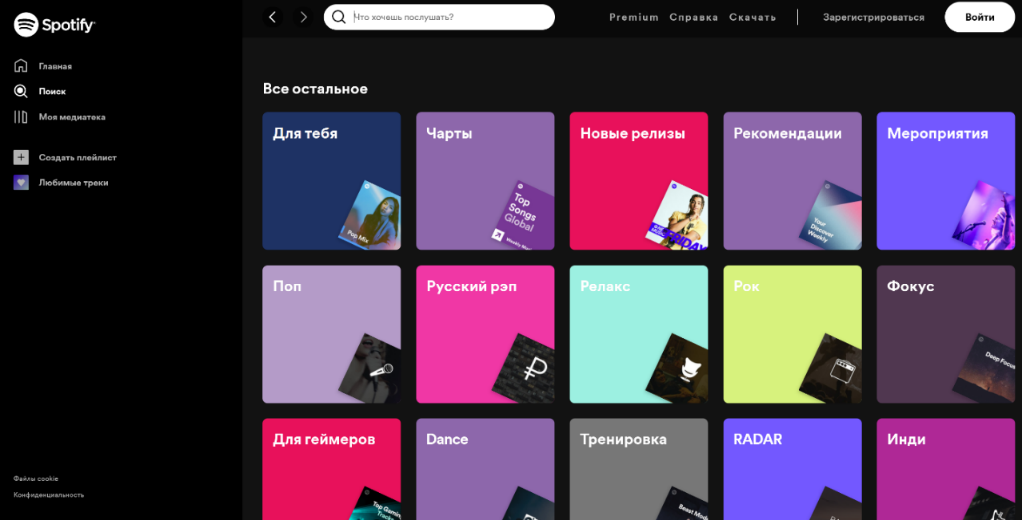


Рисунок 1.4 – Демонстрация открытия панели поиска

Затрагивая далее тему поиска информации, на рисунке 1.5 отображена детализированная комплектация интерфейса в случае появления запроса к серверу. Как можно заметить, вне зависимости от того, по какому из критериев ведется поиск, система в любом случае будет выдавать результаты каждого из фильтров, будь то название песни, альбома или исполнителя, для однозначного удовлетворения запросов пользователей.



Рисунок 1.5 – Демонстрация работы системы поиска информации

Подводя итоги рассмотрения такой полномасштабной платформы, как Spotify, были отмечены основные возможности, коррелирующие с внедрением технологий проектирования баз данных.

1.1.2. YouTube Music

С недавних пор имеются различные дополнения платформ, которые ссылаются друг на друга, при этом связанные как минимум разработкой одной организацией. YouTube Music [2] вошла в ряд тех самых немногочисленных программных объектов. В данном случае имеется прямая связь с намного ранее созданной медиа платформой YouTube. Это стратегически продуманный и окупаемый ход представителей менеджмента и разработчиков. Ибо синхронизация одной платформы с другой расширяет возможности пользователя за счёт автоматической наполненности контента на основе их предпочтений, учитывая заранее, что у подавляющего большинства наверняка имеется свой аккаунт и база косвенно затронутых просмотренных видеоклипов по мелодиям на YouTube.

Результат синхронизации аккаунтов можно увидеть на рисунке 2.6. На нем показан профиль пользователя, который предоставляет ряд смежных возможностей. Авторизация пользователя привязывалась именно к аккаунту Google, что может заметно как увеличить время в случае регистрации с нуля, так и уменьшить при условиях уже существующего аккаунта.

Первым, что увидит пользователь после манипуляций выше, будет главная страница. На рисунке 1.7 отображено предоставление ряда функционала для ориентирования клиента. Сервис автоматически подобрал существующие плейлисты на основе данных запросов пользователя на том же аккаунте. С первых секунд можно выбрать течение музыки под текущее состояние благодаря хлебным крошкам. Для более детального поиска сверху размещена панель навигации, заменяющая привычное боковое меню.

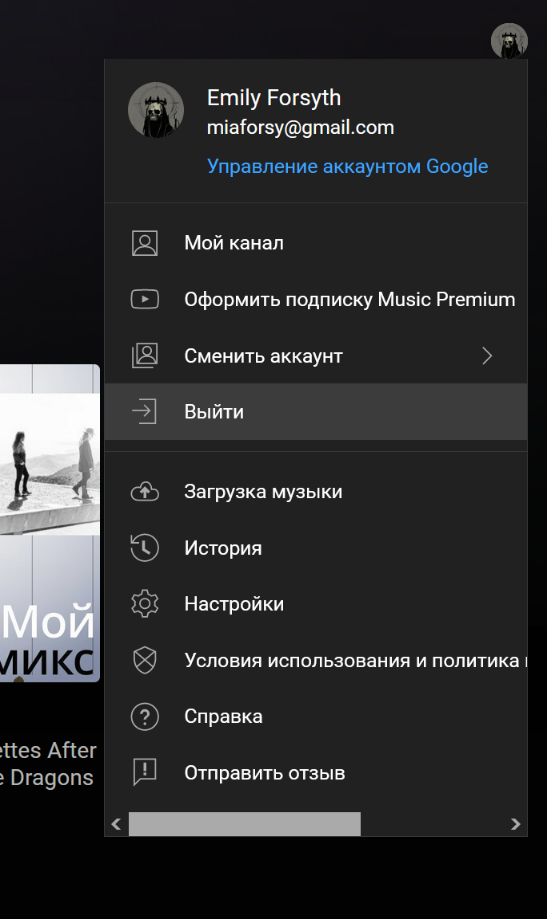


Рисунок 1.6 – Личный кабинет авторизированного пользователя

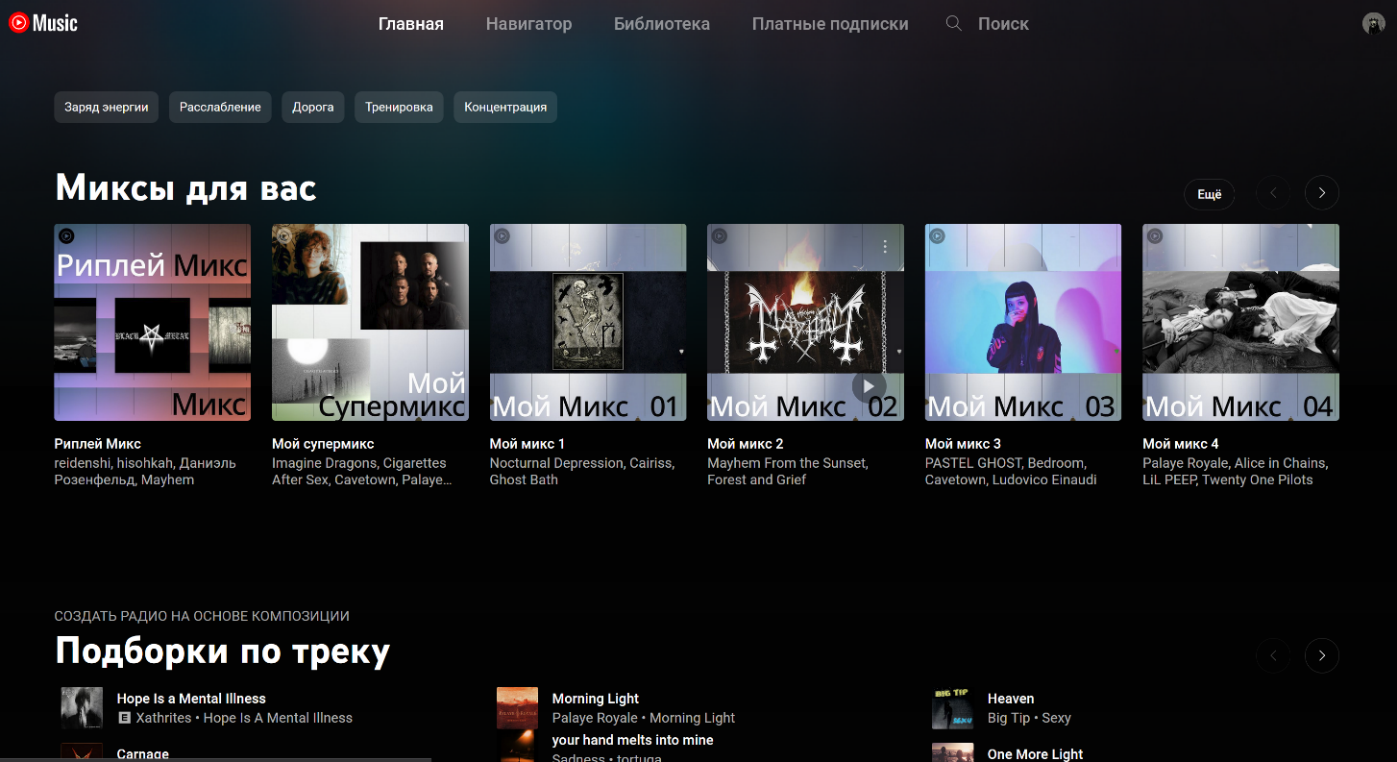


Рисунок 1.7 –Главная страница сайта YouTube Music

При переходе на панели управления по навигатору, пользователь сможет выбрать на основе более подробной базы классифицирующего материала то жанровое ответвление, которое ему будет ближе. Также автоматически генерируется список ваших часто используемых жанров, настроений или событий. Реализация показана на рисунке 1.8.

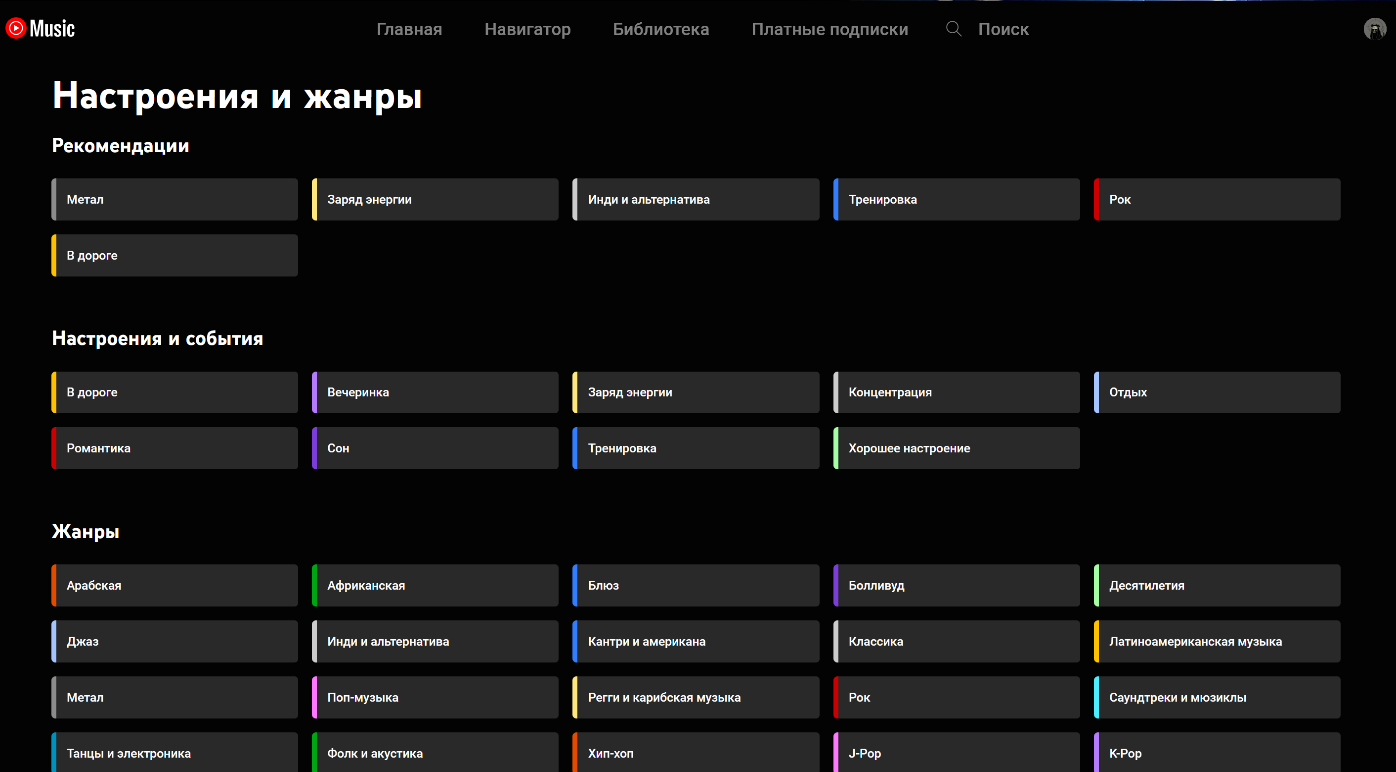


Рисунок 1.8 – Жанровая классификация

Если перейти на вкладку «Библиотека», откроется возможность поиска музыки по исполнителю. При выборе определенного исполнителя пользователь будет иметь доступ к трекам, альбомам, а также сможет просмотреть краткую информацию об исполнителе. Отображено на рисунке 1.9.

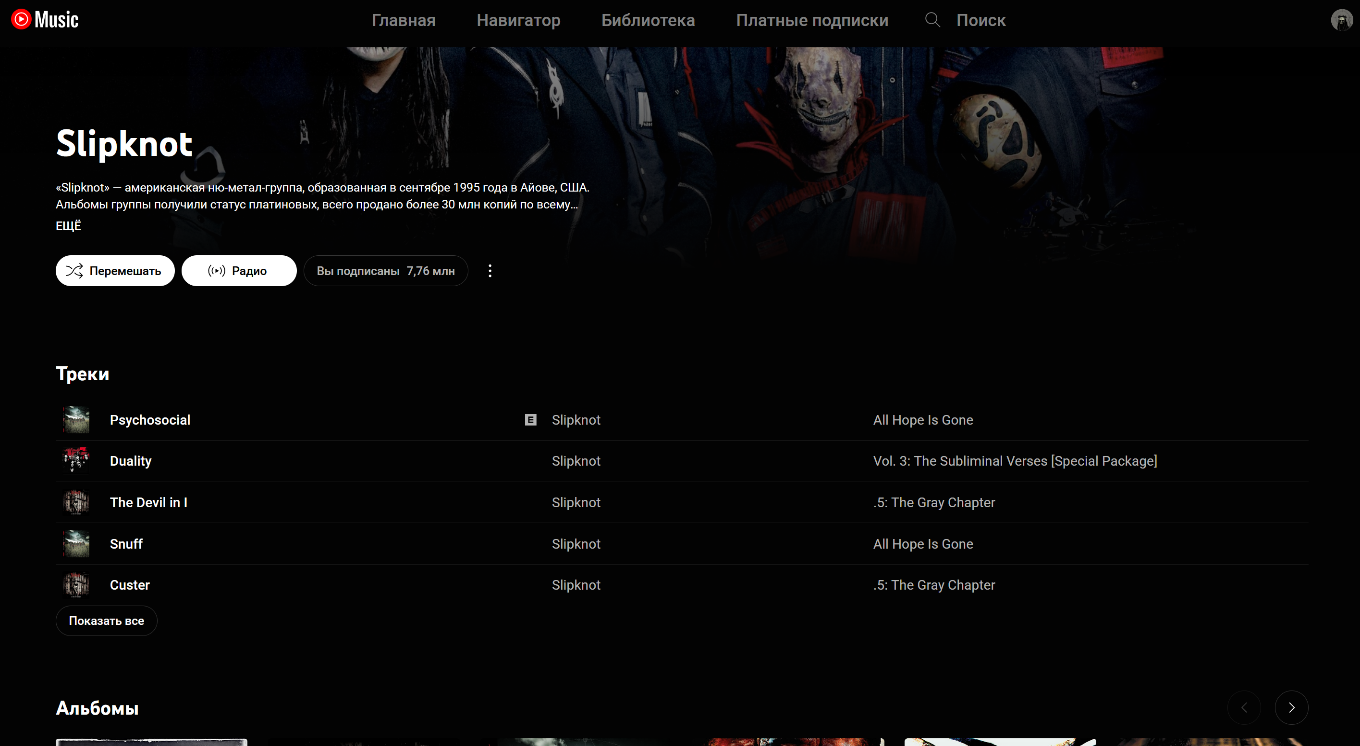


Рисунок 1.9 – Профиль исполнителя

Перейдя к теме поиска конкретной мелодии, здесь технология мало отличима от технологии, используемой в первом аналоге. Это значит, что при поиске по любому из критериев, будь то название композиции, исполнителя, альбома или плейлиста, платформа предоставит доступ к каждому из этих подразделений в своем привычном порядке, как показано на рисунке 1.10.

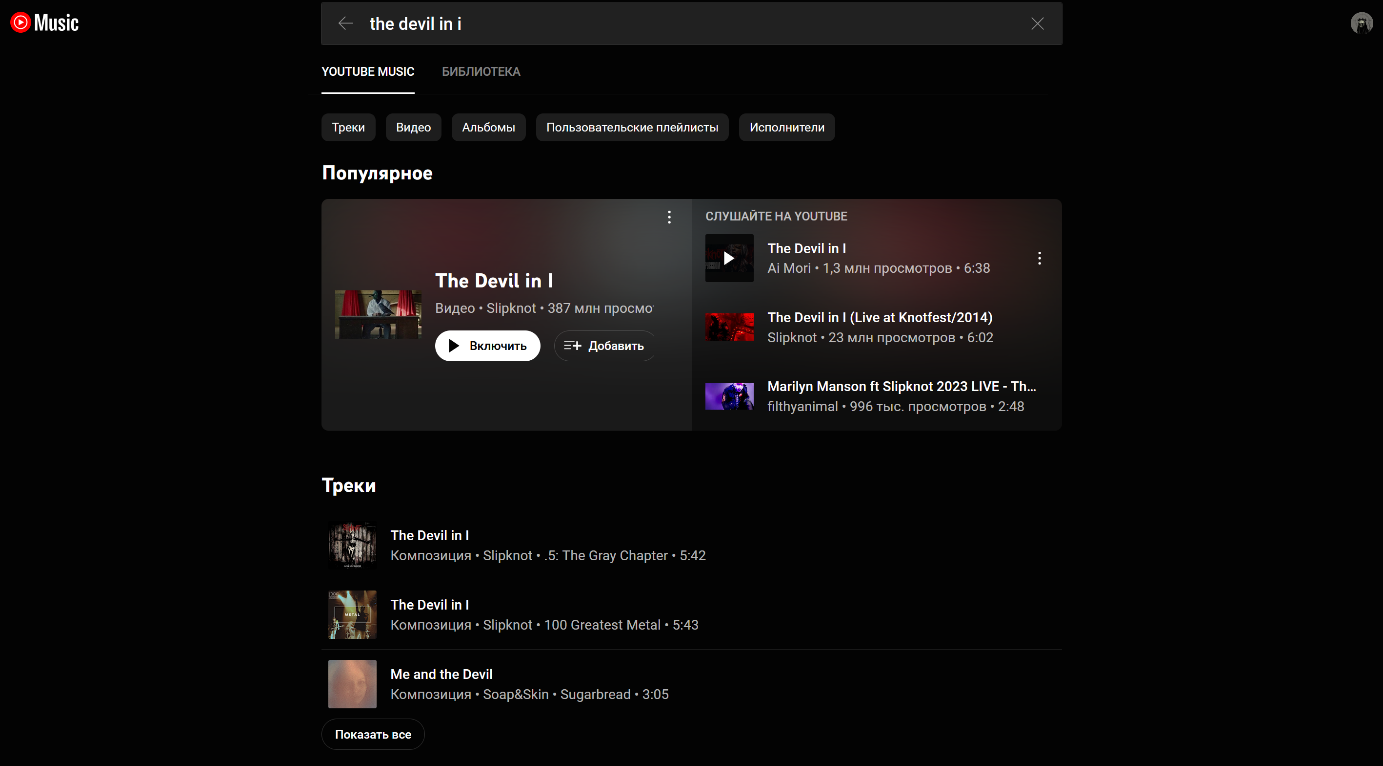


Рисунок 1.10 – Поиск по введенному критерию

Рассмотрев основной функционал такой платформы, как YouTube Music, можно отметить, что многие структурные единицы дополнены. Идея синхронизации профилей делает данный объект более конкурентноспособным и делает более практичным использование сервиса в целом.

* 1. Разработка функциональных требований, определенных вариантов использования

Функциональные требования, выполняемые приложением:

1. Выполнение регистрации и авторизации.
2. Просмотр предоставленных композиций по исполнителю или жанру.
3. Оснащение панели контроля над воспроизводимой мелодией функционалом.
4. Внедрение панели навигации.
5. Занесение в плейлист понравившейся композиции.
6. Предоставление возможности добавлять и удалять композицию.
7. Редактирование учетной записи.

При разработке приложения использованы:

* интегрированная среда разработки Android Studio.
* язык программирования Java.
* Microsoft SQLite.
  1. Вывод по разделу

При рассмотрении аналогов были выявлены основные направленности по реализации структуры платформ.

Подводя итоги рассмотрения такой полномасштабной платформы, как Spotify, были отмечены основные возможности, коррелирующие с внедрением технологий проектирования баз данных.

Рассмотрев основной функционал такой платформы, как YouTube Music, можно отметить, что многие структурные единицы дополнены. Идея синхронизации профилей делает данный объект более конкурентноспособным и делает более практичным использование сервиса в целом.

Также были отмечены функциональные требования, выполняемые приложением, и средства реализации программного продукта.

1. **Проектирования программного средства**

В этом разделе рассматриваются ключевые аспекты проектирования разрабатываемого приложения. Будет рассмотрена структура приложения, выбор технологий, а также схема навигации в приложении. Этот раздел поможет лучше понять, как разрабатываемое приложение будет работать и какие задачи должны быть выполнены для успешной реализации.

* 1. **Проектирование базы данных**

Для достижения поставленных задач была создана база данных Boom. Для ее создания использовалась система управления реляционными базами данных SQLite. Реляционные базы данных позволяют объединять информацию, принадлежащую разным сущностям, то есть объектам, в которых хранятся данные.

Единицей хранящейся в БД информации является таблица. Каждая таблица представляет собой совокупность строк и столбцов, где строки соответствуют экземпляру объекта, конкретному событию или явлению, а столбцы – атрибутам (признакам, характеристикам, параметрам) объекта, события или явления. Каждая строка содержит сведения о конкретном событии.

База данных Boom состоит из таблиц, представленных на рисунке 2.1.

Таблицы имеют первичные и вторичные ключи. Первичный ключ – это поле или набор полей, однозначно идентифицирующий запись. Вторичные ключи устанавливаются по полям, которые часто используются при поиске или сортировки данных. В отличие от первичных ключей, поля для вторичных ключей могут содержать не уникальную информацию. Всё это необходимо для установления связей между таблицами.

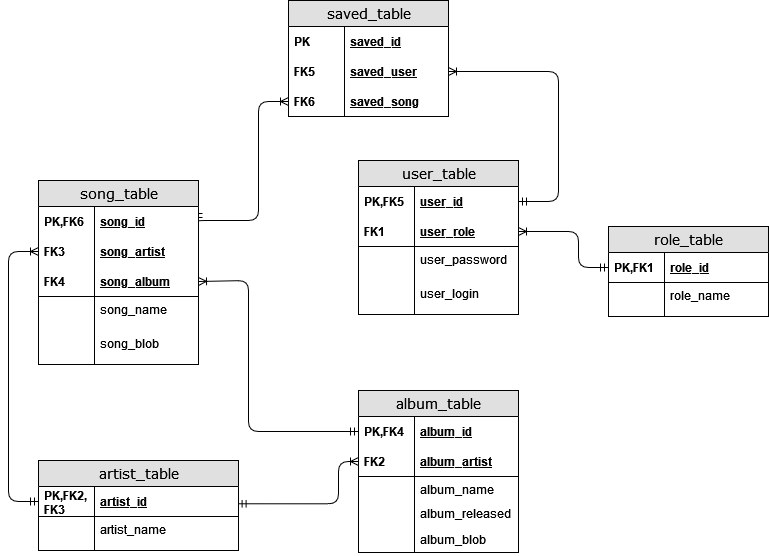


Рисунок 2.1 – ER-диаграмма представления базы данных

Рассматривая структуру базы данных Boom, можно увидеть, что чаще всего использовалось реляционное отношение один-ко-многим между таблицами. Это отношение характеризуется соответствием всякой записи в родительской таблице, записи в дочерней таблице.

Проанализируем каждую из таблиц.

Таблица album\_table представляет собой альбомы, выпущенные исполнителями, и потому содержит следующие столбцы: album\_id (идентификатор альбома, тип number(10), первичный ключ), artist\_id (идентификатор исполнителя, тип number(10), внешний ключ), album\_name (текстовое название альбома, тип varchar2(30)), album\_released (год выпуска альбома, тип number(10)) и album\_blob (столбец для хранения обложки альбома, тип данных blob).

Таблица artist\_table содержит информацию о музыкальных исполнителях: artist\_id (идентификатор исполнителя, тип number(10), первичный ключ), artist\_name (имя исполнителя, тип varchar2(10)).

Таблица role\_table представляет собой перечень ролей, которые могут иметь пользователи. Имеющиеся столбцы: role\_id (идентификатор роли, типа number(10), первичный ключ) и role\_name (текстовое название роли, тип varchar2(30)). Имеющиеся по умолчанию роли: USER (рядовой пользователь) и OWNER (пользователь с правами администратора).

Таблица song\_table отвечает за хранение информации о музыкальных композициях, имеющихся в базе данных. А именно: song\_id (идентификатор песни, тип number(10), первичный ключ), song\_artist (идентификатор исполнителя, тип number(10), внешний ключ), song\_album (идентификатор альбома, тип number(10), внешний ключ), song\_name (текстовое название песни, тип varchar2(30)) и song\_blob (столбец для хранения аудиофайла песни, тип данных blob).

Таблица user\_table содержит информацию о пользователях, а именно: user\_id (идентификатор пользователя, тип number(10), первичный ключ), user\_login (логин пользователя, типа varchar2(30)), user\_password (пароль пользователя, тип varchar2(30), шифруется в процессе регистрации и расшифровывается при проверке во время авторизации) и user\_role (идентификатор роли, тип number(10), внешний ключ).

Последняя таблица, saved\_table, используется для сопоставления пользователя с понравившейся ему композицией. Благодаря ей пользователи сохраняют песни в свой плейлист.

Информация, хранимая в данной таблице: saved\_id (идентификатор экземпляра сохраненной песни, number(10), первичный ключ), saved\_user (идентификатор пользователя, сохранившего песню; тип number(10), внешний ключ) и saved\_song (идентификатор сохраненной песни, тип number(10), внешний ключ).

* 1. **Проектирование ролей**

В качестве основной роли взят суперпользователь, т.к. приложение направлено на локальное использование. Он владеет комбинированными возможностями стандартного юзера и администратора.

* 1. **Проектирование навигации**

При проектировании навигации по страницам были использованы специальные методы и компоненты, предоставляемые средой разработки Java под Android. Схема навигации включает в себя ориентирование между страницами входа и регистрации, далее переход на главную страницу с навигационным меню.

Для реализации навигации используются Intent и компоненты, предоставляемые Android. Экран входа проверяет учетные данные пользователя и при успешной аутентификации перенаправляет его на главный экран через Intent. В случае неверных данных пользователю предлагается ввести лицензионный ключ.

Главный экран организован с использованием BottomNavigationView и ViewPager. BottomNavigationView обеспечивает переключение между вкладками, а ViewPager — плавные переходы между ними. Вкладки представляют различные разделы приложения: альбомы, исполнители, песни, плейлисты и директории.

* 1. **Спецификация функциональных требований**

Программное средство предусматривает ряд функций для каждого объекта и выстраивает логику последовательных действий.

В связи с тем, что приложение ориентировано на локальное использование, предусмотрена роль суперпользователя, обладающего правами стандартного юзера и администратора.

Суперпользователь сможет выполнять следующее: авторизация, регистрация (в случае отсутствия учетных данных), редактирование плейлиста (добавить в плейлист, удалить из плейлиста), получение списка песен, воспроизведение песни и контроль над ней (с начала, остановить), поиск (исполнителя, альбома, песни в плейлисте или в списке песен).

Для более наглядного представления функциональных возможностей были использованы средства визуального моделирования.

Для отображения более общего понимания границ возможностей пользования программным средством была построена диаграмма последовательности.

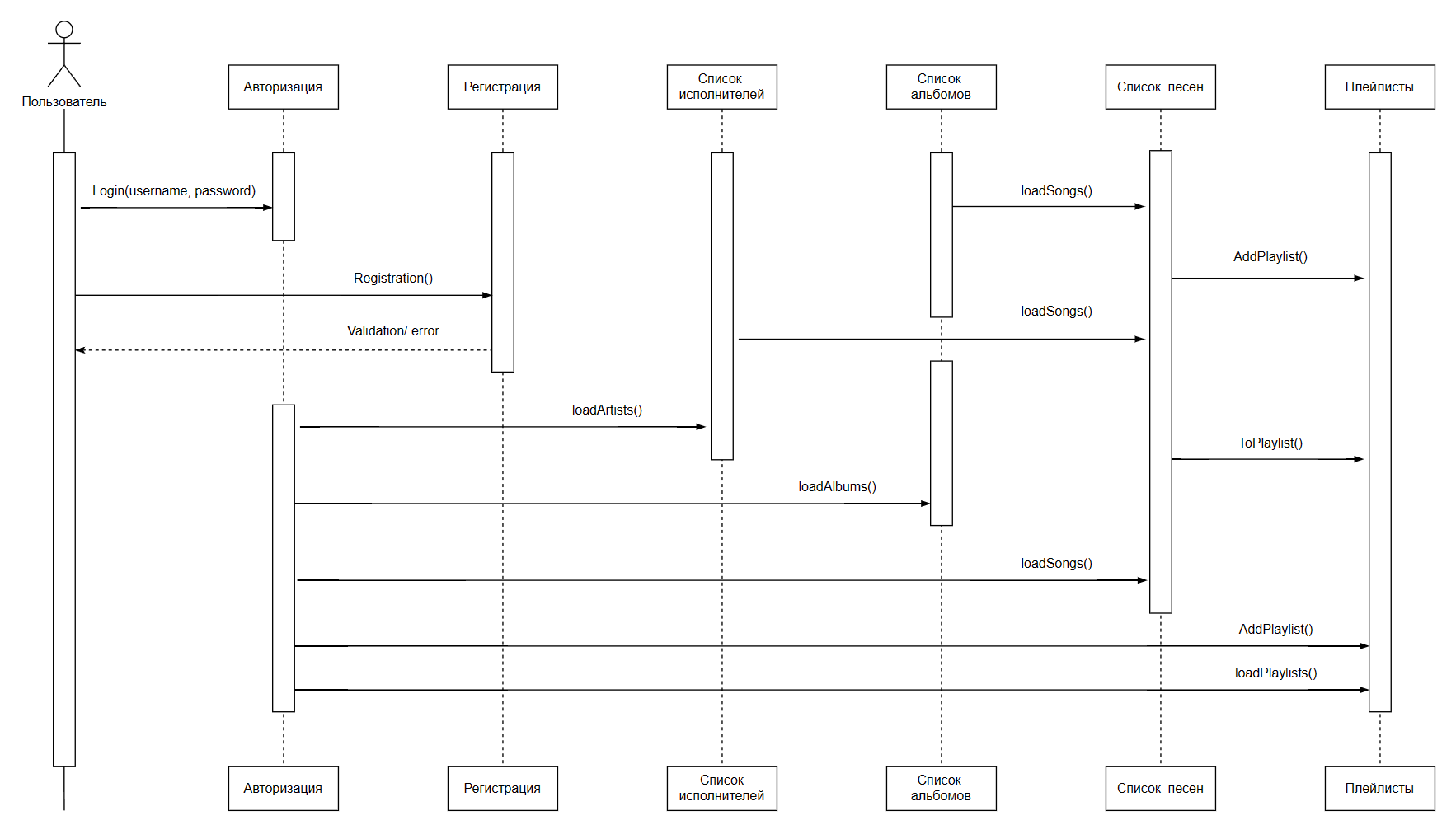


Рисунок 2.2 – Диаграмма последовательности приложения

На рисунке 2.2 изображена диаграмма последовательности данного приложения для пользователя, отображающая навигационные возможности.

* 1. **Вывод по разделу**

В данном разделе были рассмотрены ключевые элементы проекта. В частности, было выполнено проектирование базы данных приложения, включая его физическую и логическую структуру, которая будет поддерживать весь функционал. Была назначена роль, исходя из нужд приложениы, и описана структура проекта, включая подробное описание навигации. Эти шаги позволили разграничить проект и определить основные направления в создании приложения.

1. **Реализация приложения**

Данный раздел содержит в себе описание шагов разработки приложения и его элементов. Разработка приложения будет выполнена в IDE Android Studio, версия Android SDK – 3.20.2. Первоначальная реализация будет протестирована на Android-устройстве Google Pixel 8 API 30.

* 1. **База данных и взаимодействие с ней**

Для работы с базой данных разработан класс DatabaseHelper, рассмотрим его содержимое – Листинг 3.1. Полный скрипт работы с базой данных представлен в приложении А.

|  |
| --- |
| public class DataBaseHelper extends SQLiteOpenHelper  {  private DataBaseHelper(Context context) {  super(context, *DATABASE\_NAME*, null, *DATABASE\_VERSION*);  mContext = context; }  public static synchronized DataBaseHelper getDatabaseHelper(Context context) {  if (*mDatabaseHelper* == null)  *mDatabaseHelper* = new DataBaseHelper(context.getApplicationContext());  return *mDatabaseHelper*; } |

Листинг 3.1 – Содержимое класса DatabaseHelper

Класс DataBaseHelper наследуется от SQLiteOpenHelper, что позволяет использовать его функционал для работы с базой данных SQLite. Внутри класса определен приватный конструктор и статический метод.

Приватный конструктор используется для создания экземпляра DataBaseHelper. Он вызывает конструктор суперкласса SQLiteOpenHelper, передавая ему контекст приложения, имя базы данных, курсоры и версию базы данных. Кроме того, в конструкторе сохраняется контекст приложения в переменной mContext.

Статический метод getDatabaseHelper используется для получения единственного экземпляра класса DataBaseHelper. Он реализован как синхронизированный, чтобы обеспечить потокобезопасность при создании экземпляра класса. Если mDatabaseHelper равен null, то создается новый экземпляр DataBaseHelper с использованием контекста приложения context.getApplicationContext(). В противном случае возвращается существующий экземпляр mDatabaseHelper. Это позволяет избежать создания нескольких экземпляров базы данных и обеспечивает централизованный доступ к базе данных из разных частей приложения.

На листинге 3.2 представлен метод buildLibrary(), в котором происходит построение библиотеки музыкальной коллекции на основе данных из базы данных.

|  |
| --- |
| private void buildLibrary() {  fadeInFadeOut();  mCompositeDisposable.add(Observable.fromCallable(() -> CursorHelper.buildMusicLibrary(this))  .subscribeOn(Schedulers.io())  .observeOn(AndroidSchedulers.mainThread())  .subscribeWith(new DisposableObserver<Boolean>() {  @Override  public void onNext(Boolean aBoolean) {  (MainActivity или LoginActivity).  Intent intent = new Intent((Context) SplashActivity.this, LoginActivity.class);  startActivity(intent);  SplashActivity.this.finish();  }  @Override  public void onError(Throwable e) {  Logger.exp("" + e.getMessage());  }  @Override  public void onComplete() {  }  })  );  } |

Листинг 3.2 – Содержимое метода для построения библиотеки музыки

Создается объект Observable, который будет выполнять метод buildMusicLibrary() из класса CursorHelper. В этом методе, происходит извлечение данных из базы данных (таких как треки, альбомы и исполнители), формирование библиотеки музыки и возврат значения типа Boolean в зависимости от успешности построения библиотеки.

Также указываются метод subscribeOn(), который указывает, что операция должна выполняться в фоновом потоке для избежания блокировки пользовательского интерфейса.

Помимо этого создается метод subscribeWith(), который подписывается на Observable и определяет, как обрабатывать результаты операции. Если операция успешно завершается, запускается новое активити (MainActivity или LoginActivity).

В результате осуществляется переход на главный экран приложения после успешного построения библиотеки.

* 1. **Авторизация и регистрация**

В процессе регистрации пользователь предоставляет свои данные, такие как логин, пароль и адрес электронной почты. Эти данные проверяются на валидность, включая наличие обязательных полей и соответствие формату электронной почты. После успешной проверки новая учетная запись создается в системе.

В классе RegisterActivity осуществляется процесс регистрации пользователя. При вводе данных пользователем и нажатии на кнопку регистрации данные проходят валидацию. Если данные верны и пользователь с таким логином еще не зарегистрирован, создается новая учетная запись, и пользователь перенаправляется на экран входа. В случае возникновения ошибки, например, если логин уже занят, отображается соответствующее сообщение.

Авторизация представляет собой проверку подлинности пользователя. Пользователь вводит свой логин и пароль, которые затем проверяются на соответствие данным в базе данных. При успешной аутентификации пользователь получает доступ к защищенным ресурсам приложения, например, главному экрану. В случае неверных данных пользователю предлагается повторно ввести данные или, если регистрация доступна, зарегистрироваться.

В классе LoginActivity реализована функциональность входа пользователя. После ввода данных и нажатия на кнопку входа происходит проверка аутентификационных данных. При успешной аутентификации пользователь перенаправляется на главный экран. Если введены неверные данные, пользователю предлагается возможность повторного ввода или регистрации новой учетной записи при наличии доступной опции.

* 1. **Описание структуры проекта**

Организация структуры проекта имеет стандартный вид представления. Проект разделен на два блока: app с реализацией логики проекта и Gradle Scripts с установкой зависимостей и управлением пакетами. Блок app разделен на подблоки manifest, java, assets и res. Рассмотрим более подробно блок java, который содержит детализированное описание логики для каждого класса внутри каждого подблока. Структура представлена на рисунке 3.2.

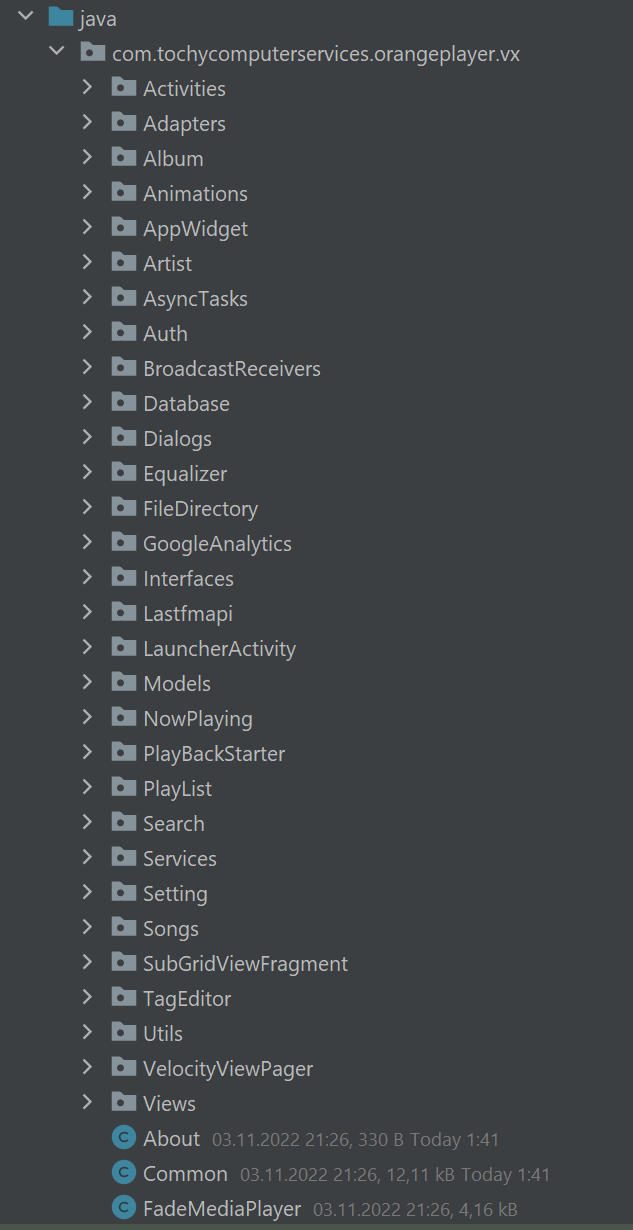


Рисунок 3.2 – Структура папки java

Компоненты проекта:

* activities: Классы активностей, отвечающие за UI и взаимодействие с пользователем.
* adapters: Адаптеры для отображения данных в списках и других представлениях.
* album: Классы, связанные с альбомами, их управлением и отображением.
* animations: Классы для создания анимаций и переходов в интерфейсе.
* appwidget: Классы для создания и управления виджетами приложения.
* artist: Классы, связанные с исполнителями, их управлением и отображением.
* asyncTasks: Асинхронные задачи для выполнения фоновых операций без блокировки UI.
* auth: Классы для аутентификации и авторизации пользователей.
* broadcastReceivers: Широковещательные приемники для получения и обработки системных трансляций.
* database: Классы для доступа и управления данными приложения.
* dialogs: Классы для создания и отображения диалоговых окон.
* equalizer: Классы для управления настройками эквалайзера.
* fileDirectory: Классы для управления файлами и каталогами.
* googleAnalytics: Классы для отслеживания активности пользователей и сбора аналитических данных.
* interfaces: Интерфейсы, определяющие стандарты взаимодействия между компонентами.
* lastfmapi: Классы для взаимодействия с API Last.fm.
* launcherActivity: Класс стартовой активности приложения.
* models: Модели данных, представляющие структуру и логику данных приложения.
* nowPlaying: Классы, связанные с воспроизведением музыки, отображением информации о текущем треке.
* permissions: Классы для запроса и управления разрешениями, необходимыми приложению.
* preferences: Классы для хранения и загрузки пользовательских настроек.
* receiver: Классы приемников для обработки различных типов событий или сообщений.
* services: Классы для фоновых процессов, работающих независимо от жизненного цикла активности.
* utils: Вспомогательные классы, предоставляющие общие функции, многократно используемые в разных частях приложения.
* fragments: Классы фрагментов, представляющих модульные компоненты UI, встраиваемые в активности.

Соответственные XML-файлы, определяющие пользовательский интерфейс, расположены в папке res подпапке layout. Также определены другие ресурсы, такие как шрифты, строки, цвета, анимация.

В папке assets находятся статичные файл.

Manifest располагает информацией о проекте.

* 1. **Вывод по разделу**

В данном разделе была рассмотрена реализация функциональных возможностей посредством доступа к средствам базы данных. Описан процесс регистрации и авторизации. Подробно очерчена структура проекта и прописаны соответствующие функциональные требования.

1. **Руководство пользователя**

В данном разделе описаны основные шаги по взаимодействию с приложением «Orange Music Player».

При открытии приложения требуется ввести свои учетные данные, как показано на рисунке 4.1.

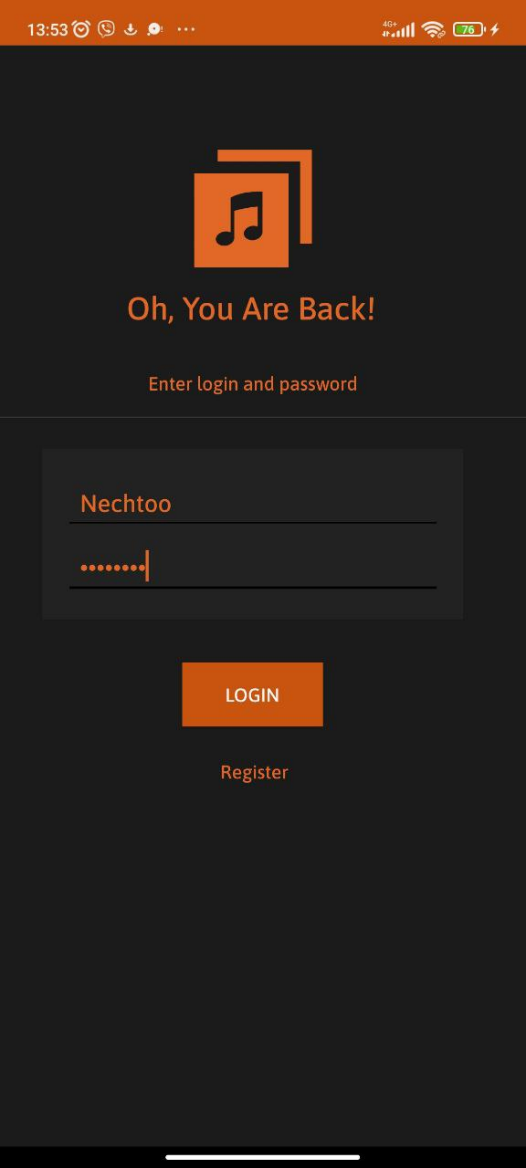


Рисунок 4.1 – Страница авторизации

В случае отсутствия этих данных в базе данных пользователя перенаправит на страницу регистрации, отображенной на рисунке 4.2, где данные, с некорректными значениями в полях Email и Password будут заблокированы для отправки, пока не будут внесены данные корректного формата.

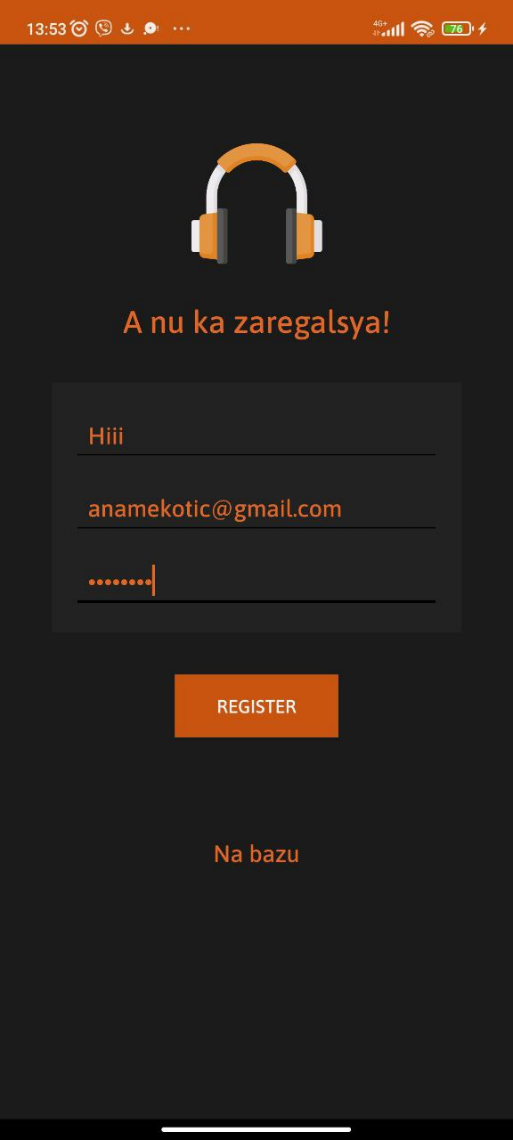


Рисунок 4.2 – Страница регистрации

После успешной авторизации пользователь оказывается на вкладке «Songs», от которой он может дальше перемещаться по приложению посредством навигации. При нажатии на песню она будет выбрана и воспроизведена. Это включает себя рисунок 4.3.

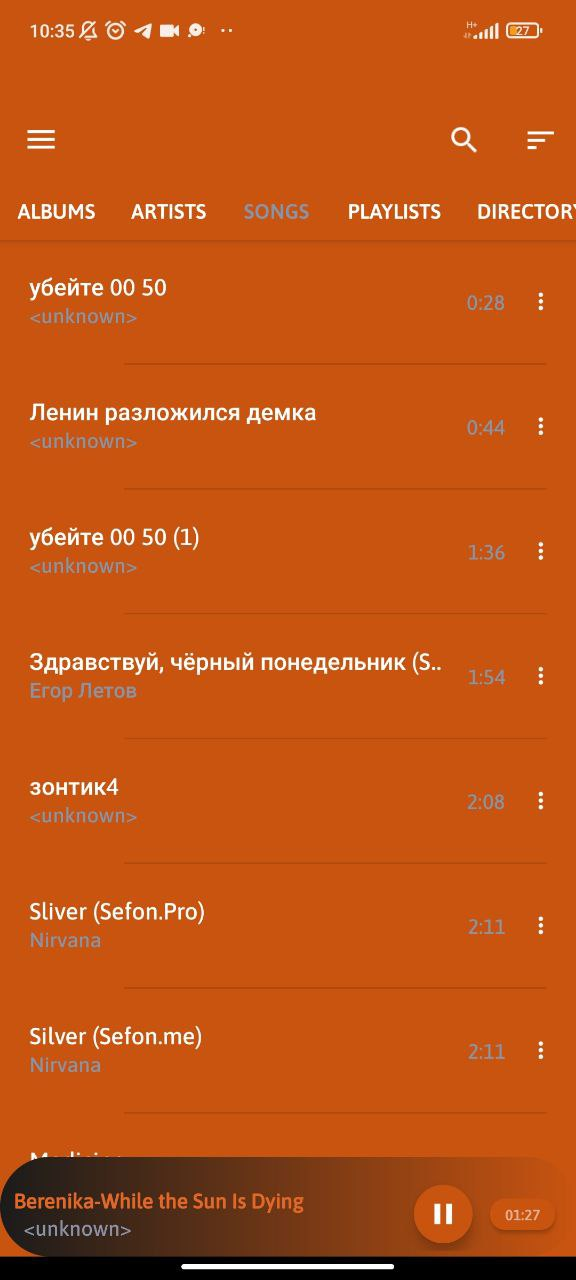


Рисунок 4.3 – Вкладка «Songs»

Контроль над воспроизводимой мелодией включает в себя: воспроизведение следующей/предыдущей композиции, пауза, сохранение позиции тайминга отыгрываемой мелодии, установка таймера, повторение фрагмента песни, повторение текущей композиции или перемешивание последующих песен в плейлисте. Данная страница отображена на рисунке 4.4.

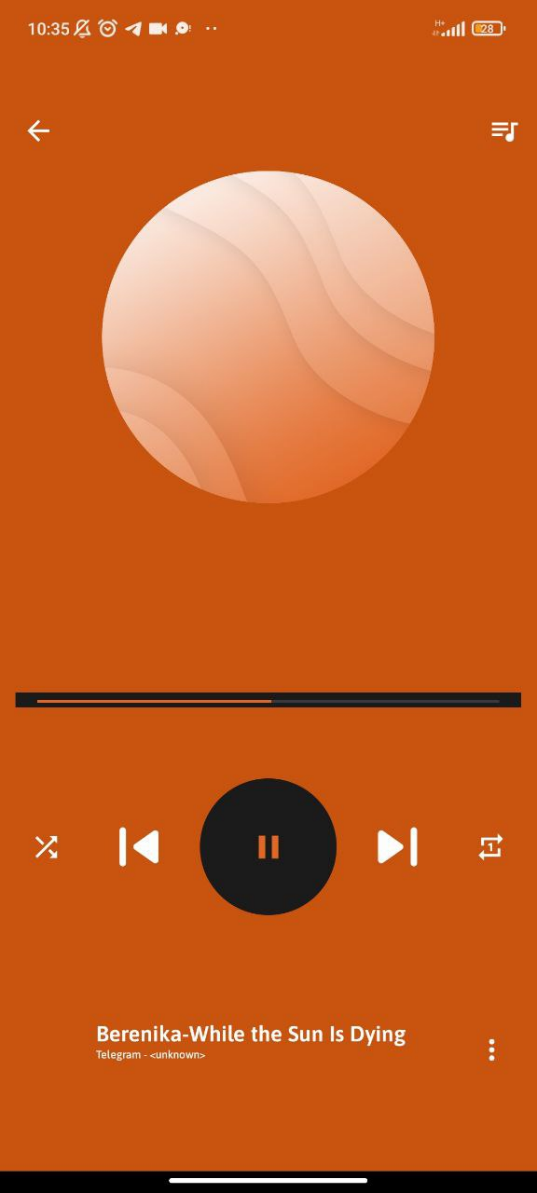


Рисунок 4.4 – Контроль над воспроизводимой композицией

Создание своего плейлиста также допустимо, включено добавление песен в имеющиеся плейлисты (избранное). Реализованы возможности удаления песни из плейлиста, а также составление плейлистов по истории прослушивания и добавления в библиотеку. Список плейлистов отображен на рисунке 4.5.

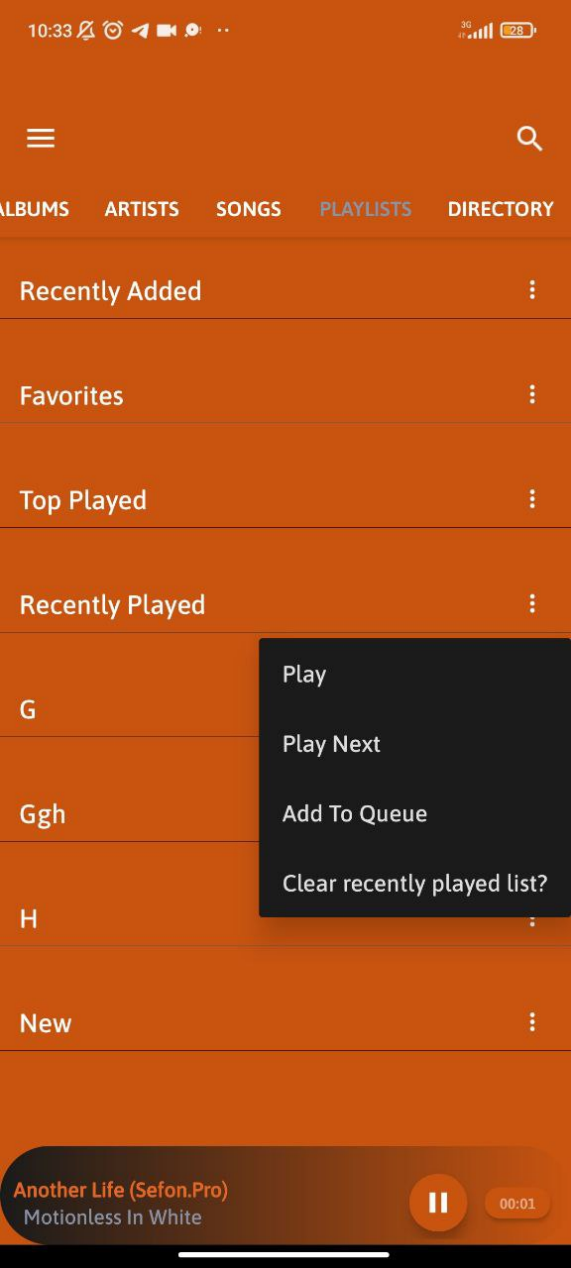


Рисунок 4.5 – Список плейлистов

Помимо этого, имеется возможность поиска исполнителя/песня. Проиллюстрировано на рисунке 4.6.

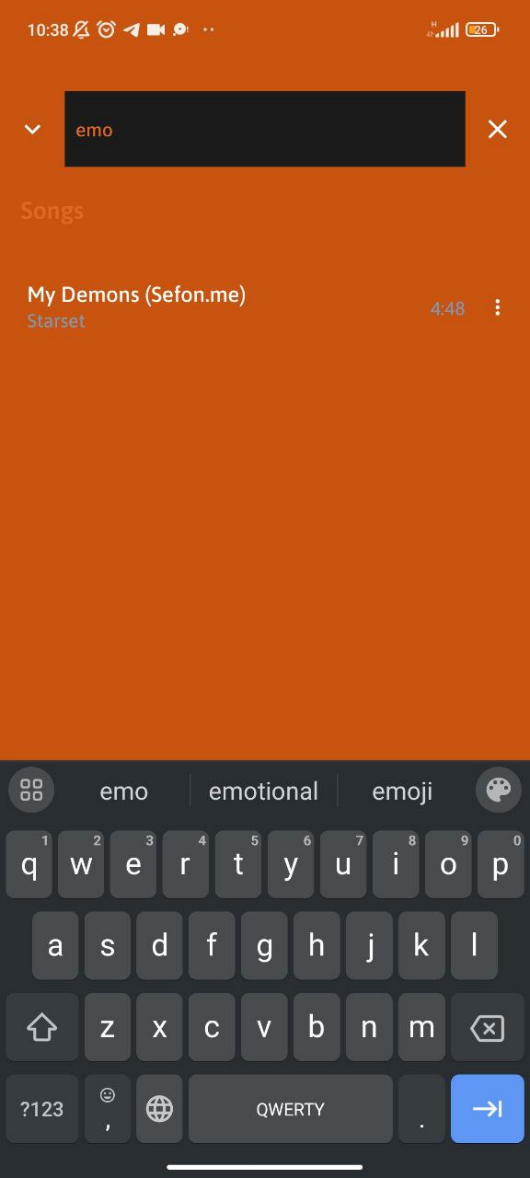


Рисунок 4.6 – Поиск песни

Таким образом пользователю были предоставлены все заявленные требованиями возможности.

* 1. **Вывод по разделу**

В данном разделе было представлено подробное руководство пользователя. Все компоненты работают исправно.

# Заключение

Результатом выполнения курсового проекта стало Android приложение с использованием технологии Java для операционной системы Windows. В ходе курсовой работы была разработана база данных SQLite на тему «Музыкальная платформа». Приложение является средством взаимодействия клиента с базой данных.

Были реализованы основные требования, указанные в листе задания вместе со следующими пунктами:

* регистрация и авторизация пользователей музыкальной площадки с учётом выданных им ролей;
* пароли при регистрации шифруются;
* мультимедийные типы данных, корректно записывающиеся в базу данных и затем воспроизводящиеся;
* поиск песен по исполнителю, альбому и названию самой песни;
* группировка песен по исполнителям, альбомам;
* возможность каждому пользователю сохранять понравившиеся песни в свой плейлист.

Проект не может окончательно претендовать на полноценный коммерческий продукт, поскольку для демонстрации технологии и работы с базой данных были опущены некоторые атрибуты современных стриминговых площадок, реализация которых не относится к основной теме курсового проекта (например, программирование искусственного интеллекта, собирающего информацию о предпочтениях пользователя и формирующее на их основе подборки).

Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

# **Список использованных источников**

1. Spotify [Электронный ресурс] / Режим доступа: <https://open.spotify.com> Дата доступа: 20.09.2023.

2. YouTube Music [Электронный ресурс] / Режим доступа: <https://music.youtube.com> Дата доступа: 20.09.2023.

3. METANIT.COM Создание и выполнение процедур [Электронный ресурс] / Режим доступа: [MS SQL Server и T-SQL | Хранимые процедуры (metanit.com)](https://metanit.com/sql/sqlserver/11.1.php?ysclid=lhr90o5hzb869594471) – Дата доступа: 27.10.2023.

5. Сертификаты SQL Server и асимметричные ключи [Электронный ресурс] / Режим доступа: [Сертификаты SQL Server и асимметричные ключи – SQL Server | Microsoft Learn](https://learn.microsoft.com/ru-Ru/sql/relational-databases/security/sql-server-certificates-and-asymmetric-keys?view=sql-server-ver15) – Дата доступа: 27.10.2023.

6. Примеры массового импорта и экспорта XML-документов (SQL Server) [Электронный ресурс] / Режим доступа: https://docs.oracle.com/cd/B1930601/ appdev.102/b14258/dxmldom.htm#i1076719 – Дата доступа: 21.11.2023.

# **Приложение A. Классы для работы с базой данных**

|  |
| --- |
| class DatabaseHelper{  late Database db;  Future init(String path) async{  db = await openDatabase(path, version: 1,  onCreate: (db, version) async{  await CityHandler(db).createTable();  await RoleHandler(db).createTable();  await ClientHandler(db).createTable();  await BusHandler(db).createTable();  await RouteHandler(db).createTable();  await BusStopHandler(db).createTable();  await TripHandler(db).createTable();  await TicketHandler(db).createTable();  await TicketHandler(db).createTriggers();  });  }  Future close() async => db.close();  }  class CityHandler{  late Database db;  CityHandler(this.db);  Future createTable() async{  await db.execute('''  create table IF NOT EXISTS $tableName ($columnCityID INTEGER PRIMARY KEY autoincrement,  $columnCityName TEXT not null,  $columnLocation TEXT not null)  ''');  }  Future<int> insert(City city) async{  return await db.insert(tableName, city.toMap());  }  Future<City?> getCity(int id) async{  List<Map> maps = await db.query(tableName,  columns: [columnCityID, columnCityName, columnLocation],  where: '$columnCityID = ?',  whereArgs: [id]);  if(maps.isNotEmpty){  return City.fromMap(maps.first);  }  return null;  }  Future<int> delete(int id) async{  return await db.delete(tableName, where: '$columnCityID = ?', whereArgs: [id]);  }  Future<int> update(City city) async{  return await db.update(tableName, city.toMap(), where: '$columnCityID = ?', whereArgs: [city.id]);  }  Future<List<City>> getAllCities(String selectedCity, String searchText) async {  List<Map<String, dynamic>> maps = await db.query(  tableName,  where: 'cityName LIKE ?',  whereArgs: ['%$searchText%']  );  List<City> cities = [];  for (var map in maps) {  City city = City.fromMap(map);  if (city.cityName != selectedCity) {  cities.add(city);  }  }  return cities;  }  Future close() async => db.close();  } |