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 технологий

Специализация 1–40 01 01 Программное обеспечение информационных технологий (программирование интернет – изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных сервиса по прослушиванию музыки с использованием мультимедийных типов данных»

Выполнил студент Хованский Тимофей Александрович

(Ф.И.О.)

Руководитель проекта асс., Нистюк О.А.

(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:  асс., Нистюк О.А.

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

Нормоконтролер: асс., Нистюк О.А.

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

Курсовой проект защищен с оценкой

**Содержание**

[Введение 3](#_Toc122118153)

[1 Разбор аналогов и постановка задачи 4](#_Toc122118154)

[2 Разработка модели базы данных 6](#_Toc122118155)

[3 Разработка необходимых объектов 7](#_Toc122118156)

[3.1 Таблицы базы данных 7](#_Toc122118157)

[3.2 Процедуры базы данных 8](#_Toc122118158)

[3.2.1 Выборка данных из таблиц 8](#_Toc122118159)

[3.2.2 Выборка данных по поисковому запросу c пагинацией 8](#_Toc122118160)

[3.2.3 Заполнение таблиц 100 000 строк 9](#_Toc122118161)

[3.2.4 Выборка данных из представлений со статистикой 9](#_Toc122118162)

[3.2.5 Удаление данных из таблиц 10](#_Toc122118163)

[3.2.6 Добавление данных в таблицы 10](#_Toc122118164)

[3.2.7 Изменение данных в таблицах 10](#_Toc122118165)

[3.3 Представления базы данных 10](#_Toc122118166)

[3.4 Индексы базы данных 11](#_Toc122118167)

[3.5 Триггеры базы данных 12](#_Toc122118168)

[3.6 Пользователи базы данных 12](#_Toc122118169)

[4 Описание процедур импорта и экспорта 14](#_Toc122118170)

[5 Тестирование производительности 15](#_Toc122118171)

[6 Описание технологии 16](#_Toc122118172)

[7 Обоснование технических приемов программирования 18](#_Toc122118173)

[8 Руководство пользователя 20](#_Toc122118174)

[Заключение 23](#_Toc122118175)

[Список литературы 24](#_Toc122118176)

[Приложение А 25](#_Toc122118177)

Введение

Целью данной работы являлась разработка реляционной базы данных для музыкальной (стриминговой) площадки. Эта база данных должна составлялась для обеспечения клиента доступом к музыкальным композициям, имеющимся на ней. Так же было необходимо разработать соответствующее приложение, для демонстрации её работы.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины. Реляционная база данных — база данных, основанная на реляционной модели данных. В качестве СУБД для базы данных была выбрана Oracle 21c, в связи с ее высокой производительностью и надежностью.

Так же было необходимо разработать приложение для демонстрации работы базы данных, взаимодействия с ней. Приложение было написано на языке программирования Java с графическим интерфейсом, выполненным с помощью фреймворка React. Для взаимодействия с сервером базы данных Oracle использовался Spring JDBC.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации работы веб-сервера с базой данных.

Для обеспечения безопасности пользователей приложения в моем курсовом проекте используется технология шифрования паролей от аккаунта перед записью их в базу данных. А также для обеспечения работы приложения использовались мультимедийные типы данных при хранении изображений и аудио файлов.

Основные требования к приложению:

* реализация ролей администратора и рядового пользователя,
* поиск аудиозаписей по жанру, автору и рейтингу пользователя,
* загрузка аудиозаписей на платформу администратором,
* скачивание песен пользователем,
* взаимодействие с базой данных при помощи хранимых процедур.

В пояснительной записке вы сможете найти краткую информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

1. Разбор аналогов и постановка задачи

Музыка играет огромную роль в жизни человека. Одной из главных ее функций является объединять людей, поскольку язык музыки понятен без перевода. Значимость музыки заключается в ее способности обогащать жизнь человека и общества в целом.

Изобретение радио и кинематографа позволило перейти на следующий уровень инженерных разработок – телевидения, которое стало играть важную роль в продвижении музыки и популяризации артистов. В начале 1980-х гг. музыкальное телевидение затмило важность радио. А сегодня стриминговые сервисы стали еще популярнее и удобнее телевидения.

Одним из основных трендов музыкальных приложений остается курируемая музыка. Это так называемые «радиостанции», создающиеся с минимальных использованием компьютерных алгоритмов. В составлении плейлистов участвуют люди – музыкальные эксперты, знаменитости и т.д.

При разработке своего приложения я обратила внимания на самые успешные примеры популярнейших музыкальных площадок. Рассмотрим их ниже.

Сегодня Яндекс Музыка – крупнейший русскоязычный музыкальный стриминговый сервис. Интерфейс этого сервиса можно увидеть на рисунке 1.1.

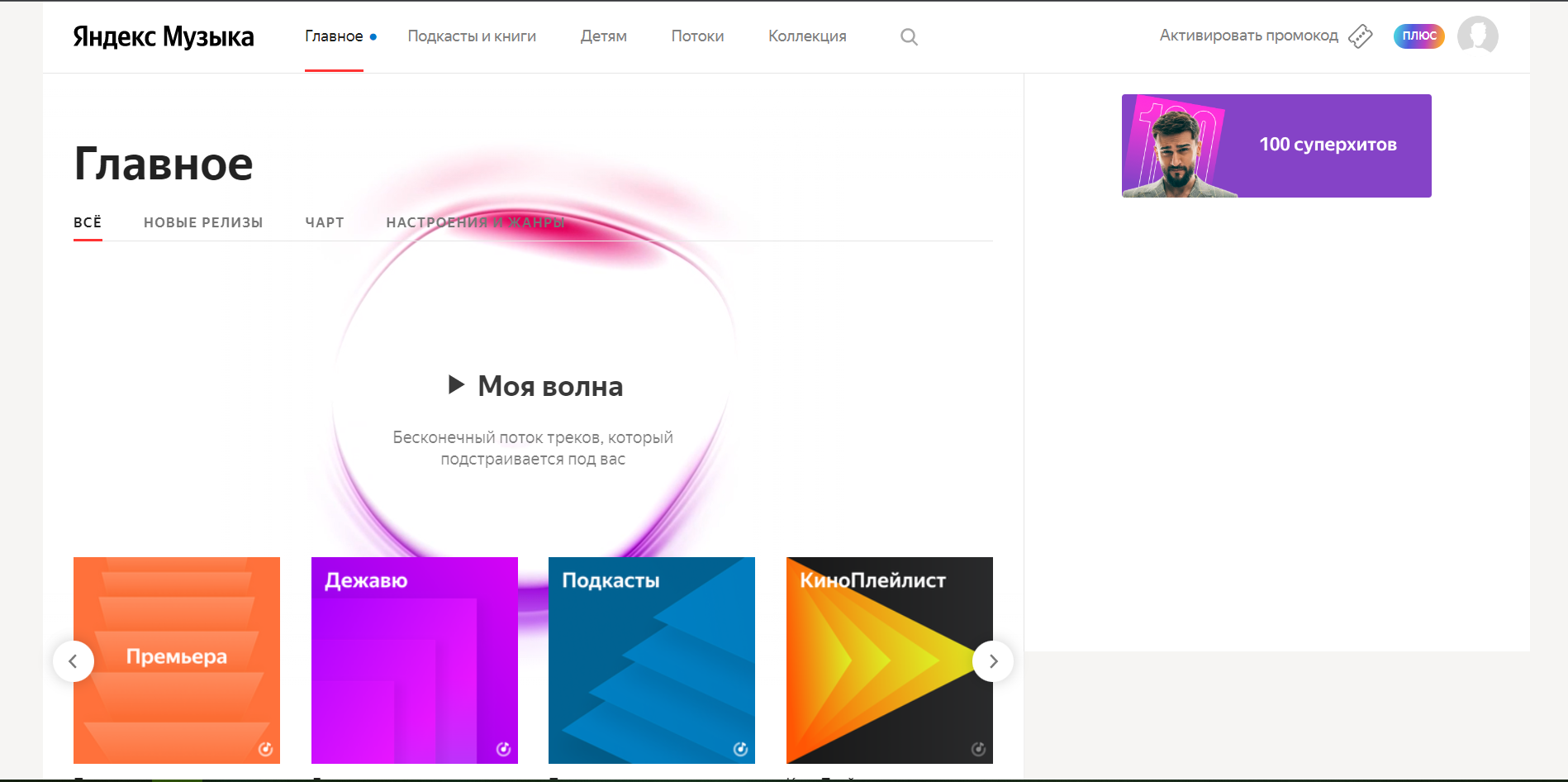


Рисунок 1.1 – Домашняя страница Яндекс Музыка

С домашней страницы в Яндекс Музыка пользователь может воспользоваться поиском, попасть в свои плейлисты, обратиться к недавно прослушанным плейлистам и открыть настойки своего аккаунта. Также он может воспользоваться сервисом «Моя волна», который подберёт коллекцию треков на основе ранее прослушанных пользователем треков.

«Зайцев.Нет» – сервис попроще, нежели Яндекс Музыка, зато полностью бесплатен и позволяет скачивать все имеющиеся композиции. Кроме музыки доступны подкасты, аудиокниги и радио – всё что нужно для требовательного слушателя.

В главном меню можно найти нужного исполнителя трек или сборник. В плейере любой песни есть специальная иконка стрелочки, направленной перпендикулярно лежащей снизу линии – нажав на неё можно скачать песню на свой компьютер. Также на сервисе есть регистрация и возможность поделиться сервисом в разнообразных социальных сетях. Интерфейс этого сервиса можно увидеть на рисунке 1.2.

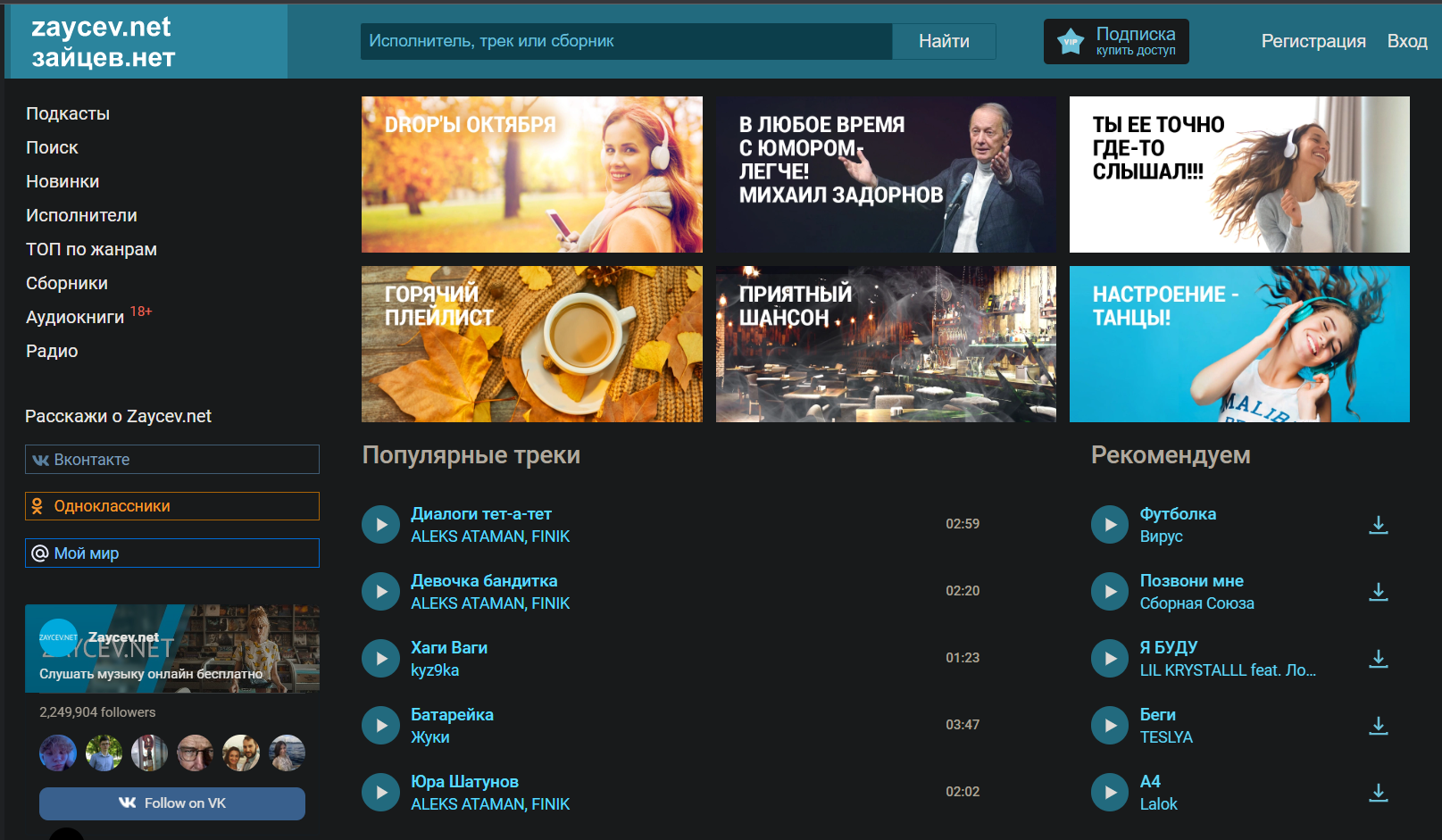


Рисунок 1.2 – Домашняя страница «Зайцев.Нет»

Разобрав два приведенных выше сервиса для стриминга музыки, можно сформировать общий функционал необходимый для работы приложения.

Прежде всего, естественно, база данных должна корректно работать с аудио файлами. Обязательно должен быть поиск песен по исполнителю и названию песни, также пользователь должен быть способен сохранять в личный плейлист понравившиеся ему песню и скачивать на свой компьютер. Пользователь должен иметь возможность оценивать музыкальную композицию, а также оставлять комментарии – таким образом авторы, смогут узнать мнение об их творчестве и качестве материала.

1. Разработка модели базы данных

Первым этапом курсового проекта будет создание логически взаимосвязанных таблиц. Чтобы составить визуальную взаимосвязанную структуру базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

При выборе модели базы данных я решил использовать реляционную модель. Реляционная модель данных — это способ описания и хранения данных, который используется в базах данных. Она описывает данные в виде таблиц (также называемых реляциями), в которых строки соответствуют объектам, а столбцы - характеристикам этих объектов. Также реляционная модель данных определяет способы взаимосвязи между таблицами с помощью ключей (первичные и внешние).

Одним из главных преимуществ реляционной модели данных является ее функциональность и возможность использования стандартных языков запросов (SQL) для работы с данными. Это позволяет легко управлять базой данных и производить различные операции над ней, такие как вставка, обновление и удаление.

Диаграмма базы данных, спроектированной в ходе разработки представлена в приложении А.

При разработке приложение получило рабочее название «Сервис по прослушиванию музыки», соответственно схема базы данных называется «Music Service». База данных, основанная на реляционной модели данных. В ее структуру входят следующие таблицы: USERS, ROLES, GENRES, AUTHORS, TRACK\_FILES, TRACKS, PLAYLISTS, RATING, PLAYLIST\_TRACKS. Таблицы хранят в себе данные необходимые для описания и представления треков пользователям. А также информацию о самих клиентах и администраторе. Ниже мы разберем каждую из них более подробно.

Разрабатывая модель базы данных, было взято во внимание соблюдений трёх первых нормальных форм, также были учтены вопросы производительности базы данных в рабочем процессе приложения или сайта, использующего её. Ещё одним важным вопрос является безопасность базы данных и разграничение доступа различных пользователей к данным – для этого были созданы различные пользователи базы данных, а доступ к таблицам был организован через представления и процедуры.

1. Разработка необходимых объектов

При разработке приложения для курсового проекта была использована база данных Oracle 21c.

* 1. Таблицы базы данных

Для реализации базы данных музыкальной площадки было разработано 9 таблиц.

В структуру схемы базы данных курсового проекта входят следующие таблицы: AUTHORS, GENRES, PLAYLIST\_TRACKS, PLAYLISTS, RATING, ROLES, TRACK\_FILES, TRACKS, USERS. Ниже мы разберем каждую из них более подробно.

Таблица ROLES представляет собой перечень ролей, которые могут иметь пользователи. Имеющиеся столбцы: ID (идентификатор роли, типа number (\*), первичный ключ) и NAME (текстовое название роли, тип varchar2(20)). Имеющиеся по умолчанию роли: user (обычный пользователь) и admin (администратор).

Таблица USERS содержит информацию о пользователях, а именно: ID (идентификатор пользователя, тип number (\*), первичный ключ), LOGIN (логин пользователя, типа varchar2(30)), PASSWORD (хеш пароля пользователя, тип varchar2(60)) и ROLE\_ID (идентификатор роли, тип number (\*), внешний ключ).

Таблица GENRES содержит информацию о музыкальных жанрах: ID (идентификатор музыкального жанра, тип number (\*), первичный ключ), NAME (имя музыкального жанра, тип varchar2(30)).

Таблица AUTHORS содержит информацию об авторах музыкальных треков: ID (идентификатор автора, тип number (\*), первичный ключ), NAME (имя автора, тип varchar2(30)).

Таблица TRACK\_FILES содержит музыкальные файлы в бинарном виде: ID (идентификатор трека, тип number (\*), первичный ключ), TRACK\_FILE (бинарный файл трека, тип blob).

Таблица TRACKS представляет собой песни, выпущенные авторами в определённом жанре, и потому содержит следующие столбцы: ID (идентификатор трека, тип number(\*), первичный ключ), AUTHOR\_ID(идентификатор автора, тип number(\*), внешний ключ), GENRE\_ID(идентификатор жанра, тип number(\*), внешний ключ)), TRACK\_FILE\_ID (идентификатор бинарного файла трека, тип number(\*), внешний ключ)) и NAME (название трека, тип varchar2(30),внешний ключ).

Таблица RATING хранит оценки пользователей трекам: ID (идентификатор трека, тип number (\*), первичный ключ), TRACK\_ID (идентификатор трека, которому поставили оценку, тип number (\*), внешний ключ), USER\_ID (идентификатор пользователя, который поставил оценку, тип number (\*), внешний ключ), RATE (цифровое значение оценки, тип number (10)).

Таблица PLAYLISTS содержит плейлисты, которые создают пользователи: ID (идентификатор трека, тип number (\*), первичный ключ), USER\_ID (идентификатор владельца плейлиста, тип number (\*), внешний ключ), NAME (имя плейлиста, varchar2(30), TRACK\_COUNT (кол-во треков в плейлисте, тип number (\*)).

Таблица PLAYLIST\_TRACKS нужна для связи между плейлистами и треками: ID (идентификатор пары плейлист-трек, тип number (\*), первичный ключ), TRACK\_ID (идентификатор трека, тип number (\*), внешний ключ), PLAYLIST\_ID (идентификатор трека, тип number (\*), внешний ключ).

* 1. Процедуры базы данных

Для управления данными через приложение пользователи и администраторы использует хранимые процедуры. Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

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

* выборка данных из таблиц,
* выборка данных из представлений со статистикой,
* выборка данных по поисковому запросу с пагинацией,
* заполнение таблиц 100 000 строк,
* добавление данных в таблицы,
* удаление данных из таблиц,
* изменение данных в таблицах,
* экспорт и импорт таблицы TRACKS в формате JSON.
  + 1. Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие процедуры: GET\_AUTHORS, GET\_GENRES, GET\_LOGINS, GET\_PLAYLIST\_BY\_ID, GET\_PLAYLISTS, GET\_TRACKS\_BY, GET\_TRACK\_FILE, GET\_USER\_BY\_ID, GET\_USER\_BY, GET\_USER\_BY\_LOGIN, GET\_USER\_ID\_BY\_LOGIN. Основная их задача – выборка данных из всех основных таблиц базы данных.

* + 1. Выборка данных по поисковому запросу c пагинацией

Для вывода данных из таблиц по поисковому запросу с использованием пагинации была написана процедура GET\_TRACKS\_FOR\_USER и GET\_TRACK\_COUNT. GET\_TRACK\_COUNT возвращает число треков согласно поисковым критериям (поле поиска, значение поля поиска, минимальная оценка пользователя, максимальная оценка пользователя), а GET\_TRACKS\_FOR\_USER возвращает треки, согласно тем же критериям.

При этом принимая дополнительные параметры в виде номера страницы вывода и кол-ва треков, которые должна содержать одна страница. GET\_TRACKS\_FOR\_USER представлена на листинге 2.1.

|  |
| --- |
| create or replace procedure *JSON\_TO\_TRACKS* as  fille utl\_file.file\_type;  line varchar2(32767); begin  fille := utl\_file.*fopen*('ORACLE\_BASE', 'import.json', 'r');  loop  begin  utl\_file.*GET\_LINE*(fille, line);  merge into TRACKS  using (select *json\_value*(line, '$.name') as name,  *json\_value*(line, '$.genre\_id') as GENRE\_ID,  *json\_value*(line, '$.author\_id') as AUTHOR\_ID,  *json\_value*(line, '$.track\_file\_id') as TRACK\_FILE\_ID  from dual) src  on (src.name = TRACKS.name)  when not matched then  insert (NAME, GENRE\_ID, AUTHOR\_ID,TRACK\_FILE\_ID)  values (src.name, src.GENRE\_ID, src.AUTHOR\_ID, src.TRACK\_FILE\_ID);  exception  when no\_data\_found then  exit;  end;  end loop;  utl\_file.*fclose*(fille); end; |

Листинг 3.1 – Процедура GET\_TRACKS\_FOR\_USER

* + 1. Заполнение таблиц 100 000 строк

Для заполнения таблиц было разработана процедура INSERT\_100000\_LINES, заполняющая таблицу TRACKS 100000 записями. Процедура представлена на листинге 3.2

|  |
| --- |
| create procedure *Insert\_100000\_lines* is BEGIN  FOR i IN 1 .. 10000  LOOP  insert into tracks (name, genre\_id, author\_id, track\_file\_id) VALUES ('Track- ' || i, 21, 21, 1);  END LOOP; END; |

Листинг 3.2 – Процедура Insert\_100000\_lines

* + 1. Выборка данных из представлений со статистикой

Сбор статистики для администратора одна из важнейших функций базы данных.

Так как статистика являет собой сложную совокупность данных из различных таблиц было принято решение создать множество представлений со статистикой. Для выборки из представлений, агрегирующим статистику данных основных таблиц были созданы следующие процедуры:

* GET\_TRACKS\_WITH\_LARGEST\_NUMBER\_OF\_RATING,
* GET\_TRACKS\_WITH\_LARGEST\_AVERAGE\_RATING,
* GET\_GENRES\_WITH\_LARGEST\_NUMBER\_OF\_TRACKS,
* GET\_GENRES\_WITH\_LARGEST\_NUMBER\_OF\_RATING,
* GET\_AUTHORS\_WITH\_LARGEST\_NUMBER\_OF\_TRACKS,
* GET\_AUTHORS\_WITH\_LARGEST\_NUMBER\_OF\_RATING,
* GET\_USERS\_WITH\_LARGEST\_NUMBER\_OF\_RATING.
  + 1. Удаление данных из таблиц

Для управления базой данных были созданы процедуры удаления: DELETE\_PLAYLIST, DELETE\_TRACK\_FROM\_PLAYLIST. Они были разработаны соответственно для удаления строк из таблиц: PLAYLISTS, PLAYLIST\_TRACKS.

* + 1. Добавление данных в таблицы

Для управления базой данных были созданы процедуры добавления новых строк в основные таблицы: ADD\_GENRE, ADD\_AUTHOR, ADD\_TRACK\_TO\_PLAYLIST, CREATE\_PLAYLIST, CREATE\_TRACK, CREATE\_TRACK\_FILE, SET\_TRACK\_RATING.

* + 1. Изменение данных в таблицах

Для изменения данных создана лишь одна процедура UPDATE\_TRACK\_RATING, обновляющая рейтинг песне, поставленный пользователем.

* 1. Представления базы данных

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, который определяет структуру и содержимое представления, в момент обращения к представлению. Оно не содержит физических данных, а лишь отображает информацию, полученную из одной или нескольких таблиц базы данных. Представление позволяет скрыть сложность запроса и предоставить пользователям более удобный интерфейс для работы с данными.

В нашей базе данных представления преимущественно представляют собой статистику данных основных таблиц доступных только пользователю администратору, а также существует представления музыкальных треков с авторами и жанрами этих треков. Для этого созданы следующие представления:

* USERS\_WITH\_LARGE\_NUMBER\_OF\_RATINGS,
* TRACKS\_WITH\_LARGE\_NUMBER\_OF\_RATINGS,
* TRACKS\_WITH\_LARGE\_AVERAGE\_RATING,
* GENRES\_WITH\_LARGE\_NUMBER\_OF\_TRACKS,
* GENRES\_WITH\_LARGE\_NUMBER\_OF\_RATINGS,
* AUTHORS\_WITH\_LARGE\_NUMBER\_OF\_TRACKS,
* AUTHORS\_WITH\_LARGE\_NUMBER\_OF\_RATINGS,
* TRACKS\_WITH\_GENRES\_AND\_AUTHORS
  + .

К примеру, представление TRACKS \_WITH \_LARGE \_NUMBER \_OF\_RATINGS являет собой выборку из 10 треков, у которых наибольшее кол-во оценок от пользователей. Представление показано на листинге 3.3

|  |
| --- |
| create view TRACKS\_WITH\_LARGE\_NUMBER\_OF\_RATINGS as select t1.ID as TRACK\_ID,t1.NAME as TRACK\_NAME,t2.NAME as GENRE\_NAME, t3.NAME as AUTHOR\_NAME, t1.RATE\_COUNT from (select TRACKS.ID, TRACKS.NAME, TRACKS.GENRE\_ID, TRACKS.AUTHOR\_ID, *count*(RATE) as RATE\_COUNT  from TRACKS  left join  RATING on TRACKS.ID = RATING.TRACK\_ID  group by TRACKS.ID, TRACKS.NAME, TRACKS.GENRE\_ID, TRACKS.AUTHOR\_ID  order by *count*(RATE) desc  fetch next 10 rows only) t1  join GENRES t2 on t1.GENRE\_ID = t2.ID  join AUTHORS t3 on t1.AUTHOR\_ID = t3.ID |

Листинг 3.3 – представление TRACKS\_WITH\_LARGE\_NUMBER\_OF\_RATINGS

* 1. Индексы базы данных

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска.

При проверке существования пользователя при авторизации, база данных должна быстро находить аккаунт по введённому логину, чтобы валидировать введённые данные неавторизованного пользователя. Для этого создан индекс USERS\_INDEX по полю NAME в таблице USERS.

Сам индекс представлен на листинге 3.4

|  |
| --- |
| create index USERS\_INDEX on USERS(LOGIN) |

Листинг 3.4 – индекс USERS\_INDEX

* 1. Триггеры базы данных

Триггер — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

В базе данных имеется таблица PLAYLISTS, представляющая плейлисты и PLAYLIST\_TRACKS, представляющая треки плейлистов. В случае, когда в определённый плейлист добавляется песня или происходит удаление действует триггер.

Триггер на удаление назван decrement\_track\_count – он представлен на листинге 3.5.

|  |
| --- |
| create or replace trigger increment\_track\_count after insert on PLAYLIST\_TRACKS for each row begin  update playlists set track\_count = track\_count + 1 where id = :new.playlist\_id; end; |

Листинг 3.5– триггер decrement\_track\_count

Триггер на добавление назван increment\_track\_count – он представлен на рисунке 3.6

|  |
| --- |
| create or replace trigger decrement\_track\_count after delete on PLAYLIST\_TRACKS for each row begin  update playlists set track\_count = track\_count - 1 where id = :old.playlist\_id; end; |

Листинг 3.6 – триггер decrement\_track\_count

* 1. Пользователи базы данных

В базе данных присутствуют три пользователя unauthorized\_user, admin и client. Каждый из них имеет свой набор доступных процедур для взаимодействия с бд.

Unauthorized\_user представлен собой неавторизованного пользователя, которому доступно лишь процедуры используемые при логине и регистрации, такие как поиск пользователя по идентификатору, идентификатора по логину, регистрация пользователя и получения списка всех логинов. Выданные процедуры пользователю unauthorized\_user можно увидеть на листинге 3.7

|  |
| --- |
| grant execute on *GET\_USER\_BY\_ID* to unauthorized\_user; grant execute on *GET\_USER\_ID\_BY\_LOGIN* to unauthorized\_user; grant execute on *REGISTER\_USER* to unauthorized\_user; grant execute on *GET\_LOGINS* to unauthorized\_user; |

Рисунок 3.7 – процедуры, выданные пользователю unauthorized\_user

Client представляет собой обычного пользователя, который может совершать процедуры, связанные с прослушиванием и оценкой треков и взаимодействием с плейлистами.

Выданные процедуры пользователю client можно увидеть на листинге 3.8

|  |
| --- |
| grant execute on *GET\_USER\_BY\_LOGIN* to client; grant execute on *GET\_PLAYLISTS* to client; grant execute on *GET\_PLAYLIST\_BY\_ID* to client; grant execute on *GET\_PLAYLIST\_TRACKS\_FOR\_USER* to client; grant execute on *ADD\_TRACK\_TO\_PLAYLIST* to client; grant execute on *DELETE\_TRACK\_FROM\_PLAYLIST* to client; grant execute on *CREATE\_PLAYLIST* to client; grant execute on *GET\_TRACKS\_FOR\_USER* to client; grant execute on *SET\_TRACK\_RATING* to client; grant execute on *UPDATE\_TRACK\_RATING* to client; grant execute on *GET\_TRACK\_FILE* to client; grant execute on *GET\_TRACK\_BY\_ID* to client; grant execute on *GET\_TRACK\_COUNT* to client; grant execute on *GET\_TRACK\_FILE* to client; grant execute on *GET\_RATING\_FOR\_TRACK\_FROM\_USER* to client; grant execute on *TRACKS\_TO\_JSON* to client; grant execute on *DELETE\_PLAYLIST* to client; |

Листинг 3.8 – процедуры, выданные пользователю client

Admin представляет собой администратор с процедурами связанными с добавлением треков, авторов и жанров, а также просмотром статистики. Выданные процедуры пользователю client можно увидеть на рисунке 3.9

|  |
| --- |
| grant execute on *CREATE\_TRACK\_FILE* to admin; grant execute on *CREATE\_TRACK* to admin; grant execute on *ADD\_AUTHOR* to admin; grant execute on *ADD\_GENRE* to admin; grant execute on *GET\_GENRES* to admin; grant execute on *GET\_AUTHORS* to admin; grant execute on *GET\_AUTHORS\_WITH\_LARGEST\_NUMBER\_OF\_RATING* to admin; grant execute on *GET\_AUTHORS\_WITH\_LARGEST\_NUMBER\_OF\_TRACKS* to admin; grant execute on *GET\_GENRES\_WITH\_LARGEST\_NUMBER\_OF\_RATING* to admin; grant execute on *GET\_USERS\_WITH\_LARGEST\_NUMBER\_OF\_RATING* to admin; |

Листинг 3.9 – процедуры, выданные пользователю admin

На этом описание необходимых объектов базы данных заканчивается.

1. Описание процедур импорта и экспорта

База данных имеет возможность экспортировать и импортировать данные для таблицы TRACKS.

Для экспорта таблицы TRACKSв формате JSON была разработана процедура TRACKS\_TO\_JSON. Для работы с файловой системой используется метод utl\_file, а для работы с JSON объект JSON\_OBJECT. Процедура представлена на листинге 4.1

|  |
| --- |
| create procedure *TRACKS\_TO\_JSON* as export\_file utl\_file.file\_type;  begin  export\_file := utl\_file.*fopen*('ORACLE\_BASE', 'export.json', 'w');  for c in (select *JSON\_OBJECT*('id' value id , 'name' value NAME, 'genre\_id' value GENRE\_ID , 'author\_id' value AUTHOR\_ID , 'track\_file\_id' value TRACK\_FILE\_ID ) as json  from TRACKS)  loop  utl\_file.*put\_line*(export\_file, c.json);  end loop;  utl\_file.*fclose*(export\_file); end; |

Листинг 4.1–процедура TRACKS\_TO\_JSON

Для импорта данных в таблицу TRACKS, из файла формата JSON, была разработана процедура JSON\_TO\_TRACKS.Процедура JSON\_TO\_TRACKS представлена ниже на листинге 4.2

|  |
| --- |
| create procedure *JSON\_TO\_TRACKS* as  fille utl\_file.file\_type;  line varchar2(32767); begin  fille := utl\_file.*fopen*('ORACLE\_BASE', 'import.json', 'r');  loop  begin  utl\_file.*GET\_LINE*(fille, line);  using (select *json\_value*(line, '$.name') as name,  *json\_value*(line, '$.genre\_id') as GENRE\_ID,  *json\_value*(line, '$.author\_id') as AUTHOR\_ID,  *json\_value*(line, '$.track\_file\_id') as TRACK\_FILE\_ID  from dual) src  when not matched then  insert (NAME, GENRE\_ID, AUTHOR\_ID,TRACK\_FILE\_ID)  values (src.name, src.GENRE\_ID, src.AUTHOR\_ID, src.TRACK\_FILE\_ID);  commit;  end;  end loop;  utl\_file.*fclose*(fille); end; |

Листинг 4.2 – процедура JSON\_TO\_TRACKS

Таким образом происходит импорт и экспорт в базе данных.

1. Тестирование производительности

Производительность базы данных — это способность базы данных обрабатывать запросы и транзакции быстро и эффективно. Высокая производительность базы данных может быть достигнута с помощью эффективного использования индексов, хорошей организации структуры базы данных и оптимизации запросов. Хорошая производительность базы данных важна для обеспечения эффективной работы системы и удовлетворения потребностей пользователей.

Для тестирования производительности была взята процедура GET\_TRACKS\_FOR\_USER, которая запрашивает нужное количество треков для определённого юзера с использованием пагинации. Процедура была вызвана со следующими параметрами: для пользователя с идентификатором 37, запрашивалось 500 треков, с пропуском 1000 первых найденных треков, которые искались по столбцу NAME, который должен был содержать значение TRACK, оценка трека могла варьироваться от 0 до 10. Учитывая, что в рамках приложения запрос идёт всего на 10 треков, подобные условия покажут производительность в крайне радикальных условиях. Условия запроса представлены на листинге 5.1

|  |
| --- |
| USER\_ID\_ number := 37; TRACKS\_TO\_SKIP\_ number := 1000; TRACKS\_TO\_FETCH\_ number := 500; SEARCH\_BY\_ varchar2(4000) := 'name'; SEARCH\_VALUE\_ varchar2(4000) := 'Track'; ORDER\_ varchar2(4000) := 'descending'; MIN\_RATE\_ number := 0; MAX\_RATE\_ number := 10; TRACKS\_ sys\_refcursor; |

Листинг 5.1 – условия запроса к процедуре GET\_TRACKS\_FOR\_USER

И результат выполнения запроса приходит всего за 182 ms, где 162 ms уходит на получение данных и 20 на исполнение процедуры. Результаты наглядно видны на рисунке 5.1



Рисунок 5.1 – результаты тестирование процедуры GET\_TRACKS\_FOR\_USER

Исходя из полученных данных можно сделать вывод, что тест производительности пройден.

1. Описание технологии

В базе данных реализовано хранение мультимедийных данных. В таблице TRACK\_FILES в столбце TRACK\_FILE хранится бинарная строка музыкального файла в формате BLOB. Фрагмент данных таблицы можно увидеть на рисунке 6.1

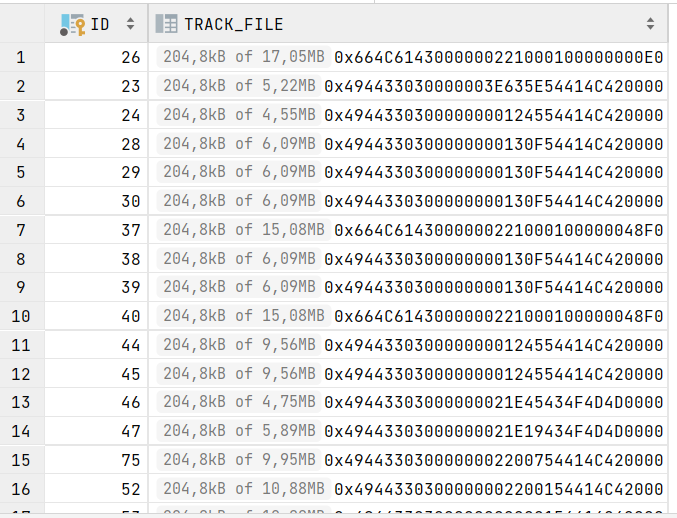


Рисунок 6.1 - Фрагмент данных таблицы TRACK\_FILES

Когда пользователь передаёт на сервер JAVA музыкальный файл, он превращается в бинарный файл и передаёт репозиторию, работающему с бд, где он и сохраняется. Фрагмент кода обработки файла можно увидеть на листинге 6.1

|  |
| --- |
| public Long uploadTrackFile(MultipartFile file) throws IOException, SQLException  {  var fileBytes = file.getBytes();  Blob blob = new SerialBlob(fileBytes);  return trackRepository.uploadTrackFile(fileBytes); } |

Листинг 6.1 – Фрагмент кода обработки музыкального файла в бинарный файл

Таким образом происходит изменение формата файла в бинарный формат.

Когда же бинарный файл запрашивается из базы, сервер обратно передаёт бинарный файл в потоке бинарного массива с указанием, что файл является музыкальным в формате mp3. Фрагмент этого кода можно увидеть на рисунке 6.2

|  |
| --- |
| Blob trackBlob = trackService.getTrackFileById(id); var trackFile = trackBlob.getBytes(1, (int) trackBlob.length()); response.setContentType("audio/mpeg"); response.setContentLength(trackFile.length); response.setHeader("Content-Disposition", "attachment; filename=" + trackService.getTrackFileNameById(id) + ".mp3"); InputStream inputStream = new ByteArrayInputStream(trackFile); IOUtils.*copy*(inputStream, response.getOutputStream()); response.flushBuffer(); |

Рисунок 6.2 - Фрагмент кода отправки бинарного файла в байтовом потоке пользователю

Таким образом реализуется хранение музыкальных файлов в базе данных. Стоит отметить, что в ходе разработки базы данных использование этой технологии являлась экспериментом, так как хорошо известно из опыта разработчиков баз данных, что хранить мультимедийные файлы лучше в файловой системе, так как при обращении, например, с музыкальными файлами, резко повышается кол-во данных, передаваемых в запросе и скорость этих самых запросов. Также обычно имеют ограничения на размер хранимых данных. Это может ограничивать количество мультимедиа, которое может быть храниться в базе данных. Даже использование бинарных форматов данных не сильно меняют эту ситуацию. Бинарные файлы, такие как изображения и видео, обычно имеют большой размер. Это может привести к тому, что база данных будет занимать большое количество места на жестком диске и потреблять больше ресурсов при бэкапах. Хоть для маленького приложения, разработанного в рамках курсового проекта такое решение выглядит неплохо, в случае расширения использование базы данных от такого подхода придётся отказаться. Возможно, в будущем в случае изобретение более эффективных по памяти способов хранения данных вопросы скорости и размеров отойдут на второй план, а на первый выйдет логическая структура хранения и тогда хранение мультимедийных файлов в базе данных станет неотъемлемой частью любого приложения.

1. Обоснование технических приемов программирования

В качестве клиента выступает веб-приложение, написанное на библиотеке React.js, позволяющей создать динамические одностраничные приложения.

Для разработки сервера приложения используются язык Java с фреймворком Spring, для работы с базой данных используется технология Spring JDBC и Oracle JDBC. Для каждого вида пользователя БД создаётся отдельный источник данных, они представлены на листинге 7.1:

|  |
| --- |
| static {  *config*.setJdbcUrl("jdbc:oracle:thin:@//localhost:1521/xe");  *config*.setUsername("ADMIN");  *config*.setPassword("admin");  *config*.addDataSourceProperty("cachePrepStmts", "true");  *config*.addDataSourceProperty("prepStmtCacheSize", "250");  *config*.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");  *config*.setMaximumPoolSize(125);  *ds* = new HikariDataSource(*config*); } static {  *config*.setJdbcUrl("jdbc:oracle:thin:@//localhost:1521/xe");  *config*.setUsername("CLIENT");  *config*.setPassword("client");  *config*.addDataSourceProperty("cachePrepStmts", "true");  *config*.addDataSourceProperty("prepStmtCacheSize", "250");  *config*.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");  *config*.setMaximumPoolSize(125);  *ds* = new HikariDataSource(*config*); } static {  *config*.setJdbcUrl("jdbc:oracle:thin:@//localhost:1521/xe");  *config*.setUsername("UNAUTHORIZED\_USER");  *config*.setPassword("qwerty");  *config*.addDataSourceProperty("cachePrepStmts", "true");  *config*.addDataSourceProperty("prepStmtCacheSize", "250");  *config*.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");  *config*.setMaximumPoolSize(25);  *ds* = new HikariDataSource(*config*); } |

Листинг 7.1 – источники данных

Сервер Java-Spring реализует слоистую архитектуру приложения “controller-service-repository”. Архитектура "controller-service-repository" представляет собой разделение приложения на три слоя: контроллер, сервис и репозиторий. Контроллер служит входной точкой для всех запросов и отвечает за обработку запросов, в том числе проверку входных данных и управление потоком работы. Сервис служит бизнес-логикой приложения и обеспечивает реализацию функциональности, которая не связана с внутренним управлением приложения. Репозиторий отвечает за работу с базой данных и предоставляет интерфейс для хранения и извлечения данных. Эта архитектура упрощает тестирование, улучшает разбиение на работы и повышает степень реализуемости функциональности.

На листинге 7.2 приведён пример запроса с использованием jdbc и источником данных администратора:

|  |
| --- |
| public Genre saveGenre(String name) throws SQLException {   Connection adminConnection = AdminDataSource.*getConnection*();   java.sql.CallableStatement prepareCall = adminConnection.prepareCall("{call KURSACH\_ADMIN.ADD\_GENRE(?,?)}");   prepareCall.setString(1, name);  prepareCall.registerOutParameter(2, REF\_CURSOR);  prepareCall.execute();   java.sql.ResultSet rs = (java.sql.ResultSet) prepareCall.getObject(2);   var genre = new Genre();  rs.next();   genre.setId(rs.getLong("id"));  genre.setName(rs.getString("name"));   adminConnection.close();  prepareCall.close();  rs.close();    return genre; } |

Листинг 7.2 – запрос с использованием jdbc и источника данных администратора

Получив данные, сервер обрабатывает их и передаёт клиенту. Использование JDBC позволяет создавать защищённые и быстрые запросы к базе данных, Spring-сервер воспроизводит всю бизнес-логику приложения, а клиент в React-приложении просит данные с помощью запросов, таким образом модель данных, контроллер данных и представление данных сепарируются друг от друга. Подобная архитектура обосновывает выбор описанных выше технологий. Пользователь получает быстрое, красивое и отзывчивое приложение, сервер приложения оказывается готов к расширению функционала в количественном и качественном векторе, а база данных получает мощный и мобильный способ интеракции с сервером.

1. Руководство пользователя

Начальный экран приложения представляет себе регистрацию нового пользователя. Необходимо указать логин, пароль и почтовый адрес. Страница регистрации представлена на рисунке 8.1

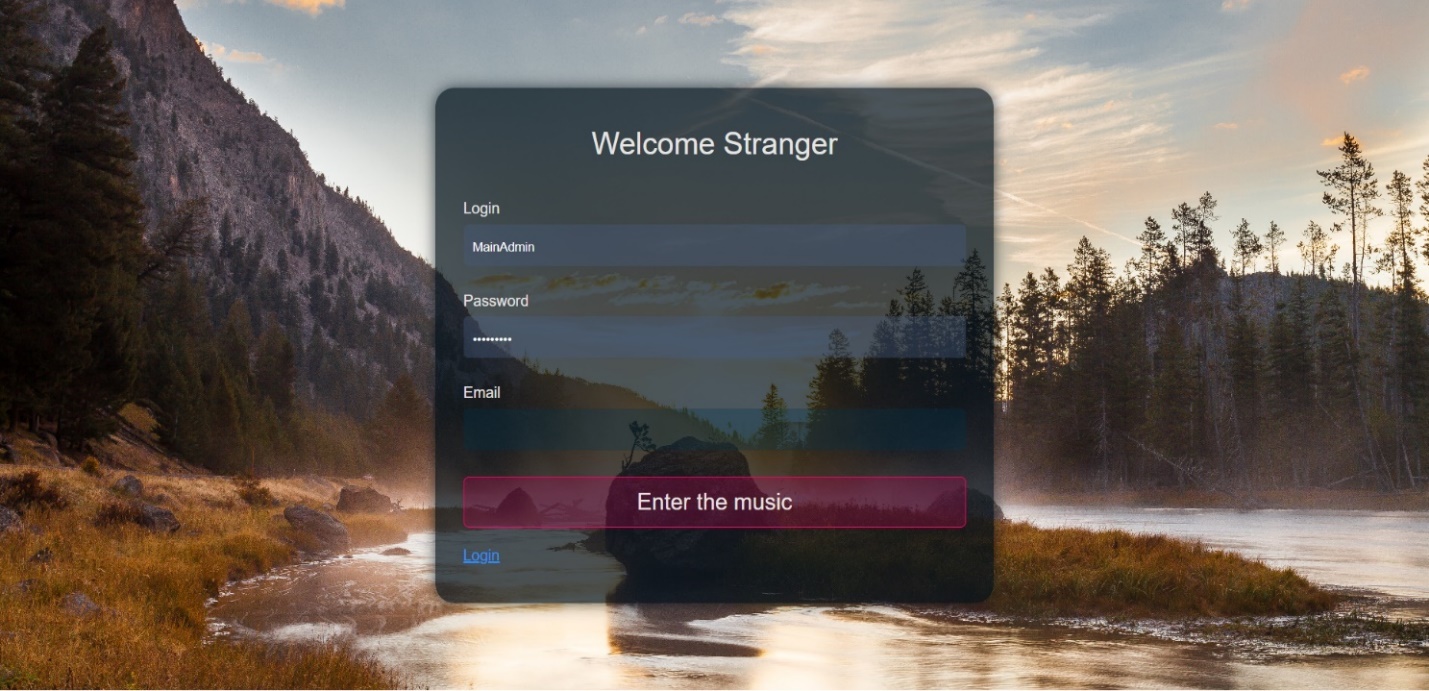


Рисунок 8.1 – Страница регистрации

Если пользователь уже зарегистрирован, он может зайти на страницу логина, указать свой логин и пароль и зайти в приложение. Страница регистрации представлена на рисунке 8.2:

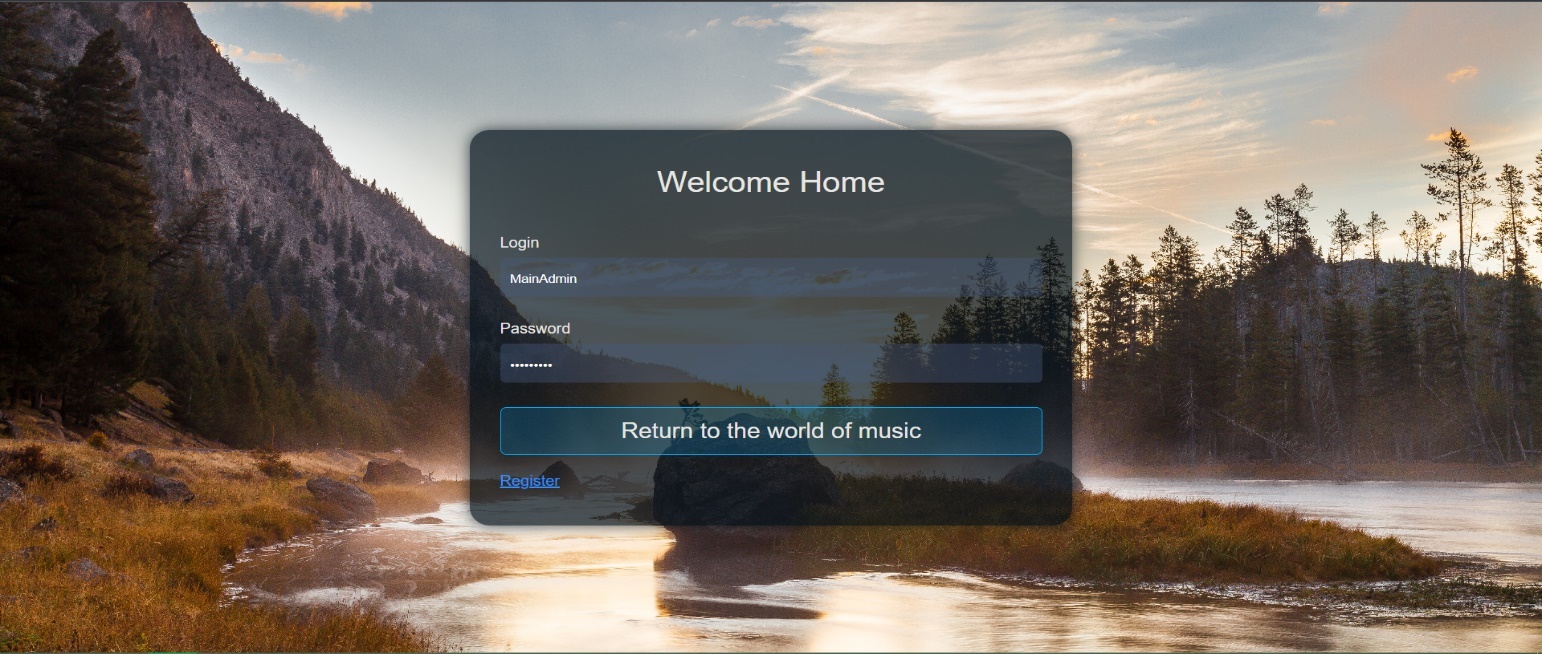


Рисунок 8.2 – страница логина

В случае если в приложение заходит юзер он оказывается на странице поиска треков. Здесь представлен поиск и фильтрация треков, сами треки выводятся в количестве 10 на страницу, а при нажатии на стрелки навигации можно выводить новые страницы треков. Трек можно прослушать и поставить ему отметку. Страница поиска представлена на рисунке 8.3

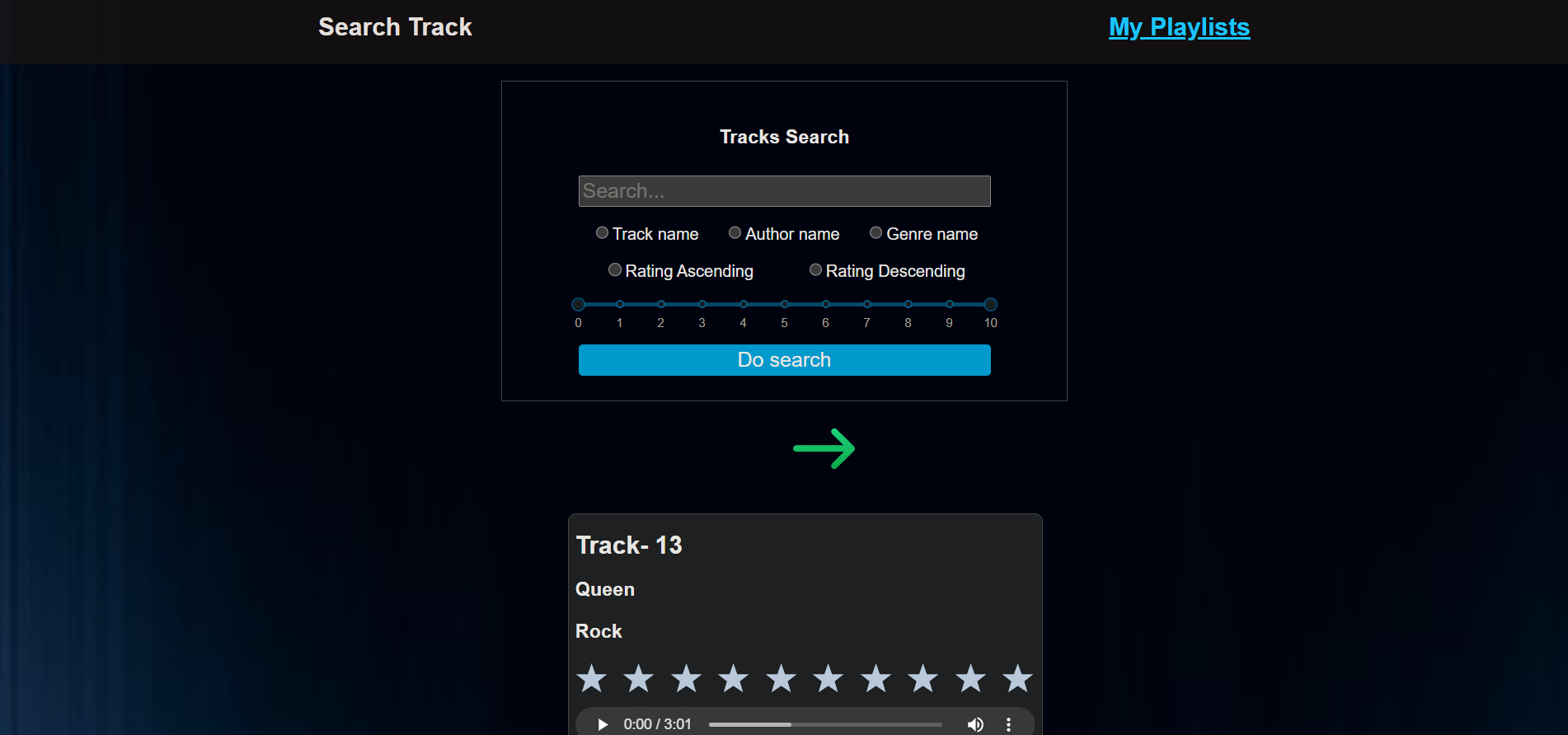


Рисунок 8.3 – страница поиска

Если нажать на ссылку My Playlists, можно перейти на страницу плейлистов пользователя. В правом столбце представлены плейлисты юзера, кол-во треков в каждом. Можно добавить плейлист или удалить его. При выборе плейлиста по центру показываются все треки этого плейлиста. Здесь любой трек можно удалить. Слева представлен тот же поиск, что и на предыдущей страницы – нужен он для поиска и добавления треков в плейлист. Страница плейлистов представлена на рисунке 8.4:

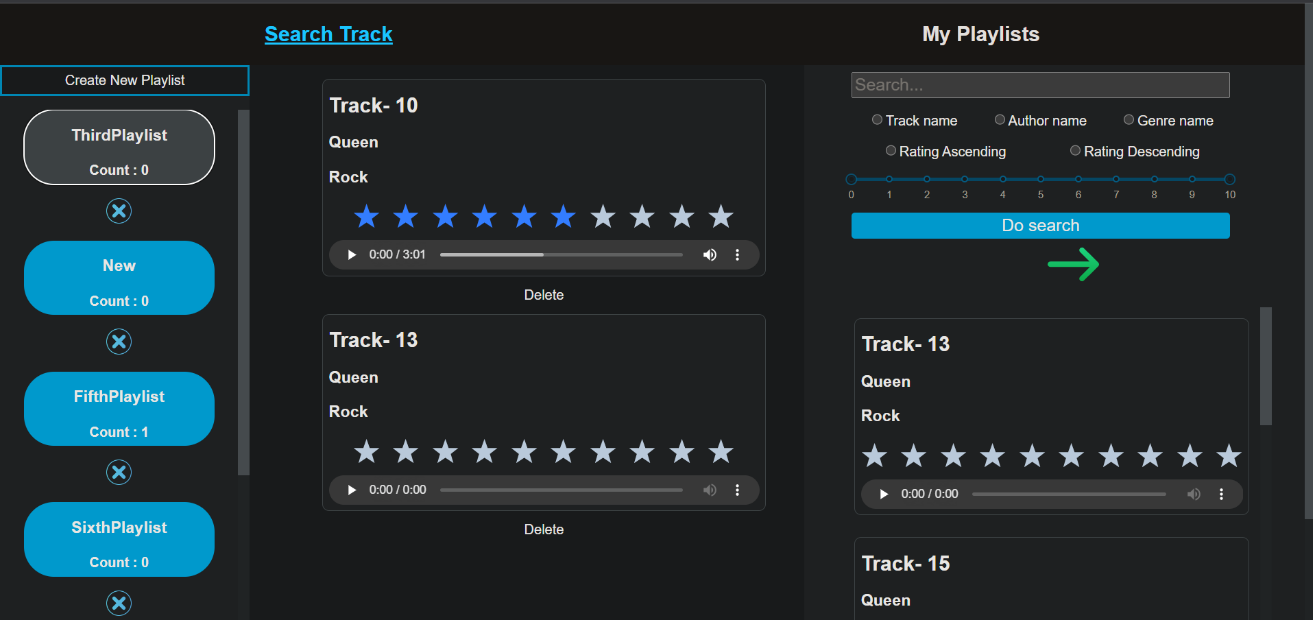


Рисунок 8.4 – страница плейлистов

В случае, если в приложение зайдёт админ, перед ним откроется страница добавления трека. Здесь он может выбрать музыкальный файл, создать или выбрать готовые Жанры и Авторов музыки и дать название треку. Страница добавления представлена на рисунке 8.5:

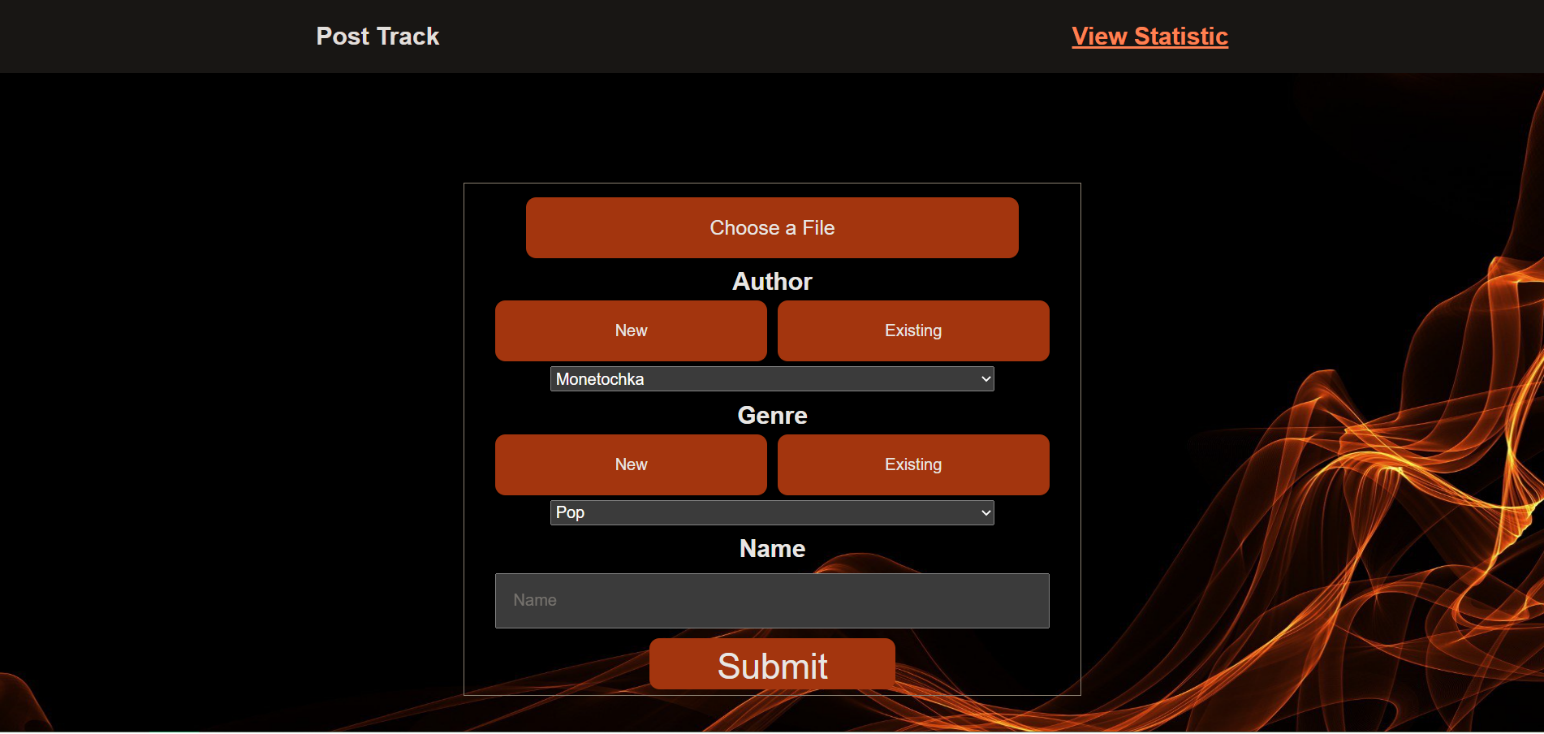


Рисунок 8.5 – страница создания треков

Второй страницей админа является страница статистики. Здесь админ может посмотреть разнообразную статистику по данным приложения. Страница статистики представлена на рисунке 8.6:

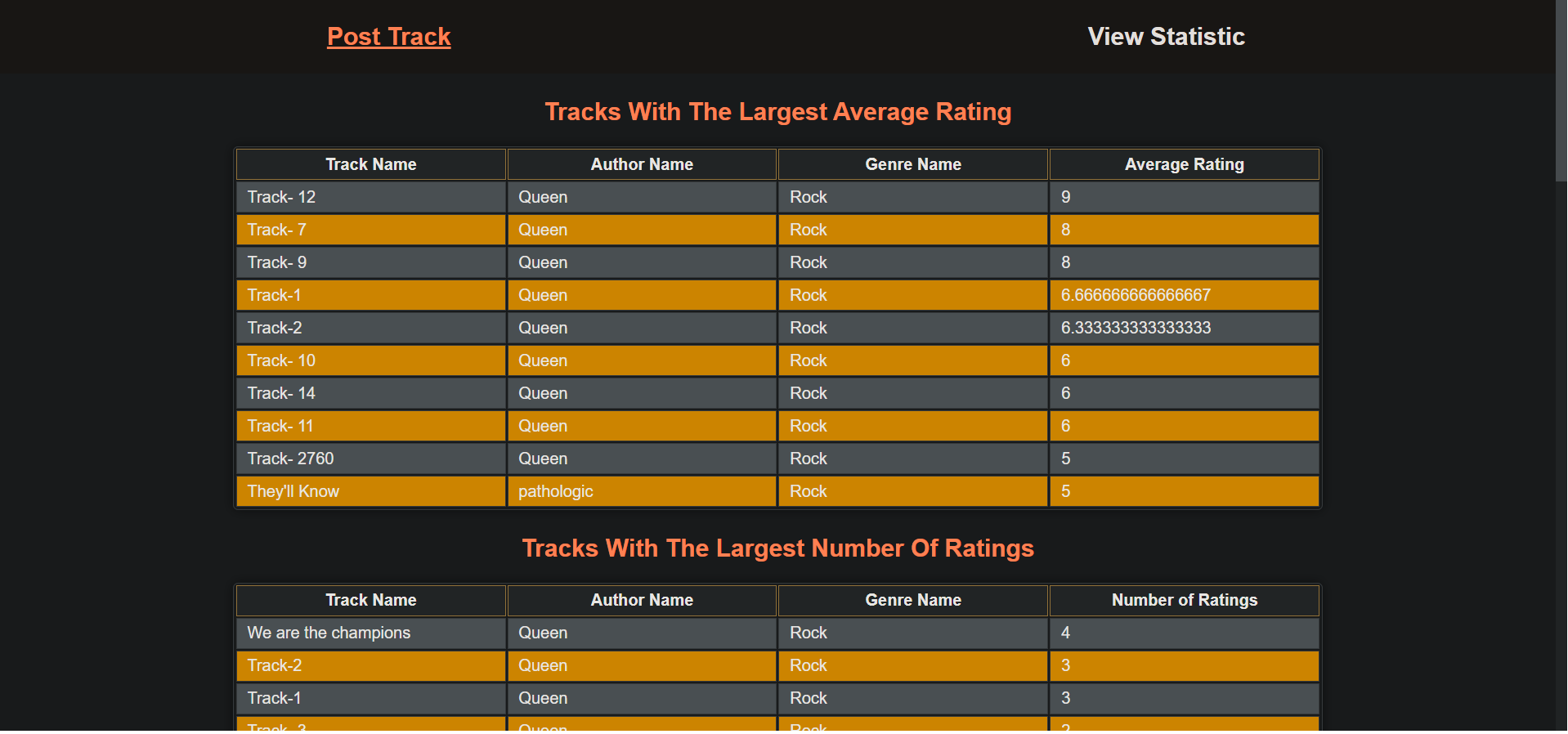


Рисунок 8.6 – страница статистики

На этом все функции сайта по прослушиванию музыки заканчиваются.

Заключение

Базы данных обеспечивают надежное хранение информации, в структурированном виде и своевременный доступ к ней. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных. Базы данных используются для хранения различных типов данных, таких как структурированные данные, текстовые файлы, изображения и аудио- и видеофайлы. Они позволяют эффективно обрабатывать и обновлять данные, что важно для бизнес-процессов, таких как обработка заказов, учет рабочего времени сотрудников и управление складами. Базы данных также могут использоваться для сбора и анализа статистических данных, что может быть полезно для принятия решений в бизнесе.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных сервиса по прослушиванию музыки. В данной работе использовалось СУБД Oracle21c.При разработке курсового проекта использовались объекты: таблицы, хранимые процедуры, триггеры, представления, индексы.

Кроме того, можно отметить, что база данных обладает хорошей надежностью и быстродействием, что важно для обеспечения эффективной работы организации. Благодаря удобному интерфейсу и гибкой системе управления, база данных оказывается простой в использовании для пользователей. Также стоит отметить, что база данных была разработана с учетом безопасности данных, что важно для защиты конфиденциальной информации.

Были созданы различные пользователи базы данных для разграничения доступа, запросы к информации исполнялись с помощью процедур.

Основной целью курсового проекта стало проектирование базы данных, которая обеспечит надежное хранение информации, в структурированном виде и своевременным доступам к ней.

БД прошла тестирование при использовании большого количества данных. Также были реализованы процедуры для импорта, экспорта данных в формат JSON.

Была реализована технология хранения мультимедийных данных.

Был создан сервер для работы с базой данных на Java Spring и клиентское приложения на библиотеке React.

В соответствии с полученным результатом работы можно сделать вывод, что разработанная база данных работает корректно, а требования технического задания выполнены в полном объёме.

Список литературы

1. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com.

2. Oracle-dba.ru [Электронный ресурс] / Режим доступа: https://oracle-dba.ru.

3. Developing and Using Stored Procedures [Электронный ресурс] /Режим доступа: <https://docs.oracle.com/cd/B2835901/appdev.111/b28843/tdddgprocedures.htm>

4. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 08.12.2021

5.Официальный сайт Oracle [Электронный ресурс] / Foundation, Inc. https://www.oracle.com/database/database-vault/index.html– Дата доступа: 10.11.2021.

6.Продукты Oracle [Электронный ресурс] [Электронный ресурс] / Foundation, Inc. http://www.interface.ru/home.asp?artId=24678 – Дата доступа: 21.11.2021.

Приложение А

