

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационные системы и технологии
Специальность 1–40 01 01 Программное обеспечение информационных техноло-
гий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных платформы для прослушивания музыки»

Выполнил студент Гиль Виталия Сергеевна
(Ф.И.О.)

Руководитель работы асс. Савельева М.Г.
(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.
(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой _____

Минск 2024

Содержание

Введение.....	5
1 Постановка задачи и анализ требований	6
1.1 Цели и задачи проекта	6
1.2 Аналитический обзор аналогов	6
1.2.1 Аналог «Spotify».....	7
1.2.2 Аналог «Яндекс Музыка»	7
1.2.3 Аналог «SoundCloud»	8
1.3 Вывод по разделу	9
2 Проектирование и разработка базы данных.....	10
2.1 Роли и пользователи	10
2.2 Привилегии	11
2.3 Вывод по разделу	12
3 Разработка необходимых объектов	13
3.1 Описание информационных объектов и ограничений целостности.	13
3.2 Процедуры	15
3.3 Функции	19
3.5 Вывод по разделу	21
4 Описание процедур импорта и экспорта	22
4.1 Процедуры импорта и экспорта.....	22
4.2 Вывод по разделу	23
5. Тестирование производительности базы данных	24
5.1 Тестирование производительности по таблице USERS.....	24
5.2 Создание индексов	25
5.3 Вывод по разделу	26
6. Описание технологии и ее применение в базе данных	27
6.1 Технология хранения мультимедийной информации.....	27
6.2 Вывод по разделу	27
7 Руководство пользователя.....	28
7.1 Тестирование клиентской части	28
7.2 Тестирование области работы администратора.....	32
7.3 Вывод по разделу	35
Заключение	36
Список использованных литературных источников	37
Приложение А	38
Приложение Б	41
Приложение В.....	42
Приложение Г	44

Введение

В современном мире стриминговые сервисы сталкиваются с огромным объемом контента и информации, которую необходимо эффективно управлять и обрабатывать. Важно иметь надежную систему хранения данных и удобный доступ к ним. В рамках данного проекта была разработана и реализована база данных PostgreSQL и соответствующий интерфейс для стримингового сервиса.

Целью данного проекта является создание интегрированной базы данных, которая позволит нам эффективно управлять информацией о наших пользователях, песнях, плейлистах и других важных данных.

Задачи проекта:

- создание структуры базы данных, включая таблицы для хранения информации о песнях, исполнителей, сообществах, пользователях;
- Реализация процедур и функций для управления данными, включая добавление, обновление, удаление и извлечение информации;
- Создание пользовательских ролей и привилегий для различных уровней доступа к данным (для пользователя, администратора).

В рамках проекта была реализована технология хранения мультимедийных данных. Важным аспектом при работе с мультимедийными данными является оптимизация процессов доступа к этим данным, чтобы обеспечить быструю загрузку и передачу информации.

Система управления базами данных (СУБД) играет важную роль в обработке и хранении информации, обеспечивая эффективное взаимодействие между базой данных и пользователями или программами. Она предоставляет удобный интерфейс для доступа к данным, управления информацией и оптимизации процессов. С использованием СУБД можно выполнять разнообразные операции, включая мониторинг производительности, настройку и создание резервных копий данных.

Выбор СУБД, такой как PostgreSQL, позволяет нам эффективно управлять информацией о доступных турах, клиентах, отелях и других аспектах нашего бизнеса. Системы управления базами данных обеспечивают безопасность, надежность и удобство работы с данными, что является ключевыми факторами для успешной деятельности стримингового сервиса.

1 Постановка задачи и анализ требований

1.1 Цели и задачи проекта

Функционально должно быть выполнено:

- Определение ролей (администратор и клиент);
- Администратор может управлять песнями (добавление, редактирование, удаление);
- Клиент может управлять аккаунтом (создание, изменение, удаление).
- Клиент может управлять избранным (добавление, удаление);
- Клиент может управлять плейлистами (создание, изменение, удаление);
- Фильтрация музыки по количеству прослушиваний;
- Поиск музыки по названию, по артисту;
- Приложение для взаимодействия с БД.

Должны быть выполнены следующие требования:

- База данных должна быть спроектирована в СУБД PostgreSQL;
- Доступ к данным должен осуществляться только через соответствующие процедуры;
- Должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;
- Необходимо протестировать производительность базы данных (на таблицах, содержащих не менее 100 000 строк) и внести изменения в структуру в случае необходимости.

Применить технологию хранения мультимедийной информации в базе данных: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

1.2 Аналитический обзор аналогов

В процессе разработки программного продукта ключевым этапом является анализ прототипов и исследование литературных источников. В области стриминговых сервисов наблюдается растущая популярность и запросы от пользователей. Одной из основных задач таких платформ является предоставление музыкальных треков и разнообразного контента для удовлетворения музыкальных предпочтений пользователей.

Среди веб-приложений, успешно функционирующих в этой сфере, выделяются «Spotify» и «Яндекс Музыка». Эти платформы предлагают широкий выбор музыкального контента, персонализированные рекомендации и удобные функции для потребителей. В результате стремительного развития цифровых технологий и увеличения доступности интернета, стриминговые сервисы стали неотъемлемой частью повседневной жизни многих людей, обеспечивая им удобный доступ к музыке в любое время и в любом месте.

1.2.1 Аналог «Spotify»

«Spotify» – популярный стриминговый сервис, для прослушивания музыки [1].

Spotify включает в себя ряд баз данных, которые хранят информацию о различных аспектах. Некоторые из них:

- Таблица «Пользователи»: содержит информацию о пользователях сайта, включая их имя, адрес электронной почты, дату регистрации и роль (пользователь и исполнитель);
- Таблица «Треки»: хранит информацию о музыкальных треках, загруженных на сайт. Включает название трека, жанр, длительность и ссылку на файл;
- Таблица «Исполнители»: содержит информацию о исполнителях, которые загружают свои треки на сайте. Включает имя исполнителя, контактные данные, и количество загруженных треков. Интерфейс «Spotify» представлен на рисунке 1.1.

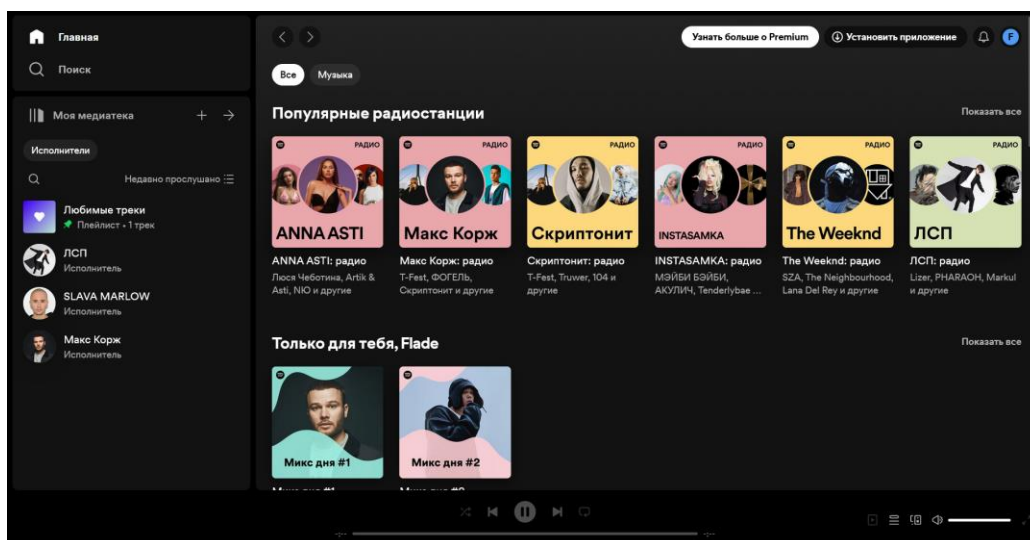


Рисунок 1.1 – Интерфейс «Spotify»

Проанализировав «Spotify», можно выделить основные минусы и плюсы данного программного средства.

Основные минусы:

- Ограничения на типы данных и размеры;
- Ограниченные аналитические возможности;
- Ограниченный функционал бесплатной версии.

Основные плюсы:

- Огромное хранилище песен;
- Эффективный функционал для прослушивания;
- Процедуры для поиска и фильтрации.

1.2.2 Аналог «Яндекс Музыка»

Яндекс Музыка – это сервис для поиска и прослушивания музыки с рекомендациями для каждого пользователя [2]. Для некоторых стран сервис доступен без подписки, после оплаты подписки Музыкой можно пользоваться по всему миру.

Наличие множества подкастов дополняет аудиоопыт пользователей. Однако, реклама без подписки и отсутствие возможности скачивания треков для прослушивания офлайн являются недостатками, которые ухудшают общее впечатление от использования сервиса.

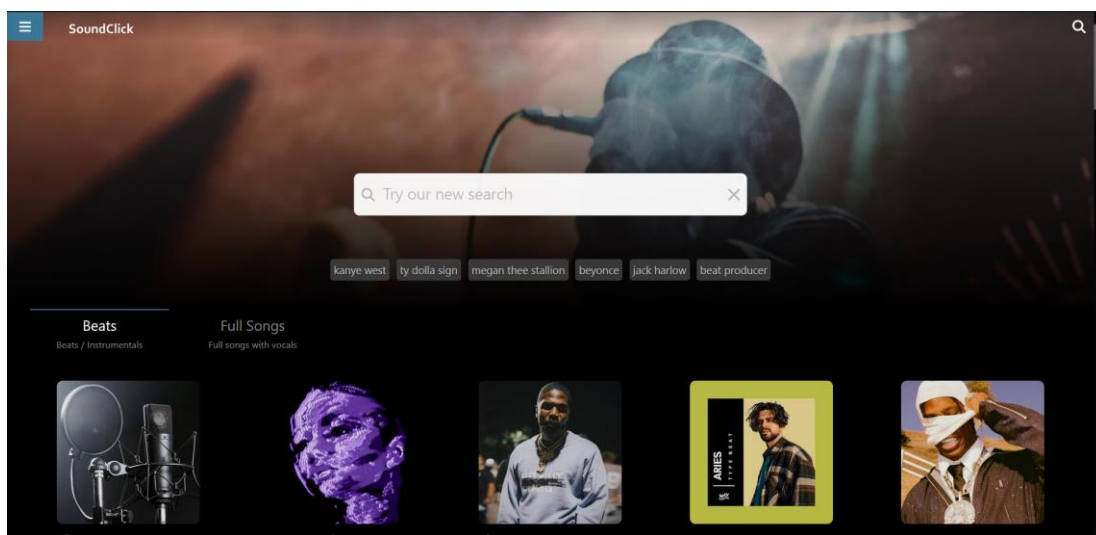


Рисунок 1.2 – Сайт «Яндекс Музыка»

Проанализировав «Яндекс Музыка», можно выделить её основные минусы и плюсы.

Основные минусы:

- Без подписки на «Яндекс Плюс» на экране много рекламы;
- Без подписки на «Яндекс Плюс» во время прослушивания будет проигрываться реклама;
- Нельзя скачать музыку на устройство.

Основные плюсы:

- Наличие мобильного приложения;
- Удобный интерфейс;
- Различные плейлисты, с подборками музыки по вкусу пользователя;
- Множество подкастов для прослушивания.

1.2.3 Аналог «SoundCloud»

SoundCloud – сервис для прослушивания новой музыки, общения с другими пользователями и размещения собственных треков [3].

SoundCloud акцентирует внимание на социальных функциях, позволяя пользователям комментировать треки, ставить лайки, делиться музыкой и взаимодействовать с другими участниками сообщества, что создает более тесные связи между пользователями и артистами.

SoundCloud является популярной платформой для начинающих музыкантов и независимых исполнителей, которые могут загружать свои треки и найти публику для своего творчества.

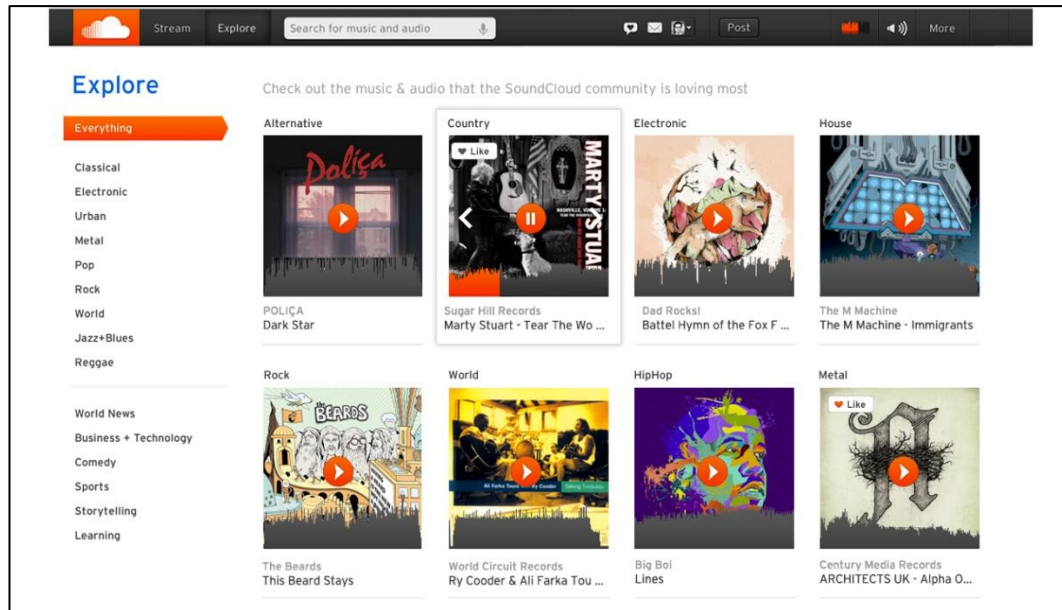


Рисунок 1.3 – Интерфейс «SoundCloud»

Проанализировав «SoundCloud», можно выделить основные минусы и плюсы.

Основные минусы:

- Реклама и ограничения бесплатной версии приложения;
- Качество звука ниже чем на других платформах;
- Некоторые аудиозаписи могут быть недоступны из-за ограничений лицензии.

Основные плюсы:

- SoundCloud предлагает огромное количество треков различных жанров;
- Пользователи могут взаимодействовать с другими пользователями, оставлять комментарии, ставить лайки и репосты, что создает сообщество вокруг музыкального контента;
- Интуитивно понятный интерфейс приложения делает его удобным в использовании как для новичков, так и для опытных пользователей;
- Артисты могут загружать свои собственные треки на SoundCloud, что способствует распространению их музыкального творчества.

1.3 Вывод по разделу

Были проведены исследования аналогичных решений, анализируя их достоинства и недостатки, с целью определения необходимой функциональности. На основе этого анализа были разработаны функциональные требования, которые являются ключевыми для достижения поставленной цели и создания высококачественного продукта в будущем.

2 Проектирование и разработка базы данных

2.1 Роли и пользователи

При выполнении команды CREATE USER в PostgreSQL создается пользователь базы данных, и автоматически создается роль с тем же названием. В PostgreSQL пользователь и роль совпадают по имени и используются взаимозаменяемо.

Были разработаны следующие пользователи:

- Администратор;
- Пользователь.

Диаграмма вариантов использования для роли Users представлена на рисунке 2.1.

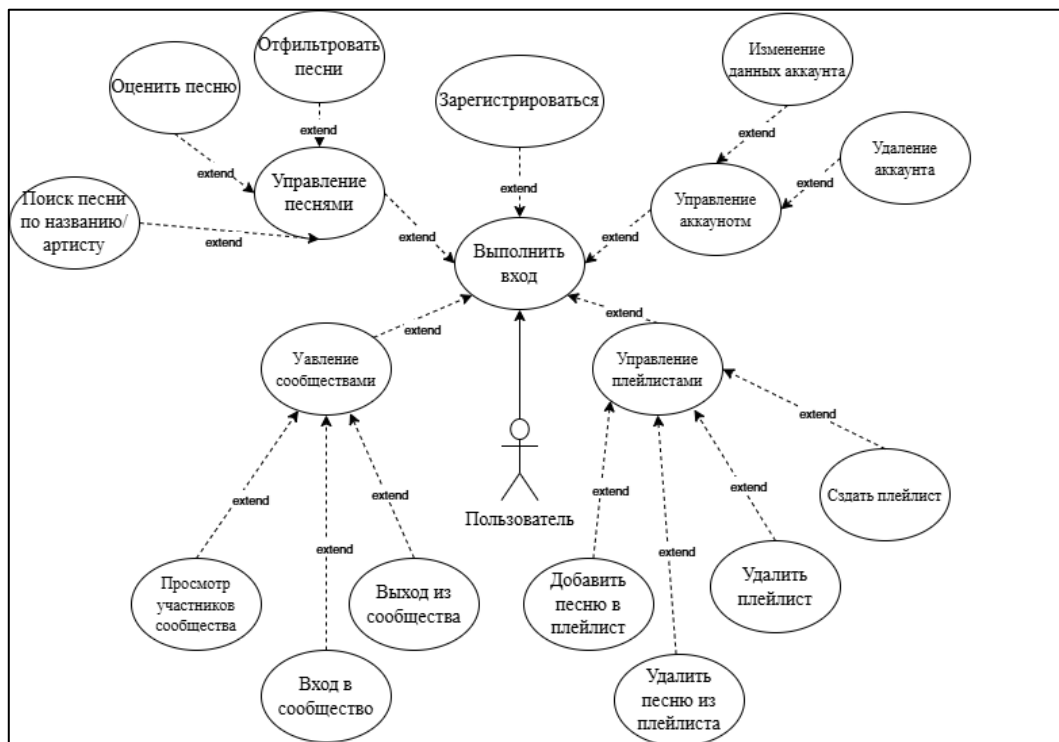


Рисунок 2.1 – Диаграмма вариантов использования для роли users

Возможности пользователя:

- Управление аккаунтом (создание, изменение, удаление);
- Управление песнями (добавление в плейлисты, удаление из плейлистов);
- Вход в сообщества;
- Просмотр участников сообществ;
- Управление плейлистами (создание, изменение, удаление);
- Фильтрация песен по прослушиваниям;
- Поиск песен по названию и по исполнителю.

Диаграмма вариантов использования для роли admin представлена на рисунке 2.2.

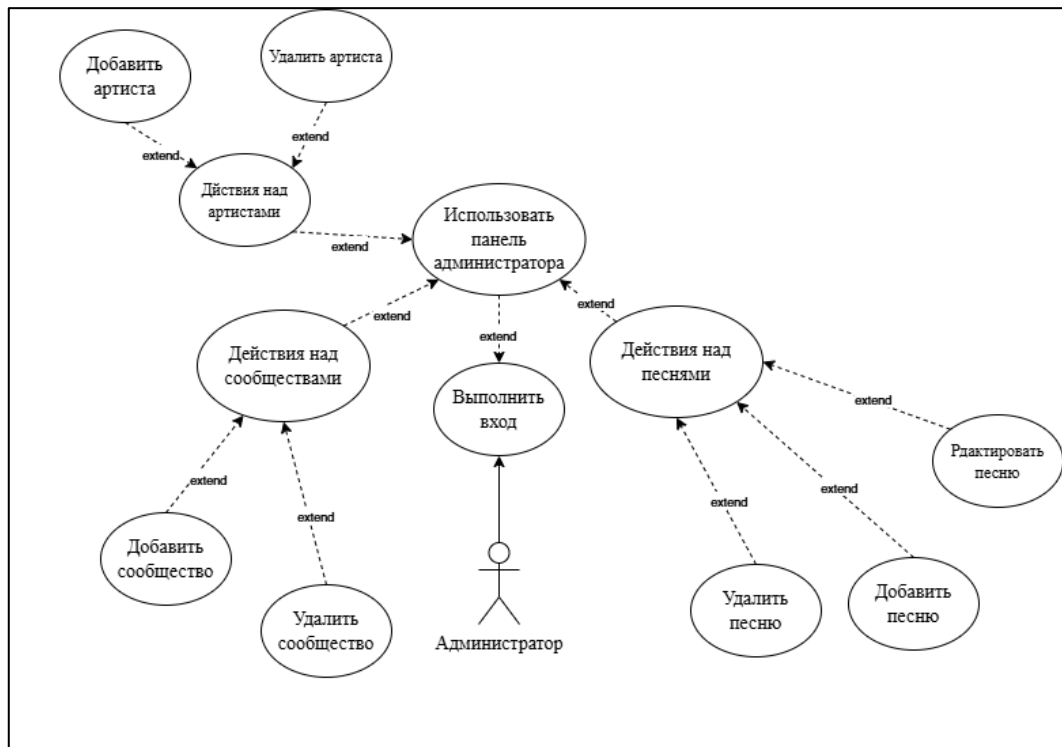


Рисунок 2.2 – Диаграмма вариантов использования для роли admin

Возможности администратора:

- Добавление и удаление исполнителей;
- Управление песнями (добавление, изменение, удаление);
- Создание и удаление сообществ по жанрам музыки;
- Управление плейлистами (создание, изменения, удаление);
- Фильтрация музыки по прослушиванием;
- Поиск музыки по названию и по исполнителю.

2.2 Привилегии

Привилегии, выдаваемые ролям, определяют, какие действия могут быть выполнены в отношении определенных объектов базы данных (например, таблиц, представлений и т.д.) и контролируют доступ пользователей к этим объектам. Таким образом, выдача привилегий ролям позволяет определить права доступа на уровне ролей, что упрощает управление безопасностью и снижает риски нарушения безопасности базы данных.

Предоставление привилегий роли администратор представлено на листинге 2.2.

```

Grant execute on procedure insert_artist to admin;
Grant execute on procedure delete_artist_by_name to admin;
Grant execute on procedure add_song_and_link_artists to admin;
  
```

```
Grant execute on procedure delete_song_and_unlink_artists to admin;  
Grant execute on procedure create_genre_community to admin;  
Grant execute on procedure delete_community to admin;  
Grant execute on procedure UpdateSongAndArtistsById to admin;
```

Листинг 2.2– Предоставление привилегий

Администратору были предоставлены различные привилегии для эффективного управления базой данных. Среди них — возможность выполнения процедур, таких как добавление и удаление информации об артистах, создание жанровых сообществ и обновление данных о песнях и исполнителях. Эти привилегии предоставляют администратору полный контроль над базой данных, позволяя ему эффективно координировать операции по управлению информацией.

Предоставление администратору указанных привилегий не только обеспечивает ему возможность управления данными, но и помогает обеспечить безопасность и целостность информации. Этот набор привилегий дает администратору необходимые инструменты для обеспечения правильной работы базы данных, обеспечивая доступ к ключевым функциям, которые помогают поддерживать порядок и эффективность в обработке информации о исполнителях, песнях и жанровых сообществах.

2.3 Вывод по разделу

В разделе обсуждается процесс создания пользователей и ролей в PostgreSQL, а также выдача привилегий ролям для управления доступом к объектам базы данных. В PostgreSQL пользователи и роли могут быть созданы через команду CREATE USER, где пользователь и роль имеют одинаковое имя и могут использоваться взаимозаменяемо. Привилегии, назначаемые ролям, определяют возможные действия относительно объектов базы данных, обеспечивая удобное управление безопасностью и контролем доступа к данным в системе.

3 Разработка необходимых объектов

3.1 Описание информационных объектов и ограничений целостности

Схема базы данных ограничения целостности, связи и поля представлены на рисунке 3.1.

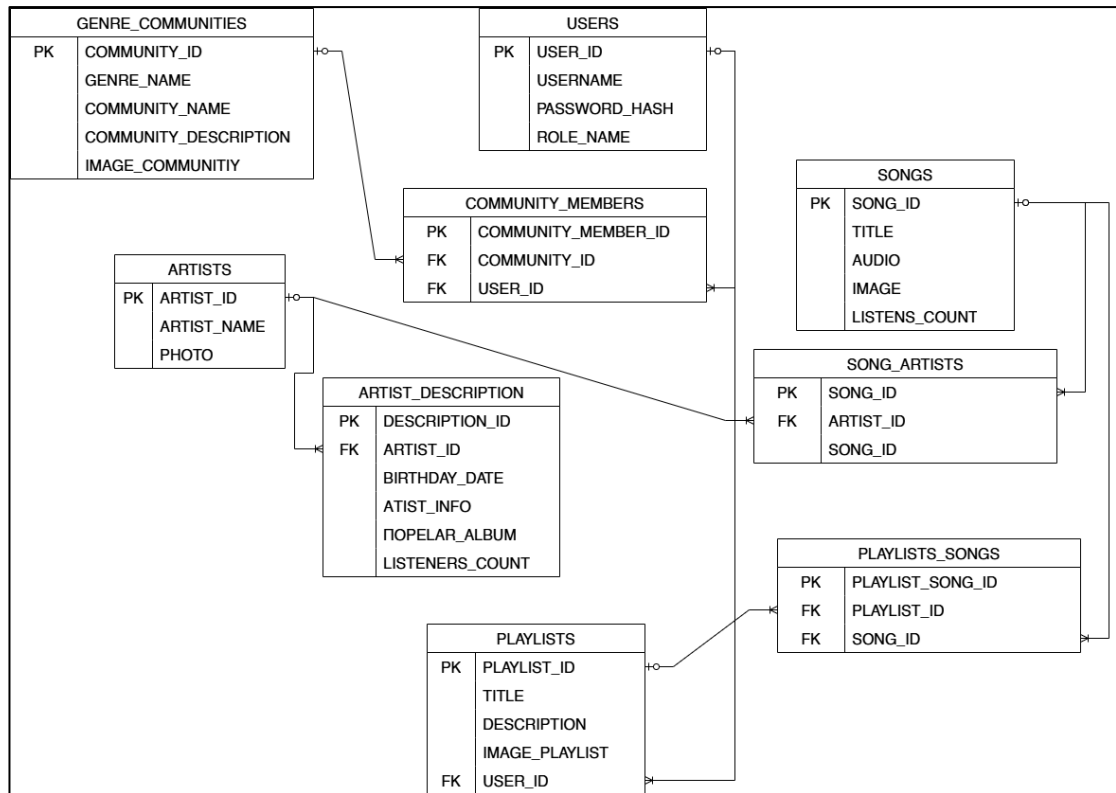


Рисунок 3.1 – Схема БД проекта

Таблица Artists:

- **ARTIST_ID**: Хранит уникальный идентификатор исполнителя. Это авто-инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

- **ARTIST_NAME**: Хранит название исполнителя VARCHAR(255);

- **PHOTO**: Хранит ссылку на фото VARCHAR(255).

Таблица Artists_Description:

- **DESCRIPTION_ID**: Хранит уникальный идентификатор описания исполнителя. Это авто-инкрементируемое поле типа SERIAL.

- **ARTIST_ID**: Хранит уникальный идентификатор для исполнителя Типа VARCHAR(255);

- **BIRTH_DATE**: Хранит дату рождения исполнителя тип DATE;

- **ARTIST_INFO**: Хранит описание исполнителя VARCHAR(255);

- **POPULAR_ALBUM**: Хранит название популярного альбома VARCHAR(255);

– LISTENERS_COUNT: Хранит количество слушателей исполнителя
Тип INTEGER.

Таблица Genre_Communities:

– COMMUNITY_ID: Хранит уникальный идентификатор сообщества. Это авто-инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

– GENRE_NAME: Хранит название жанра типа VARCHAR(255);

– COMMUNITY_NAME: Хранит название сообщества типа VARCHAR(255);

– COMMUNITY_DESCRIPTION : Хранит описание сообщества типа VARCHAR(255);

– IMAGE_COMMUNITY: Хранит фото сообщества типа VARCHAR(255).

Таблица Users:

–USER_ID: Хранит уникальный идентификатор пользователя. Это авто-инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

–USERNAME: Хранит имя пользователя тип VARCHAR(255);

–PASSWORD_HASH:Хранит пароль пользователя тип VARCHAR(255);

–ROLE_NAME: Хранит роль пользователя тип VARCHAR(255).

Таблица Users:

– SONG_ARTISTS_ID: Хранит уникальный идентификатор связи между песней и исполнителем. Это авто-инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

– SONG_ID: Хранит уникальный идентификатор песни типа SERIAL. Это внешний ключ, который ссылается на поле SONG_ID в таблице SONGS. Для столбца SONG_ID существует ограничение по внешнему ключу;

– ARTIST_ID: Хранит уникальный идентификатор исполнителя типа INT. Это внешний ключ, который ссылается на поле ARTIST_ID в таблице ARTISTS.

Таблица Songs:

– SONG_ID: Хранит уникальный идентификатор песни в таблице SONGS. Это авто-инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

– TITLE: Хранит название песни типа VARCHAR(255);

– AUDIO: Хранит ссылку на песню типа VARCHAR(255);

– IMAGE: Хранит ссылку на фото песни типа VARCHAR(255);

– LISTENS_COUNT: Хранит количество прослушиваний на песне типа VARCHAR(255).

Таблица Playlist_Songs:

– PLAYLIST_SONG_ID: Хранит уникальный идентификатор который указывает в каком плейлисте есть песня. Это авто инкрементируемое поле типа SERIAL. В данной таблице присутствует ограничение первичного ключа для столбца ID;

- **PLAYLIST_ID**: Хранит уникальный идентификатор плейлиста, в котором находится песня типа INT. Это внешний ключ, который ссылается на поле **PLAYLIST_ID** в таблице **PLAYLISTS**. Для столбца **PLAYLIST_ID** существует ограничение по внешнему ключу, связывающее его с таблицей **PLAYLISTS**;

- **SONG_ID**: Хранит уникальный идентификатор песни, которая находится в определённом плейлисте типа INT. Это внешний ключ, который ссылается на поле **SONG_ID** в таблице **SONGS**. Для столбца **SONG_ID** существует ограничение по внешнему ключу, связывающее его с таблицей **SONGS**.

Таблица **Community_members**:

- **COMMUNITY_MEMBERS_ID**: Хранит уникальный идентификатор пользователя в сообществе. Это авто-инкрементируемое поле типа **SERIAL**. В данной таблице присутствует ограничение первичного ключа для столбца **ID**;

- **COMMUNITY_ID**: Хранит уникальный идентификатор сообщества, в котором находится пользователь рейтинг типа INT;

- **USER_ID**: Хранит уникальный идентификатор пользователя, который вступил в сообщество типа INT. Это внешний ключ, который ссылается на поле **USER_ID** в таблице **USERS**. Для столбца **USER_ID** существует ограничение по внешнему ключу, связывающее его с таблицей **USERS**.

Таблица **Playlists**:

- **PLAYLIST_ID**: Хранит уникальный идентификатор плейлиста. Это авто-инкрементируемое поле типа **SERIAL**. В данной таблице присутствует ограничение первичного ключа для столбца **ID**;

- **TITLE**: Хранит название плейлиста тип **VARCHAR(255)**;

- **DESCRIPTION**: Хранит описание плейлиста тип **VARCHAR(255)**;

- **IMAGE_PLAYLIST**: Хранит ссылку на фото плейлиста **VARCHAR(255)**;

- **USER_ID**: Хранит уникальный идентификатор пользователя, который создал плейлист типа INT. Это внешний ключ, который ссылается на поле **USER_ID** в таблице **USERS**. Для столбца **USER_ID** существует ограничение по внешнему ключу, связывающее его с таблицей **USERS**.

3.2 Процедуры

Процедуры в базе данных могут быть использованы для выполнения различных задач, таких как обновление, удаление или выборка данных, а также для обработки данных внутри базы данных. Они могут быть вызваны из других процедур, функций, триггеров или приложений, что позволяет уменьшить нагрузку на сеть при обращении к базе данных.

Кроме того, процедуры могут содержать в себе параметры, которые могут быть переданы в качестве аргументов при вызове процедуры. Это позволяет создавать более гибкие и универсальные процедуры, которые могут быть использованы для выполнения различных задач с разными наборами данных.

Процедуры, разработанные в рамках курсового проекта:

- **add_user**. Регистрация;

- Login. Авторизация;
- edit_user_info. Редактирование аккаунта;
- delete_user. Удаление аккаунта;
- insert_artist. Добавление исполнителя;
- delete_artist_by_name. Удаление исполнителя;
- add_song_and_link_artists. Добавление песни и связи ее с исполнителем;
- delete_song_and_unlink_artists. Удаление песни у исполнителей;
- update_listen_count. Обновление количества прослушиваний;
- add_users_from_json_file. Импорт данных в json;
- export_users_to_json. Экспорт данных из json;
- add_song_to_favorites. Добавление в избранное;
- add_song_to_playlist. Добавление песен в плейлист;
- Removesongfromplaylist. Удаление песни из плейлиста;
- create_genre_community. Добавление сообщества;
- create_playlist. Добавление плейлиста;
- delete_user_playlist. Удаление плейлиста пользователя;
- Join_community. Вход в сообщество;
- Remove_user_from_community. Выход с сообщества;
- updatesongandartistbyid. Редактирование песен;
- delete_community. Удаление сообществ;

Создание процедуры add_song_and_link_artists представлено на листинге 3.1.

```
CREATE OR REPLACE PROCEDURE add_song_and_link_artists(
    title_param VARCHAR,
    audio_param VARCHAR,
    image_param VARCHAR,
    artist_names_param VARCHAR[]
)
AS $$
DECLARE
    artist_id_param INTEGER[];
    song_exists BOOLEAN;
    song_id_param INTEGER;
BEGIN
    BEGIN
        -- Проверка существования исполнителей
        SELECT array_agg(artist_id) INTO artist_id_param
        FROM artists
        WHERE artist_name = ANY (artist_names_param);

        IF artist_id_param IS NULL THEN
            RAISE NOTICE 'Один или несколько исполнителей не существуют';
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
```

```

        RAISE NOTICE 'Один или несколько исполнителей не
существуют';
    END;

    -- Проверка наличия песни у указанных исполнителей
    SELECT EXISTS (
        SELECT 1
        FROM songs s
        INNER JOIN song_artists sa ON s.song_id = sa.song_id
        WHERE s.title = title_param AND sa.artist_id = ANY (art-
ist_id_param)
    ) INTO song_exists;

    IF song_exists THEN
        RAISE NOTICE 'Песня уже существует у указанных исполните-
лей';
    ELSE
        BEGIN
            -- Вставка записи в таблицу songs
            INSERT INTO songs (title, audio, image)
            VALUES (title_param, audio_param, image_param)
            RETURNING song_id INTO song_id_param;

            -- Вставка записей в таблицу song_artists
            INSERT INTO song_artists (song_id, artist_id)
            SELECT song_id_param, unnest(artist_id_param) AS art-
ist_id;
        EXCEPTION
            WHEN unique_violation THEN
                RAISE NOTICE 'Такая песня уже существует';
        END;
    END IF;
    RAISE NOTICE 'Песня успешно добавлена!';
END;
$$ LANGUAGE plpgsql;

```

Листинг 3.1 – Создание процедуры add_song_and_link_artists

Данная процедура add_song_and_link_artists была разработана для добавления новой песни в базу данных и связывания ее с указанными исполнителями.

Процедура начинается с проверки существования всех указанных исполнителей в таблице artists. В случае, если один или несколько исполнителей не найдены, генерируется уведомление об этом. После этого происходит проверка наличия песни у указанных исполнителей: если песня уже существует, также генерируется соответствующее уведомление.

Создание процедуры add_user представлено в листинге 3.2.

```

CREATE OR REPLACE PROCEDURE add_user(
    username_param VARCHAR(255),
    password_hash_param VARCHAR(255),
    role_name_param VARCHAR(255),
    playlist_photo_url_param VARCHAR(255),
    OUT user_id_param INT,
    OUT user_exists BOOLEAN
)
SECURITY DEFINER
AS $$
DECLARE
    user_id_output INTEGER;
    playlist_id_output INTEGER;
BEGIN
    -- Проверка наличия пользователя с таким именем
    BEGIN
        SELECT user_id INTO STRICT user_id_output
        FROM users
        WHERE username = username_param;

        -- Если пользователь найден, устанавливаем флаг
user_exists в true
        user_exists := TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            -- Если пользователь не найден, продолжаем выполне-
ние
            user_exists := FALSE;
    END;
    -- Если пользователь существует, выходим из процедуры
    IF user_exists THEN
        RETURN;
    END IF;
    -- Создаем нового пользователя
    INSERT INTO users (username, password_hash, role_name)
    VALUES (username_param, password_hash_param,
role_name_param)
    RETURNING user_id INTO STRICT user_id_output
    -- Создаем плейлист "Понравившиеся" для нового пользователя
    INSERT INTO playlists (user_id, title, descriprion, im-
age_playlist)
    VALUES (user_id_output, 'Понравившиеся', '',
playlist_photo_url_param)
    RETURNING playlist_id INTO STRICT playlist_id_output;
    -- Возвращаем ID пользователя и флаг user_exists
    user_id_param := user_id_output;
    user_exists := FALSE;
    RAISE NOTICE 'Вы успешно зарегистрировались под именем:
% ', username_param;
END;
$$ LANGUAGE plpgsql;

```

Листинг 3.2 – Создание процедуры add_user

Процедура `add_user` предназначена для добавления нового пользователя в базу данных, а также создания для него плейлиста "Понравившиеся".

Сначала процедура проверяет наличие пользователя с указанным именем в таблице `users`. Если пользователь уже существует, устанавливается флаг `user_exists` в `TRUE`, и выполнение процедуры завершается. В случае отсутствия пользователя с таким именем, процедура создает новую запись о пользователе, используя переданные параметры.

3.3 Функции

Функции являются объектами базы данных, которые позволяют выполнять определенные действия на стороне сервера базы данных. Функции могут использоваться для выполнения различных задач, таких как обработка данных, преобразование данных, агрегация данных и т. д.

Функции, разработанные в рамках курсового проекта:

- `delete_playlist`. Удаление плейлиста;
- `get_all_communities`. Получение всех сообществ;
- `get_all_community_names`. Вывод всех сообществ;
- `get_artist_description`. Получить описание исполнителя;
- `get_artist_info`. Получить информацию о исполнителе;
- `get_community_users`. Получение участников сообщества;
- `get_playlist_info_by_name`. Получение плейлиста по имени;
- `get_playlist_songs`. Получение всех песен в плейлисте;
- `get_songs_by_listens_count_range`. Получить отфильтрованные песни.
- `get_user_playlists`. Получить плейлисты пользователя;
- `get_user_playlist_without_like`. Получение всех плейлистов, кроме «избранное»;
- `getcommunitybyuser`. Получение сообществ;
- `getsongswithartists`. Вывод всех песен;
- `getusercredentialstitlebyid`. Получение имени и пароля пользователя;
- `searchsongsbyartist`. Поиск по исполнителю;
- `searchsongsbytitle`. Поиск по названию;
- `select_artist`. Вывод исполнителей;

Пример функции `getsongswithartists`, которая выводит все песни, представлена в листинге 3.3.

```
CREATE OR REPLACE FUNCTION GetSongsWithArtists(lim INT, off INT)
RETURNS TABLE (song_id INT, title VARCHAR(255), artists TEXT,
photo_path VARCHAR(255), audio_path VARCHAR(255), listens_count
INT)
LANGUAGE plpgsql
AS $$
BEGIN RETURN QUERY
    SELECT s.song_id, s.title, string_agg(a.artist_name, ' & ')
AS artists, s.image::VARCHAR(255), s.audio, s.listens_count
```

```

FROM songs s
JOIN song_artists sa ON s.song_id = sa.song_id
JOIN artists a ON sa.artist_id = a.artist_id
GROUP BY s.song_id, s.title, s.image, s.audio, s.lis-
tens_count
LIMIT lim OFFSET off;
END;
$$;
CREATE OR REPLACE FUNCTION GetSongsWithArtists(lim INT, off INT)
RETURNS TABLE (song_id INT, title VARCHAR(255), artists TEXT,
photo_path VARCHAR(255), audio_path VARCHAR(255), listens_count
INT)
LANGUAGE plpgsql
AS $$
BEGIN RETURN QUERY
    SELECT subquery.song_id, subquery.title, subquery.artists,
subquery.photo_path, subquery.audio_path, subquery.listens_count
    FROM (
        SELECT s.song_id, s.title, string_agg(a.artist_name, ' &
') AS artists, s.image::VARCHAR(255) AS photo_path, s.audio AS
audio_path, s.listens_count,
            ROW_NUMBER() OVER (ORDER BY s.song_id) AS row_num
        FROM songs s
        JOIN song_artists sa ON s.song_id = sa.song_id
        JOIN artists a ON sa.artist_id = a.artist_id
        GROUP BY s.song_id, s.title, s.image, s.audio, s.lis-
tens_count
    ) AS subquery
    WHERE subquery.row_num > (off - 1) * lim
        AND subquery.row_num <= off * lim;
END;
$$;

```

Листинг 3.3 – Создание функции getsongswithartists

Эта функция позволяет получить список песен с информацией о каждой песне и ее исполнителях, что может быть полезно для отображения списка песен на веб-странице, в мобильном приложении или другом пользовательском интерфейсе. Параметры `lim` и `off` используются для управления количеством возвращаемых записей и сдвига в результирующем наборе данных, что позволяет реализовать постраничную загрузку данных или другие методы пагинации.

Пример функции `delete_playlist.`, которая выводит все песни, представлена в листинге 3.4.

```

CREATE OR REPLACE FUNCTION delete_playlist(
    p_user_id INT,
    p_title VARCHAR(255)
) RETURNS VOID AS
$$

```

```

BEGINIF EXISTS (
    SELECT 1
    FROM playlists
    WHERE user_id = p_user_id
    AND title = p_title) THEN
    DELETE FROM playlists
    WHERE user_id = p_user_id
    AND title = p_title;
    RAISE INFO 'Плейлист успешно удален.';ELSE
    RAISE NOTICE 'У пользователя нет плейлиста с указанным
именем.';END IF;
EXCEPTION WHEN OTHERS THEN
    RAISE NOTICE 'Ошибка при удалении плейлиста:';END;
$$ LANGUAGE plpgsql;

```

Листинг 3.4 – Создание функции delete_playlist

Функция delete_playlist разработана для обеспечения возможности удаления плейлистов пользователей из базы данных по заданному идентификатору пользователя и названию плейлиста.

3.5 Вывод по разделу

В рамках разработки модели базы данных для приложения были созданы следующие объекты:

- 10 таблиц, содержащих данные о песнях, плейлистах, авторах, сообществах и др;
- 21 процедура выполнения различных операций с данными, таких как добавление, удаление, обновление песен, добавление плейлистов и др;
- 17 функций для получения различной информации из базы данных, например, получение списка всех песен, плейлистов, сообществ и др.

4 Описание процедур импорта и экспорта

4.1 Процедуры импорта и экспорта

JSON – это формат, который хранит структурированную информацию и в основном используется для передачи данных между сервером и клиентом.

Файл JSON представляет собой более простую и лёгкую альтернативу расширению с аналогичными функциями XML (Extensive Markup Language).

Часто возникает необходимость импортировать и экспортировать JSON-файлы, в данной курсовой работе используются функция `row_to_json()` для экспорта. Процедура экспорта используется для таблицы `USERS`, так как эта таблица является важнейшей в базе данных. Пример создания процедуры можно посмотреть в листинге 4.1.

```
CREATE OR REPLACE PROCEDURE export_users_to_json(p_file_path
text)
LANGUAGE plpgsql
AS $$
BEGIN
    EXECUTE format('COPY (
        SELECT json_agg(row_to_json(t))
        FROM (
            SELECT u.*
            FROM users u
        ) t
    ) TO %L', p_file_path);
END;
$$;

CALL export_users_to_json('C:\3course\COURSE_PROJ\online_mu-
sic\infoUsers.json');
```

Листинг 4.1 – Пример создания процедуры экспорта в json файл

Импорт так же происходит в таблицу `USERS`. Пример создания процедуры можно посмотреть в листинге 4.2.

```
CREATE OR REPLACE PROCEDURE add_users_from_json_file(
    json_file_path TEXT
)
AS $$
DECLARE
    json_data JSON;
    user_data JSON;
    username_param VARCHAR;
    password_hash_param VARCHAR;
    role_name_param VARCHAR;
    playlist_photo_url_param VARCHAR;
```

```

    user_id_output INT;
    user_exists BOOLEAN;
BEGIN
    -- Чтение JSON-файла
    json_data := jsonb(PG_READ_FILE(json_file_path));
    -- Проверка, удалось ли прочитать JSON-файл
    IF json_data IS NULL THEN
        RAISE EXCEPTION 'Не удалось прочитать JSON-файл';
    END IF;
    -- Перебор пользователей в JSON
    FOR user_data IN SELECT * FROM json_array_ele-
ments(json_data)
    LOOP
        -- Извлечение значений из JSON
        username_param := user_data->>'username';
        password_hash_param := user_data->>'password';
        role_name_param := user_data->>'role';
        playlist_photo_url_param := user_data-
>>'playlist_photo_url';
        -- Вызов процедуры add_user для каждого пользователя из
JSON
        CALL add_user(username_param, password_hash_param,
role_name_param, playlist_photo_url_param, user_id_output,
user_exists);
    END LOOP;
END;
$$ LANGUAGE plpgsql;
SELECT * FROM USERS;
CALL add_users_from_json_file(C:\3course\COURSE_PROJ\online_mu-
sic\addUsers.json');

```

Листинг 4.2 – Пример создания процедуры импорта из json файла

Процедура `add_users_from_json_file` представляет собой инструмент для эффективного импорта данных о пользователях из JSON-файла в таблицу базы данных. Этот подход позволяет автоматизировать процесс добавления большого количества пользовательской информации из внешнего источника.

4.2 Вывод по разделу

Разработанные процедуры импорта и экспорта данных в формате JSON предоставляют эффективные инструменты для обмена информацией между базой данных и внешними источниками. Использование подобных механизмов упрощает процессы обновления и синхронизации данных, обеспечивая более гибкую работу с информацией в приложении.

5. Тестирование производительности базы данных

5.1 Тестирование производительности по таблице USERS

Производительность БД является решающим фактором эффективности управленческих и коммерческих приложений. Если поиск или запись данных выполняется медленно – способность к нормальной работе приложения падает. Существует единственный путь выяснить причину плохой производительности – выполнить количественные измерения и определить, что является причиной проблемы производительности.

В PostgreSQL оптимизация запросом в основном заключается в построение индексов над таблицами.

Для тестирования производительности в таблицу USERS были добавлены 100 000 записей.

На рисунке 5.1 представлен результат Select-запроса к таблице USERS до создания индекса с выборкой user_name, содержащих «7777». Согласно заданному условию было найдено 10 строк. Время выполнения составляет 13,906 мс.

The screenshot shows the PostgreSQL query planner interface. The SQL query entered is: `explain analyze select * from users where username like '7777';`. The query plan shows a sequential scan on the 'users' table. The execution time is 13.906 ms.

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Loops
1	Seq Scan on users	(cost=0.00..2083.04 rows=10 width=34)	10	34	1.424..13.884	10	10
2	Filter: ((username)::text ~~ '7777':text)						
3	Rows Removed by Filter		99993				
4	Planning Time				0.311 ms		
5	Execution Time				13.906 ms		

Рисунок 5.1 – Результат Select-запроса к таблице Users без индекса

На рисунке 5.2 представлен результат Select-запроса к таблице USERS после создания индекса. Время выполнения уже составляет 9,596 мс.

The screenshot shows the PostgreSQL query planner interface. The SQL query entered is: `explain analyze select * from users where username like '7777';`. The query plan shows a sequential scan on the 'users' table. The execution time is 9.596 ms.

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Loops
1	Seq Scan on users	(cost=0.00..2083.04 rows=10 width=34)	10	34	1.298..9.577	10	10
2	Filter: ((username)::text ~~ '7777':text)						
3	Rows Removed by Filter		99993				
4	Planning Time				0.107 ms		
5	Execution Time				9.596 ms		

Рисунок 5.2 – Результат Select-запроса к таблице Users с индексом

На рисунке 5.3 представлен результат Select-запроса к таблице USERS до создания индекса с выборкой user_name, содержащих «7». Согласно заданному условию было найдено 7071 строка. Время выполнения составляет 15,479 мс.

The screenshot shows a SQL query execution interface. At the top, the query is: `49 explain analyze select * from users where username like '%7';`. Below the query, there are tabs for 'Data Output', 'Messages', 'Explain', and 'Notifications'. The 'Explain' tab is selected, showing a 'QUERY PLAN' table. The table has 5 rows. Row 1: 'Seq Scan on users (cost=0.00..2083.04 rows=7071 width=34) (actual time=0.014..15.143 rows=10000 loops...'. Row 2: 'Filter: ((username)::text ~~ '%7':text)'. Row 3: 'Rows Removed by Filter: 90003'. Row 4: 'Planning Time: 0.078 ms'. Row 5: 'Execution Time: 15.479 ms'. The interface also shows 'Showing rows: 1 to 5' and various icons for saving, copying, and deleting.

	QUERY PLAN
1	Seq Scan on users (cost=0.00..2083.04 rows=7071 width=34) (actual time=0.014..15.143 rows=10000 loops...
2	Filter: ((username)::text ~~ '%7':text)
3	Rows Removed by Filter: 90003
4	Planning Time: 0.078 ms
5	Execution Time: 15.479 ms

Рисунок 5.3 – Результат Select-запроса к таблице Users без индекса

На рисунке 5.4 представлен результат Select-запроса к таблице USERS после создания индекса. Время выполнения уже составляет 9,764 мс.

The screenshot shows a SQL query execution interface. At the top, the query is: `49 explain analyze select * from users where username like '%7';` and `50 create index user_ind on users (username)`. Below the query, there are tabs for 'Data Output', 'Messages', 'Explain', and 'Notifications'. The 'Explain' tab is selected, showing a 'QUERY PLAN' table. The table has 5 rows. Row 1: 'Seq Scan on users (cost=0.00..2083.04 rows=7071 width=34) (actual time=0.020..9.477 rows=10000 loops...'. Row 2: 'Filter: ((username)::text ~~ '%7':text)'. Row 3: 'Rows Removed by Filter: 90003'. Row 4: 'Planning Time: 0.129 ms'. Row 5: 'Execution Time: 9.764 ms'. The interface also shows 'Showing rows: 1 to 5' and various icons for saving, copying, and deleting.

	QUERY PLAN
1	Seq Scan on users (cost=0.00..2083.04 rows=7071 width=34) (actual time=0.020..9.477 rows=10000 loops...
2	Filter: ((username)::text ~~ '%7':text)
3	Rows Removed by Filter: 90003
4	Planning Time: 0.129 ms
5	Execution Time: 9.764 ms

Рисунок 5.4 – Результат Select-запроса к таблице Users с индексом

Оптимизация производительности баз данных, особенно в PostgreSQL, часто связана с использованием индексов для улучшения скорости выполнения запросов. Результаты тестирования эффективности показывают, что создание индексов над таблицами, особенно при больших объемах данных, значительно сокращает время выполнения запросов, что существенно повышает производительность системы.

5.2 Создание индексов

Индексы являются структурами данных, которые ускоряют выполнение запросов на поиск, сортировку или группировку данных в таблицах. Индексы

создаются на основе значений столбцов в таблице и предоставляют быстрый доступ к записям, удовлетворяющим определенным условиям.

Индексы, разработанные в рамках курсового проекта:

- index user_ind. Индекс для имени пользователя;

Пример создания индексов представлен в листинге 5.5.

```
create index user_ind on users (username)
```

Листинг 5.5 – Создания индекса user_ind

В рамках курсового проекта был разработан индекс user_ind для ускорения доступа к данным по имени пользователя в таблице. Создание индексов на столбцах, по которым выполняются частые запросы, значительно повышает производительность базы данных путем оптимизации поиска и фильтрации записей.

5.3 Вывод по разделу

Анализ результатов тестирования производительности базы данных PostgreSQL показал, что создание индексов на таблицах существенно сокращает время выполнения запросов, особенно при работе с большими объемами данных. Эффективное использование индексов является ключевым аспектом оптимизации производительности системы и обеспечивает более быстрый доступ к данным, что в конечном итоге повышает эффективность управленческих и коммерческих приложений.

6. Описание технологии и ее применение в базе данных

6.1 Технология хранения мультимедийной информации

Для создания базы данных для локального фотохостинга была выбрана технология мультимедийные типы в БД.

Мультимедийные типы в базах данных позволяют хранить и обрабатывать различные типы мультимедийных данных, таких как изображения, аудио- и видеофайлы. Для каждого типа мультимедийных данных существуют специальные технологии, которые позволяют эффективно хранить и обрабатывать эти данные в базе данных.

В частности, для хранения изображений используется тип данных `bytea`, который позволяет хранить бинарные данные прямо в записях таблицы, обеспечивая удобство и эффективность доступа. Аудиофайлы могут быть сохранены в базе данных в виде ссылок на файлы, расположенные на сервере. Для обработки таких мультимедийных типов в БД применяются различные технологии, включая URL-адреса и ссылки на серверные файлы. В рамках данного проекта используются два метода для хранения мультимедийных данных: биты и изображения к ним хранятся как ссылки на файлы, а профильные изображения — в виде бинарных данных `bytea`. Пример создания таблицы `artists` с применением технологии мультимедийных типов данных представлен в листинге 6.1.

```
CREATE TABLE artists (  
  artist_id SERIAL PRIMARY KEY,  
  artist_name VARCHAR(255) NOT NULL UNIQUE,  
  photo BYTEA NOT NULL  
);
```

Листинг 6.1 – Создание таблицы `artists`

Выбранный подход к хранению мультимедийных данных в базе данных для фотохостинга обеспечивает гибкость и эффективность при работе с различными типами медиа-контента. Использование типа данных `bytea` для хранения изображений непосредственно в записях таблицы упрощает доступ к данным и обеспечивает компактное хранение.

6.2 Вывод по разделу

Использование мультимедийных типов данных в БД позволяет упростить хранение, управление и обработку мультимедийных данных, а также повысить производительность и надежность системы. Однако, при использовании таких типов данных необходимо учитывать особенности их обработки и хранения в БД и выбирать соответствующие технологии в зависимости от конкретных требований и задач системы.

7 Руководство пользователя

7.1 Тестирование клиентской части

Для начала тестирования клиентской части необходимо создать пользователя, используя регистрацию в разработанном приложении. Также будет создано несколько других пользователей и админ для проверки работы некоторых функций. Регистрация пользователя представлена на рисунке 7.1.

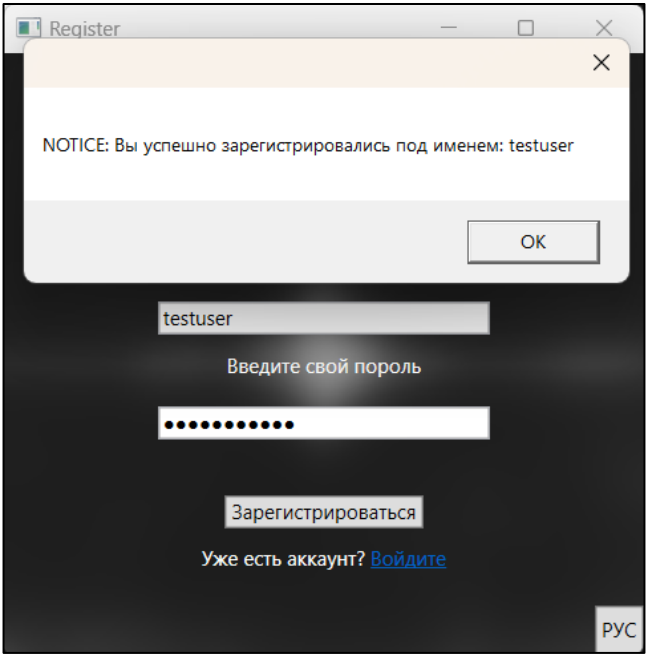


Рисунок 7.1 – Регистрация пользователя

На рисунке 7.2 отображено главное окно приложения со список треков, которые присутствуют на сервисе.

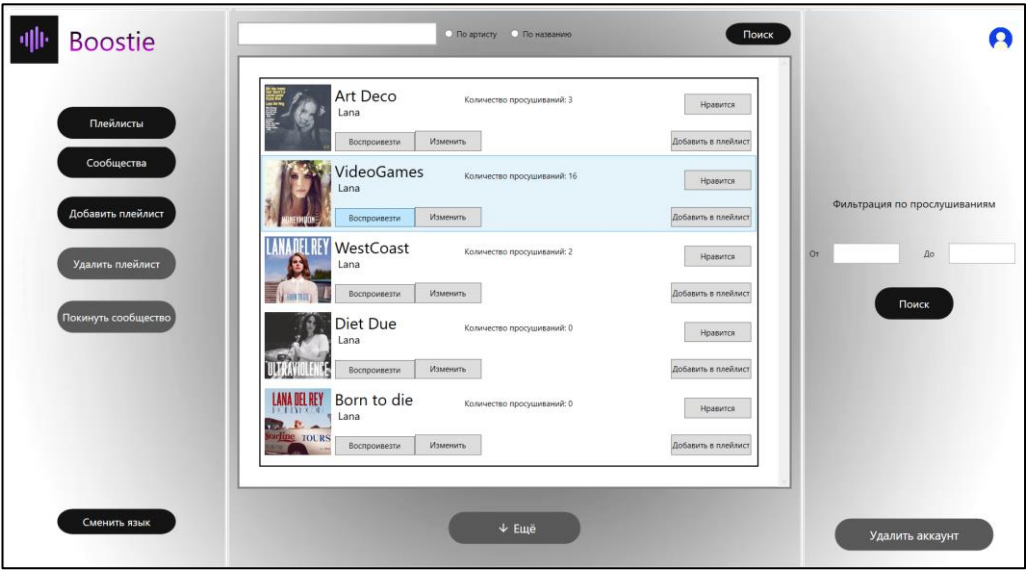


Рисунок 7.2 – Главный экран приложения

Теперь протестируем добавление плейлиста. Для этого необходимо нажать на «Добавить плейлист» в левом меню. В данное окно передается «ID» пользователя, из-за чего созданный плейлист связан с нашим клиентом. Продемонстрировано на рисунке 7.3.

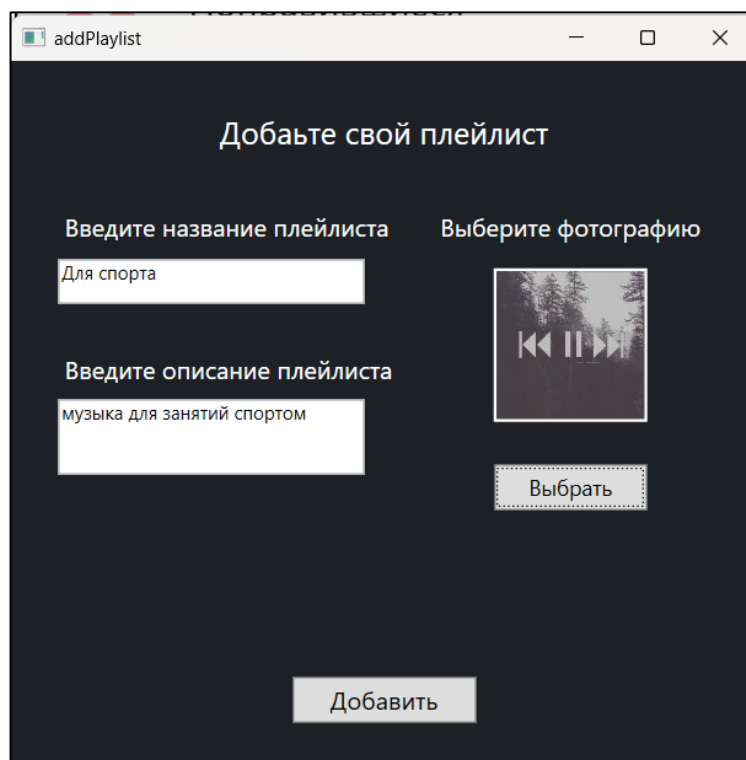


Рисунок 7.3 – Окно добавления плейлиста

После нажатия кнопки «Добавить» вызывается процедура добавления плейлиста. Добавленный плейлист отображается на странице с плейлистами. Страница плейлистов показана на рисунке 7.4.

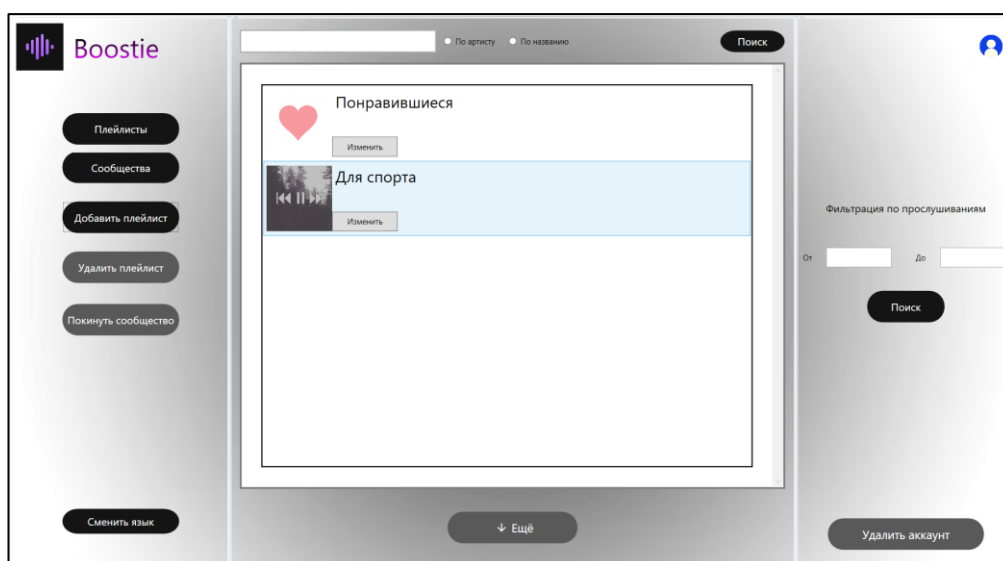


Рисунок 7.4 – Страница плейлистов

После добавления плейлиста мы можем добавить в него любую песню. Для этого нам нужно нажать на кнопку «Добавить в плейлист», которая отображена на рисунке 7.2. Далее нужно в открывшемся окне выбрать плейлист из списка предложенных. Окно с выбором плейлиста показано на рисунке 7.5.

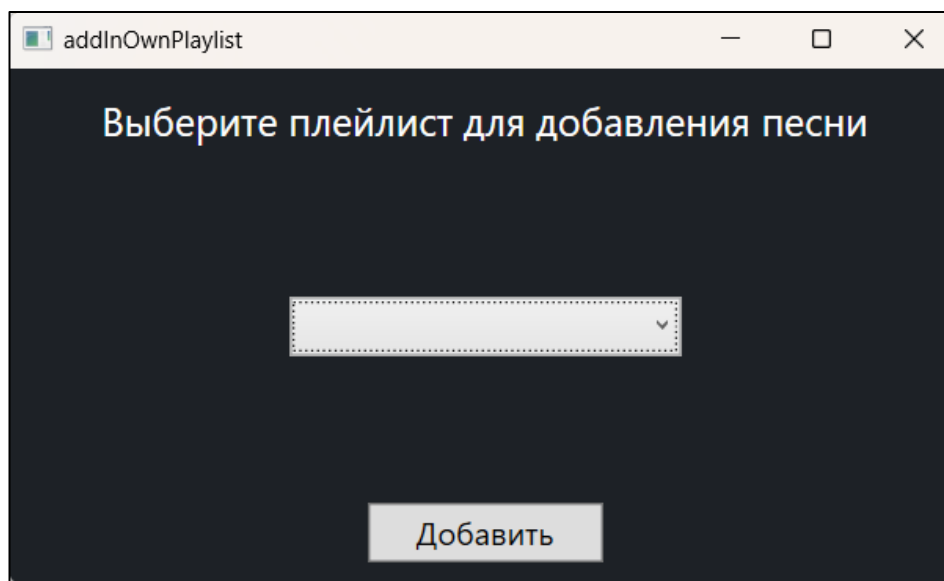


Рисунок 7.5 – Окно выбора плейлиста

Далее в существующем плейлисте появится добавленный трек. Содержимое плейлиста показано на рисунке 7.6.

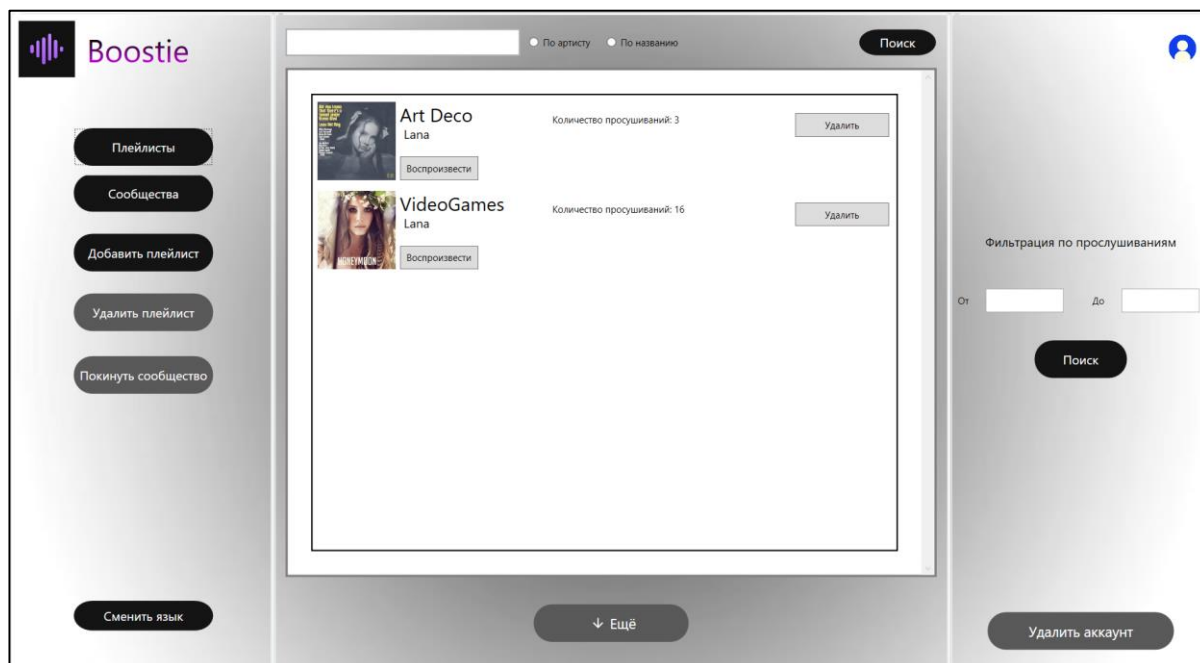


Рисунок 7.6 – Окно выбора плейлиста

Пользователь может вступить в сообщество по жанрам музыки и просматривать список учащихся. Список сообществ показан на рисунке 7.7.

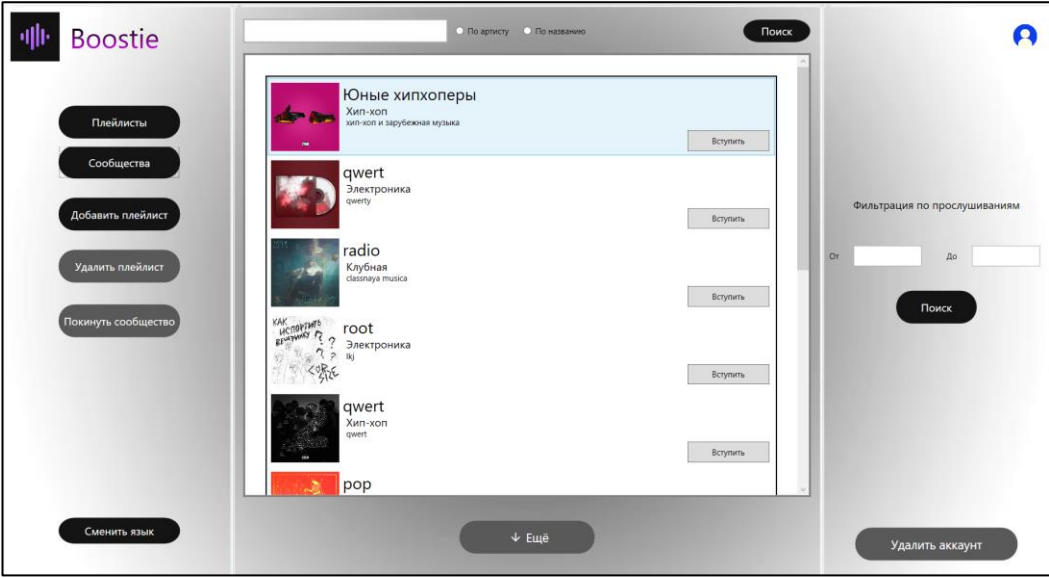


Рисунок 7.7 – Страница сообществ

На рисунке 7.8 показаны участники конкретного сообщества.

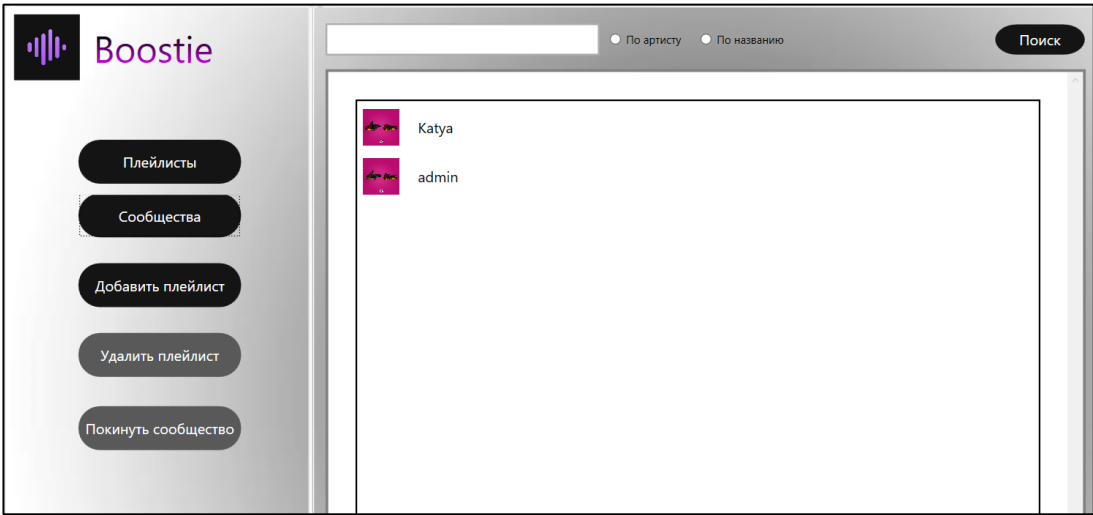


Рисунок 7.8 – Страница с участниками сообщества

Пользователь может выполнять поиск музыки по артисту и по названию, выбирая. Пример поиска по артисту показан на рисунке 7.9.

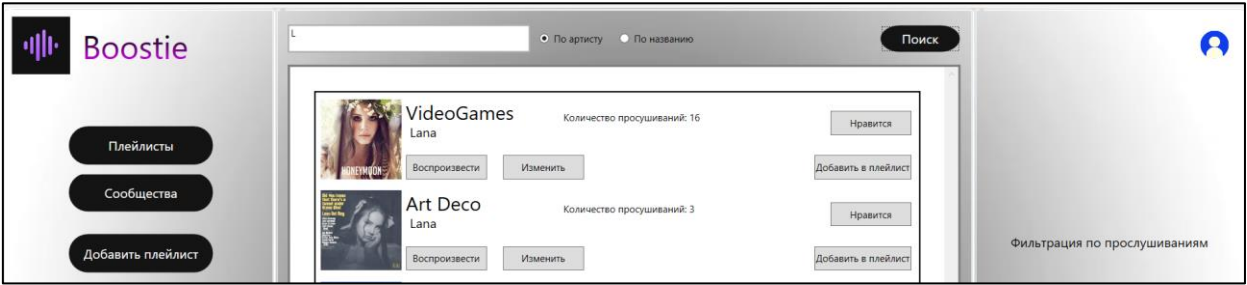


Рисунок 7.9 – Поиск по артисту

Фильтрация музыки по количеству прослушиваний показана на рисунке 7.10.

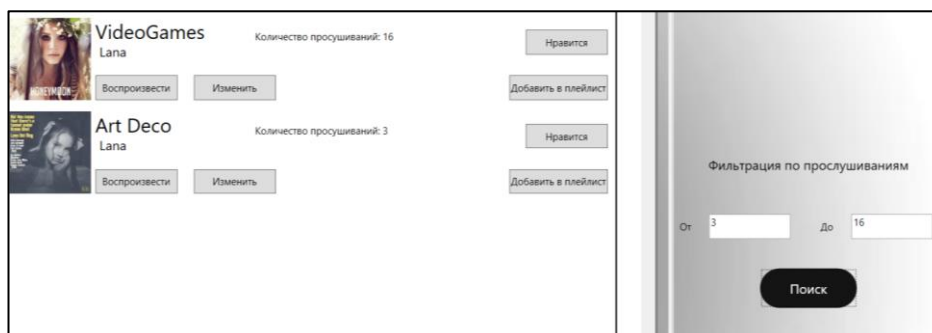


Рисунок 7.10 – Фильтрация по прослушиваниям

У пользователя есть возможность редактировать аккаунт. Редактирование информации об аккаунте показано на рисунке 7.11.

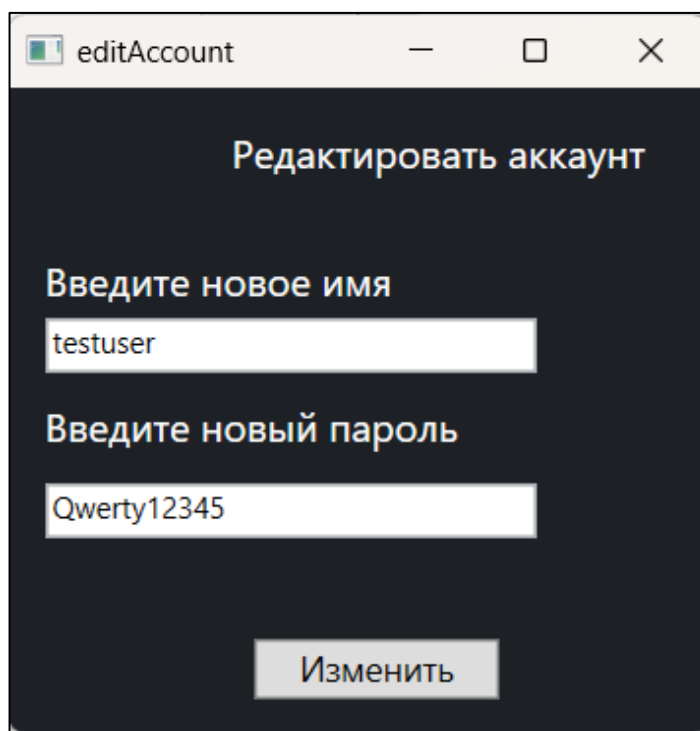


Рисунок 7.11 – Редактирование информации о пользователе

После нажатия на кнопку изменить вызовется процедура `edit_userinfo`, которая изменит в бд данные о пользователе.

7.2 Тестирование области работы администратора

Для получения доступа к аккаунту администратора, нужно войти в приложение под соответствующим логином и паролем. Данные администратора отображены на рисунке 7.12.

	user_id [PK] integer	username character varying (255)	password_hash character varying (255)	role_name character varying (255)
1	5	admin	Adminadmin	admin

Рисунок 7.12 – Данные администратора

Вход в приложение от имени администратора показан на рисунке 7.13.

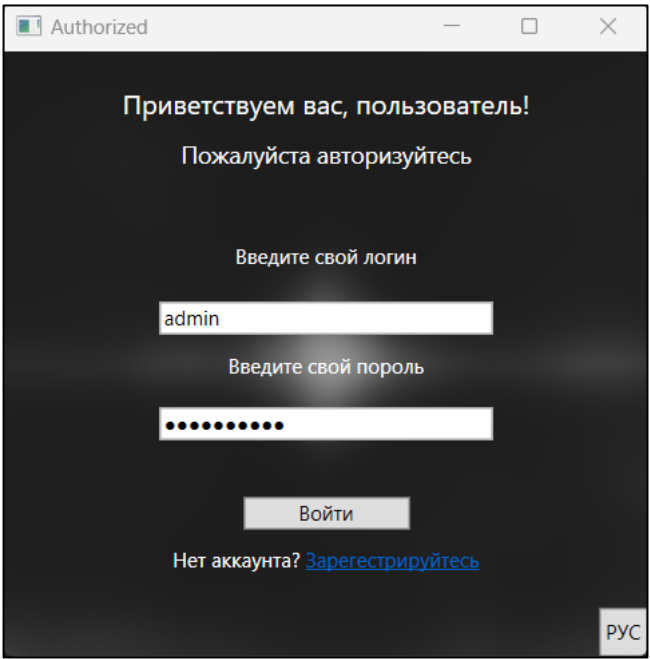


Рисунок 7.13 – Вход в приложение от имени администратора

Страница главного экрана приложения роли admin представлена на рисунке 7.14.

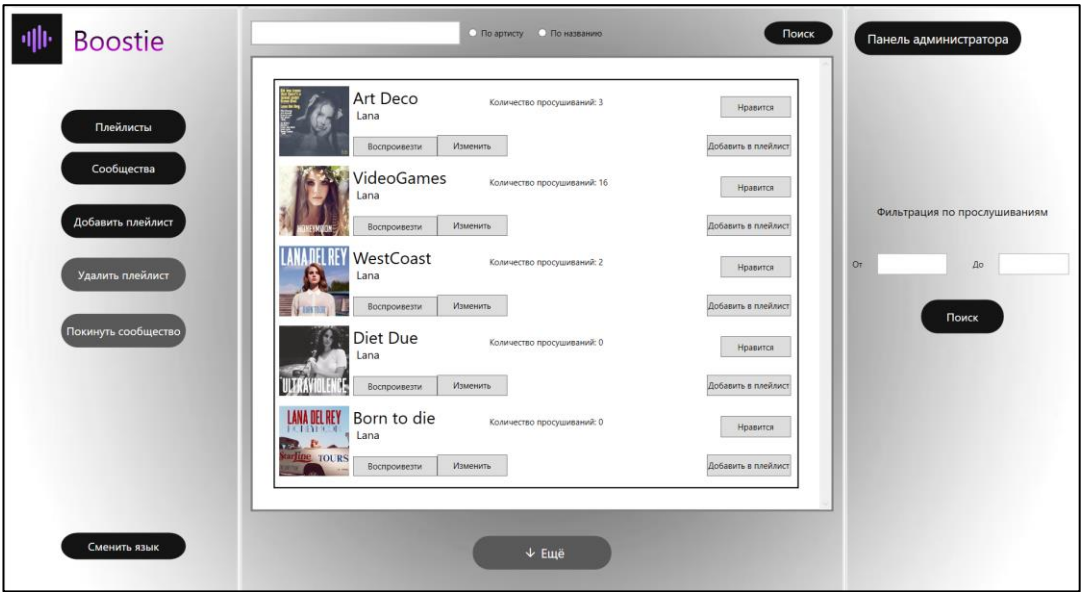


Рисунок 7.14 – Главный экран администратора

Основная возможность модератора – добавление песен. Для этого нужно нажать на кнопку «Панель администратора», а далее на кнопку «Добавить песню». После чего заполнить все данные песни, и если такой песни не существует и при этом существует такой автор, песня успешно добавится. Панель администратора показана на рисунке 7.15.

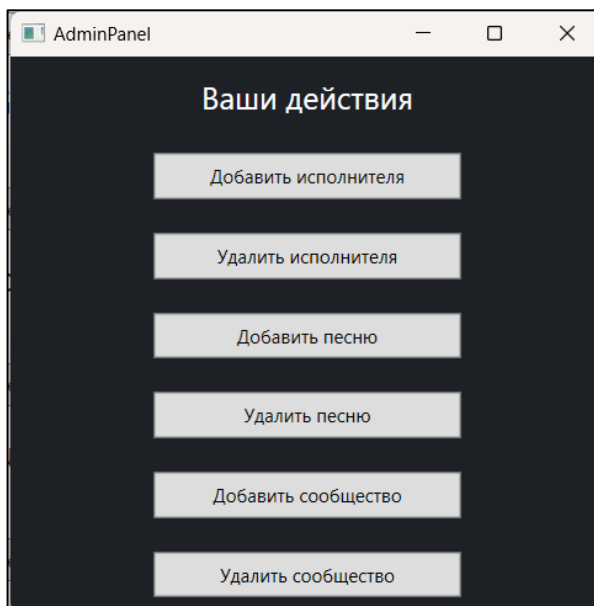


Рисунок 7.15 – Панель администратора

После нажатия на кнопку «Добавить песню» откроется окно добавления песни, отображенное на рисунке 7.16.

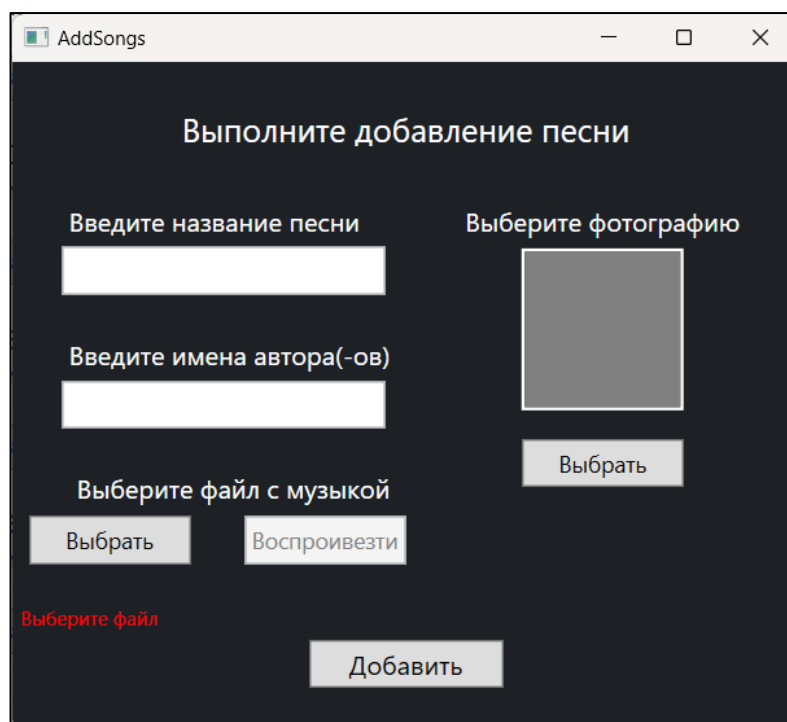


Рисунок 7.16 – Окно добавления песни

Песня будет добавлена в список после заполнения всех полей, а также после нажатия на кнопку «Добавить».

Администратор может использовать весь функционал из панели администратора, показанный на рисунке 7.15.

7.3 Вывод по разделу

Данный раздел описывает функционал приложения, включающий в себя возможности авторизации и регистрации, просмотра списка песен, создания плейлистов, изменение информации песен, сообществ по жанрам и списком людей в них, а также поиск и фильтрации песен.

Из описания интерфейса приложения видно, что оно имеет удобный и интуитивно понятный интерфейс для пользователей и администраторов.

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных для прослушивания музыки, которая в совокупности с приложением формирует полноценное desktop-приложение для прослушивания музыки. Основной целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением, которое могло облегчить взаимодействие с базой данных посредством программного интерфейса. При разработке выполнены следующие пункты:

- Регистрация и авторизация пользователей;
- Поиск, фильтрация, добавление, удаление плейлистов или их содержимого, с возможностью вступать в сообщество со стороны пользователей;
- Удаление и добавление: песен, исполнителей, сообществ со стороны администратора;
- Изменение песен со стороны администратора.

Приложение прошло тестирование при использовании в БД большого количество данных.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список использованных литературных источников

1. Spotify [Электронный ресурс] / Режим доступа: <https://open.spotify.com> – Дата доступа: 20.11.2024.
2. Yandex music [Электронный ресурс] / Режим доступа: <https://music.yandex.by/home> – Дата доступа: 20.1.2024.
3. SoundCloud [Электронный ресурс] / Режим доступа: <https://soundcloud.com> – Дата доступа: 20.1.2024.

Приложение А

```

-- Создание таблицы "Исполнители"
CREATE TABLE artists (
  artist_id SERIAL PRIMARY KEY,
  artist_name VARCHAR(255) NOT NULL UNIQUE,
  photo BYTEA NOT NULL
);
select * from playlist_songs;
select * from songs
-- Создание таблицы "Песни"
CREATE TABLE songs (
  song_id SERIAL PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  audio VARCHAR(255) NOT NULL,
  image VARCHAR(255) NOT NULL,
  listens_count INT DEFAULT 0
);

drop INDEX idx_song_artists_song_id
drop INDEX idx_song_artists_artist_id
CREATE INDEX idx_song_artists_song_id ON song_artists
(song_id);
CREATE INDEX idx_song_artists_artist_id ON song_artists (artist_id);

-- Создание таблицы "Связь песен и исполнителей"
CREATE TABLE song_artists (
  song_artist_id SERIAL PRIMARY KEY,
  song_id INT NOT NULL,
  artist_id INT NOT NULL,
  FOREIGN KEY (song_id) REFERENCES songs(song_id) ON DELETE CASCADE,
  FOREIGN KEY (artist_id) REFERENCES artists(artist_id)
);

select * from users;
-- Создание таблицы "Пользователи"
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role_name VARCHAR(255) NOT NULL
);

-- Создание таблицы "Лайки"
CREATE TABLE likes (
  like_id SERIAL PRIMARY KEY,
  user_id INT NOT NULL,
  song_id INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users(user_id),
  FOREIGN KEY (song_id) REFERENCES songs(song_id) ON DELETE CASCADE

```

```

);

-- Создание таблицы "Плейлисты"
CREATE TABLE playlists (
playlist_id SERIAL PRIMARY KEY,
user_id INT NOT NULL,
title VARCHAR(255) NOT NULL,
descriprion VARCHAR(255),
image_playlist VARCHAR(255),
FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CAS-
CADE
);

-- Создание таблицы "Состав плейлиста"
CREATE TABLE playlist_songs (
playlist_song_id SERIAL PRIMARY KEY,
playlist_id INT NOT NULL,
song_id INT NOT NULL,
FOREIGN KEY (playlist_id) REFERENCES playlists(playlist_id) ON
DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES songs(song_id) ON DELETE CAS-
CADE
);

-- Создание таблицы "Сообщества по жанрам"
CREATE TABLE genre_communities (
community_id SERIAL PRIMARY KEY,
genre_name VARCHAR(255) NOT NULL,
community_name VARCHAR(255) NOT NULL,
community_description VARCHAR(255) NOT NULL,
image_community VARCHAR(255) NOT NULL
);

-- Создание таблицы "Участники сообществ"
CREATE TABLE community_members (
community_member_id SERIAL PRIMARY KEY,
community_id INT NOT NULL,
user_id INT NOT NULL,
FOREIGN KEY (community_id) REFERENCES genre_communities(commu-
nity_id) ON DELETE CASCADE,
FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE artist_descriptions (
description_id SERIAL PRIMARY KEY,
artist_id INT NOT NULL,
birth_date DATE NOT NULL,
artist_info TEXT,
popular_album VARCHAR(255),
listeners_count INT,
FOREIGN KEY (artist_id) REFERENCES artists(artist_id)
);

```

```
-- Удаление таблицы "Сообщества по жанрам"
DROP TABLE IF EXISTS genre_communities CASCADE;

-- Удаление таблицы "Участники сообществ"
DROP TABLE IF EXISTS community_members CASCADE;

-- Удаление таблицы "Лайки"
DROP TABLE IF EXISTS likes CASCADE;

-- Удаление таблицы "Состав плейлиста"
DROP TABLE IF EXISTS playlist_songs CASCADE;

-- Удаление таблицы "Плейлисты"
DROP TABLE IF EXISTS playlists CASCADE;

-- Удаление таблицы "Связь песен и исполнителей"
DROP TABLE IF EXISTS song_artists CASCADE;

-- Удаление таблицы "Песни"
DROP TABLE IF EXISTS songs CASCADE;
-- Удаление таблицы "Исполнители"
DROP TABLE IF EXISTS artists CASCADE;
-- Удаление таблицы "Описания исполнителей"
DROP TABLE IF EXISTS artist_descriptions;
DROP TABLE IF EXISTS users;
```

Листинг А.1 – Создание таблиц

Приложение Б

```
CREATE OR REPLACE PROCEDURE insert_artist(  
    p_artist_name VARCHAR(255),  
    p_photo BYTEA,  
    p_birth_date DATE,  
    p_artist_info TEXT,  
    p_popular_album VARCHAR(255),  
    p_listeners_count INT  
)  
LANGUAGE plpgsql  
SECURITY DEFINER  
AS $$  
DECLARE  
    v_artist_id INT;  
BEGIN  
    -- Проверка наличия артиста с таким же именем  
    IF EXISTS (SELECT 1 FROM artists WHERE artist_name = p_art-  
ist_name) THEN  
        RAISE NOTICE 'Артист с именем % уже существует',  
p_artist_name;  
    ELSE  
        -- Вставка данных в таблицу artists  
        INSERT INTO artists (artist_name, photo)  
VALUES (p_artist_name, p_photo)  
RETURNING artist_id INTO v_artist_id;  
  
        -- Вставка данных в таблицу artist_descriptions  
        INSERT INTO artist_descriptions (artist_id, birth_date,  
artist_info, popular_album, listeners_count)  
VALUES (v_artist_id, p_birth_date, p_artist_info,  
p_popular_album, p_listeners_count);  
  
        RAISE NOTICE 'Исполнитель успешно добавлен!';  
    END IF;  
END;  
$$;
```

Листинг Б.1 – Создание процедуры добавления исполнителя

Приложение В

```

CREATE OR REPLACE PROCEDURE add_song_and_link_artists(
    title_param VARCHAR,
    audio_param VARCHAR,
    image_param VARCHAR,
    artist_names_param VARCHAR[]
)
SECURITY DEFINER
AS $$
DECLARE
    artist_id_param INTEGER[];
    song_exists BOOLEAN;
    song_id_param INTEGER;
BEGIN
    BEGIN
        -- Проверка существования исполнителей
        SELECT array_agg(artist_id) INTO artist_id_param
        FROM artists
        WHERE artist_name = ANY (artist_names_param);

        IF artist_id_param IS NULL THEN
            RAISE NOTICE 'Один или несколько исполнителей не
существуют';
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE NOTICE 'Один или несколько исполнителей не
существуют';
    END;

    -- Проверка наличия песни у указанных исполнителей
    SELECT EXISTS (
        SELECT 1
        FROM songs s
        INNER JOIN song_artists sa ON s.song_id = sa.song_id
        WHERE s.title = title_param AND sa.artist_id = ANY
(artist_id_param)
    ) INTO song_exists;

    IF song_exists THEN
        RAISE NOTICE 'Песня уже существует у указанных исполни-
телей';
    ELSE
        BEGIN
            -- Вставка записи в таблицу songs
            INSERT INTO songs (title, audio, image)
            VALUES (title_param, audio_param, image_param)
            RETURNING song_id INTO song_id_param;

            -- Вставка записей в таблицу song_artists
            INSERT INTO song_artists (song_id, artist_id)

```



```
        SELECT song_id_param, unnest(artist_id_param) AS
artist_id;
    EXCEPTION
        WHEN unique_violation THEN
            RAISE NOTICE 'Такая песня уже существует';
        END;
    END IF;
    RAISE NOTICE 'Песня успешно добавлена!';
END;
$$ LANGUAGE plpgsql;
```

Листинг В.1 – Создание процедуры добавления песни

Приложение Г

```
CREATE OR REPLACE PROCEDURE export_users_to_json(p_file_path
text)
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
    EXECUTE format('COPY (
        SELECT json_agg(row_to_json(t))
        FROM (
            SELECT u.*
            FROM users u
        ) t
    ) TO %L', p_file_path);
END;
$$;
```

Листинг Г.1 – Процедура импорта для таблицы users

```
CREATE OR REPLACE PROCEDURE add_users_from_json_file(
    json_file_path TEXT
)
AS $$
DECLARE
    json_data JSON;
    user_data JSON;
    username_param VARCHAR;
    password_hash_param VARCHAR;
    role_name_param VARCHAR;
    playlist_photo_url_param VARCHAR;
    user_id_output INT;
    user_exists BOOLEAN;
BEGIN
    -- Чтение JSON-файла
    json_data := jsonb(PG_READ_FILE(json_file_path));

    -- Проверка, удалось ли прочитать JSON-файл
    IF json_data IS NULL THEN
        RAISE EXCEPTION 'Не удалось прочитать JSON-файл';
    END IF;

    -- Перебор пользователей в JSON
    FOR user_data IN SELECT * FROM json_array_elements(json_data)
    LOOP
        -- Извлечение значений из JSON
        username_param := user_data->>'username';
        password_hash_param := user_data->>'password';
        role_name_param := user_data->>'role';
        playlist_photo_url_param := user_data->>'playlist_photo_url';
    END LOOP;
END;
```

```
-- Вызов процедуры add_user для каждого пользователя из
JSON
      CALL    add_user(username_param,    password_hash_param,
role_name_param,    playlist_photo_url_param,    user_id_output,
user_exists);
      END LOOP;
END;
$$ LANGUAGE plpgsql;
```

Листинг Г.2 – Процедура экспорта для таблицы users