МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

Выполнил студент

(Ф.И.О.)

Руководитель проекта

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант:

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер:

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[Введение 4](#_Toc132675085)

[1 Анализ требований к программному средству 5](#_Toc132675086)

[1.1 Аналитический обзор аналогов 5](#_Toc132675087)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc132675088)

[1.3 Вывод 8](#_Toc132675090)

[2 Разработка архитектуры проекта 10](#_Toc132675091)

[2.1 Обобщенная структура управлением приложения 10](#_Toc132675092)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 10](#_Toc132675093)

[3 Разработка модели базы данных 13](#_Toc132675094)

[3.1 Создание необходимых объектов 13](#_Toc132675095)

[3.2 Описание используемой технологии 14](#_Toc132675096)

[3.3 Вывод 15](#_Toc132675097)

[4 Установка, настройка и использование PosgtreSQL 14.5 16](#_Toc132675098)

[4.1 Установка PostgreSQL 16](#_Toc132675099)

[4.2 Создание таблиц 16](#_Toc132675100)

[4.3 Создание ролей для разграничения 17](#_Toc132675101)

[4.4 Создание пакетов процедур для базы данных 19](#_Toc132675102)

[4.4.1 Выборка данных из таблиц 20](#_Toc132675103)

[4.4.2 Выборка данных по поисковому запросу 20](#_Toc132675104)

[4.4.3 Заполнение таблиц 100 000 строк 21](#_Toc132675105)

[4.4.4 Добавление данных в таблицы 22](#_Toc132675106)

[4.4.5 Удаление данных в таблицы 22](#_Toc132675107)

[4.4.6 Изменение данных в таблицы 22](#_Toc132675108)

[4.4.7 Дополнительные функции 22](#_Toc132675109)

[4.5 Описание процедур экспорта и импорта 24](#_Toc132675110)

[4.6 Тестирование производительности базы данных 26](#_Toc132675111)

[4.7 Вывод 27](#_Toc132675112)

# Введение

Цель данной работы заключается в создании реляционной базы данных для музыкальной платформы, которая обеспечивает пользователя доступом к имеющимся на ней музыкальным композициям. В рамках работы также необходимо разработать соответствующее приложение, которое будет демонстрировать функциональность базы данных и обеспечивать клиентов доступом к музыкальным трекам.

База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Postgres SQL, поскольку эта система обладает высокой надежностью и производительностью, что позволяет обеспечить эффективное хранение, обработку и управление музыкальными данными.

Также необходимо разработать приложение для демонстрации функциональности базы данных и взаимодействия с ней. Приложение было реализовано с использованием языка программирования Node.js и фреймворка React с TypeScript.

Для гарантированной безопасности пользователей приложения в моей курсовой работе применяется метод шифрования паролей перед их сохранением в базу данных. Также для обеспечения функциональности приложения используются мультимедийные форматы данных при сохранении аудио и картинок.

Основные требования к приложению:

* Реализация ролей администратора и пользователя.
* Поиск аудиозаписей по альбому, исполнителю или жанру.
* Загрузка аудиозаписей на платформу пользователем.
* Взаимодействие с базой данных при помощи хранимых процедур и функций.

В пояснительной записке содержится информация о сопоставимых продуктах, структуре и реализации проекта, а также инструкции по использованию приложения.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Музыка - это одна из самых популярных и распространенных форм искусства, которая имеет давнюю историю и оказывает значительное влияние на культуру и общество в целом. В настоящее время музыкальные платформы являются важной частью музыкальной индустрии, обеспечивая людям доступ к огромной библиотеке музыкальных произведений различных жанров и эпох.

Одним из основных преимуществ музыкальных платформ является возможность слушать музыку в любое время и в любом месте с помощью смартфона, компьютера или другого устройства с доступом в Интернет. Это делает музыкальные платформы очень популярными среди миллионов пользователей по всему миру.

Ниже будут описаны аналоги.

Spotify - это популярная музыкальная платформа, которая позволяет пользователям стримить музыку онлайн, создавать плейлисты и делиться ими с друзьями, слушать радио, а также находить новую музыку, основываясь на своих предпочтениях. Пример интерфейса данного сервиса представлен на рисунке 1.1.

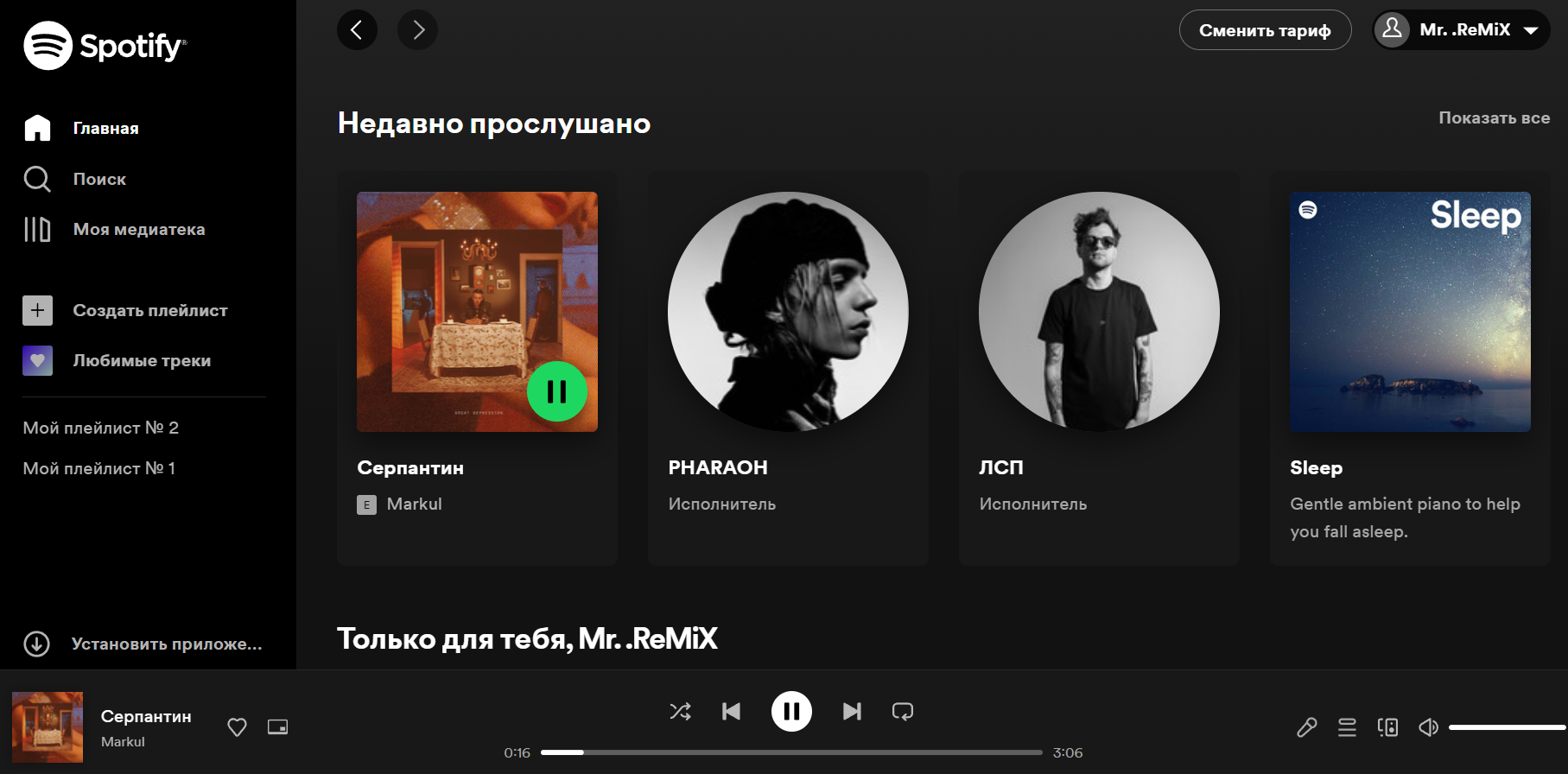


Рисунок 1.1 – Интерфейс сервиса Spotify

Одной из основных функций Spotify является поиск музыки. Пользователи могут искать музыку по альбому, исполнителю, жанру или песне. Кроме того, Spotify предлагает персонализированные рекомендации в соответствии с предпочтениями пользователя, а также плейлисты, созданные другими пользователями и кураторами платформы.

Пользователи могут создавать свои собственные плейлисты и делиться ими с друзьями или сохранять музыку для офлайн-воспроизведения. Spotify также позволяет пользователям слушать радио и подкасты, включая оригинальный контент, созданный самой платформой.

Другая функция Spotify - это возможность использовать платформу как социальную сеть, где пользователи могут подписываться на друг друга, просматривать их плейлисты и рекомендации, а также обмениваться сообщениями.

Spotify также предлагает два варианта подписки: бесплатную и платную. Бесплатная версия содержит рекламу и ограничения в использовании, в то время как платная версия позволяет получить неограниченный доступ к музыке, отсутствие рекламы и другие функции.

Один из аналогов музыкальной платформы, рассмотренной в ходе обзора, - Apple Music. Этот сервис был создан компанией Apple и запущен в 2015 году. Он позволяет пользователям прослушивать музыку на различных устройствах, а также создавать собственные плейлисты и подборки.

Apple Music предлагает пользователю возможность доступа к более чем 75 миллионам песен, а также к подкастам и аудиокнигам. Сервис также предлагает персонализированные рекомендации, основанные на предпочтениях пользователя.

Пользователи могут создавать свои плейлисты и делиться ими с другими пользователями. Apple Music также предлагает радиостанции, созданные известными диджеями и музыкантами, а также живые трансляции концертов и мероприятий.

Интерфейс Apple Music схож с интерфейсом Spotify и предлагает удобный поиск по альбомам, песням и исполнителям. Также сервис позволяет скачивать музыку для офлайн прослушивания. Пример интерфейса данного сервиса представлен на рисунке 1.2.

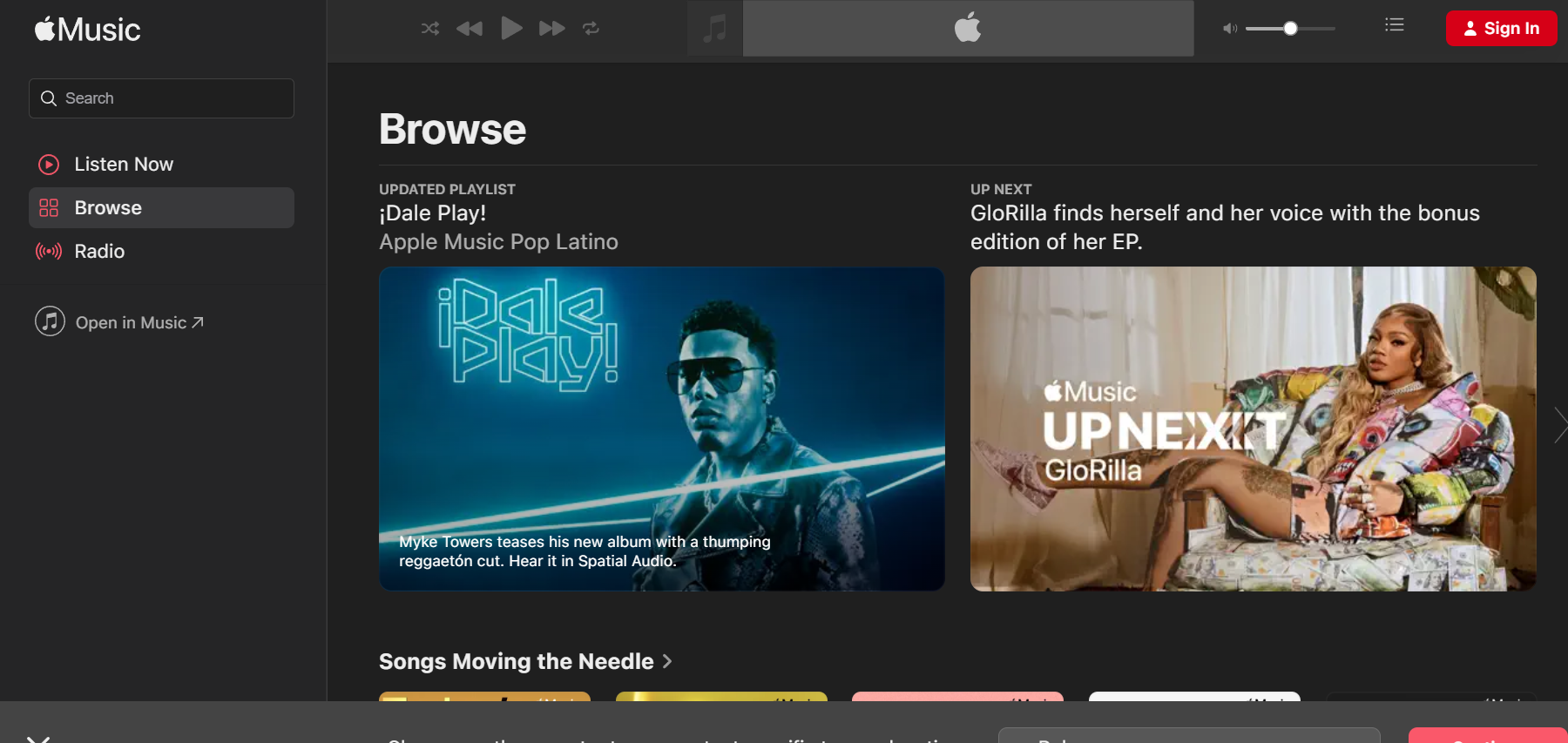


Рисунок 1.2 – Интерфейс сервиса Apple Music

В данном разделе были проведены аналитические обзоры двух популярных музыкальных сервисов - Spotify и Apple Music. Оба сервиса предоставляют пользователям возможность прослушивать музыку онлайн, создавать персональные плейлисты и получать рекомендации на основе своих предпочтений.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами. Например, для музыкальной площадки функциональные требования могут включать в себя функции для хранения информации о музыкальных треках и пользователях, поиска музыки по категориям и критериям, создания и управления плейлистами, а также функции для оценки и прослушивания музыки.

Помимо функциональных требований, важно также определить роли пользователей и их варианты использования системы. Варианты использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Роли пользователя — это набор прав, которые пользователь может получить в системе. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Guest.
* User.
* Manager.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.3.

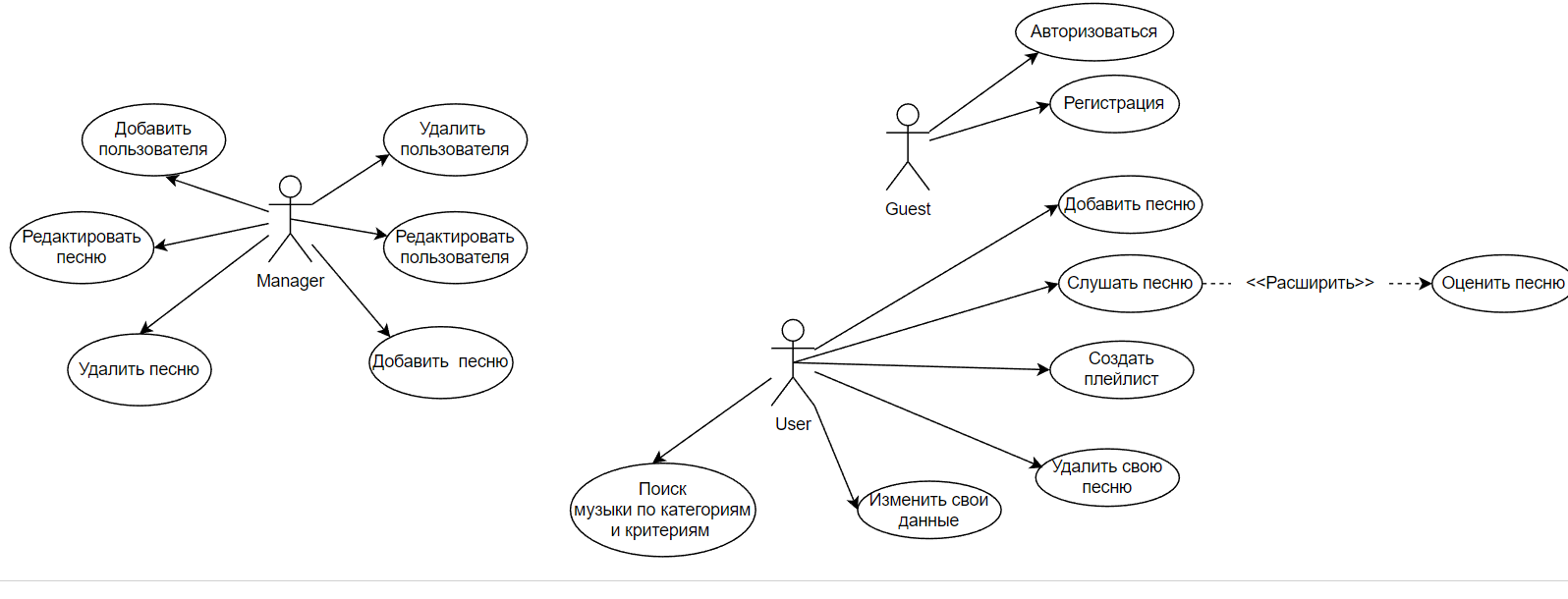


Рисунок 1.3 – UML диаграмма вариантов использования

В начале работы с приложением пользователь является гостем. Ему будет доступна только просмотр всей информации без возможности слушать музыку. После регистрации пользователь становится пользователем (User).

Роль User получает возможность слушать музыку, добавлять понравившиеся треки в свой персональный плейлист, оценивать музыкальные треки и осуществлять поиск музыки по категориям и критериям. Кроме того, User может добавлять в свою библиотеку любимые треки в соответствии с настроением или жанром.

Роль Manager заключается в управлении площадкой и мониторинге действий пользователей. Администратор имеет доступ к просмотру информации о всех пользователях, в том числе их действиях на площадке. При необходимости администратор может редактировать информацию о треках и пользователях, например, изменять категории и теги у треков, а также изменять личную информацию о пользователях.

## В данном разделе были определены роли пользователей и разработаны варианты использования системы в зависимости от этих ролей. Гость может только просматривать информацию, пользователь получает доступ к функциям системы, таким как прослушивание музыки, добавление треков в плейлист и поиск музыки, а администратор имеет права на управление площадкой и редактирование информации о пользователях и треках. Варианты использования системы изображены на UML диаграмме.

## 1.3 Вывод

Итого, был проведен аналитический обзор аналогов музыкальных платформ и сервисов, которые уже существуют на рынке. Этот обзор позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть в разрабатываемой системе. Также были определены функциональные требования базы данных, а также роли пользователей и варианты использования системы в зависимости от этих ролей. Была разработана UML-диаграмма, на которой отображены основные функции, которые доступны для каждой из ролей пользователей.

1. Разработка архитектуры проекта

## Обобщенная структура управлением приложения

Для обеспечения управления приложением с использованием базы данных необходимо разработать удобный и интуитивно понятный интерфейс, который позволит пользователю взаимодействовать с базой данных и эффективно управлять данными. Это может включать в себя разработку оптимизированных запросов для вставки, обновления и удаления данных, а также разработку механизмов для извлечения и обработки информации из базы данных.

В функциональность приложения для прослушивания музыки может входить возможность быстрого и удобного поиска музыкальных композиций, исполнителей и альбомов по различным параметрам, таким как жанр, год выпуска, рейтинг и т.д. Также пользователи должны иметь возможность проигрывать музыкальные композиции на сайте без необходимости скачивания или установки специального программного обеспечения. Для удобства пользователей также должна быть предусмотрена возможность создания собственных плейлистов с возможностью настройки порядка воспроизведения.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма базы данных таблиц (Database Table Diagram) - это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

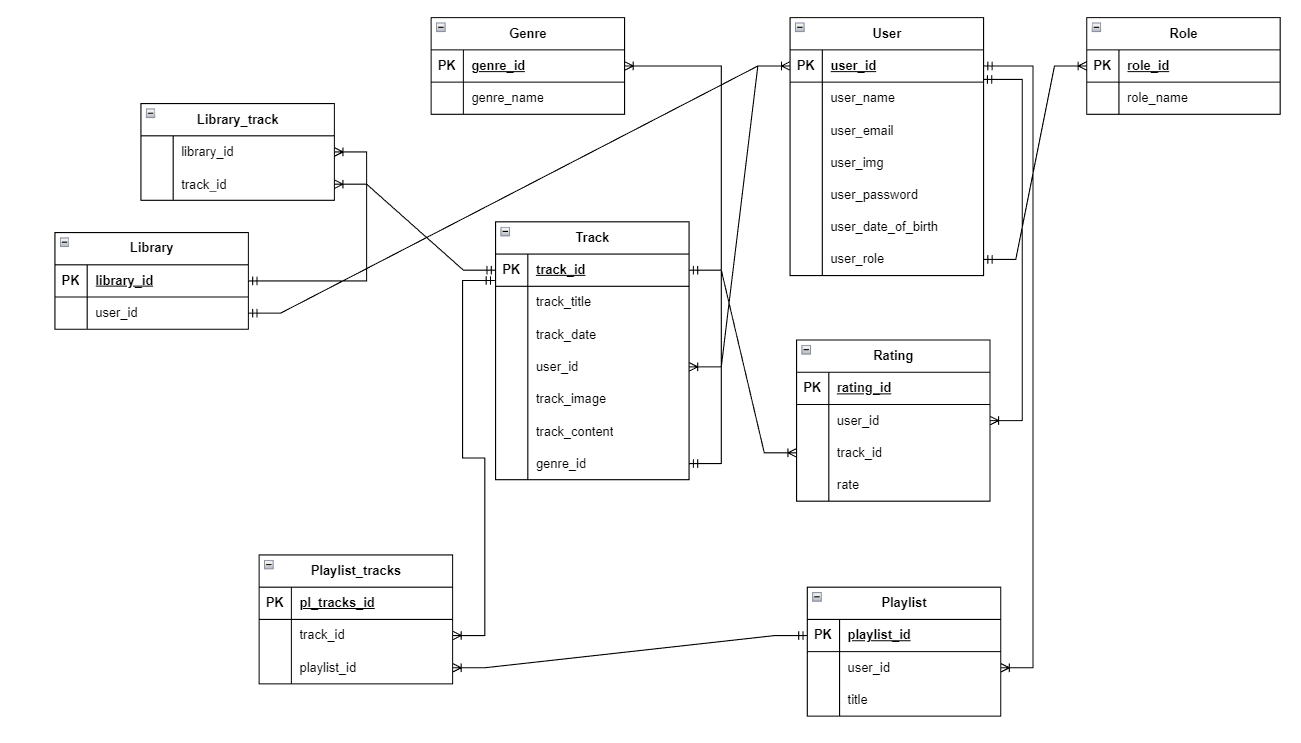


Рисунок 2.1 – Диаграмма базы данных

Таким образом, диаграмма показывает связи между таблицами и полями, а также отношения между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному". Например, таблица Users связана с таблицами Rating, Playlist и Library\_user через внешние ключи user\_id. Также видно, что таблицы Track, Rating, Playlist\_tracks и Library\_track связаны с таблицей Users, а таблицы User\_track, Track и Genre связаны друг с другом через внешние ключи.

**2.3 Описание информационных объектов**

Для реализации базы данных было разработано 8 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Role, Users, Genre, User\_track, Track, Rating, Playlist, Playlist\_tracks, Library\_track. Ниже мы разберем каждую из них более подробно.

Таблица Role представляет собой перечень ролей, которые могут иметь пользователи. Имеющиеся поля: role\_id (идентификатор роли, типа SERIAL, первичный ключ), role\_name (текстовое название роли, тип VARCHAR(255)).

Таблица Users содержит информацию о пользователях. Имеющиеся поля: user\_id (идентификатор пользователя, типа SERIAL, первичный ключ), user\_name (имя пользователя, тип VARCHAR(255)), user\_img (изображение пользователя, тип BYTEA), user\_email (адрес электронной почты пользователя, тип VARCHAR(255)), user\_password (пароль пользователя, тип VARCHAR(255)), user\_date\_of\_birth (дата рождения пользователя, тип DATE), user\_role\_id (идентификатор роли пользователя, тип INTEGER, внешний ключ ссылающийся на role\_id в таблице Role).

Таблица Genre содержит информацию о жанрах музыки. Имеющиеся поля: genre\_id (идентификатор жанра, типа SERIAL, первичный ключ), genre\_name (название жанра, тип VARCHAR(255)).

Таблица User\_track содержит информацию об исполнителях. Имеющиеся поля: track\_id (идентификатор трека, тип INTEGER, внешний ключ ссылающийся на track\_id в таблице Track), user\_id (идентификатор пользователя-владельца, тип INTEGER, внешний ключ ссылающийся на user\_id в таблице Users).

Таблица Track содержит информацию о треках. Имеющиеся поля: track\_id (идентификатор трека, типа SERIAL, первичный ключ), track\_title (название трека, тип VARCHAR(255)), track\_date (дата создания трека, тип DATE, track\_image (изображение трека, тип BYTEA), track\_content (содержание трека, тип BYTEA), genre\_id (идентификатор жанра, тип INTEGER, внешний ключ ссылающийся на genre\_id в таблице Genre).

Таблица Rating содержит информацию о рейтинге пользователей для треков. Имеющиеся поля: rating\_id (идентификатор рейтинга, типа SERIAL, первичный ключ), user\_id (идентификатор пользователя, который оценил трек, тип INTEGER, внешний ключ ссылающийся на user\_id в таблице Users), track\_id (идентификатор трека, который был оценен, тип INTEGER, внешний ключ ссылающийся на track\_id в таблице Track), rate (оценка, которую поставил пользователь, тип INTEGER).

Таблица Playlist содержит информацию о плейлистах пользователей. Имеющиеся поля: playlist\_id (идентификатор плейлиста, типа SERIAL, первичный ключ), user\_id (идентификикатор пользователя, создавшего плейлист, тип INTEGER, внешний ключ ссылающийся на user\_id в таблице Users) и title (название плейлиста, тип VARCHAR(255) NOT NULL).

Таблица Playlist\_tracks связывает плейлисты и треки. Имеющиеся поля: id (идентификатор записи, типа SERIAL, первичный ключ), track\_id (идентификатор трека, тип INTEGER, внешний ключ ссылающийся на track\_id в таблице Track) и playlist\_id (идентификатор плейлиста, тип INTEGER, внешний ключ ссылающийся на playlist\_id в таблице Playlist).

Таблица Library\_track содержит информацию о треках, которые добавлены в библиотеку пользователей. Имеющиеся поля: id (идентификатор записи, типа SERIAL, первичный ключ), user\_id (идентификатор пользователя, который добавил трек в свою библиотеку, тип INTEGER, внешний ключ ссылающийся на user\_id в таблице Users) и track\_id (идентификатор трека, тип INTEGER, внешний ключ ссылающийся на track\_id в таблице Track).

**2.4 Вывод**

Разработка архитектуры проекта необходима для определения структуры и функциональности приложения. Обобщенная структура управления приложения позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать между собой.

Описание информационных объектов является важной частью архитектуры проекта, так как это помогает понять, какие данные будут использоваться в приложении, и как они будут храниться и обрабатываться. В данном проекте были описаны объекты, такие как пользователи, роли, жанры, артисты, треки, рейтинги, плейлисты и библиотека пользователя.

В целом, разработка архитектуры проекта является важным шагом в процессе разработки приложения, так как это позволяет определить необходимые компоненты и информационные объекты, которые помогут создать функциональное и эффективное приложение.

3 Разработка модели базы данных

## Создание необходимых объектов

Для музыкальной площадки необходимо создать несколько таблиц, которые будут хранить данные о пользователях, жанрах, аудиофайлах, плейлистах и прослушиваниях.

Для музыкальной площадки необходимо создать следующие таблицы:

Role - таблица, содержащая роли пользователей. В этой таблице будут храниться записи с ролями, такими как администратор, модератор, пользователь и т.д.

Users - таблица, содержащая данные о пользователе. В этой таблице будут храниться информация о пользователе, такая как имя, фотография, электронная почта, дата рождения и т.д.

Genre - таблица, содержащая жанры музыки. В этой таблице будут храниться записи о жанрах музыки, такие как рок, поп, рэп и т.д.

Track - таблица, содержащая информацию о музыкальных треках. В этой таблице будут храниться данные о музыкальных треках, такие как название, дата выпуска, изображение обложки, файл с музыкой и т.д.

User\_Track - таблица, содержащая отношение между пользователем и музыкальными треками. В этой таблице будут храниться записи, связывающие пользователя с его любимыми треками.

Rating - таблица, содержащая информацию об оценках пользователей музыкальных треков. В этой таблице будут храниться данные об оценках, которые пользователи выставляют трекам.

Playlist - таблица, содержащая информацию о плейлистах пользователей. В этой таблице будут храниться данные о плейлистах, такие как название и пользователь, создавший плейлист.

Playlist\_tracks - таблица, содержащая отношение между плейлистом и музыкальными треками. В этой таблице будут храниться записи, связывающие плейлист с его треками.

Library\_user - таблица, содержащая информацию о библиотеке музыки пользователя. В этой таблице будут храниться записи, связывающие пользователя с его музыкальной библиотекой.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, можно создать индексы на столбцах user\_email в таблице Users, чтобы быстро находить информацию о пользователе по его электронной почте.

Для более удобной работы с базой данных можно создать несколько функций и процедур. Например, можно создать функцию для поиска треков по заданным критериям, таким как жанр, дата выпуска, название и т.д. Также можно создать процедуру для добавления новых треков в базу данных

* + 1. Представления базы данных

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы три представления:

* all\_info\_user, которое объединяет данные таблиц Users и Role;
* all\_info\_track, которое содержит информацию о треках и среднюю оценку каждого трека;
* playlist\_tracks\_info, которое содержит информацию о треках, входящих в определенный плейлист, и их среднюю оценку.

Представление all\_info\_user было создано для того, чтобы получить полную информацию о пользователях и их ролях, объединив данные из двух таблиц.

Представление all\_info\_track было создано для того, чтобы получить информацию о треках и их средней оценке, объединив данные из таблицы Track и вызова функции average\_rating(), которая рассчитывает среднюю оценку каждого трека на основе данных из таблицы Rating.

Представление playlist\_tracks\_info было создано для того, чтобы получить информацию о треках, входящих в определенный плейлист, и их средней оценке, объединив данные из таблиц Playlist\_tracks, Track и вызова функции average\_rating().

* + 1. Индексы базы данных

Индекс — объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Для того, чтобы быстро находить треки по их названию, были созданы два индекса: IDX\_TITLE на таблице TRACK по полю TRACK\_TITLE и idx\_user\_name на таблице USERS по полю USER\_NAME. Сами индексы представлены на листинге 3.4

|  |
| --- |
| CREATE INDEX IDX\_TITLE ON TRACK(track\_title);  CREATE INDEX idx\_user\_name on users(user\_name); |

Листинг 3.1 – Индексы базы данных

* + 1. Триггеры базы данных

Триггер базы данных — это объект базы данных, который выполняет некоторое действие автоматически при определенных событиях в таблице или представлении базы данных. Триггер может быть запрограммирован на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для обеспечения целостности данных и контроля доступа к данным, а также для автоматической обработки данных при выполнении определенных операций в таблице.

Первый триггер, созданный в таблице Users, проверяет, что дата рождения нового пользователя не является будущей датой. Если дата рождения нового пользователя больше или равна текущей дате, то триггер генерирует исключение с сообщением об ошибке "Invalid date of birth". Триггер создается с помощью функции check\_date\_of\_birth(), которая возвращает значение NEW (новые значения строк в таблице) при выполнении вставки или обновления. Скрипт триггера будет представлен на листинге 4.3.

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_date\_of\_birth() RETURNS TRIGGER AS $$  BEGIN  IF NEW.user\_date\_of\_birth >= CURRENT\_DATE THEN  RAISE EXCEPTION 'Invalid date of birth';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_check\_date\_of\_birth  BEFORE INSERT OR UPDATE ON Users  FOR EACH ROW  EXECUTE FUNCTION check\_date\_of\_birth(); |

Листинг 3.1 – Скрип триггера trigger\_check\_date\_of\_birth

Второй триггер проверяет, что длина пароля нового пользователя не менее 6 символов. Если длина пароля меньше 6 символов, то триггер генерирует исключение с сообщением об ошибке "Password must be at least 6 characters long". Триггер создается с помощью функции check\_password\_length(), которая также возвращает значение NEW при выполнении вставки или обновления.

|  |
| --- |
| BEGIN  IF CHAR\_LENGTH(NEW.user\_password) < 6 THEN  RAISE EXCEPTION 'Password must be at least 6 characters long';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_check\_password\_length  BEFORE INSERT OR UPDATE ON Users  FOR EACH ROW  EXECUTE FUNCTION check\_password\_length(); |

Листинг 3.1 – Скрип триггера check\_password\_length

Таким образом, оба триггера выполняют валидацию данных перед их вставкой или обновлением в таблице Users, обеспечивая целостность данных.

## 3.2 Описание используемой технологии

В данной базе данных используется мультимедийность для хранения музыкальных файлов и изображений. Например, в таблице Users в столбце user\_img хранится изображение профиля пользователя в формате BYTEA. В таблице Track в столбце track\_image также хранятся изображения, а в столбце track\_content хранятся музыкальные файлы в формате BYTEA. Также в таблице Library\_user используется связь с таблицей Track для хранения треков, добавленных пользователем в его библиотеку. Таким образом, использование мультимедийности в базе данных позволяет хранить различные мультимедийные данные, необходимые для работы музыкальной площадки.

Когда пользователь на стороне клиента загружает мультимедийные файлы (например, изображения), они отправляются на сервер Node.js для обработки и сохранения в базе данных. Для этого файлы передаются на сервер Node.js, который конвертирует файл в байты, которые затем сохраняются в базу данных в виде типа столбца bytea. Cкрипт на Node.js для конвертации загруженного файла в бинарный формат представлен на листинге 3.1.

|  |
| --- |
| const storage = multer.diskStorage({  destination: function (req, file, cb) {  if (file.mimetype.startsWith("image/")) {  cb(null, "uploads/images/");  } else if (file.mimetype.startsWith("audio/")) {  cb(null, "uploads/music/");  } else {  cb(new Error("Invalid file type"));  }  }, |

Листинг 3.1 – Cкрипт конвертации файла в бинарный формат

Когда же бинарные файлы запрашиваются из базы данных на сервере, они получаются в бинарном формате. Для того, чтобы отправить эти данные на клиент (например, для вывода изображения в браузере), сервер должен преобразовать их в нужный формат, например, в формат utf-8. Cкрипт на Node.js для конвертации бинарного файла представлен на листинге 3.2.

|  |
| --- |
| song.track\_image = `http://localhost:3001/images/${song?.track\_image?.toString("utf-8")}`; |

Листинг 3.2 – Cкрипт конвертации бинарного файла

Таким образом, была рассмотрена тема хранения мультимедийных данных в базах данных.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для музыкальной площадки. Tакже была описана использованная технология мультимедийности для хранения музыкальных файлов и изображений в формате BYTEA. Использование мультимедийности в базе данных позволяет хранить различные мультимедийные данные, необходимые для работы музыкальной площадки.

4 Установка, настройка и использование PosgtreSQL 14.5

## 4.1 Установка PostgreSQL

PostgreSQL - это мощная реляционная база данных с открытым исходным кодом, которая предлагает множество функций, таких как многопоточность, транзакционность, контроль целостности данных, масштабируемость и многое другое. PostgreSQL широко используется в коммерческих и научных проектах, а также веб-приложениях и мобильных приложениях.

Чтобы установить PostgreSQL, нужно выполнить следующие шаги:

1. Загрузить установочный файл PostgreSQL с официального сайта;
2. Запустить установочный файл;
3. Следовать инструкциям установщика, выбрав необходимые опции, такие как расположение установки, язык, компоненты и т.д.;
4. Завершить установку, перезапустив компьютер (в зависимости от настроек установщика).

После установки PostgreSQL, необходимо настроить сервер. Это включает в себя создание пользователя, базы данных и таблиц, а также настройку параметров конфигурации для оптимальной производительности и безопасности.

## Создание таблиц

В данном разделе мы создадим таблицы для нашей базы данных. Но перед тем, как приступить к созданию таблиц, нам нужно создать табличное пространство.

Табличное пространство - это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств будет представлен на листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE TS\_USER LOCATION 'E:/POSTGRESQL/14/DATA/DBS/TS\_USER';  CREATE TABLESPACE TS\_TRACK LOCATION 'E:/POSTGRESQL/14/DATA/DBS/TS\_TRACK';  CREATE TABLESPACE TS\_PLAYLIST LOCATION 'E:/POSTGRESQL/14/DATA/DBS/TS\_PLAYLIST';  CREATE TABLESPACE TS\_LIBRARY LOCATION 'E:/POSTGRESQL/14/DATA/DBS/TS\_LIBRARY'; |

Листинг 4.1 – Cкрипт для создания табличных пространств

Для базы данных создадим шесть основных таблиц: Role, Users, Genre, Track, User\_Track, Rating, Playlist, Playlist\_tracks и Library\_user.

Таблицы Role, Users и Playlist будут храниться в табличном пространстве TS\_USER, таблицы Genre, Track и User\_Track - в табличном пространстве TS\_TRACK, таблица Playlist\_tracks - в табличном пространстве TS\_PLAYLIST, а таблица Library\_user - в табличном пространстве TS\_LIBRARY.

Каждая таблица будет содержать свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Например, ограничение FOREIGN KEY определяет связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, в базе данных будут присутствовать связи между таблицами. Одна из основных связей - это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице. Например, у каждого пользователя может быть множество треков, которые он добавил в свою библиотеку. Для этого мы добавим в таблицу User\_Track внешний ключ (FOREIGN KEY) на таблицу Users, который будет указывать на идентификатор пользователя.

Другой тип связи - это связь "многие ко многим" (many-to-many), которая определяет отношение между множеством записей в одной таблице и множеством записей в другой таблице. Например, у каждого пользователя может быть множество плейлистов, и каждый плейлист может содержать множество треков. Для этого мы создадим таблицу Playlist\_tracks, которая будет содержать внешние ключи на таблицы Playlist и Track. Скрипт создание таблицы User будет представлен на листинге 4.2.

|  |
| --- |
| CREATE TABLE Users (  user\_id SERIAL PRIMARY KEY,  user\_name VARCHAR(255) NOT NULL,  user\_img BYTEA,  user\_email VARCHAR(255) UNIQUE NOT NULL,  user\_password VARCHAR(255) NOT NULL,  user\_date\_of\_birth DATE,  user\_role\_id INTEGER,  CONSTRAINT fk\_user\_role FOREIGN KEY (user\_role\_id) REFERENCES Role(role\_id)  ) TABLESPACE ts\_user; |

Листинг 4.2 – Cкрипт создание таблицы User

Таким образом, было описано создание табличного пространства для базы данных, а также таблиц, которые будут храниться в этих пространствах. Были созданы четыре табличных пространства: TS\_USER, TS\_TRACK, TS\_PLAYLIST и TS\_LIBRARY, в каждом из которых будут храниться соответствующие таблицы.

## Создание ролей для разграничения

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Будут созданы три роли для разграничения доступа к базе данных: user\_role, manager\_role и programmer\_role. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли user\_role и manager\_role являются основными, в то время как роль programmer\_role будет иметь более широкий набор привилегий.

Роль programmer\_role имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли programmer\_role можно увидеть на листинге 4.3.

|  |
| --- |
| GRANT ALL PRIVILEGES ON DATABASE "BD\_MUSIC" TO PROGRAMMER\_ROLE;  GRANT CONNECT ON DATABASE "BD\_MUSIC" TO PROGRAMMER\_ROLE;  GRANT ALL ON TABLESPACE TS\_USER TO PROGRAMMER\_ROLE;  GRANT ALL ON TABLESPACE TS\_TRACK TO PROGRAMMER\_ROLE;  GRANT ALL ON TABLESPACE TS\_PLAYLIST TO PROGRAMMER\_ROLE;  GRANT ALL ON TABLESPACE TS\_LIBRARY TO PROGRAMMER\_ROLE; |

Листинг 4.3 – Привилегии, выданные роли programmer\_role

Роль manager\_role имеет права на выполнение различных функций и процедур, связанных с управлением пользователями и треками в базе данных. Эта роль может выполнять функции добавления, редактирования и удаления треков, а также обновления информации о пользователях. Выданные привилегии роли manager\_role можно увидеть на листинге 4.4.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION REGISTER TO MANAGER\_ROLE;  GRANT EXECUTE ON FUNCTION GETUSERBYID TO MANAGER\_ROLE;  GRANT EXECUTEON FUNCTION SEARCH\_TRACK\_BY\_TITLE\_OR\_USER\_NAME TO MANAGER\_ROLE;  GRANT EXECUTE ON FUNCTION ADDTRACK TO MANAGER\_ROLE;  GRANT EXECUTE ON FUNCTION GETTRACKS TO MANAGER\_ROLE;  GRANT EXECUTE ON FUNCTION GETUSERS TO MANAGER\_ROLE;  GRANT EXECUTE ON FUNCTION GETTRACKBYID TO MANAGER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER TO MANAGER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER\_PASSWORD TO MANAGER\_ROLE;  GRANT EXECUTE ON PROCEDURE DELETE\_TRACK TO MANAGER\_ROLE;  GRANT EXECUTE ON PROCEDURE DELETE\_USER TO MANAGER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_TRACK TO MANAGER\_ROLE;  GRANT SELECT, INSERT, UPDATE ON TABLE USERS TO MANAGER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE TRACK TO MANAGER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE USER\_TRACK TO MANAGER\_ROLE; |

Листинг 4.4 – Привилегии, выданные роли manager\_role

Роль user может добавлять свои треки и редактировать их, создавать плейлисты и добавлять в них треки или добавлять треки в свою библиотеку. Кроме того, user имеет возможность изменять свой профиль и пароль. Роль также может оценивать треки и удалять их из своих плейлистов или библиотеки. Пользователь может просматривать профили других пользователей, искать треки по названию или имени пользователя и просматривать информацию о треках, такую как автор, название, жанр и т.д. Выданные привилегии роли user\_role можно увидеть на листинге 4.5.

|  |
| --- |
| GRANT EXECUTE ON FUNCTION REGISTER TO USER\_ROLE;  GRANT EXECUTE ON FUNCTION LOGIN TO USER\_ROLE;  GRANT EXECUTE ON FUNCTION ADDTRACK TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE ADD\_RATING TO USER\_ROLE;  GRANT EXECUTE ON FUNCTION CREATEPLAYLIST TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE ADD\_TRACK\_TO\_PLAYLIST TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE ADD\_TRACK\_TO\_LIBRARY TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE ADD\_TRACK\_TO\_LIBRARY TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_USER\_PASSWORD TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_TRACK TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE UPDATE\_PLAYLIST TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE DELETE\_TRACK TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE REMOVE\_TRACK\_FROM\_PLAYLIST TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE DELETE\_PLAYLIST TO USER\_ROLE;  GRANT EXECUTE ON PROCEDURE DELETE\_TRACK\_FROM\_LIBRARY TO USER\_ROLE;  GRANT EXECUTE ON FUNCTION GETUSERBYID TO USER\_ROLE;  GRANT EXECUTE ON FUNCTION SEARCH\_TRACK\_BY\_TITLE\_OR\_USER\_NAME TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE ON TABLE USERS TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE TRACK TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE RATING TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE USER\_TRACK TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE PLAYLIST TO USER\_ROLE;  GRANT SELECT, INSERT, UPDATE ON TABLE LIBRARY\_USER TO USER\_ROLE; |

Листинг 4.5 – Привилегии, выданные роли user\_role

Таким образом, были созданы три роли для ограничения доступа к базе данных: user\_role, manager\_role и programmer\_role. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль programmer\_role имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль user\_role имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию и добавлять свои треки. Роль manager\_role имеет средний уровень привилегий и может выполнять функции, связанные с управлением пользователями и треками в базе данных.

## Создание пакетов процедур для базы данных

Для управления данными через приложение пользователи и администраторы используют хранимые процедуры и функции. Хранимая процедура представляет собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Функция также представляет собой набор SQL-инструкций, но возвращает значение, которое может быть использовано внутри другой инструкции SQL.

Написанные в ходе разработки курсового проекта процедуры и функции можно разбить на несколько категорий:

1. Выборка данных из таблиц;
2. Выборка данных по поисковому запросу;
3. Заполнение таблиц 100 000 строк;
4. Добавление данных в таблицы;
5. Удаление данных из таблиц;
6. Изменение данных в таблицах;
7. Дополнительные функции.

Отличие функций от процедур состоит в том, что функции возвращают значение, которое может быть использовано в других SQL-запросах, а процедуры не возвращают значение. Кроме того, функции могут быть использованы в выражениях SQL, например, для вычисления значения поля в запросе SELECT.

В зависимости от того, какую задачу необходимо выполнить, следует использовать хранимую процедуру или функцию. Хранимые процедуры могут использоваться для выполнения сложных операций над данными, таких как массовые изменения в таблицах, а также для оптимизации производительности приложения. Функции же наиболее полезны в случаях, когда требуется выполнить вычисление на основе данных в базе данных, например, для подсчета статистики или фильтрации данных.

### 4.4.1 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры и функции: GetUsers, GetUserById, GetTracks, get\_users\_from\_track, GetTracksUser, GetTrackById,GetPlaylistTracksByID, GetAllPlayListByUserId, GetAllPlaylists, GetTrackFromLibraryByUserID, GetRatingUsers, get\_recent\_tracks. Основная их задача – выборка данных из всех основных таблиц базы данных.

### 4.4.2 Выборка данных по поисковому запросу

Для поиска определенной музыкальной композиции или пользователя, можно использовать функцию search\_track\_by\_title\_or\_user\_name. Данная функция принимает на вход поисковый запрос в виде текста и возвращает таблицу с данными о найденных композициях или пользователях. Функция представлена на листинге 4.6.

|  |
| --- |
| CREATE OR REPLACE FUNCTION search\_track\_by\_title\_or\_user\_name(query\_text VARCHAR) RETURNS TABLE( track\_id INTEGER, track\_title VARCHAR, track\_date DATE, genre\_name VARCHAR, track\_image BYTEA, track\_content BYTEA, avg\_rating NUMERIC) AS $$  BEGIN  RETURN QUERY  SELECT  t.track\_id,  t.track\_title,  t.track\_date,  g.genre\_name,  t.track\_image,  t.track\_content,  average\_rating(t.track\_id) AS avg\_rating  FROM  Track t  JOIN Genre g ON t.genre\_id = g.genre\_id  LEFT JOIN User\_Track ut ON t.track\_id = ut.track\_id  LEFT JOIN Users u ON ut.user\_id = u.user\_id  WHERE  t.track\_title ILIKE '%' || query\_text || '%'  OR u.user\_name ILIKE '%' || query\_text || '%';  END;  $$ LANGUAGE plpgsql; |

Листинг 4.6 – Функция search\_track\_by\_title\_or\_user\_name

Для поиска используется оператор ILIKE, который позволяет выполнить поиск без учета регистра символов. Поисковый запрос в виде текста передается в функцию в качестве аргумента query\_text. Затем возвращается таблица с данными о найденных композициях или пользователях.

### 4.4.3 Заполнение таблиц 100 000 строк

Для заполнения таблицы GENRE была разработана функция INSERT\_GENRES, которая вставляет 100000 строк в таблицу. Функция представлена на листинге 4.7.

|  |
| --- |
| CREATE OR REPLACE FUNCTION INSERT\_GENRES()  RETURNS VOID AS $$  DECLARE  I INTEGER := 1;  BEGIN  WHILE I <= 100000 LOOP  INSERT INTO GENRE (GENRE\_NAME) VALUES ('GENRE ' || I);  I := I + 1;  END LOOP;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.7 – Функция заполнения таблицы GENRE

Функция INSERT\_GENRES была создана для заполнения таблицы GENRE 100000 строками. В теле функции используется цикл WHILE, который проходит по значениям от 1 до 100000 и для каждого значения выполняет вставку новой строки в таблицу GENRE с именем 'GENRE' и порядковым номером из цикла. Функция не возвращает значение и не принимает аргументов. Для выполнения функции необходимо выполнить SELECT INSERT\_GENRES().

### 4.4.4 Добавление данных в таблицы

Были разработаны следующие процедуры и функции для добавления новых строк в основные таблицы базы данных: Register, AddTrack, add\_rating, CreatePlaylist, add\_track\_to\_playlist, add\_track\_to\_library.

### 4.4.5 Удаление данных в таблицы

Для удаления данных из базы данных были созданы процедуры: DELETE\_USER, DELETE\_TRACK, REMOVE\_TRACK\_FROM\_PLAYLIST, DELETE\_PLAYLIST, DELETE\_TRACK\_FROM\_LIBRARY.

### 4.4.6 Изменение данных в таблицы

Для изменение данных в базе данных были созданы следующие процедуры: UPDATE\_USER, UPDATE\_USER\_PASSWORD, UPDATE\_TRACK, UPDATE\_TRACK\_TITLE, UPDATE\_PLAYLIST.

### 4.4.7 Дополнительные функции

Дополнительные функции в базе данных могут быть полезны для решения различных задач, которые не решаются стандартными запросами.

Функция encrypt\_password позволяет зашифровать пароль с помощью ключа, что может повысить безопасность хранения паролей пользователей в базе данных. Функция представлена на листинге 4.8.

|  |
| --- |
| CREATE OR REPLACE FUNCTION encrypt\_password(password TEXT)  RETURNS TEXT  AS $$  DECLARE  i INTEGER := 1;  key TEXT := '323232'; --key  key\_length INTEGER := LENGTH(key);  encrypted\_password TEXT := '';  BEGIN  IF password IS NULL THEN  RETURN NULL;  END IF;  FOR i IN 1..LENGTH(password) LOOP  encrypted\_password := encrypted\_password || CHR(ASCII(SUBSTRING(password, i, 1)) # ASCII(SUBSTRING(key, i % key\_length + 1, 1)));  END LOOP;  RETURN encrypted\_password;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.8 – Функция encrypt\_password

Функция decrypt\_password позволяет расшифровать зашифрованный пароль, что может быть полезно для проверки правильности введенного пароля при аутентификации. Функция представлена на листинге 4.9.

|  |
| --- |
| CREATE OR REPLACE FUNCTION decrypt\_password(encrypted\_password TEXT)  RETURNS TEXT  AS $$  DECLARE  i INTEGER := 1;  key TEXT := '323232';  key\_length INTEGER := LENGTH(key);  password TEXT := '';  BEGIN  IF encrypted\_password IS NULL THEN  RETURN NULL;  END IF;  FOR i IN 1..LENGTH(encrypted\_password) LOOP  password := password || CHR(ASCII(SUBSTRING(encrypted\_password, i, 1)) # ASCII(SUBSTRING(key, i % key\_length + 1, 1)));  END LOOP;  RETURN password;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.9 – Функция decrypt\_password

Функция average\_rating вычисляет средний рейтинг трека на основе данных таблицы Rating, что может использоваться для отображения рейтинга трека в приложении. Функция представлена на листинге 4.10.

|  |
| --- |
| CREATE OR REPLACE FUNCTION average\_rating(track\_id INTEGER) RETURNS NUMERIC AS $$  DECLARE  total\_rating INTEGER;  num\_ratings INTEGER;  BEGIN  SELECT SUM(rate), COUNT(\*) INTO total\_rating, num\_ratings  FROM Rating  WHERE Rating.track\_id = $1;  IF num\_ratings = 0 THEN  RETURN 0;  END IF;  RETURN total\_rating::NUMERIC / num\_ratings;  END;  $$ LANGUAGE plpgsql; |

Листинг 4.10 – Функция average\_rating

В целом, эти функции могут быть полезны для повышения безопасности хранения паролей пользователей, а также для получения дополнительной информации о треках в базе данных.

## Описание процедур экспорта и импорта

База данных имеет возможность экспортировать и импортировать данные для таблицы Users в форматах JSON и XML. Это может быть полезно в случае необходимости переноса данных на другой сервер или резервного копирования данных.

Для экспорта данных в формате JSON была создана функция EXPORT\_USERS\_TO\_JSON\_FILE, которая принимает имя файла. Функция представлена на листинге 4.11.

|  |
| --- |
| CREATE OR REPLACE FUNCTION EXPORT\_USERS\_TO\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS VOID AS  $$  DECLARE  JSON\_DATA JSON;  BEGIN  SELECT JSON\_AGG(ROW\_TO\_JSON(USERS)) INTO JSON\_DATA FROM USERS;  PERFORM PG\_FILE\_WRITE(FILE\_PATH, JSON\_DATA::TEXT,'TRUE');  END;  $$  LANGUAGE PLPGSQL; |

Листинг 4.10 – Функция EXPORT\_USERS\_TO\_JSON\_FILE

Для экспорта данных в формате XML была создана функция EXPORT\_USERS\_TO\_XML\_FILE, которая принимает имя файла. Функция представлена на листинге 4.11.

|  |
| --- |
| CREATE OR REPLACE FUNCTION EXPORT\_USERS\_TO\_XML\_FILE(FILE\_PATH TEXT)  RETURNS VOID AS  $$  DECLARE  XML\_DATA XML;  BEGIN  SELECT XMLELEMENT(NAME "USERS", XMLAGG(XMLELEMENT(NAME "USER",  XMLFOREST(USER\_ID, USER\_NAME, USER\_EMAIL, USER\_DATE\_OF\_BIRTH, USER\_ROLE\_ID)))) INTO XML\_DATA FROM USERS;  XML\_DATA := FORMAT('<?XML VERSION="1.0" ENCODING="UTF-8"?>%S', XML\_DATA::TEXT);  PERFORM PG\_FILE\_WRITE(FILE\_PATH, XML\_DATA::TEXT,'TRUE');  END;  $$  LANGUAGE PLPGSQL; |

Листинг 4.11 – Функция EXPORT\_USERS\_TO\_XML\_FILE

Для импорта данных из файла в формате JSON была создана процедура IMPORT\_USERS\_TO\_JSON\_FILE которая принимает имя файла и импортирует данные из файла в таблицу Users. Функция представлена на листинге 4.12.

|  |
| --- |
| CREATE OR REPLACE FUNCTION IMPORT\_USERS\_FROM\_JSON\_FILE(FILE\_PATH TEXT)  RETURNS TABLE (  USER\_ID INTEGER,  USER\_NAME VARCHAR(255),  ) AS $$  DECLARE  JSON\_DATA JSON;  USER\_DATA JSON;  BEGIN  JSON\_DATA := PG\_READ\_FILE(FILE\_PATH, 0, 1000000000)::JSON;  FOR USER\_DATA IN SELECT \* FROM JSON\_ARRAY\_ELEMENTS(JSON\_DATA)  LOOP  INSERT INTO TEMP\_USERS (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_ROLE\_ID)  VALUES (  USER\_DATA->>'USER\_NAME',  USER\_DATA->>'USER\_EMAIL',  USER\_DATA->>'USER\_PASSWORD',  TO\_DATE(USER\_DATA->>'USER\_DATE\_OF\_BIRTH', 'YYYY-MM-DD'),  CAST(USER\_DATA->>'USER\_ROLE\_ID' AS INTEGER)  );  END LOOP;  RETURN QUERY SELECT \* FROM TEMP\_USERS;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.12 – Функция IMPORT\_USERS\_TO\_JSON\_FILE

Для импорта данных из файла в формате XML была создана процедура IMPORT\_USERS\_TO\_XML\_FILE которая принимает имя файла и импортирует данные из файла в таблицу Users. Функция представлена на листинге 4.13.

|  |
| --- |
| CREATE OR REPLACE FUNCTION IMPORT\_USERS\_FROM\_XML\_FILE(FILE\_PATH TEXT)  RETURNS TABLE (  USER\_NAME VARCHAR(255),  USER\_EMAIL VARCHAR(255),  USER\_PASSWORD VARCHAR(255),  ) AS $$  DECLARE  XML\_DATA XML;  USER\_DATA RECORD;  BEGIN  XML\_DATA := XMLPARSE(DOCUMENT CONVERT\_FROM(PG\_READ\_BINARY\_FILE(FILE\_PATH), 'UTF8'));    FOR USER\_DATA IN SELECT \* FROM XMLTABLE('/USERS/USER' PASSING XML\_DATA COLUMNS  USER\_NAME VARCHAR(255) PATH 'USER\_NAME',  USER\_EMAIL VARCHAR(255) PATH 'USER\_EMAIL',  USER\_PASSWORD VARCHAR(255) PATH 'USER\_PASSWORD',  USER\_DATE\_OF\_BIRTH DATE PATH 'USER\_DATE\_OF\_BIRTH',  USER\_ROLE\_ID INTEGER PATH 'USER\_ROLE\_ID'  ) LOOP  INSERT INTO TEMP\_USERS (USER\_NAME, USER\_EMAIL, USER\_PASSWORD, USER\_DATE\_OF\_BIRTH, USER\_ROLE\_ID)  VALUES (  USER\_DATA.USER\_NAME,  USER\_DATA.USER\_EMAIL,  USER\_DATA.USER\_PASSWORD,  USER\_DATA.USER\_DATE\_OF\_BIRTH,  USER\_DATA.USER\_ROLE\_ID  );  END LOOP;  RETURN QUERY SELECT \* FROM TEMP\_USERS;  END;  $$ LANGUAGE PLPGSQL; |

Листинг 4.13 – Функция IMPORT\_USERS\_TO\_XML\_FILE

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON и XML, что делает управление базой данных более удобным и эффективным.

## Тестирование производительности базы данных

Тестирование производительности является важным этапом разработки, поскольку позволяет определить, насколько хорошо база данных может обрабатывать запросы и как быстро она может возвращать результаты.

Для тестирования производительности базы данных была выбрана таблица genre, содержащая больше всего данных. Для получения выборки данных использовался запрос, который представлен на листинге 4.14.

|  |
| --- |
| select \* from genre limit 500 |

Листинг 4.14 – Запрос к таблице genre

Время выполнения этого запроса составило 402 миллисекунды. Результат будет на рисунке 4.1.

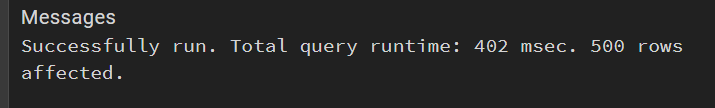


Рисунок 4.1 – Результат выполнения запроса

Тестирование производительности базы данных показало, что выбранный запрос к таблице genre был выполнен достаточно быстро (402 миллисекунды). Однако, для полного тестирования производительности базы данных необходимо провести более широкий диапазон тестов, включая запросы к другим таблицам с различными условиями и параметрами. Также следует обращать внимание на количество данных в таблицах и оптимизацию запросов для достижения максимальной производительности.

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования PostgreSQL. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и пакетов процедур для выполнения различных операций с данными.

Были также представлены процедуры экспорта и импорта данных в форматах Json и Xml, а также были проведены тесты производительности базы данных. В результате тестирования было установлено, что база данных PostgreSQL обладает высокой производительностью и способна быстро обрабатывать запросы на получение данных.

Итак, можно сделать вывод, что PostgreSQL является мощной и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка PostgreSQL, а также оптимизация запросов, позволят обеспечить высокую производительность и эффективность работы с базой данных.

5 Тестирование, проверка работоспособности и анализ полученных результатов

## 5.1 Тестирование клиентской области

## 5.2 Тестирование области работы администратора

## 5.3 Вывод

6 Руководство по использованию программного средства

## 6.1 Установка приложения

## 6.2 Вывод

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для музыкальной площадки с использованием технологии применения мультимедийных типов данных в СУБД PostgreSQL.

В процессе выполнения работы были использованы различные объекты, включая таблицы, триггеры и функции, чтобы обеспечить структурированное хранение данных и своевременный доступ к ним. В результате, цель работы была успешно достигнута, и база данных готова к использованию.

Тестирование базы данных было проведено при использовании большого объема данных, и результаты были положительными. Были реализованы процедуры для импорта и экспорта данных в формате JSON и XML, что обеспечило удобство использования и управления данными.

Одной из ключевых особенностей разработанной базы данных является технология хранения мультимедийных данных, что позволяет эффективно управлять медиа-файлами на площадке.

В целом, результаты работы говорят о том, что база данных полностью соответствует поставленным требованиям и может успешно использоваться в рамках музыкальной площадки.

# Список литературных источников

1. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.04.2023

2. Postgresqltutorial.com [Электронный ресурс] / Режим доступа: https://www.postgresqltutorial.com/ – Дата доступа: 18.04.2023

3. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 18.04.2023

# Приложение А