|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_ ***ИУК «Информатика и управление»\_\_\_\_\_\_\_\_\_***

**КАФЕДРА** \_\_ ***ИУК5 «Системы обработки информации»***

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе на тему:**

***Веб-приложение для прослушивания музыкальных композиций.***

по дисциплине ***Базы данных***

Студент гр. ИУК5-52Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Панченко А.П. )

(подпись) (Ф.И.О.)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Кириллов В.Ю. )

(подпись) (Ф.И.О.)

Оценка руководителя \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка защиты \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка проекта \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка по пятибалльной шкале)

Комиссия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

Калуга, 2021

Калужский филиал   
федерального государственного бюджетного образовательного учреждения высшего образования

***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»   
(КФ МГТУ им. Н.Э. Баумана)***

УТВЕРЖДАЮ

Заведующий кафедрой **\_\_ИУК5\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_(Е.В. Вершинин)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине ***Базы данных***

Студент Панченко А.П. ИУК5-52Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, инициалы, индекс группы)

Руководитель Кириллов В.Ю.

(фамилия, инициалы)

График выполнения проекта: 25% к\_4\_нед., 50% к\_7\_нед., 75% к\_10\_нед., 100% к\_14\_нед.

***1. Тема курсового проекта***

***Веб-приложение для прослушивания музыкальных композиций***

***2. Техническое задание***

*Разработать веб-приложение для прослушивания музыкальных композиций использованием базы данных.* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

***3. Оформление курсового проекта***

3.1. Расчетно-пояснительная записка на\_\_\_\_\_\_\_ листах формата А4.

3.2. Перечень графического материала КП (плакаты, схемы, чертежи и т.п.)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021г.

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись) (Ф.И.О.)

Задание получил\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021г.

(подпись) (Ф.И.О.)

Примечание:

Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc93542085)

[1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 5](#_Toc93542086)

[1.1. Общие сведения 5](#_Toc93542087)

[1.1.1. Полное наименование системы 5](#_Toc93542088)

[1.1.2. Плановые сроки начала и окончания работы по созданию системы 5](#_Toc93542089)

[1.2. Назначение и цели создания (развития) системы 5](#_Toc93542090)

[1.2.1. Назначение системы 5](#_Toc93542091)

[1.2.2. Цели создания системы 5](#_Toc93542092)

[1.3. Характеристики объекта автоматизации 5](#_Toc93542093)

[1.4. Требования к системе 6](#_Toc93542094)

[1.4.1. Требования к системе в целом 6](#_Toc93542095)

[1.4.2. Требования к структуре и функционированию системы 6](#_Toc93542096)

[1.4.3. Требования к надежности 6](#_Toc93542097)

[1.5. Состав и содержание работ по созданию системы 7](#_Toc93542098)

[1.6. Порядок контроля и приемки системы 8](#_Toc93542099)

[1.6.1. Состав, объем и методы испытаний системы и ее составных частей 8](#_Toc93542100)

[1.6.2. Общие требования к приемке работ 8](#_Toc93542101)

[1.7. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие 9](#_Toc93542102)

[1.8. Требования к документированию 9](#_Toc93542103)

[1.9. Источники разработки 9](#_Toc93542104)

[2. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 10](#_Toc93542105)

[2.1. Постановка задачи проектирования 10](#_Toc93542106)

[2.2. Описание предметной области 11](#_Toc93542107)

[2.3. Анализ аналогов 12](#_Toc93542108)

[2.4. Перечень задач, подлежащих решению в процессе разработки. 15](#_Toc93542109)

[2.5. Обоснование выбора инструментов и платформы для разработки 16](#_Toc93542110)

[3. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ 19](#_Toc93542111)

[3.1. Разработка алгоритмов обработки информации 19](#_Toc93542112)

[3.2. Логическая схема базы данных 19](#_Toc93542113)

[3.3. Разработка архитектуры приложения 20](#_Toc93542114)

[3.4. Реализация функционала приложения 20](#_Toc93542115)

[4. ПРОЕТНО-ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ 23](#_Toc93542116)

[4.1. Проектирование начального и тестового наполнения базы данных. 23](#_Toc93542117)

[4.2. Технологические решения, поддерживающие эксплуатационный цикл программы. 23](#_Toc93542118)

[4.3. Порядок развертывания системы. 24](#_Toc93542119)

[4.4. Разработка руководства пользователя и руководства администратора. 24](#_Toc93542120)

[ЗАКЛЮЧЕНИЕ 31](#_Toc93542121)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 32](#_Toc93542122)

ВВЕДЕНИЕ

Главными преимуществами стриминговых музыкальных сервисов считаются небольшая стоимость использования и универсальность. Кроме того, пользователи прислушиваются к музыкальным рекомендациям по стилям и жанрам и слушают плейлисты, созданные при помощи анализа их предпочтений. Размещение музыки на стриминговых сервисах выгодно и музыкантам. Кроме лицензионного вознаграждения(роялти) за прослушивания, они получают новых слушателей.

Стриминговые сервисы работают по принципу передачи контента от провайдера к пользователю. Весь контент уже загружен на стороннем сервере, конечному пользователю не требуется ничего скачивать для просмотра или воспроизведения. Контент транслируется в режиме реального времени, скорость загрузки напрямую зависит от скорости интернета пользователя. С нынешним даже самым простейшим интернетом можно без проблем прослушать музыку. Онлайн контент стал отличной заменой скачиванию файлов.

Все показатели указывают на то, что рост музыкальной индустрии в мире продолжится. И ключевую роль в этом будет играть стриминг – пока самый выгодный легальный способ слушать музыку.

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ
   1. Общие сведения
      1. Полное наименование системы

Веб-приложение для поиска и прослушивания музыкальных композиций.

* + 1. Плановые сроки начала и окончания работы по созданию системы

Начало работы: 01.09.2021

Окончание работы: 01.12.2021

* 1. Назначение и цели создания (развития) системы
     1. Назначение системы

Назначением системы является автоматизация поиска и прослушивания музыкальных композиций с помощью веб-приложения с использованием базы данных.

* + 1. Цели создания системы

Цель создания системы – получить прибыль за счет предоставления платных подписок на музыкальные композиции.

* 1. Характеристики объекта автоматизации

Объектом автоматизации является музыкальный плеер. Автоматизированию подлежит демонстрация, поиск и прослушивание имеющихся музыкальных альбомов и композиций.

* 1. Требования к системе
     1. Требования к системе в целом

Программный продукт должен представлять собой веб-приложение.

* + 1. Требования к структуре и функционированию системы

Приложение должно быть разделено на 3 слоя:

- Слой клиентского приложения в виде веб-страницы;

- Слой бизнес-логики приложения (сервер);

- Слой хранения данных (база данных).

Функционал приложения должен содержать:

* Авторизацию пользователей;
* Поиск музыкальных композиций и альбомов;
* Хранение пользовательских плейлистов и списков альбомов;
* Воспроизведение музыкальных композиций;
* Создание недельных чартов, основанных на количестве прослушиваний музыкальных композиций за текущую неделю;
* Хранение недельных чартов предыдущих недель.
  + 1. Требования к надежности

Система должна обеспечивать корректную обработку исключительных ситуаций, вызванных, например, вводом неверных пользовательских данных или неоднозначной информации в базу данных.

* 1. Состав и содержание работ по созданию системы

|  |  |  |
| --- | --- | --- |
| Стадии | Этапы работ | Сроки исполнения |
| 1. Формирование требований к АС | 1.1. Исследование объекта и подтверждение необходимости создания АС.  1.2. Формирование требований пользователя к АС.  1.3. Оформление отчёта о  выполненной работе и заявки на разработку АС. |  |
| 2. Разработка концепции АС. | 2.1. Изучение выбранного объекта.  2.2. Проведение научно- исследовательских работ.  2.3. Проектирования концепции АС, удовлетворяющей потребности пользователя.  2.4. Оформление отчёта о выполненной работе. |  |
| 3. Техническое задание. | Разработка и утверждение технического задания на создание АС. |  |
| 4. Разработка системы. | Разработка системы согласно  техническому заданию. |  |
| 5. Защита курсовой работы. | 5.1. Создание грамотной презентации и речи для защиты курсовой работы.  . Защита курсовой работы. |  |

* 1. Порядок контроля и приемки системы
     1. Состав, объем и методы испытаний системы и ее составных частей

Приложение должно пройти предварительные испытания, состоящие из отладки и минимального набора тестов.

В результате предварительных испытаний, должны быть исправлены недочеты, замечания на которые были получены в ходе предварительных испытаний.

Для проверки корректной работы внесённых изменений должны быть проведены повторные испытания разработанной системы.

* + 1. Общие требования к приемке работ

В процессе приемки работ должна быть осуществлена проверка на соответствие требованиям настоящего «Технического задания». По результатам испытаний возможны доработки и исправления.

* 1. Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие

При подготовке объекта автоматизации к вводу системы в действие следует выполнить:

* Обеспечение работ по вводу данных в систему;
* Развертывание системы на сервере;
* Настройка системы доступа и создание учетных записей.
  1. Требования к документированию

По окончанию работы предъявлена расчетно-пояснительная записка, в состав которой входят:

* Техническое задание;
* Научно-исследовательская часть;
* Проектно-конструкторская часть;
* Проектно-технологическая часть.

Также должна быть предоставлена графическая часть работы, выполненная в формате А1 на 2 листах, в которую входят:

* Демонстрационные чертежи;
* Алгоритмические схемы.
  1. Источники разработки
* Гост 34.601-90;
* Гост 34.602-89.

1. ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ
   1. Постановка задачи проектирования

Задачей данного проекта является разработка и проектирование веб-приложения, отвечающего следующим требованиям:

* Возможность отслеживания новых альбомов и музыкальных композиций;
* Возможность поиска интересующих композиций по авторам, названиям альбомов и композиций;
* Возможность отслеживания недельных чартов текущей недели и предыдущих недель;
* Прослушивание музыкальных композиций.

База данных будет содержать информацию о пользователях, альбомах, исполнителях, песнях и недельных чартах.

Макет главной страницы представлен на Рисунке 2.1.



Рисунок 2.1 – Макет главной страницы

* 1. Описание предметной области

В данном приложении существуют песни, альбомы, исполнители и недельные чарты.

* У каждого альбома есть обложка, название, список песен, список исполнителей, список пользователей, которые добавили альбом в свою фонотеку и дата релиза;
* Каждая песня содержит обложку, название песни, свой альбом, список исполнителей, файл песни, список пользователей, которые добавили песню в свою фонотеку, дату релиза, количество прослушиваний(всего) и количество прослушиваний в рамках недельного чарта;
* Каждый исполнитель содержит фотографию, псевдоним и списки выпущенных альбомов и песен;
* Недельные чарты представляют собой дату проведения чарта и список песен, вошедших в каждый из них с позиционированием.

Благодаря внедрению панели администратора в систему облегчается взаимодействие с данными.

Описание и функционал со стороны администратора:

Администратор имеет доступ ко всем данным системы, а также может с легкостью добавлять, изменять, удалять их.

Описание и функционал со стороны пользователя:

* Отслеживание новинок недели, месяца, года;
* Просмотр чарта как текущей недели, так и предшествующих недель;
* Добавление альбомов и песен в личную фонотеку, для удобства (только для авторизованных пользователей);
* Поиск по ключевым словам(названиям альбомов, песен, именам исполнителей);
* Авторизация / регистрация в системе;
* Прослушивание песен.

Для данного проекта была сформирована концептуальная схема (Рисунок 2.2).



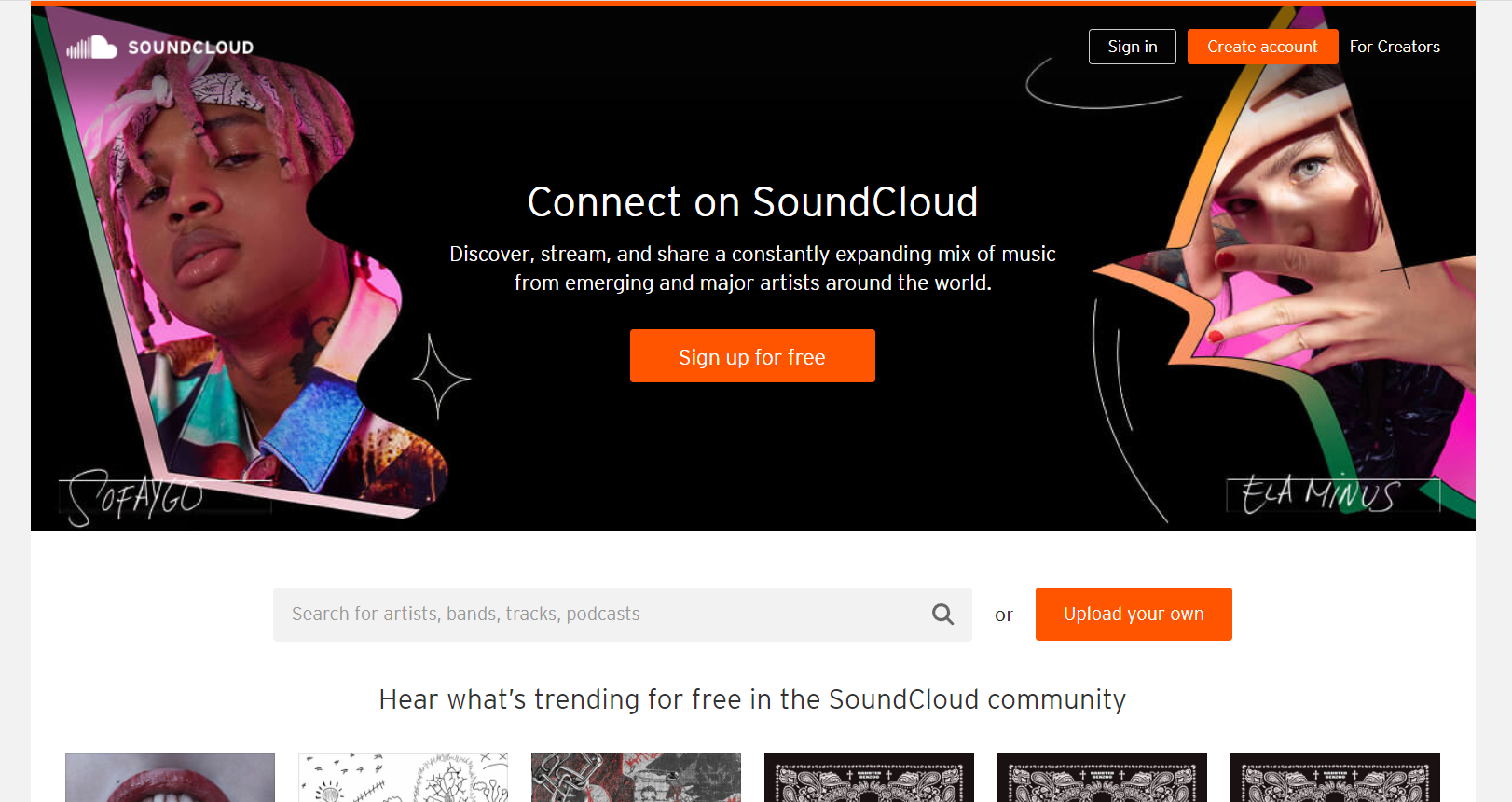
Рисунок 2.2 – Концептуальная схема

* 1. Анализ аналогов

**SoundCloud**

Главная задача – это прослушивание музыки, плеер интегрирован в приложение. Здесь есть возможность потокового прослушивания аудиозаписей и создания плейлистов неограниченного объёма. [[3]](#Литература_3)

Главная страница SoundCloud представлена на Рисунке 2.3.1.

  
Рисунок 2.3.1 - Главная страница SoundCloud [[1]](#Литература_1)

**Функционал:**

* Проигрывание станций – каждый пользователь сервиса может публиковать свои аудиоподборки, которые будут доступны широкой публики. Чтобы послушать композиции, достаточно кликнуть по станции.
* Обзор – на данной странице пользователю предлагаются чарты от SoundCloud, а также фильтр композиций по жанрам.
* Создание плейлистов – музыкальная социальная сеть позволяет формировать свои собственные плейлисты.
* Воспроизведение похожих дорожек.

**Плюсы:**

* Уникальность композиций.
* Возможность прослушивания + составления собственного альбома
* Отображение статистики прослушиваний, просмотров.
* Возможность комментировать альбомы и песни.

**Минусы:**

* Ограничение по аккаунту.
* Ограничение количества скачиваний.

**Spotify**

Spotify – это сервис для стриминга различного контента (миллионов треков, подкастов и видеороликов) от авторов со всего мира.

Основной функционал, такой как воспроизведение музыки – бесплатный. Также можно оформить Premium-подписку. [[5]](#Литература_5)

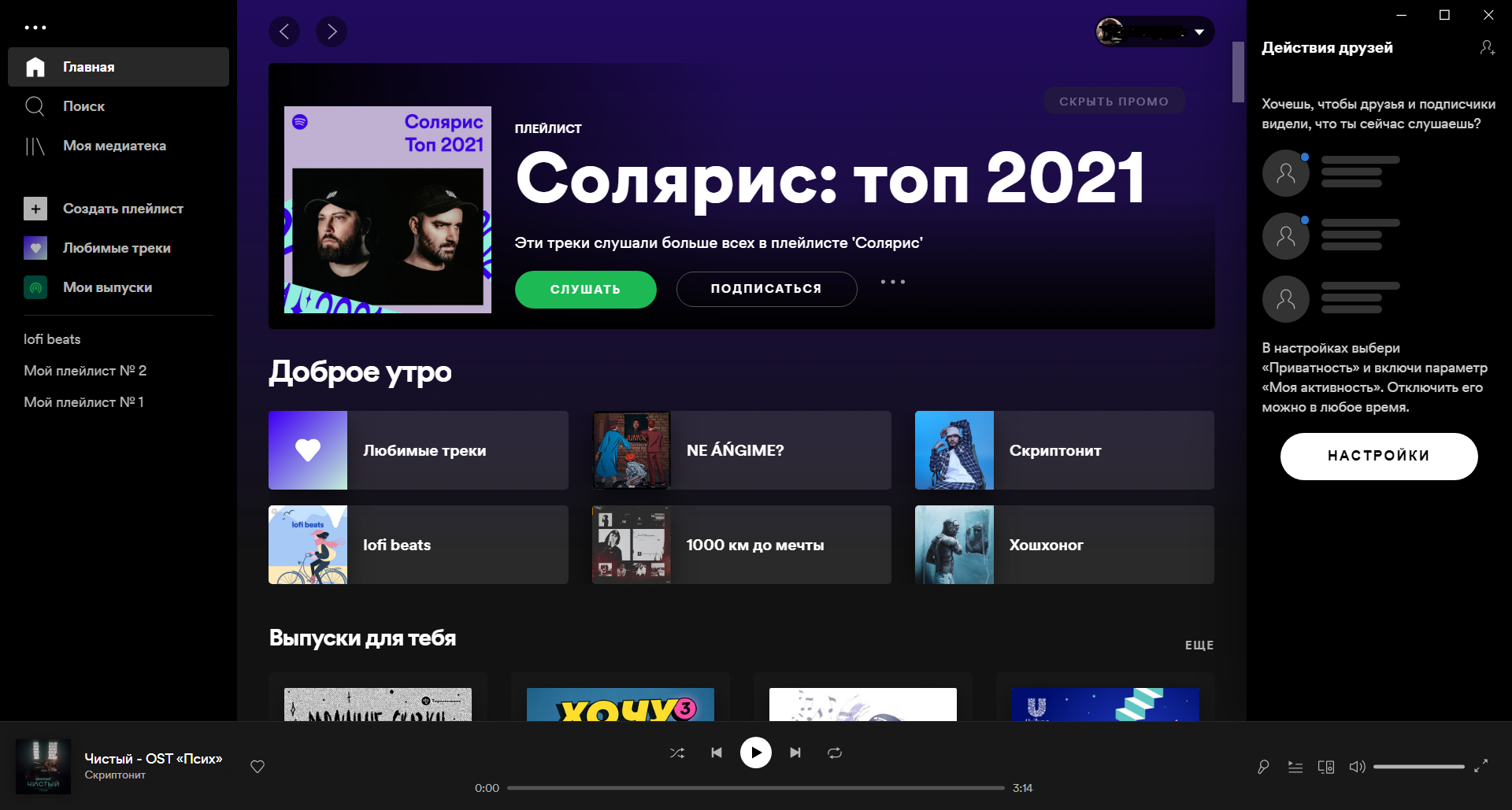


Рисунок 2.3.2 – Главная страница Spotify для ПК [[2]](#Литература_2)

**Функционал:**

* Получение рекомендаций на основе своих предпочтений.
* Создание своих коллекций музыки и подкастов.
* Главное преимущество – алгоритмы подбора музыки, которые практически идеально подстраиваются под вкусы слушателей.

Сервис Spotify доступен на разных устройствах, включая компьютеры, телефоны и планшеты, а также динамики, телевизоры и автомобильные аудиосистемы.

**Преимущества:**

* Добавление подписчиков.
* Создание плейлистов.
* Наличие официального приложения для смартфона и компьютера. И основные плюсы Spotify связаны с тем, что программу можно без проблем установить на любое устройство.
* Рекомендации.
* Высокое качество звучания.

**Недостатки:**

* Большое количество ограничений в бесплатной версии.
  1. Перечень задач, подлежащих решению в процессе разработки.
* Определить состав и структуру данных;
* Разработать пользовательский интерфейс;
* Определить архитектуру приложения;
* Разработать и реализовать базу данных;
* Разработать функционал WEB-приложения;
* Протестировать и проверить разработанную систему;
* Исправить выявленные ошибки.
  1. Обоснование выбора инструментов и платформы для разработки

1. Для создания базы данных была выбрана Объектно-реляционная система управления базами данных PostgreSQL.

Из характеристик СУБД, которые могут определить выбор, одной из важнейших является модель данных. Теоретически любую информацию можно представить в виде реляционной модели. Сила реляционных баз данных в том, что эта модель очень хорошо подходит для предприятий, которые располагают немалыми средствами для активного внедрения передовых систем. Эта модель имеет наиболее проработанное математическое основание и хорошо проработанные стандарты. Реляционная модель данных отличается большой гибкостью с точки зрения изменения структуры данных. Здесь можно менять физическую структуру данных, не переписывая приложения.

Важным критерием выбора с точки зрения перспектив становится наличие эффективных средств разработки. Такие средства, обладающие

удобным интерфейсом, позволяют специалистам предприятия самостоятельно и быстро настраивать информационные системы в соответствии с требованиями бизнеса.

PostgreSQL - это мощная объектно-реляционная система управления базами данных. Она включает большинство типов данных SQL92 и SQL99, включая integer, numeric, boolean, char, varchar, date, interval, и timestamp. Она также поддерживает хранение больших двоичных объектов (BLOB's), включая картинки, звук, или видео. А также имеет API для C# и др.

Являясь СУБД класса предприятия, PostgreSQL предоставляет такие особенности как Multi-Version Concurrency Control (MVCC), восстановление по точке во времени, табличное пространство, асинхронная репликация, вложенные транзакции (точки сохранения), горячее резервирование, планировщик/оптимизатор запросов, и упреждающее журналирование на случай поломки.

Средства обеспечения целостности данных включают составные первичные ключи, внешние ключи с поддержкой запрета и каскадирования изменений/удалений, проверку ограничений (constraints), ограничения уникальности и ограничения на непустые значения.

Для разработки системы была выбрана среда разработки Visual Studio с кроссплатформенным фреймворком ASP.NET Core с использованием Entity Framework.

1. В качестве среды разработки была выбрана Microsoft Visual Studio 2019

Microsoft Visual Studio 2019— это набор инструментов разработки, основанных на использовании компонентов, и других технологий для создания мощных, производительных приложений. Кроме того, среда Visual Studio оптимизирована для совместного проектирования, разработки и развертывания корпоративных решений. Visual Studio предоставляет средства для проектирования, разработки b отладки.

Среда разработки Visual Studio представляет собой полный набор средств разработки для создания настольных приложений. Visual C# использует единую интегрированную среду разработки (IDE), которая позволяет совместно использовать средства и упрощает создание решений на базе нескольких языков.

Почему именно эта версия продукта, потому что в нем есть такие удобные нововведения как:

1. Visual Studio IntelliCode повышает эффективность разработки программного обеспечения с помощью искусственного интеллекта (ИИ). Для создания рекомендаций IntelliCode анализирует 2000 проектов с открытым кодом на GitHub

2. Рефакторинг В C#. Есть много новых удобных возможностей рефакторинга, которые помогают упорядочить код. Они отображаются как предложения со значком лампочки и включают такие действия, как перемещение элементов в интерфейс или базовый класс, настройку пространств имен в соответствии со структурой папок.

1. Основной фреймворк разработки – ASP.NET Core с использованием Entity Framework:

Основным преимуществом платформы .NET в отношении баз данных является наличие LINQ.

Аббревиатура LINQ обозначает целый набор технологий, создающих и использующих возможности интеграции запросов непосредственно в язык C#. Традиционно запросы к данным выражаются в виде простых строк без проверки типов при компиляции или поддержки IntelliSense. Кроме того, разработчику приходится изучать различные языки запросов для каждого типа источников данных: баз данных SQL, XML-документов, различных веб-служб и т. д. Технологии LINQ превращают запросы в удобную языковую конструкцию, которая применяется аналогично классам, методам и событиям. [[4]](#Литература_4)

1. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ
   1. Разработка алгоритмов обработки информации

Основные нетривиальные задачей разработки системы является создание алгоритмов обработки информации.

В системе были реализованы интерфейсы репозиторий и их реализации для каждой из сущностей. Данные интерфейсы включают в себя все возможные операции по манипуляции данными.

Репозитории предоставляют минимально необходимые данные о сущностях, но если есть необходимость получения дополнительной информации о сущности, как например в список песен, для каждой песни получить списки пользовательских фонотек, то в дело вступает Менеджер данных (DataManager) позволяющий расширить сущность и получить вложенные данные.

Также DataManager служит для предотвращения избыточности данных посредством контроля создаваемых сущностей.

* 1. Логическая схема базы данных

Логическая схема базы данных представлена на Рисунке 3.2.

Рисунок 3.2 – Логическая схема базы данных

* 1. Разработка архитектуры приложения

Для данного проекта подошла бы любая платформа разработки веб-приложений. Мною была выбрана, как уже указывалось в исследовательской части ASP.NET Core.

Частью платформы .NET является Entity Framework. Данный ORM-фреймворк тоже имеет ряд преимуществ. Он позволяет создавать таблицы базы данных и саму базу данных напрямую из кода проекта (Code First). Entity Framework выполняет соответствующий запрос в базу данных, а затем предоставляет результат в экземплярах объектов веб-приложения, для того чтобы далее с ним было проще работать. Данный фреймворк позволяет выполнять запросы LINQ к базе данных.

В данном проекте используется СУБД PostgreSQL. Эта СУБД не особо отличается от Microsoft SQL Server.

* 1. Реализация функционала приложения

Описание сущностей базы данных:

**“Songs”** – Сущность для композиций. Имеет следующие поля:

"Id" – PRIMARY KEY , text, NOT NULL,

"AlbumId" – FOREIGN KEY => Albums.Id, text , NOT NULL,

"Title" – character varying(256), NOT NULL,

"DurationSec" – integer, NOT NULL,

"ListensNumber" – integer, DEFAULT 0,

"ChartListensNumber" – integer, DEFAULT 0,

"ReleaseDate" – date, NOT NULL,

"CoverPath" – character varying(256), NOT NULL,

"SongPath” – character varying(256), NOT NULL;

**“Albums”** – Сущность для альбомов. Имеет следующие поля:

"Id" – PRIMARY KEY, text, NOT NULL,

"Title" – character varying(256), NOT NULL,

"ReleaseDate" – date, NOT NULL,

"CoverPath" – character varying(256), NOT NULL;

**“Performers”** – Сущность для исполнителей. Имеет следующие поля:

"Id" – PRIMARY KEY, text, NOT NULL,

"PerformerName" – character varying(256), NOT NULL,

"AvatarPath" – character varying(256), NOT NULL;

**“SongPerformers”** – Сущность для связывания “многие ко многим” сущностей Songs и Performers. Имеет следующие поля:

"SongId" – PRIMARY KEY, FOREIGN KEY => Songs.Id, text, NOT NULL,

"PerformerId" – PRIMARY KEY, FOREIGN KEY => Peformer.Id, text, NOT NULL;

**“AlbumPerformers”** – Сущность для связывания “многие ко многим” сущностей Albums и Performers. Имеет следующие поля:

"AlbumId" – PRIMARY KEY, FOREIGN KEY => Album.Id, text, NOT NULL,

"PerformerId" – PRIMARY KEY, FOREIGN KEY => Peformer.Id, text, NOT NULL;

**“AspNetUsers”** – Сущность для пользователей. Имеет следующие поля:

"Id" PRIMARY KEY, text, NOT NULL,

"UserName" – character varying(256), NOT NULL,

"Email" – character varying(256), NOT NULL,

"PasswordHash" – text, ……;

**“AspNetRoles” -** Сущность для ролей, используемых пользователями. Имеет следующие поля:

"Id" – PRIMARY KEY, text, NOT NULL,

"Name" – character varying(256);

**“AspNetUserRoles”** – Сущность для связывания “Многие ко многим” сущностей AspNetUsers и AspNetRoles. Имеет следующие поля:

"RoleId" – PRIMARY KEY, FOREIGN KEY => AspNetRoles.Id, text, NOT NULL,

"UserId" – PRIMARY KEY, FOREIGN KEY => AspNetUsers.Id, text, NOT NULL;

**“UserSongsLibrary” –** сущность для связывания “Многие ко многим” сущностей AspNetUsers и Songs. Имеет следующие поля:

"UserId" – PRIMARY KEY, FOREIGN KEY => AspNetRoles.Id, text, NOT NULL,

"SongId" – PRIMARY KEY, FOREIGN KEY => Songs.Id, text, NOT NULL,

"AdditionDate" – date, NOT NULL;

**“UserAlbumsLibrary” –** сущность для связывания “Многие ко многим” сущностей AspNetUsers и Albums. Имеет следующие поля:

"UserId" – PRIMARY KEY, FOREIGN KEY => AspNetRoles.Id, text, NOT NULL,

"AlbumId" – PRIMARY KEY, FOREIGN KEY => Album.Id, text, NOT NULL,

"AdditionDate" – date, NOT NULL;

**“WeeklyCharts” –** Сущность для недельных чартов. Имеет следующие поля:

"Id" – text, NOT NULL,

"ReleaseDate" – date, NOT NULL;

**“ChartSongs” –** сущность для связывания “Многие ко многим” сущностей WeeklyCharts и Songs. Имеет следующие поля:

"ChartId" – PRIMARY KEY, FOREIGN KEY => WeeklyCharts.Id, text, NOT NULL,

"SongId" – PRIMARY KEY, FOREIGN KEY => Songs.Id, text, NOT NULL,

"Spot" – integer, NOT NULL;

Подробное описание сущностей в виде DLL описано в [Приложении А](#Приложение_А).

1. ПРОЕТНО-ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

* 1. Проектирование начального и тестового наполнения базы данных.

Процедура автоматизации.

При разработке системы встал вопрос об удобном заполнении системы данными. Для тестового наполнения базы данных был использован осуществленный механизм добавления через само веб-приложение (вкладка “Администрирование”). Причиной выбора такого решения стало удобство выбора списка объектов для заполнения других объектов, например заполнение списка исполнителей для альбомов и композиций.

Данная автоматизация позволяет без особых навыков и усилий взаимодействовать с базой данных, без прямого взаимодействия с СУБД.

* 1. Технологические решения, поддерживающие эксплуатационный цикл программы.

Данное приложение можно улучшить в будущем многими способами. Так как разработка приложения была ограничена небольшим количеством времени, то большинство алгоритмов были реализованы не идеально. Большинство из них можно оптимизировать. Алгоритмы взаимодействия с базой данных можно усовершенствовать, добившись наилучшей оптимизации.

В данном приложении многие страницы не в полной мере адаптивны, и неверно отображаются на некоторых моделях мобильных устройств.

Также в данное приложение можно добавить систему комментирования альбомов и композиций, ленивую загрузку страницы (Lazy load), которая позволит пользователю быстрее получить требуемую информацию.

* 1. Порядок развертывания системы.

Данное приложение было развернуто в облачной платформе Azure компании Microsoft. Данное действие является весьма простым и быстрым в реализации, из-за использования Visual Studio(Рисунок 4.3.1).

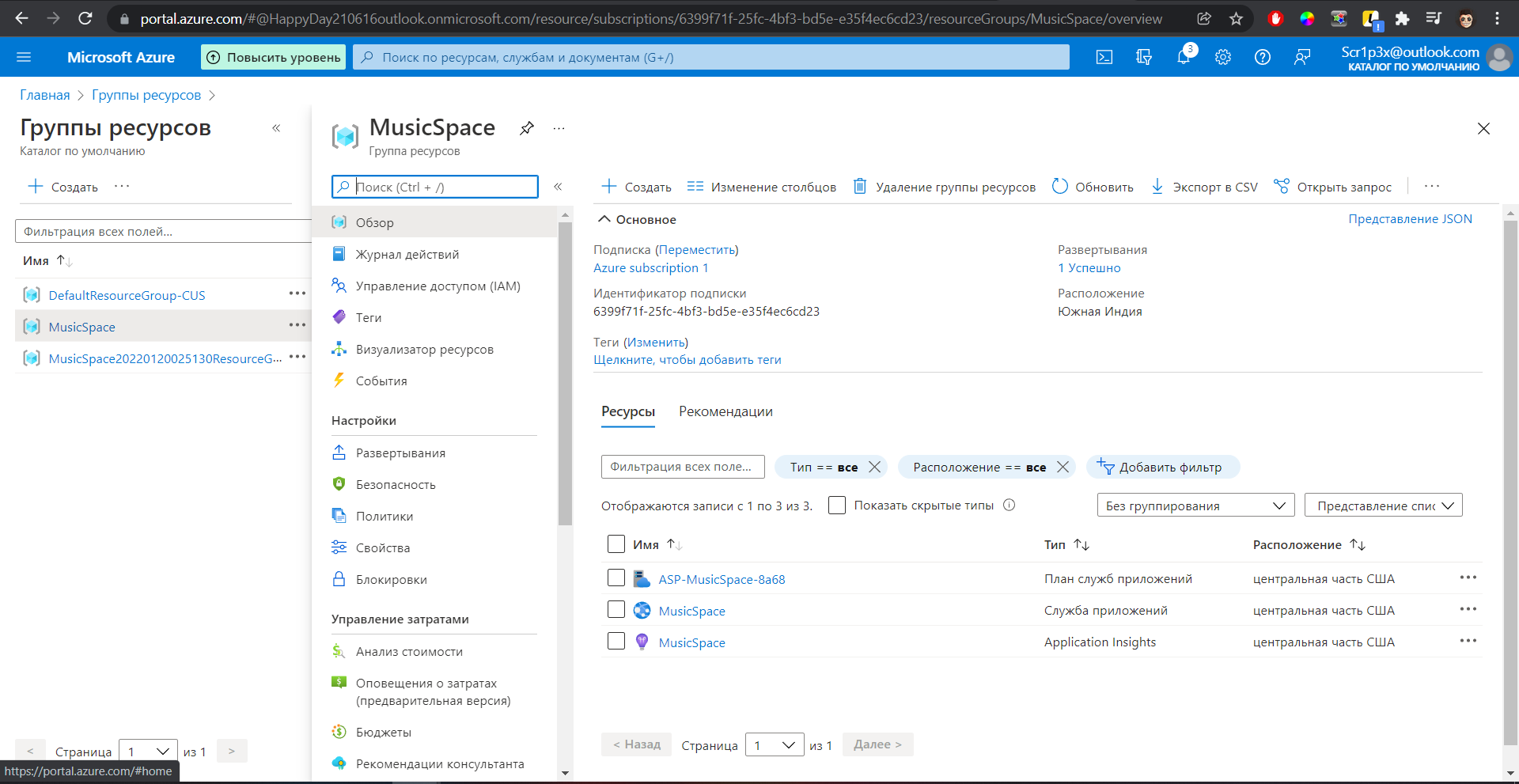


Рисунок 4.3.1 – Профиль опубликованного приложения

Для корректной публикации в Azure была создана группа ресурсов для приложения и базы данных. База данных была так же настроена на связь с Entity Framework для корректного применения миграций.

* 1. Разработка руководства пользователя и руководства администратора.

**Руководство администратора:**

Для возможности администрирования веб приложения требуется выполнить авторизацию с логином и паролем пользователя, имеющего доступ к администрированию сайта (Рисунок 4.4.1). Если нужно установить данную роль пользователя, то требуется сделать это в СУБД. В дальнейшем же можно добавлять роли в самом приложении имея доступ к администрированию.

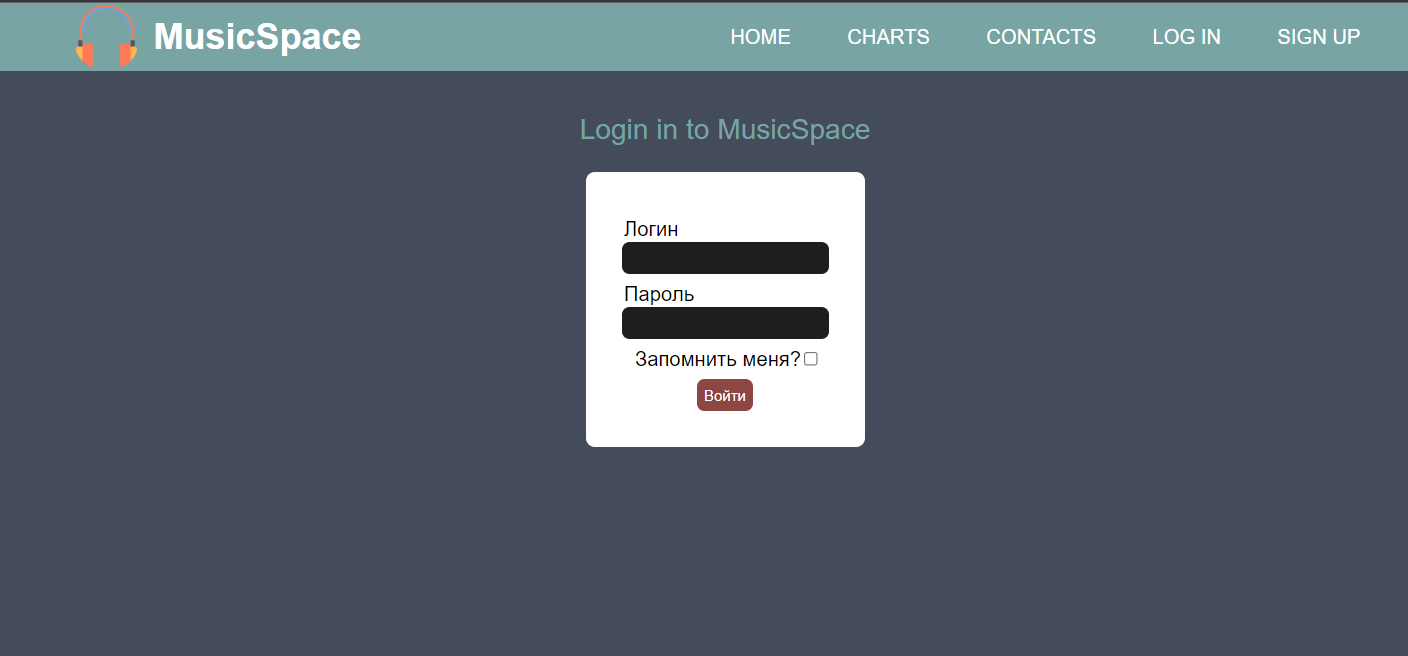


Рисунок 4.4.1 – Форма авторизации

1. После авторизации в профиле администратора появится кнопка “Панель администратора” (Рисунок 4.4.2 и Рисунок 4.4.3). При нажатии на кнопку появится несколько опций выбора. Подробнее о них далее.

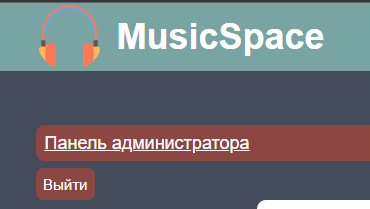


Рисунок 4.4.2 – Кнопка панели администратора

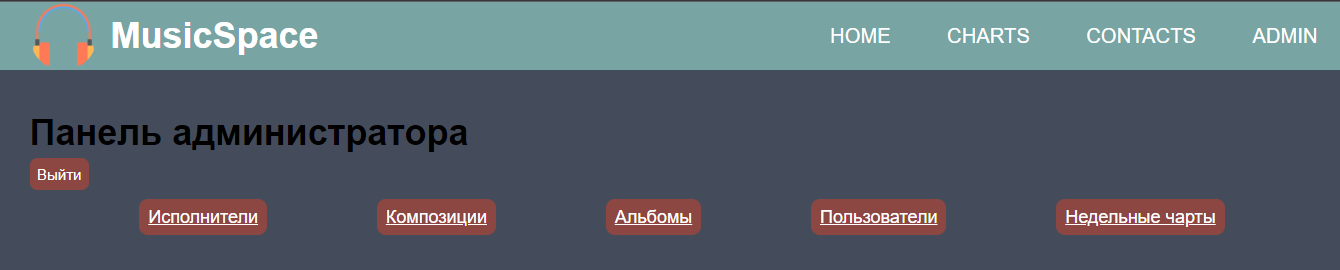


Рисунок 4.4.3 – Панель администратора

1. Кнопка Исполнители (Рисунок 4.4.3) – предоставляет таблицу всех имеющихся в базе данных исполнителей, а также все возможные действия по изменению данных исполнителей (Рисунок 4.4.4).

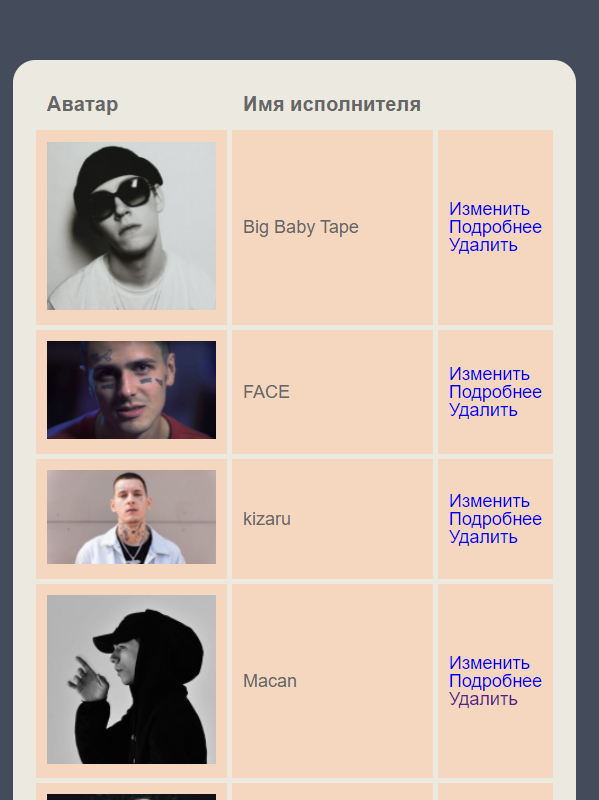


Рисунок 4.4.4 – Таблица исполнителей

1. Кнопка композиции (Рисунок 4.4.3) – предоставляет таблицу всех имеющихся в базе данных песен, а также все возможные действия по изменению данных композиций (Рисунок 4.4.5).

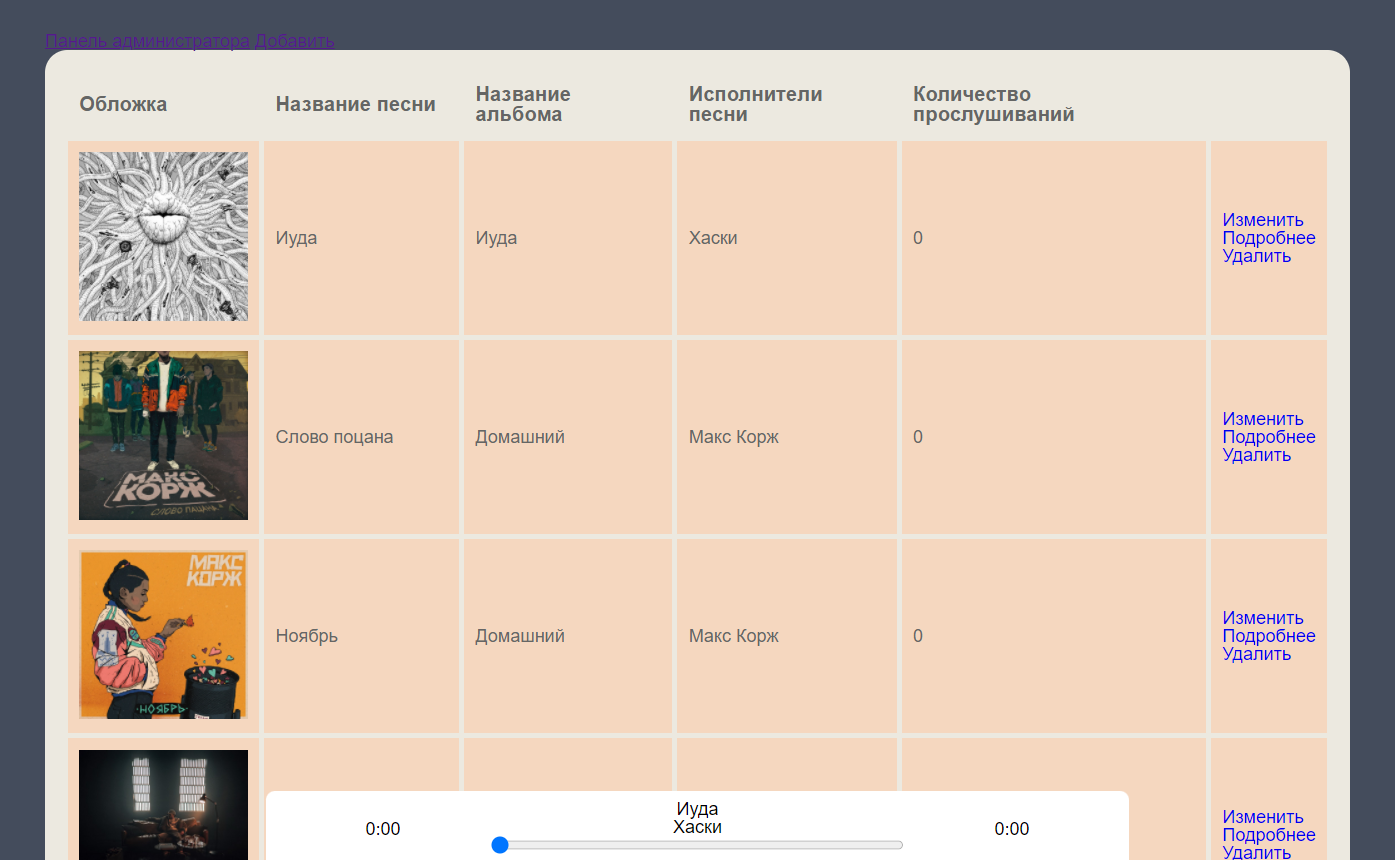


Рисунок 4.4.5 – Таблица композиций

1. Кнопка Альбомы (Рисунок 4.4.3) – представляет таблицу всех имеющихся в базе данных альбомов, а также все возможные действия по изменению данных альбомов (Рисунок 4.4.6).

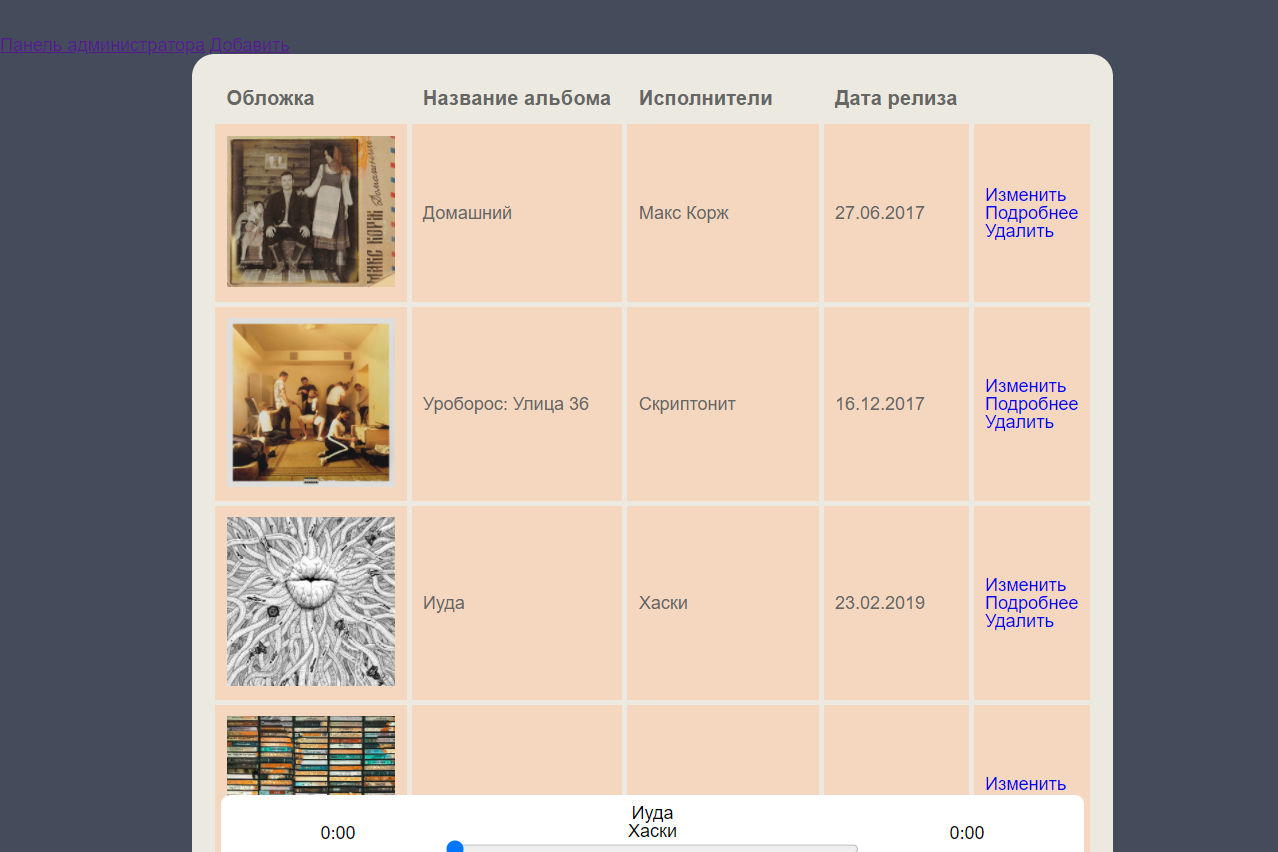


Рисунок 4.4.6 – Таблица альбомов

1. Кнопка пользователей (Рисунок 4.4.3) – представляет таблицу всех имеющихся в базе данных пользователей, а также все возможные действия по изменению данных пользователей (Рисунок 4.4.7).

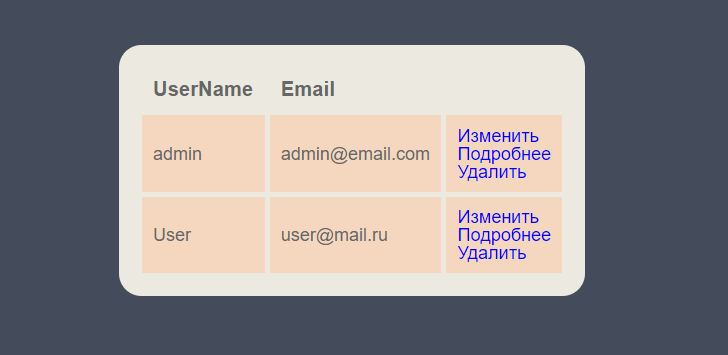


Рисунок 4.4.7 – Таблица пользователей

1. Кнопка недельные чарты (Рисунок 4.4.3) – отображает таблицу всех имеющихся в базе данных недельных чартов, а также все возможные действия по изменению данных недельных чартов (Рисунок 4.4.8).

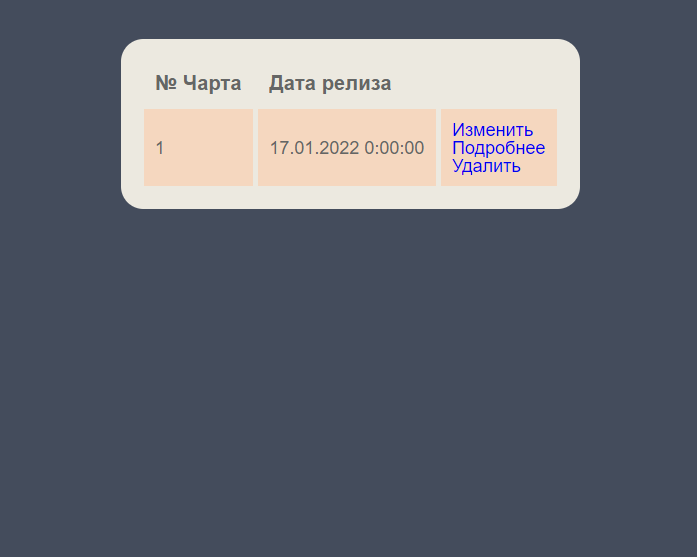


Рисунок 4.4.8 – Таблица недельных чартов

**Руководство пользователя:**

Для возможности прослушивания композиций главной страницы и недельных пользователь может не проходить авторизацию, но если пользователю потребуется воспользоваться личным хранилищем альбомов и композиций, то он будет вынужден пройти авторизацию с логином и паролем пользователя (Рисунок 4.4.1). Если же пользователь ранее не регистрировался на платформе, ему необходимо пройти регистрацию.

1. В ходе регистрации пользователю необходимо ввести электронную почту, логин (по которому он в дальнейшем будет производить авторизацию) и пароль (Рисунок 4.4.9). Всем регистрирующимся пользователям система автоматически присваивает роль “Пользователь”.

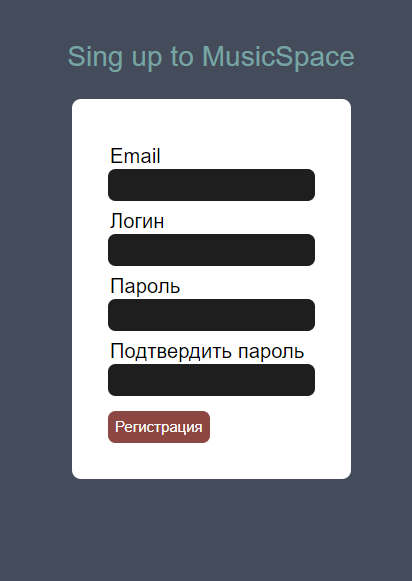


Рисунок 4.4.9 – Форма регистрации

1. После авторизации/регистрации пользователю становится доступным добавление альбомов и музыкальных композиций в личное хранилище (профиль), который отображен на Рисунке 4.4.10.

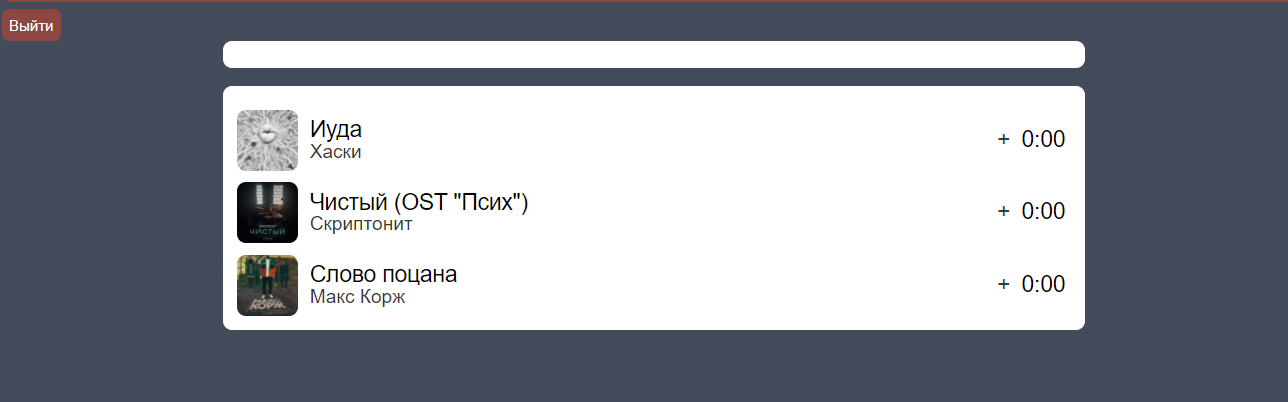


Рисунок 4.4.10 – Профиль пользователя

1. Для просмотра и прослушивания композиций недельных чартов пользователю необходимо перейти на вкладку главного меню “Charts” (Рисунок 4.4.3). После этого пользователю станут доступны композиции текущего недельного чарта в убывающем порядке количества недельных прослушиваний (Рисунок 4.4.11).

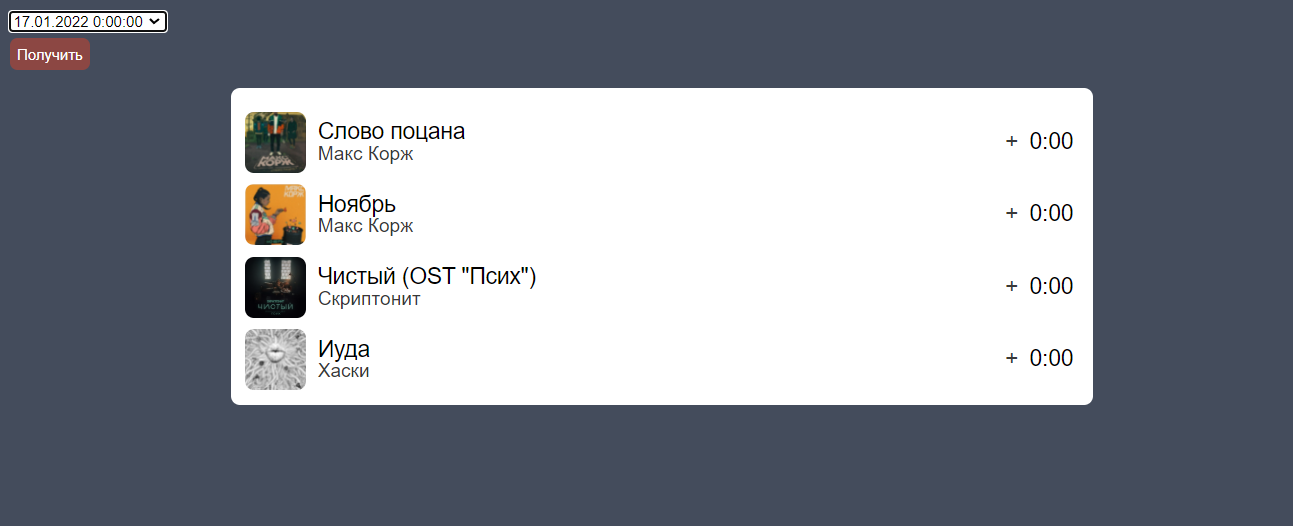


Рисунок 4.4.11 – Страница недельных чартов

1. Для отслеживания предыдущих чартов пользователю необходимо выбрать интересующую дату в выпадающем списке в левом верхнем углу страница (Рисунок 4.4.11) и нажать на кнопку “Получить”.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы было разработано веб-приложение для прослушивания музыки.

Работа выполнена в несколько этапов: была выбрана архитектура, СУБД и разработана структура системы, реализовано веб-приложение.

Были сформированы навыки по разработке и реализации программного приложения с базой данных.

Были реализованы все поставленные задачи.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. <https://soundcloud.com/>

1. <https://www.spotify.com/ru-en/>

1. <https://www.audiomania.ru/content/art-5640.html>

1. <https://docs.microsoft.com/ru-ru/dotnet/csharp/linq/>

1. <https://ru.wikipedia.org/wiki/Spotify>
2. Шнырёв, С.Л. Базы данных: учебное пособие для вузов - М. : НИЯУ МИФИ, 2011. — 224 с. — Режим доступа: <http://e.lanbook.com/book/75809>
3. Ревунков, Г.И. Базы и банки данных - М.: МГТУ им. Н.Э. Баумана, 2011. — 68 с. — Режим доступа: <http://e.lanbook.com/book/52425>
4. Ревунков, Г.И. Проектирование баз данных - М.: МГТУ им. Н.Э. Баумана, 2009. — 20 с. — Режим доступа: <http://e.lanbook.com/book/52390>
5. Кудрявцев, К.Я. Создание баз данных: учебное пособие — М.: НИЯУ МИФИ, 2010. — 155 с. — Режим доступа: <http://e.lanbook.com/book/75822>
6. Сидоров В.Н., Сломинская Е.Н., Полникова Т.В., Макарова О.Ю. Оформление графической части выпускной квалификационной работы. Учебное пособие. М.: МГТУ им. Н.Э. Баумана, 2016.

ПРИЛОЖЕНИЕ А

-- Table: public.AspNetRoles

CREATE TABLE IF NOT EXISTS public."AspNetRoles"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "Name" character varying(256) COLLATE pg\_catalog."default",

    "NormalizedName" character varying(256) COLLATE pg\_catalog."default",

    "ConcurrencyStamp" text COLLATE pg\_catalog."default",

    CONSTRAINT "PK\_AspNetRoles" PRIMARY KEY ("Id")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetRoles"

    OWNER to postgres;

-- Index: RoleNameIndex

CREATE UNIQUE INDEX IF NOT EXISTS "RoleNameIndex"

    ON public."AspNetRoles" USING btree

    ("NormalizedName" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.AspNetUsers

CREATE TABLE IF NOT EXISTS public."AspNetUsers"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "UserName" character varying(256) COLLATE pg\_catalog."default",

    "NormalizedUserName" character varying(256) COLLATE pg\_catalog."default",

    "Email" character varying(256) COLLATE pg\_catalog."default",

    "NormalizedEmail" character varying(256) COLLATE pg\_catalog."default",

    "EmailConfirmed" boolean NOT NULL,

    "PasswordHash" text COLLATE pg\_catalog."default",

    "SecurityStamp" text COLLATE pg\_catalog."default",

    "ConcurrencyStamp" text COLLATE pg\_catalog."default",

    "PhoneNumber" text COLLATE pg\_catalog."default",

    "PhoneNumberConfirmed" boolean NOT NULL,

    "TwoFactorEnabled" boolean NOT NULL,

    "LockoutEnd" timestamp with time zone,

    "LockoutEnabled" boolean NOT NULL,

    "AccessFailedCount" integer NOT NULL,

    CONSTRAINT "PK\_AspNetUsers" PRIMARY KEY ("Id")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetUsers"

    OWNER to postgres;

-- Index: EmailIndex

CREATE INDEX IF NOT EXISTS "EmailIndex"

    ON public."AspNetUsers" USING btree

    ("NormalizedEmail" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Index: UserNameIndex

CREATE UNIQUE INDEX IF NOT EXISTS "UserNameIndex"

    ON public."AspNetUsers" USING btree

    ("NormalizedUserName" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.AspNetUserRoles

CREATE TABLE IF NOT EXISTS public."AspNetUserRoles"

(

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    "RoleId" text COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_AspNetUserRoles" PRIMARY KEY ("UserId", "RoleId"),

    CONSTRAINT "FK\_AspNetUserRoles\_AspNetRoles\_RoleId" FOREIGN KEY ("RoleId")

        REFERENCES public."AspNetRoles" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE,

    CONSTRAINT "FK\_AspNetUserRoles\_AspNetUsers\_UserId" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetUserRoles"

    OWNER to postgres;

-- Index: IX\_AspNetUserRoles\_RoleId

CREATE INDEX IF NOT EXISTS "IX\_AspNetUserRoles\_RoleId"

    ON public."AspNetUserRoles" USING btree

    ("RoleId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.AspNetUserTokens

CREATE TABLE IF NOT EXISTS public."AspNetUserTokens"

(

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    "LoginProvider" text COLLATE pg\_catalog."default" NOT NULL,

    "Name" text COLLATE pg\_catalog."default" NOT NULL,

    "Value" text COLLATE pg\_catalog."default",

    CONSTRAINT "PK\_AspNetUserTokens" PRIMARY KEY ("UserId", "LoginProvider", "Name"),

    CONSTRAINT "FK\_AspNetUserTokens\_AspNetUsers\_UserId" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetUserTokens"

    OWNER to postgres;

-- Table: public.AspNetUserLogins

CREATE TABLE IF NOT EXISTS public."AspNetUserLogins"

(

    "LoginProvider" text COLLATE pg\_catalog."default" NOT NULL,

    "ProviderKey" text COLLATE pg\_catalog."default" NOT NULL,

    "ProviderDisplayName" text COLLATE pg\_catalog."default",

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_AspNetUserLogins" PRIMARY KEY ("LoginProvider", "ProviderKey"),

    CONSTRAINT "FK\_AspNetUserLogins\_AspNetUsers\_UserId" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetUserLogins"

    OWNER to postgres;

-- Index: IX\_AspNetUserLogins\_UserId

CREATE INDEX IF NOT EXISTS "IX\_AspNetUserLogins\_UserId"

    ON public."AspNetUserLogins" USING btree

    ("UserId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.AspNetUserClaims

CREATE TABLE IF NOT EXISTS public."AspNetUserClaims"

(

    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    "ClaimType" text COLLATE pg\_catalog."default",

    "ClaimValue" text COLLATE pg\_catalog."default",

    CONSTRAINT "PK\_AspNetUserClaims" PRIMARY KEY ("Id"),

    CONSTRAINT "FK\_AspNetUserClaims\_AspNetUsers\_UserId" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetUserClaims"

    OWNER to postgres;

-- Index: IX\_AspNetUserClaims\_UserId

CREATE INDEX IF NOT EXISTS "IX\_AspNetUserClaims\_UserId"

    ON public."AspNetUserClaims" USING btree

    ("UserId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.AspNetRoleClaims

CREATE TABLE IF NOT EXISTS public."AspNetRoleClaims"

(

    "Id" integer NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START 1 MINVALUE 1 MAXVALUE 2147483647 CACHE 1 ),

    "RoleId" text COLLATE pg\_catalog."default" NOT NULL,

    "ClaimType" text COLLATE pg\_catalog."default",

    "ClaimValue" text COLLATE pg\_catalog."default",

    CONSTRAINT "PK\_AspNetRoleClaims" PRIMARY KEY ("Id"),

    CONSTRAINT "FK\_AspNetRoleClaims\_AspNetRoles\_RoleId" FOREIGN KEY ("RoleId")

        REFERENCES public."AspNetRoles" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE CASCADE

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AspNetRoleClaims"

    OWNER to postgres;

-- Index: IX\_AspNetRoleClaims\_RoleId

CREATE INDEX IF NOT EXISTS "IX\_AspNetRoleClaims\_RoleId"

    ON public."AspNetRoleClaims" USING btree

    ("RoleId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.Albums

CREATE TABLE IF NOT EXISTS public."Albums"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "Title" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    "ReleaseDate" date NOT NULL,

    "CoverPath" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_Albums" PRIMARY KEY ("Id")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."Albums"

    OWNER to postgres;

-- Table: public.AlbumPerformers

CREATE TABLE IF NOT EXISTS public."AlbumPerformers"

(

    "AlbumId" text COLLATE pg\_catalog."default" NOT NULL,

    "PerformerId" text COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "AlbumPerformer\_pkey" PRIMARY KEY ("AlbumId", "PerformerId"),

    CONSTRAINT "Album\_fkey" FOREIGN KEY ("AlbumId")

        REFERENCES public."Albums" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT,

    CONSTRAINT "Performer\_fkey" FOREIGN KEY ("PerformerId")

        REFERENCES public."Performers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."AlbumPerformers"

    OWNER to postgres;

-- Index: IX\_AlbumPerformers\_PerformerId

CREATE INDEX IF NOT EXISTS "IX\_AlbumPerformers\_PerformerId"

    ON public."AlbumPerformers" USING btree

    ("PerformerId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.ChartSongs

CREATE TABLE IF NOT EXISTS public."ChartSongs"

(

    "ChartId" text COLLATE pg\_catalog."default" NOT NULL,

    "SongId" text COLLATE pg\_catalog."default" NOT NULL,

    "Spot" integer NOT NULL,

    CONSTRAINT "ChartSong\_pkey" PRIMARY KEY ("ChartId", "SongId"),

    CONSTRAINT "Chart\_fkey" FOREIGN KEY ("ChartId")

        REFERENCES public."WeeklyCharts" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT,

    CONSTRAINT "Song\_fkey" FOREIGN KEY ("SongId")

        REFERENCES public."Songs" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."ChartSongs"

    OWNER to postgres;

-- Index: IX\_ChartSongs\_SongId

CREATE INDEX IF NOT EXISTS "IX\_ChartSongs\_SongId"

    ON public."ChartSongs" USING btree

    ("SongId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.Performers

CREATE TABLE IF NOT EXISTS public."Performers"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "PerformerName" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    "AvatarPath" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_Performers" PRIMARY KEY ("Id")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."Performers"

    OWNER to postgres;

-- Index: PerformerName\_unique

CREATE UNIQUE INDEX IF NOT EXISTS "PerformerName\_unique"

    ON public."Performers" USING btree

    ("PerformerName" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.SongPerformers

CREATE TABLE IF NOT EXISTS public."SongPerformers"

(

    "SongId" text COLLATE pg\_catalog."default" NOT NULL,

    "PerformerId" text COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "SongPerformer\_pkey" PRIMARY KEY ("SongId", "PerformerId"),

    CONSTRAINT "Performer\_fkey" FOREIGN KEY ("PerformerId")

        REFERENCES public."Performers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT,

    CONSTRAINT "Song\_fkey" FOREIGN KEY ("SongId")

        REFERENCES public."Songs" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."SongPerformers"

    OWNER to postgres;

-- Index: IX\_SongPerformers\_PerformerId

CREATE INDEX IF NOT EXISTS "IX\_SongPerformers\_PerformerId"

    ON public."SongPerformers" USING btree

    ("PerformerId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.Songs

CREATE TABLE IF NOT EXISTS public."Songs"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "AlbumId" text COLLATE pg\_catalog."default" NOT NULL,

    "Title" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    "DurationSec" integer NOT NULL,

    "ListensNumber" integer DEFAULT 0,

    "ChartListensNumber" integer DEFAULT 0,

    "ReleaseDate" date NOT NULL,

    "CoverPath" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    "SongPath" character varying(256) COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_Songs" PRIMARY KEY ("Id"),

    CONSTRAINT "Album\_fkey" FOREIGN KEY ("AlbumId")

        REFERENCES public."Albums" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."Songs"

    OWNER to postgres;

-- Index: IX\_Songs\_AlbumId

CREATE INDEX IF NOT EXISTS "IX\_Songs\_AlbumId"

    ON public."Songs" USING btree

    ("AlbumId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.UserLibraryAlbums

CREATE TABLE IF NOT EXISTS public."UserLibraryAlbums"

(

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    "AlbumId" text COLLATE pg\_catalog."default" NOT NULL,

    "AdditionDate" date NOT NULL,

    "UserLibraryAlbumId" text COLLATE pg\_catalog."default",

    CONSTRAINT "UserLibraryAlbum\_pkey" PRIMARY KEY ("UserId", "AlbumId"),

    CONSTRAINT "Album\_fkey" FOREIGN KEY ("AlbumId")

        REFERENCES public."Albums" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT,

    CONSTRAINT "User\_fkey" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."UserLibraryAlbums"

    OWNER to postgres;

-- Index: IX\_UserLibraryAlbums\_AlbumId

CREATE INDEX IF NOT EXISTS "IX\_UserLibraryAlbums\_AlbumId"

    ON public."UserLibraryAlbums" USING btree

    ("AlbumId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.UserLibrarySongs

CREATE TABLE IF NOT EXISTS public."UserLibrarySongs"

(

    "UserId" text COLLATE pg\_catalog."default" NOT NULL,

    "SongId" text COLLATE pg\_catalog."default" NOT NULL,

    "AdditionDate" date NOT NULL,

    "UserLibrarySongId" text COLLATE pg\_catalog."default",

    CONSTRAINT "UserLibrarySong\_pkey" PRIMARY KEY ("UserId", "SongId"),

    CONSTRAINT "Song\_fkey" FOREIGN KEY ("SongId")

        REFERENCES public."Songs" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT,

    CONSTRAINT "User\_fkey" FOREIGN KEY ("UserId")

        REFERENCES public."AspNetUsers" ("Id") MATCH SIMPLE

        ON UPDATE NO ACTION

        ON DELETE RESTRICT

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."UserLibrarySongs"

    OWNER to postgres;

-- Index: IX\_UserLibrarySongs\_SongId

CREATE INDEX IF NOT EXISTS "IX\_UserLibrarySongs\_SongId"

    ON public."UserLibrarySongs" USING btree

    ("SongId" COLLATE pg\_catalog."default" ASC NULLS LAST)

    TABLESPACE pg\_default;

-- Table: public.WeeklyCharts

CREATE TABLE IF NOT EXISTS public."WeeklyCharts"

(

    "Id" text COLLATE pg\_catalog."default" NOT NULL,

    "ReleaseDate" date NOT NULL,

    CONSTRAINT "PK\_WeeklyCharts" PRIMARY KEY ("Id")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."WeeklyCharts"

    OWNER to postgres;

-- Table: public.\_\_EFMigrationsHistory

CREATE TABLE IF NOT EXISTS public."\_\_EFMigrationsHistory"

(

    "MigrationId" character varying(150) COLLATE pg\_catalog."default" NOT NULL,

    "ProductVersion" character varying(32) COLLATE pg\_catalog."default" NOT NULL,

    CONSTRAINT "PK\_\_\_EFMigrationsHistory" PRIMARY KEY ("MigrationId")

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public."\_\_EFMigrationsHistory"

    OWNER to postgres;