Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заместитель директора  по учебной работе  \_\_\_\_\_\_\_\_\_ З.З. Курмашева  « » 2024 г. |
|  |  |

ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ И РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ВЕДЕНИЯ СПРАВОЧНИКОВ МУЗЫКАЛЬНЫХ ИСПОЛНИТЕЛЕЙ

Пояснительная записка к курсовому проекту

МДК 11.01 Технология разработки и защиты баз данных

|  |  |
| --- | --- |
|  | Руководитель проекта  Р.Ф. Каримова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
|  | Студентка гр. 21П-1  С.А. Шамилова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

2024

Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение Уфимский колледж статистики, информатики и вычислительной техники

УТВЕРЖДАЮ

Заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_З.З. Курмашева

«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2024г.

ЗАДАНИЕ

на курсовой проект студентки дневного отделения, группы 21П-1, специальности 09.02.07 Информационные системы и программирование.

Фамилия, имя, отчество: Шамилова Сабина Анасовна

Тема курсового проекта: «Проектирование базы данных и разработка программы для ведения справочников музыкальных исполнителей».

Текст задания:

при выполнении курсового проекта должны быть решены следующие задачи:

а) спроектирована база данных;

б) разработана структура программы;

в) реализованы функции регистрации на платформу

В результате выполнения курсового проекта должны быть представлены:

а) пояснительная записка, состоящая из следующих разделов:

Введение

1 Постановка задачи

2 Экспериментальный раздел

Заключение

Приложения

Список сокращений

Список источников

б) электронный носитель, содержащий разработанный программный продукт;

в) презентация курсового проекта в электронном виде.

Список рекомендуемых источников:

1. Голицына, О. Л. Основы проектирования баз данных: учеб. пособие / О.Л. Голицына, Т.Л. Партыка, И.И. Попов. — 2-е изд., перераб. и доп. — Москва : ФОРУМ : ИНФРА-М, 2019. — 416 с.: ил. — (Cреднее профессиональное образование). – ISBN 978-5-91134-655-3. - Текст: электронный. - URL: https://znanium.com/catalog/product/1018906 (дата обращения: 22.03.2023). –Режим доступа: по подписке.
2. Гордеев, С. И. Организация баз данных в 2 ч. Часть 1: учебник для среднего профессионального образования / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. И доп. — Москва: Издательство Юрайт, 2023. — 310 с. — (Профессиональное образование). — ISBN 978-5-534-11626-7. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/518510 (дата обращения: 14.03.2023)
3. Карпова, И. П. Базы данных: учебное пособие / И. П. Карпова. – Санкт-Петербург: Питер, 2021. - 240 с. - (Серия «Учебное пособие»). - ISBN 978-5-4461- 9681-4. - Текст: электронный. - URL: https://znanium.com/catalog/product/1857026 (дата обращения: 22.03.2023). – Режим доступа: по подписке.
4. Документация по MySQL. URL: https://dev.mysql.com/doc/ (дата обращения: 14.03.2023).
5. Metanit: Сайт о программировании. URL: https://metanit.com/ (дата обращения: 14.03.2023).

Задание к выполнению получил «31» января 2024 г.

Студент Шамилова Сабина Анасовна

Срок окончания «31» мая 2024 г.

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.Ф. Каримова

Задание рассмотрено на заседании цикловой комиссии информатики

протокол № 4 от «11» января 2024 г.

Председатель цикловой комиссии информатики \_\_\_\_\_\_\_ О.В. Фатхулова

Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение Уфимский колледж статистики, информатики и вычислительной техники

ЗАКЛЮЧЕНИЕ

на курсовой проект

Студент Шамилова Сабина Анасовна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Группа 21П-1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность 09.02.07 Информационные системы и программирование

Тема Проектирование базы данных и разработка программы для ведения справочников музыкальных исполнителей \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Объем курсового проекта:

количество листов пояснительной записки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

количество листов графической части \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Заключение о степени соответствия заданию на курсовое проектирование

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

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

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

Характеристика качеств, проявленных студентом при работе над проектом:

самостоятельность, дисциплинированность, умение планировать работу и

пользоваться литературным материалом и т.д.

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

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

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

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

Положительные стороны курсового проекта

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

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

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

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

Недостатки курсового проекта

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

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

Характеристика общетехнической и специальной подготовки студента

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

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

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

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

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

Заключение и предлагаемая оценка за курсовой проект

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

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

Руководитель курсового проекта Каримова Резида Флюновна \_\_\_\_\_\_\_\_\_\_\_

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

АННОТАЦИЯ

Пояснительная записка к курсовому проекту содержит постановку и программу решения задачи «Проектирование базы данных и разработка программы для ведения справочников музыкальных исполнителей».

Программа Nover.exe написана на языке C# в среде программирования Visual Studio 2022 с использование системы управления базой данных MySQL, предназначена для работы в операционной системе Microsoft Windows 10 и выше, отлажена на данных контрольного примера.

СОДЕРЖАНИЕ

лист

[ВВЕДЕНИЕ 4](#_Toc169799821)

[1 Постановка задачи 6](#_Toc169799822)

[1.1 Описание предметной области 6](#_Toc169799823)

[1.2 Описание входной информации 9](#_Toc169799824)

[1.3 Описание выходной информации 9](#_Toc169799825)

[1.4 Диаграмма прецедентов 9](#_Toc169799826)

[1.5 Концептуальное моделирование 10](#_Toc169799827)

[1.6 Логическое моделирование 11](#_Toc169799828)

[1.7 Описание структуры базы данных 13](#_Toc169799829)

[1.8 Общие требования к программному продукту **Ошибка! Закладка не определена.**](#_Toc169799830)

[2 Экспериментальный раздел 21](#_Toc169799831)

[2.1 Описание программы 21](#_Toc169799832)

[2.2 Протокол тестирования программного продукта 26](#_Toc169799833)

[ЗАКЛЮЧЕНИЕ 42](#_Toc169799834)

[Приложение А 43](#_Toc169799835)

[Приложение Б 46](#_Toc169799836)

[Приложение В 48](#_Toc169799837)

[Приложение Г **Ошибка! Закладка не определена.**](#_Toc169799838)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 51](#_Toc169799839)

## ВВЕДЕНИЕ

С развитием интернет-технологий и цифровизации контента музыкальная индустрия столкнулась с необходимостью адаптации к новым реалиям. В настоящее время потребители музыкального контента все больше обращаются к онлайн-платформам для поиска, прослушивания и организации своей музыкальной библиотеки. В этом контексте создание музыкального приложения становится ключевым для удовлетворения запросов современного пользователя.

Данная работа представляет собой комплексный аналитический и практический проект, направленный на создание инновационного музыкального приложения, способного эффективно конкурировать на рынке цифрового контента.

Целью данной работы является проектирование и разработка приложения, предоставляющего возможность прослушивания музыки, создания персональных плейлистов, организации музыкальной библиотеки и других функций, характерных для современных музыкальных платформ.

С развитием технологий и доступом к интернету количество пользователей, предпочитающих потребление музыкального контента в онлайн-формате, постоянно растет. Музыкальные стриминговые платформы становятся основным источником для прослушивания и открытия новых музыкальных произведений. Именно поэтому задача создания собственного музыкального приложения становится актуальной, так как позволяет удовлетворить потребности пользователей в удобном доступе к музыкальному контенту.

Для достижения поставленной цели необходимо выполнить следующие задачи:

* изучить предметную область;
* описать предметную область;
* спроектировать базу данных;
* разработать дизайн приложения;
* разработать функции регистрации, авторизации;
* реализовать функции для пользователей: прослушивание треков, добавление их в медиатеку, создание плейлистов;
* разработать и протестировать приложение.

## Постановка задачи

## 1.1 Описание предметной области

Требуется разработать справочную системы для прослушивания музыкальных композиций, просмотра информации об исполнителях, создания и управления плейлистами, а также для возможности просматривать и оценивать музыку, оставлять отзывы и следить за рейтингами. Это создаст удобное пространство для взаимодействия с музыкой и сообществом, включая как персональные рекомендации, так и топовые композиции.

Данная информационная система предполагает наличие двух групп пользователей: авторизированный пользователь и неавторизированный пользователь.

Авторизированный (аутентифицированный) пользователь имеет возможность:

* прослушивать музыкальные композиции;
* добавлять музыкальные композиции в личную медиатеку;
* создавать плейлисты;
* добавлять треки в плейлисты;
* просматривать информацию об альбомах и плейлистах других пользователях;
* просматривать информацию об исполнителях и жанрах;
* входить в систему под своим аккаунтом;
* просматривать информацию о своём профиле.

Неавторизированный пользователь имеет возможность:

* прослушивать музыкальные композиции;
* просматривать информацию об альбомах и плейлистах других пользователях;
* просматривать информацию об исполнителях и жанрах;
* создать аккаунт в системе.

Пользователь имеет возможность создавать аккаунт в музыкальном приложении, просматривать, воспроизводить треки, альбомы и плейлисты. Для использования приложения пользователь должен авторизоваться или зарегистрироваться. Каждый пользователь будет характеризоваться следующими параметрами:

* идентификатор пользователя;
* имя пользователя;
* логин пользователя;
* пароль пользователя;
* аватар пользователя;
* роль пользователя (аутентифицированный/не аутентифицированный/администратор).

В приложении каждый пользователь имеет возможность прослушивать треки. Любой трек обладает следующими характеристиками:

* идентификатор трека;
* название трека;
* идентификатор альбома;
* длительность;
* жанр трека;
* ссылка на аудиофайл;
* ссылка на обложку.

Любой трек имеет своего исполнителя или исполнителей, которые в свою очередь имеют следующие характеристики:

* идентификатор исполнителя;
* имя исполнителя;
* ссылка на фотографию;
* описание исполнителя;
* количество просмотров исполнителя.

Некоторые треки содержатся в альбоме – сборнике музыкальных композиций одного исполнителя. Основные атрибуты альбома включают:

* идентификатор альбома;
* название альбома;
* исполнитель альбома;
* обложка альбома;
* дата выпуска.

Пользователь имеет возможность создавать плейлист, который представляет собой набор треков, сгруппированных пользователем по определенному критерию или тематике. Каждый плейлист содержит следующие данные:

* идентификатор плейлиста;
* идентификатор создателя;
* название плейлиста;
* обложка плейлиста;
* описание плейлиста;
* дата создания плейлиста;
* идентификатор пользователя;
* тип плейлиста (приватный/публичный).

Каждый пользователь имеет возможность добавлять треки в свою медиатеку. Медиатека – это персональное пространство пользователя в музыкальном приложении, где он может хранить и организовывать свою музыкальную коллекцию. В медиатеку входят:

* идентификатор медиатеки;
* идентификатор пользователя.

Жанр представляет собой классификацию музыкальных композиций по общим стилистическим чертам. Любой трек имеет один или несколько жанров. Каждый жанр имеет следующие характеристики:

* идентификатор жанра;
* название жанра;
* описание.

Ограничения на информацию в системе:

* пользователь должен быть старше 7 лет;
* пароль должен содержать больше 6 символов;
* логины и имена пользователей не могут совпадать.

## 1.2 Описание входной информации

Входной информацией для выполнения задачи являются:

* информация о пользователях;
* информация о плейлистах;
* справочники музыкальных композиций.

## 1.3 Описание выходной информации

Выходной информацией будут являться:

* статистика популярных артистов и треков;
* список новинок;
* рекомендованные для пользователей плейлисты;
* отзывы пользователей;
* информация о плейлистах;
* медиатеки пользователей.

Шаблоны выходной информации описаны в приложении А.

## 1.4 Диаграмма прецедентов

Диаграмма прецедентов является одним из видов диаграмм UML и используется для описания функциональности системы с точки зрения ее пользователей. Она показывает, какие действия (прецеденты) могут быть выполнены пользователями и как они взаимодействуют с системой.

На диаграмме прецедентов используются следующие элементы:

* актеры - пользователи или внешние системы, взаимодействующие с системой;
* прецеденты - описывают функциональные возможности системы и действия, которые могут быть выполнены пользователями или актерами;
* отношения - показывают связи между актерами и прецедентами, а также между различными прецедентами.

Диаграмма прецедентов помогает понять основные функциональные требования к системе, определить основные сценарии использования и взаимодействие пользователей с системой. Она является важным инструментом в процессе анализа требований и проектирования информационных систем.

Диаграмма прецедентов представлена в рисунке 1.4.1

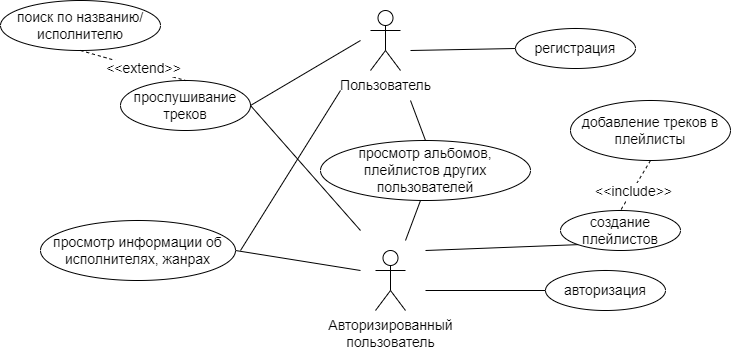


Рисунок 1.4.1 - Диаграмма прецедентов

## 1.5 Концептуальное моделирование

Концептуальная модель базы данных - это абстрактное представление структуры данных и их взаимосвязей в информационной системе. Она описывает основные сущности, их атрибуты и связи между этими сущностями.

Концептуальная модель базы данных обычно создается на раннем этапе проектирования информационной системы и служит основой для дальнейшего проектирования и разработки физической структуры базы данных. Основные элементы, используемые в концептуальной модели, включают в себя:

* сущности: представляют различные объекты или концепции, которые нужно хранить в базе данных;
* атрибуты: описывают свойства сущностей и характеристики, которые необходимо хранить;
* связи: определяют отношения между различными сущностями.

Концептуальная модель базы данных помогает разработчикам и аналитикам понять структуру и логику хранения данных в информационной системе, а также обеспечить консистентность и эффективность базы данных. Она является основой для создания физической модели базы данных и дальнейшей реализации базы данных на конкретной платформе или с использованием определенных технологий.

Концептуальная модель базы данных представлена в рисунке 1.5.1

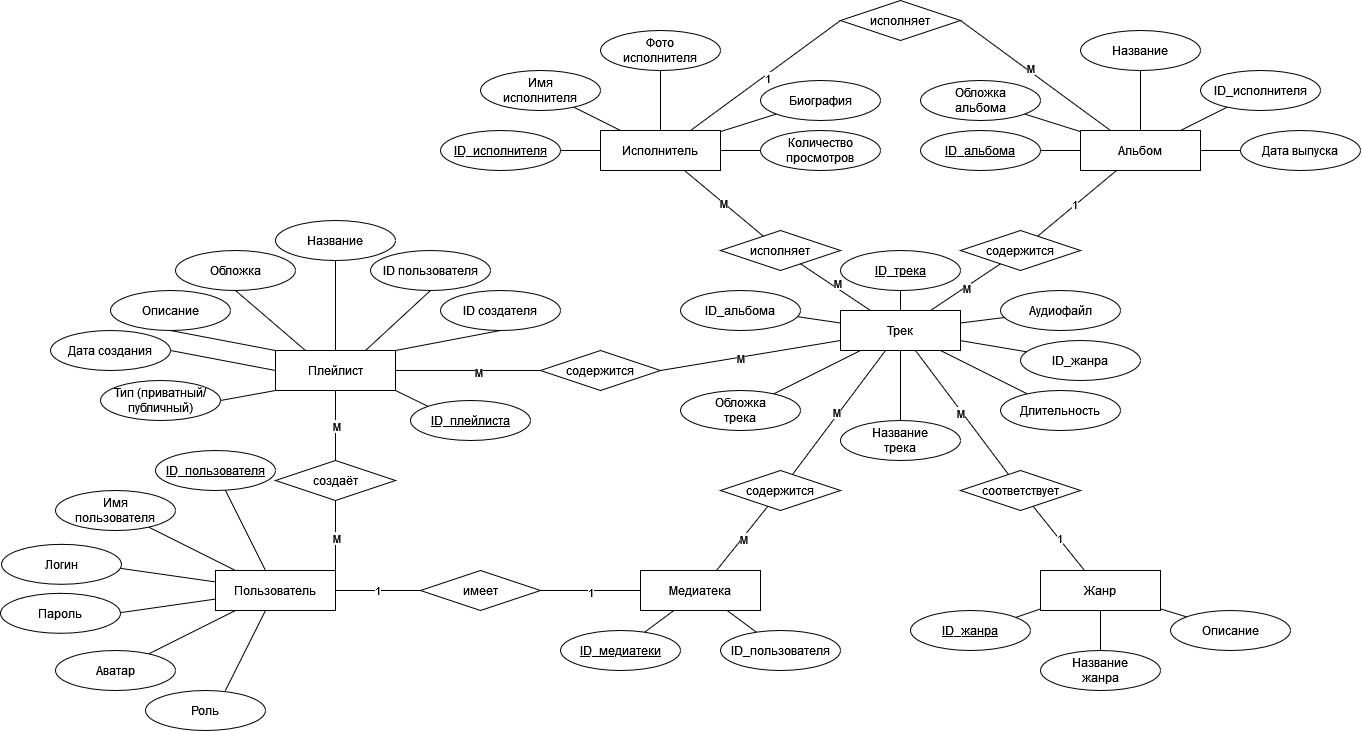


Рисунок 1.5.1 - Концептуальная модель БД

## 1.6 Логическое моделирование

Логическая модель базы данных представляет собой более конкретное описание структуры данных, которое включает в себя определение таблиц, полей, ключей и связей между таблицами. В отличие от концептуальной модели, логическая модель описывает данные с точки зрения конкретной СУБД и учитывает их специфичные особенности.

В логической модели базы данных используются следующие элементы:

* таблицы: представляют собой структурированные наборы данных, организованные в виде строк и столбцов. Каждая таблица имеет свое имя и набор полей;
* поля: описывают отдельные атрибуты сущностей и хранят конкретные значения. Каждое поле имеет свое имя, тип данных и другие свойства;
* ключи: используются для идентификации уникальных записей в таблице и для установления связей между таблицами. В логической модели могут использоваться первичные ключи, внешние ключи и другие типы ключей;
* связи: определяют отношения между таблицами и указывают на связи между сущностями.

Логическая модель базы данных позволяет перейти от абстрактного описания данных к их конкретной структуре, что облегчает процесс проектирования и реализации базы данных.

Схема данных – это структура базы данных, описанная на формальном языке, поддерживаемом СУБД (системой управления базы данных). В реляционных базах данных схема определяет таблицы, поля в каждой таблице и ограничения целостности, такие как первичный и внешний ключи.

Схема данных представлена на рисунке 1.6.1

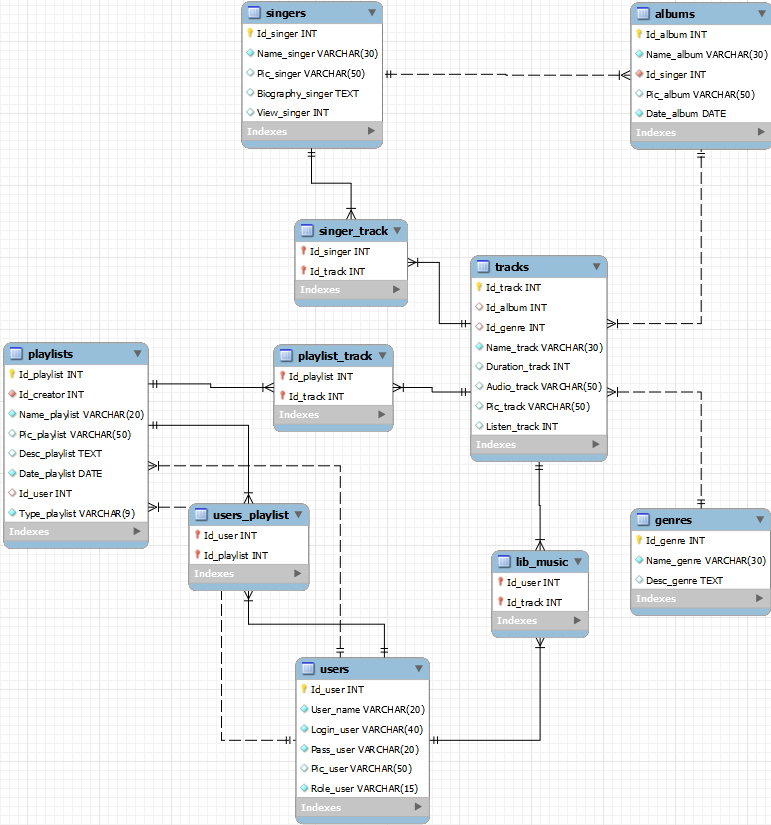


Рисунок 1.6.1 – Схема базы данных

## 1.7 Описание структуры базы данных

Описание структуры базы данных представлено в таблицах 1.7.1 – 1.7.8.

Таблица 1.7.1 - Users (пользователи)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_user | ID пользователя | INT |  | PK |
| User\_name | Имя пользователя | VARCHAR | 20 |  |
| Login\_user | Логин | VARCHAR | 40 |  |
| Pass\_user | Пароль | VARCHAR | 20 |  |
| Pic\_user | Аватар | LONGTEXT |  |  |
| Role\_user | Роль | VARCHAR | 15 |  |

Таблица 1.7.2 - Lib\_music (медиатеки пользователей)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_lib | ID медиатеки | INT |  | FK |
| Id\_user | ID пользователя | INT |  | FK |

Таблица 1.7.3 - Lib-track (промежуточная между медиатекой и треками)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_lib | ID медиатеки | INT |  | FK |
| Id\_track | ID трека | INT |  | FK |

Таблица 1.7.4 - Singers (исполнители)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_singer | ID исполнителя | INT |  | PK |
| Name\_singer | Имя исполнителя | VARCHAR | 30 |  |
| Pic\_singer | Фото исполнителя | LONGTEXT |  |  |
| Biography\_  singer | Биография | LONGTEXT |  |  |
| View\_singer | Количество просмотров | INT |  |  |

Таблица 1.7.5 - Albums (альбомы)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_album | ID альбома | INT |  | PK |
| Name\_album | Название | VARCHAR | 30 |  |
| Id\_singer | ID исполнителя | INT |  | FK |
| Pic\_album | Обложка | LONGTEXT |  |  |
| Date\_album | Дата выпуска | DATE |  |  |

Таблица 1.7.6 - Tracks (треки)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_track | ID трека | INT |  | PK |
| Id\_album | ID альбома | INT |  | FK |
| Id\_genre | ID жанра | INT |  | FK |
| Name\_track | Название | VARCHAR | 30 |  |
| Duration\_  track | Длительность | TIME |  |  |
| Audio\_track | Аудиофайл | LONGTEXT |  |  |
| Pic\_track | Обложка | LONGTEXT |  |  |

Таблица 1.7.7 – Singer-track (промежуточная между исполнителями и треками)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_singer | ID исполнителя | INT |  | FK |
| Id\_track | ID трека | INT |  | FK |

Таблица 1.7.8 - Genres (жанры)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_genre | ID жанра | INT |  | PK |
| Name\_genre | Название | VARCHAR | 30 |  |
| Desc\_genre | Описание | LONGTEXT |  |  |

Таблица 1.7.9 - Playlists (плейлисты)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_playlist | ID плейлиста | INT |  | PK |
| Id\_creator | ID создателя | INT |  | FK |
| Name\_playlist | Название | VARCHAR | 20 |  |
| Pic\_playlist | Обложка | LONGTEXT |  |  |
| Desc\_playlist | Описание | LONGTEXT |  |  |
| Date\_playlist | Дата создания | DATE |  |  |
| Type\_playlist | Тип (приватный/  публичный) | VARCHAR | 9 |  |

Таблица 1.7.10 - Users-Playlists (промежуточная между пользователями и плейлистам)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_user | ID пользователя | INT |  | FK |
| Id\_playlist | ID плейлиста | INT |  | FK |

Таблица 1.7.11 – Playlist-Track (промежуточная между плейлистами и треками)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа (РК первичный,  FK внешний) |
| Id\_playlist | ID плейлиста | INT |  | FK |
| Id\_track | ID трека | INT |  | FK |

## 1.8 Контрольный пример

Контрольный пример представляет собой ручной подсчёт задачи. Программа обрабатывает исходные данные, соответствующие контрольному примеру. Полученные результаты сравниваются с известными результатами контрольного примера. При несовпадении результатов производится поиск и исправление ошибок, после чего программа запускается повторно.

Входные данные контрольного примера приведены в приложении Б, а выходные данные — в приложении В.

## 1.9 Общие требования к программному продукту

Требования к пользователям:

* пользователи должны иметь базовые навыки работы с персональным компьютером.

Минимальные требования к техническому обеспечению:

* операционная система: Windows 7 Service Pack 1 / Windows 7 64Bit Service Pack 1 / Windows 8.1 64Bit / Windows 10 64Bit / Windows 10 64Bit / Mac OS;
* процессор: 2.40 ГГц (четырёхъядерный) / AMD Phenom 9850 (четырёхъядерный) @ 2.5 ГГц;
* оперативная память: 512 МБ (Win 7/Win 8 и выше);
* видеокарта: NVIDIA 9800 GT с 512 МБ видеопамяти / AMD HD 4870 с 1 ГБ видеопамяти (DX 9, 10, 10.1);
* жёсткий диск: 10 ГБ свободного места;
* Microsoft DirectX: версия 9.0c.

Функциональные возможности:

* приложение должно формировать и отображать выходные данные пользователю;
* в приложении должен быть обеспечен просмотр таблиц (справочников) базы данных с возможностью добавления, редактирования и удаления данных.

Требования к надёжности:

* приложение должно обрабатывать ошибочные действия пользователя и сообщать ему об этом;
* приложение должно обеспечивать контроль входной и выходной информации;
* требования к информационной и программной совместимости;
* приложение должно работать с таблицами СУБД MySQL.

## Экспериментальный раздел

## Описание программы

MVVM (Model-View-ViewModel) — это паттерн проектирования, который разделяет логику приложения на три компонента: Model (модель), View (представление) и ViewModel (модель представления). Model содержит данные и логику, View отвечает за отображение интерфейса пользователя, а ViewModel связывает View с Model, обеспечивая передачу данных и команд.

В данном проекте паттерн MVVM использовался для упрощения разработки, разделения ответственности. Также программа имеет модульную структуру. При ее запуске выполняется проект Nover.exe. Схема взаимодействия модулей программы представлена на рисунке 2.1.1. Описание модулей и методов представлено в таблице 2.1.1.

Бочку перевернуть на 90 градусов.

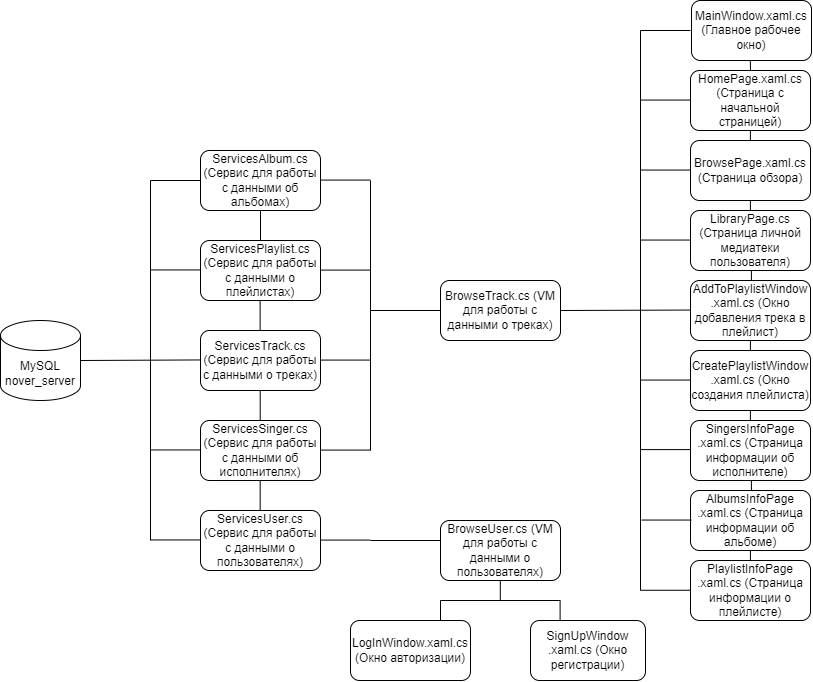


Рисунок 2.1.1 – Схема взаимодействия модулей

Таблица 2.1.1 – Описание модулей

|  |  |
| --- | --- |
| Методы | Назначение |
| 1 | 2 |
| BrowseTrack.cs – model для работы с треками | |
| private void UpdateTrackList() | Обновление списка треков |

Продолжение таблицы 2.2.1

|  |  |
| --- | --- |
| 1 | 2 |
| public static List<Track> GetTracksCurrentUser() | Получение треков из личной медиатеки пользователя |
| public static void OpenHomePageMethod() | Переход к главной странице приложения |
| public static void OpenBrowsePageMethod() | Переход к странице обзора |
| private static void PlayTrackMethod(Track selectedTrack, BrowseTrack browseTrack) | Проигрывание трека |
| private static void MediaPlayer\_MediaEnded(object sender, EventArgs e) | Проигрывание следующего трека, если нынешний закончился |
| private void PlayNextTrack() | Воспроизведение следующего трека |
| private void ResumeTrackMethod() | Метод для продолжения воспроизведения трека с момента паузы |
| public static void OpenLogInWindowMethod() | Переход к окну регистрации пользователя |
| private static void OpenSingerInfoPageMethod() | Переход к странице с информацией о выбранном исполнителе |
| private void OpenAlbumsInfoPageMethod() | Переход к странице с информацией о выбранном альбоме |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void OpenPlaylistsInfoPageMethod() | Переход к странице с информацией о выбранном плейлисте |
| private void OpenAddToPlaylistWindow() | Переход к окну добавления трека в плейлист или создания плейлиста |
| public void OpenCreatePlaylistWindowMethod() | Переход к окну создания плейлиста |
| private void OpenFileDialogMethod() | Переход к окну выбора фотографии |
| BrowseUser.cs | |
| public static List<User> GetUsers() | Получение списка пользователей из базы данных |
| public static void SignUpMethod(User newUser) | Регистрация пользователя в базу данных |
| public static void UpdateUser(User user) | Обновление информации о пользователе |
| public static void AddToLibraryMethod(Track track) | Добавление трека в личную медиатеку пользователя в список |
| public static void AddTracksToUserLibrary(User user, List<Track> tracks) | Добавление трека в личную медиатеку пользователя в базу данных |
| public static void RemoveTrackFromLibrary(User user, Track track) | Удаление трека из личной медиатеки пользователя |
| public static List<Playlist> GetPlaylistsCurrentUser() | Получение плейлистов для текущего пользователя |
| public static List<User> GetUsers() | Получение списка пользователей из базы данных |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| private void LogOutMethod() | Выход из аккаунта |
| ServicesAlbum.cs |  |
| public static List<Album> GetAlbums() | Метод для получения списка альбомов из базы данных |
| ServicesPlaylist.cs – model для работы с плейлистами | |
| public static List<Playlist> GetPlaylists() | Метод для получения списка плейлистов из базы данных |
| public static void AddPlaylistMethod(Playlist NewPlaylist) | Метод для добавления плейлиста в базу данных |
| public static void AddToPlaylistMethod() | Метод для добавления трека в плейлист в базу данных |
| ServicesSinger.cs | |
| public static List<Singer> GetSingers() | Метод для получения списка исполнителей из базы данных |
| public static void UpdateSingerView(Singer singer) | Метод для обновления количества просмотров у исполнителя |
| ServicesTrack.cs | |
| public static List<Track> GetTracks() | Получение списка треков из базы данных |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| public static List<Genre> GetGenres() | Получение списка жанров трека из базы данных |
| public static void UpdateListenTrack(Track track) | Обновление количества прослушивания трека |
| public static List<String> GetTypesPlaylists() | Получение списка типов плейлистов |
| ServicesUser.cs – model для работы с данными о пользователях | |
| public static List<User> GetUsers() | Получение списка пользователей из базы данных |
| public static void SignUpMethod(User newUser) | Регистрация пользователя в базу данных |
| public static void UpdateUser(User user) | Обновление информации о пользователе |
| public static void AddToLibraryMethod(Track track) | Добавление трека в личную медиатеку пользователя в список |
| public static void AddTracksToUserLibrary(User user, List<Track> tracks) | Добавление трека в личную медиатеку пользователя в базу данных |
| public static void RemoveTrackFromLibrary(User user, Track track) | Удаление трека из личной медиатеки пользователя |
| public static List<Genre> GetGenres() | Получение списка жанров трека из базы данных |
| public static void UpdateListenTrack(Track track) | Обновление количества прослушивания трека |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| public static List<String> GetTypesPlaylists() | Получение списка типов плейлистов |

Код программы представлен в приложении Г.

## Протокол тестирования программного продукта

В протоколе тестирования отражаются:

* тестирование на корректных данных;
* тестирование на некорректных данных;
* тестирование продукта на данных контрольного примера.

Общая информация о тестировании представлена в таблице 2.2.1

Таблица 2.2.1 – Тестирование добавления пользователя с корректными данными

|  |  |
| --- | --- |
| 1 | 2 |
| Test Case # | 1 |
| Приоритет тестирования | Высокий |
| Название тестирования | Тестирование добавления пользователя |
| Резюме испытания | Необходимо достичь корректного добавления пользователя |
| Шаги тестирования | - Перейти на страницу регистрации пользователя;  - Заполнить корректно все поля;  - Нажать «Зарегистрироваться». |
| Данные тестирования | Имя пользователя: assa  Логин: assasasa@gmail.com  Пароль: sdaos  Фото профиля: 860281.png |

Продолжение таблицы 2.2.1

|  |  |
| --- | --- |
| 1 | 2 |
| Ожидаемый результат | Отобразится уведомление об успешном добавлении пользователя. Пользователь добавится в БД. |
| Фактический результат | Отобразилось уведомление об успешном добавлении пользователя. Пользователь добавлен в БД. |
| Предпосылки | Программа запущена |
| Постусловия | Осуществится переход на главное окно приложения |
| Статус | Pass |
| Комментарии | Результат тестирования представлен на рисунке 2.2.1. |
| Ожидаемый результат | Отобразится уведомление об успешном добавлении пользователя. Пользователь добавится в БД. |

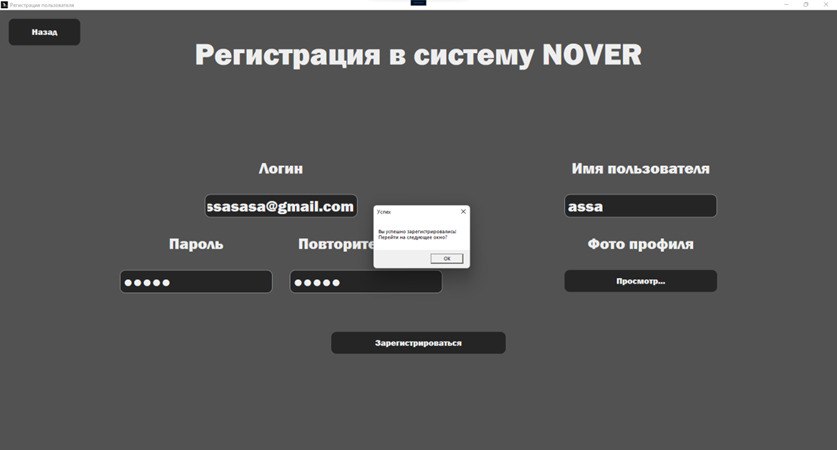


Рисунок 2.2.1 – Сообщение о добавлении пользователя

Таблица 2.2.2 – Тестирование добавления пользователя с пустыми полями

|  |  |
| --- | --- |
| 1 | 2 |
| Test Case # | 2 |
| Приоритет тестирования | Средний |
| Название тестирования | Тестирование добавления пользователя с пустыми полями |
| Резюме испытания | Нужно достичь отображения ошибки о пустых полях |
| Шаги тестирования | - Перейти на страницу добавления клиента;  - Оставить поля пустыми;  - Нажать «Зарегистрироваться». |
| Данные тестирования | Имя пользователя:  Логин:  Пароль:  Фото профиля: |
| Ожидаемый результат | Отобразится сообщение о пустых полях |
| Фактический результат | Отобразилось сообщение о пустых полях |
| Предпосылки | Программа запущена; |
| Постусловия | Останется та же страница |
| Статус | Pass |
| Комментарии | Результат тестирования представлен на рисунке 2.2.2. |

Тестирование на добавление клиента при пустых полях, ожидаемое сообщение: «Все поля должны быть заполнены!» (рисунок 2.2.2).

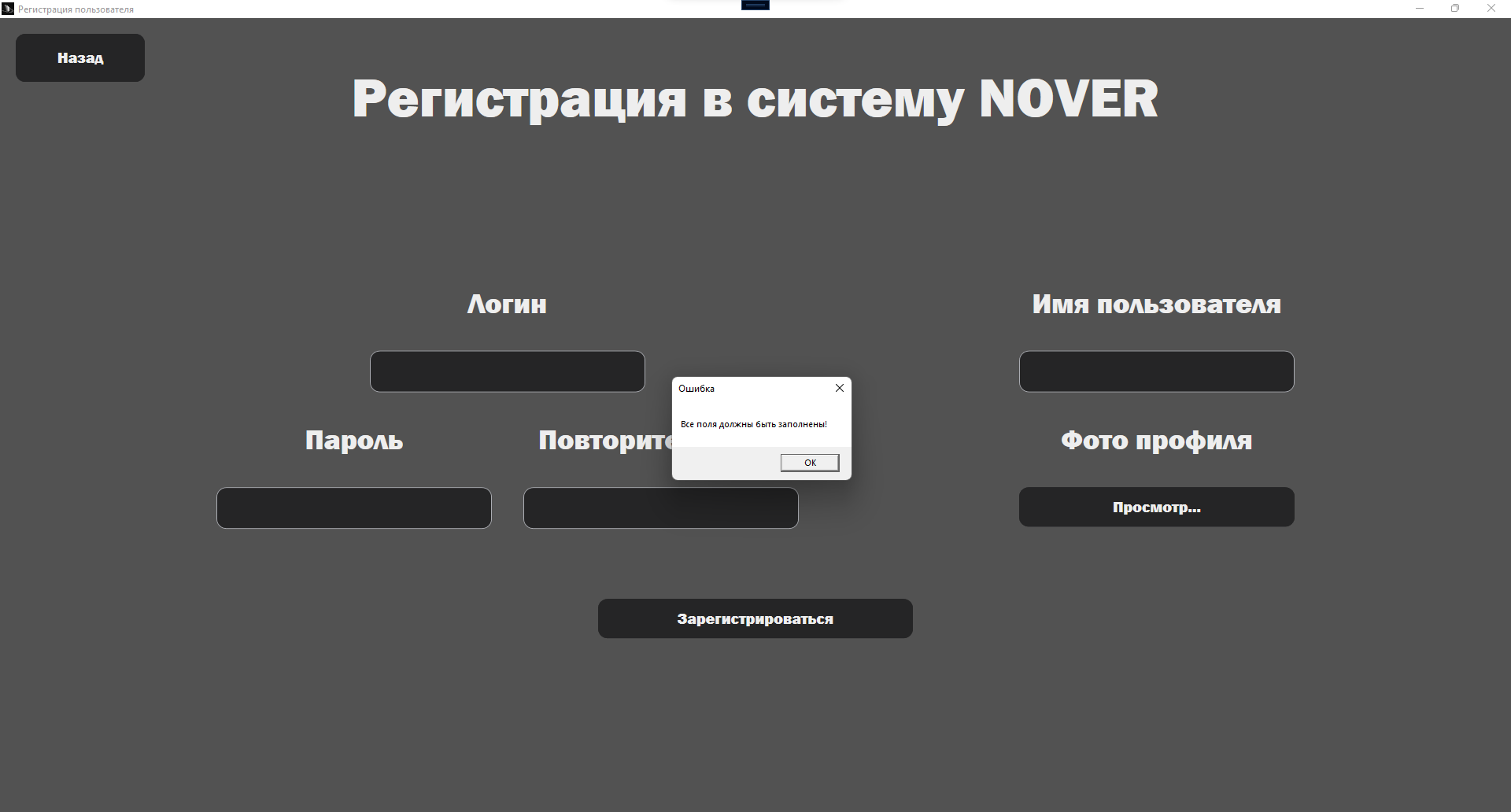


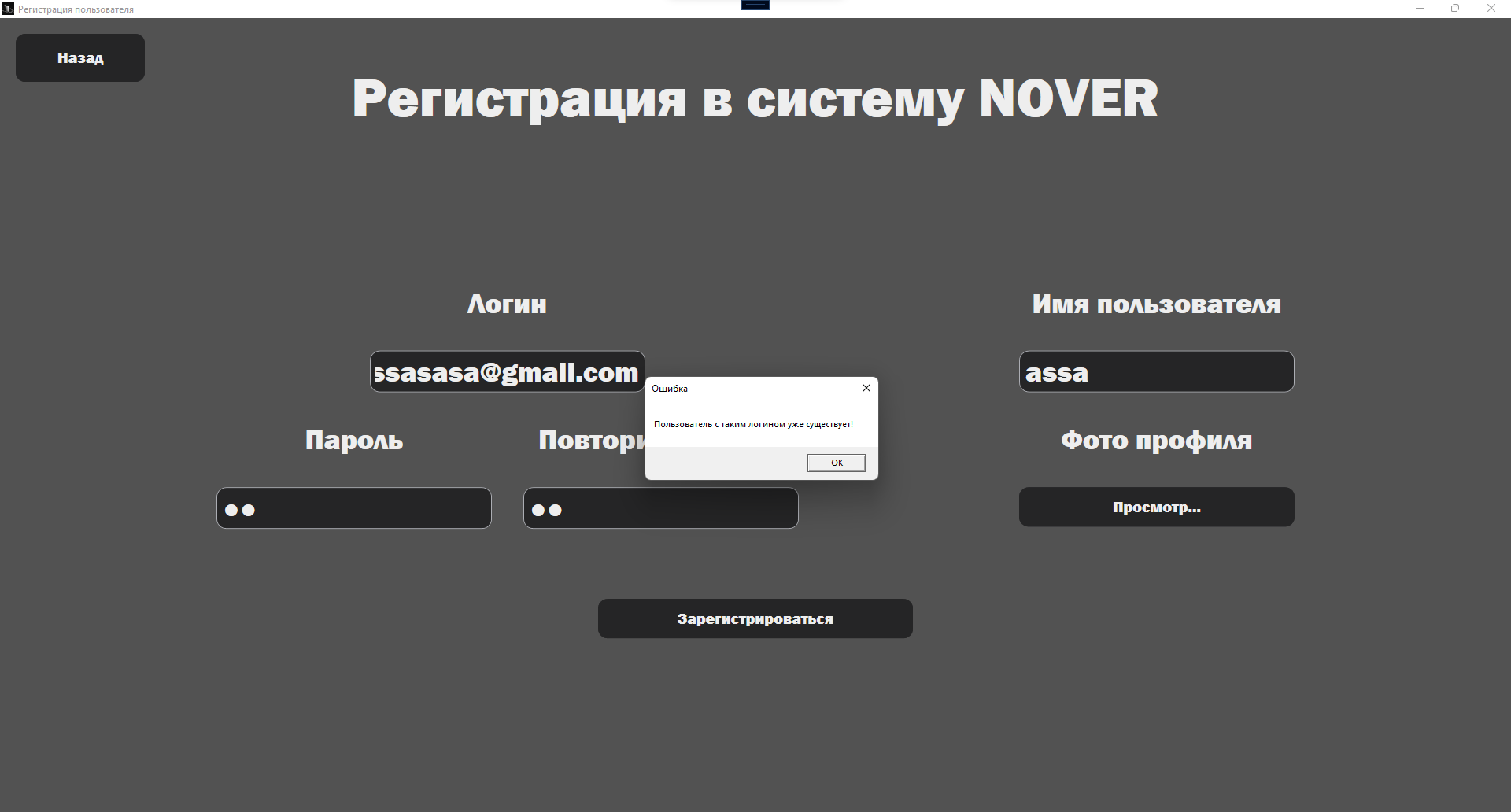
Рисунок 2.2.2 – Сообщение о пустых полях

Таблица 2.2.3 – Тестирование добавления клиента с существующими данными

|  |  |
| --- | --- |
| 1 | 2 |
| Test Case # | 3 |
| Приоритет тестирования | Средний |
| Название тестирования | Тестирование добавления пользователя с существующими данными |
| Резюме испытания | Нужно достичь отображения ошибки о существующем пользователе |
| Шаги тестирования | - Перейти на страницу добавления клиента;  - Заполнить поля существующими данными;  - Нажать «Добавить». |

Продолжение таблицы 2.2.3

|  |  |
| --- | --- |
| 1 | 2 |
| Данные тестирования | Имя пользователя: assa  Логин: assasasa@gmail.com  Пароль: sdaos  Фото профиля: 860281.png |
| Ожидаемый результат | Отобразится сообщение о существующем пользователе |
| Фактический результат | Отобразилось сообщение о существующем пользователе |
| Предпосылки | Программа запущена; |
| Постусловия | Останется та же страница |
| Статус | Pass |
| Комментарии | Результат тестирования представлен на рисунке 2.2.3. |
| Данные тестирования | Имя пользователя: assa  Логин: assasasa@gmail.com  Пароль: sdaos  Фото профиля: 860281.png |

Рисунок 2.2.3 – Сообщение, что такой пользователь уже существует

* 1. Руководство пользователя

#### Назначение системы

Программа предназначена для упрощения управления базой данных музыкальных композиций и информации об исполнителях. Основной целью данной информационной системы является предоставление пользователям возможности прослушивания музыки, создания и управления личной медиатекой, плейлистами и получения информации об исполнителях и жанрах.

#### Условия применения системы

Программное обеспечение разрабатывается для персональной вычислительной техники со следующими характеристиками:

* Microsoft Windows 7 / 8 / 10
* процессор 1 ГГц
* 128 МБ ОЗУ
* 60 МБ свободного пространства на диске
* разрешение экрана монитора не менее 1920 × 1080

Программа предназначена для пользователей, имеющих как минимум первоначальные навыки работы с графической операционной системой.

#### Подготовка системы к работе

Для запуска программы необходимо запустить приложение NOVER.exe из каталога, в котором установлен данный программный продукт. После этого открывается главное окно с главной страницей (рисунок 2.3.1).

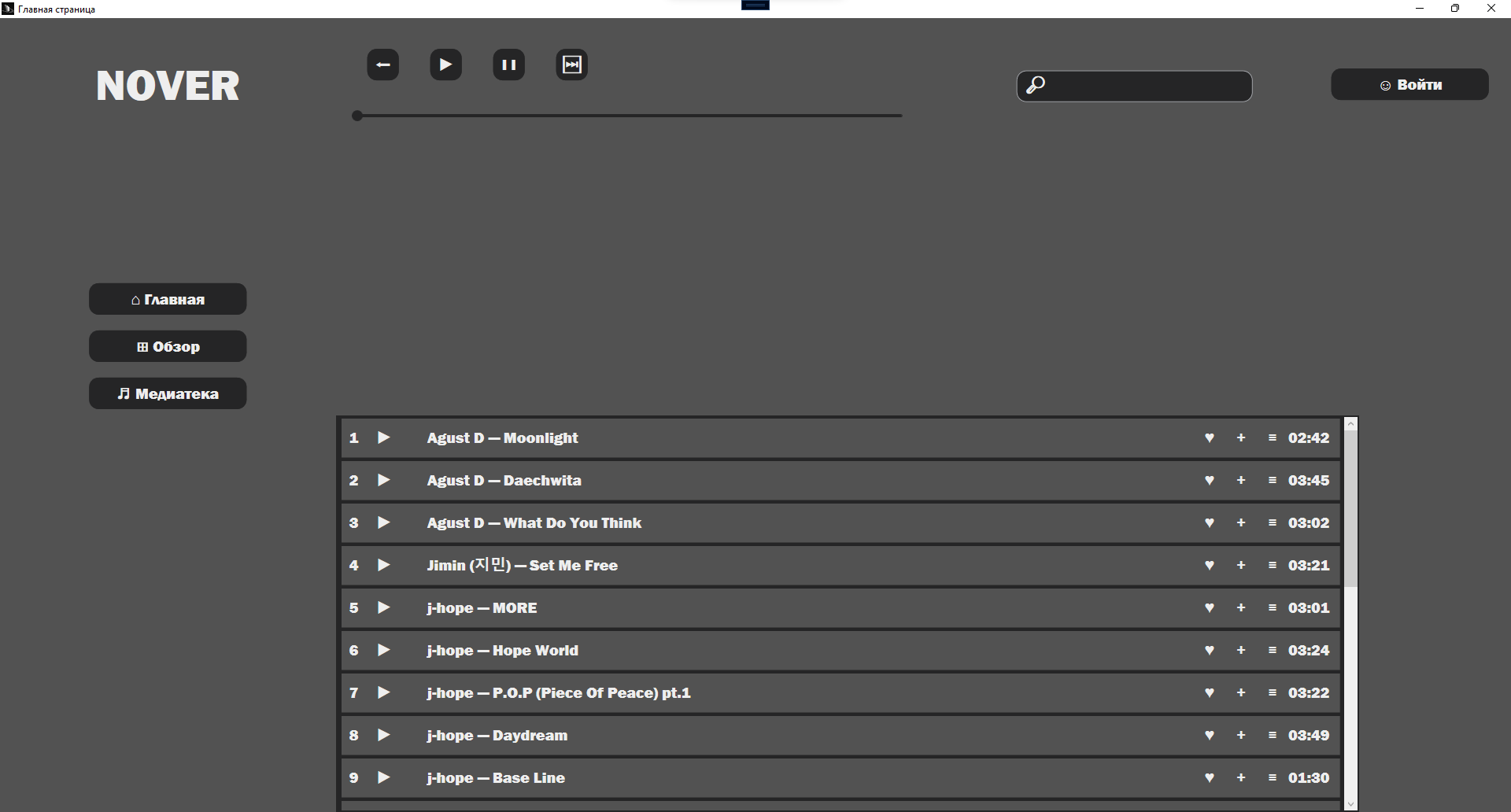


Рисунок 2.3.1 - Главное окно программы

Описание операций

После загрузки приложения пользователь может выбрать определенную вкладку: Медиатека, если пользователь авторизирован, Обзор или вернуться к странице «Главная» (рисунки 2.3.1-2.3.3).

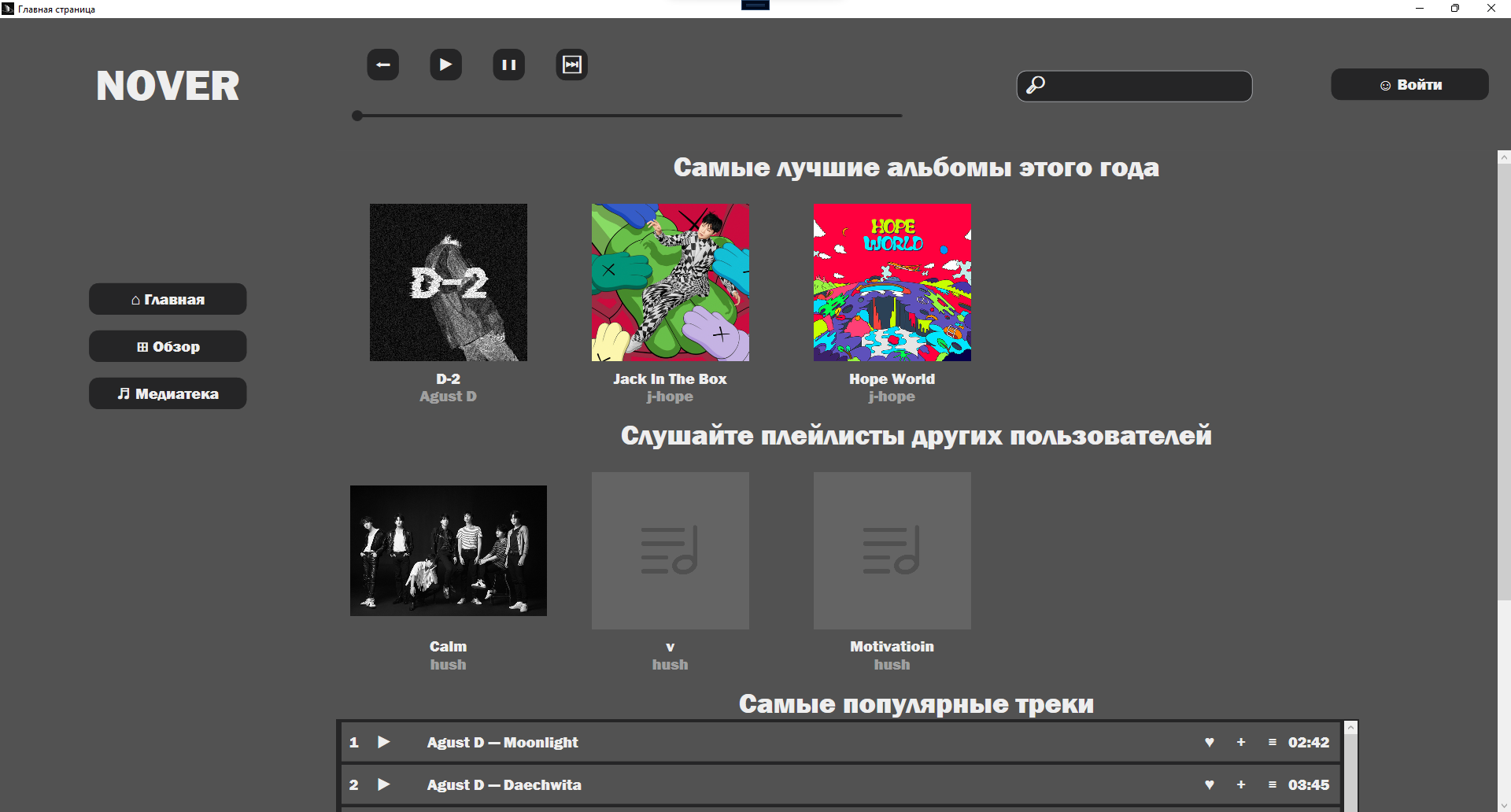


Рисунок 2.3.2 - Окно «Обзор»

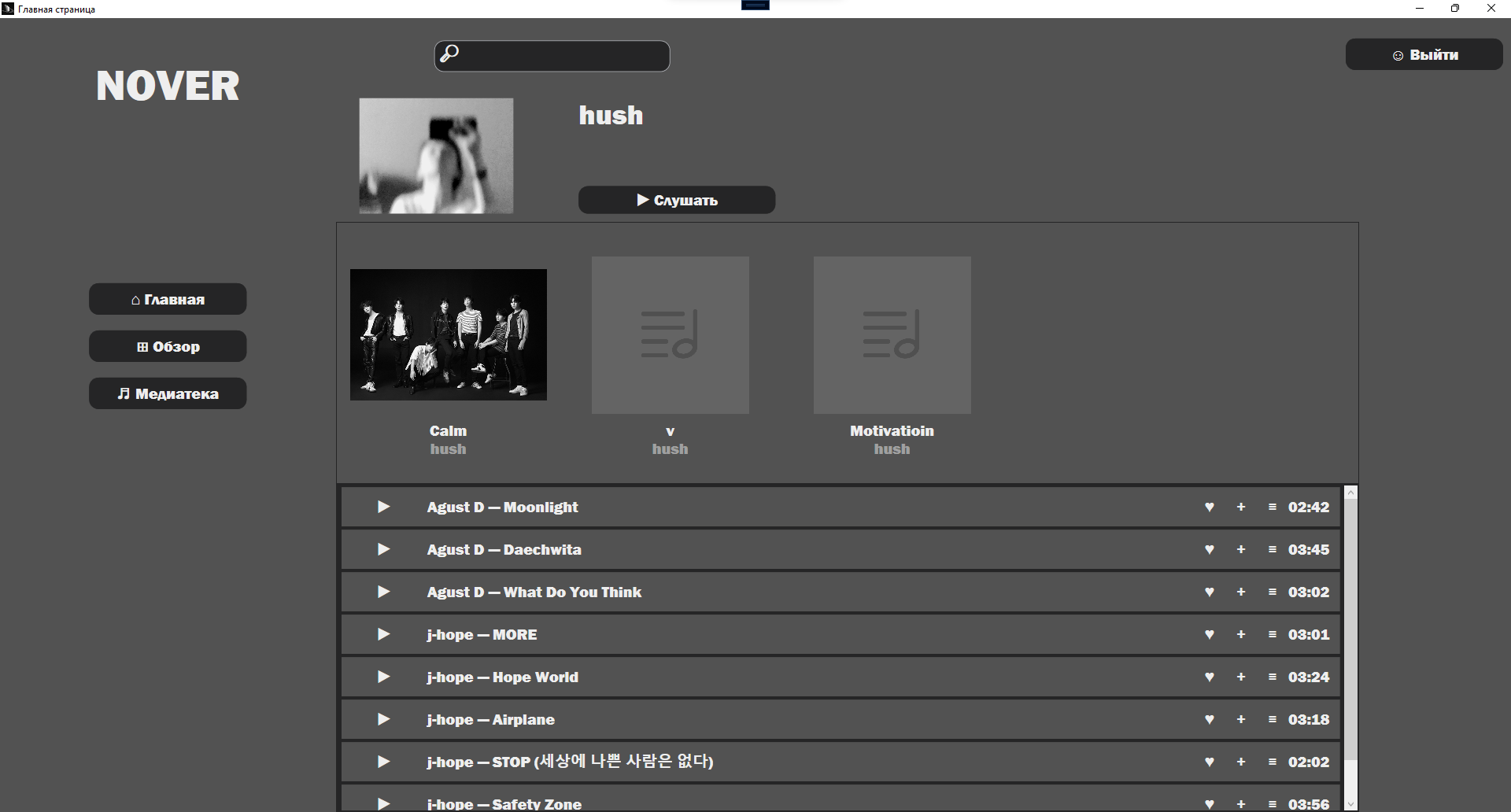


Рисунок 2.3.3 - Окно «Медиатека»

На вкладке «Главная» можно воспроизвести трек, выбрав его в списке музыкальных композиций и нажав на кнопку «▶». После этого будет воспроизведена выбранная пользователем песня.

Также пользователь может добавить трек в медиатеку, нажав на кнопку «♥». После этого трек добавится в личную медиатеку пользователя и появится всплывающее сообщение, представленное на рисунке 2.3.4.

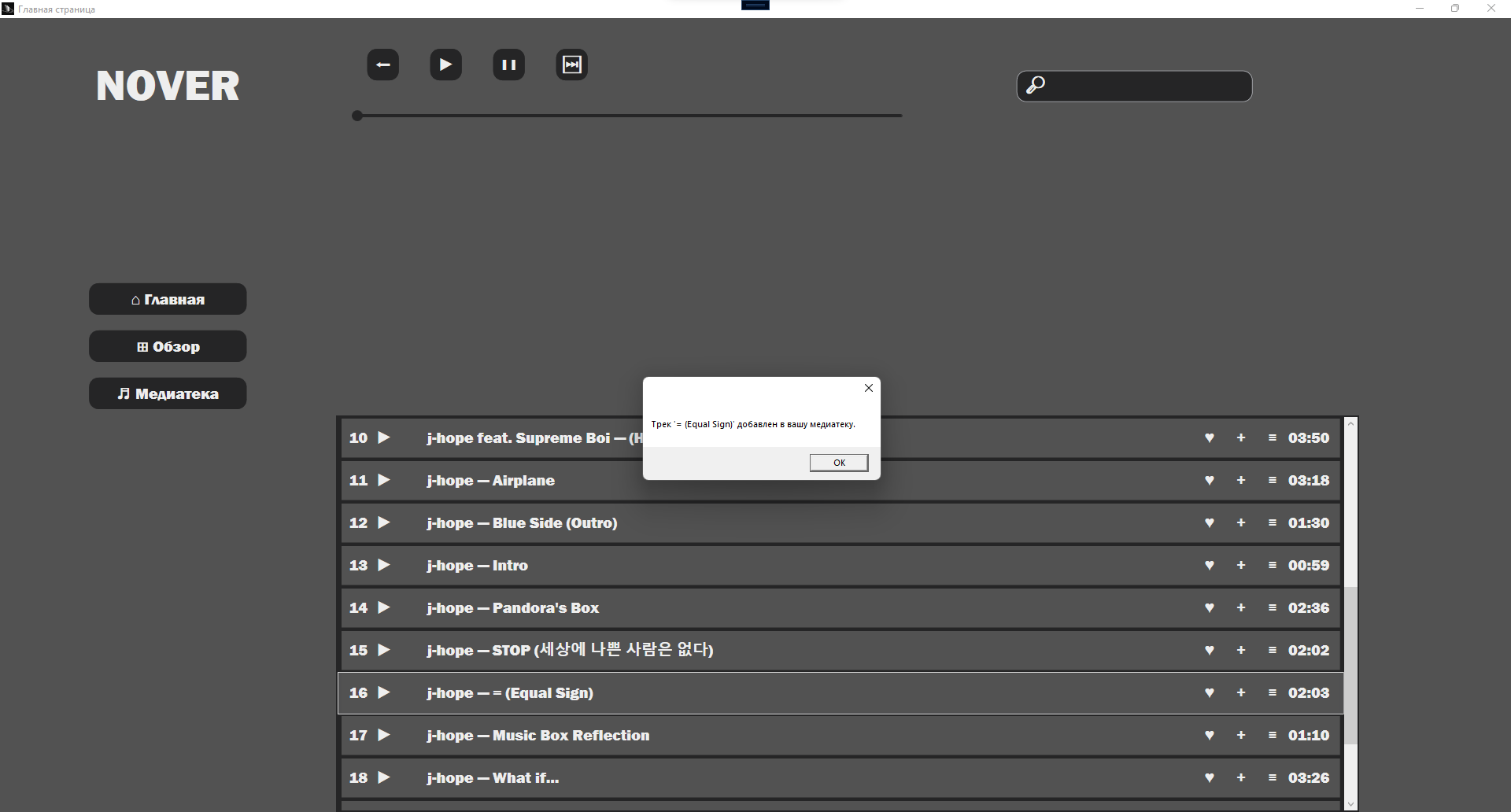


Рисунок 2.3.4 - Уведомление о том, что трек добавлен в медиатеку

Если же трек уже присутствует в медиатеке пользователя, а он нажал кнопку «♥», то тогда появится сообщение, где пользователь будет информирован о присутствии трека в медиатеке, и предоставится выбор удалить или оставить трек в медиатеке (рисунок 2.3.5).

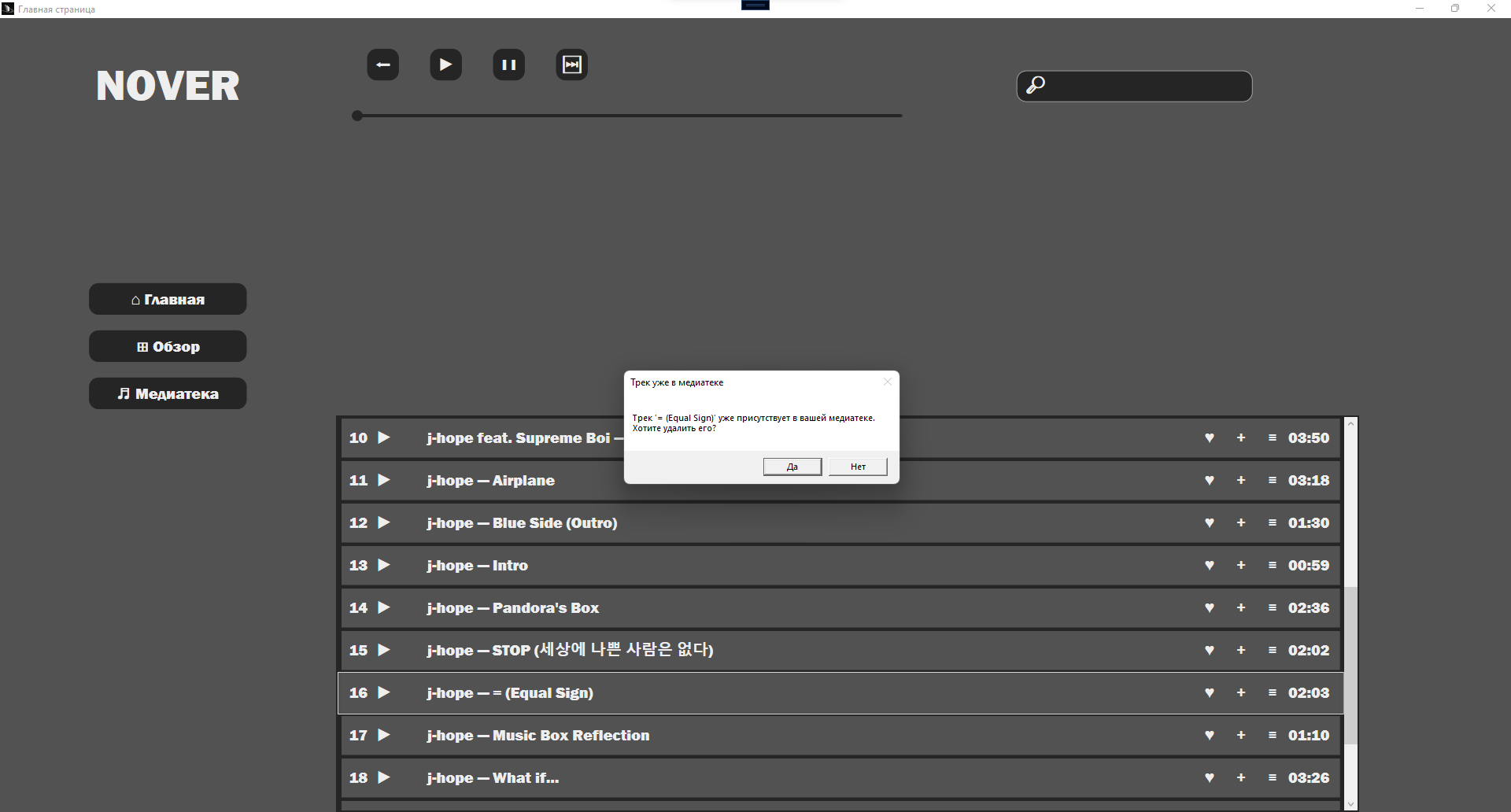


Рисунок 2.3. 5 - Сообщение, если трек уже в медиатеке

Нажав на кнопку «+», пользователь сможет добавить трек в свой добавленный или созданный плейлист, либо предоставится возможность создать новый плейлист и добавить выбранный трек туда (рисунки 2.3.6-2.3.8).

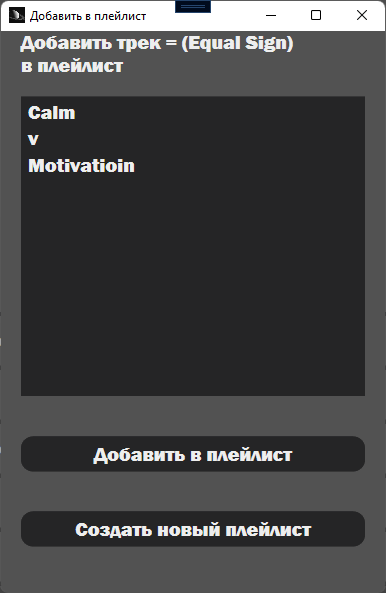


Рисунок 2.3.6 - Добавление песни в существующий плейлист

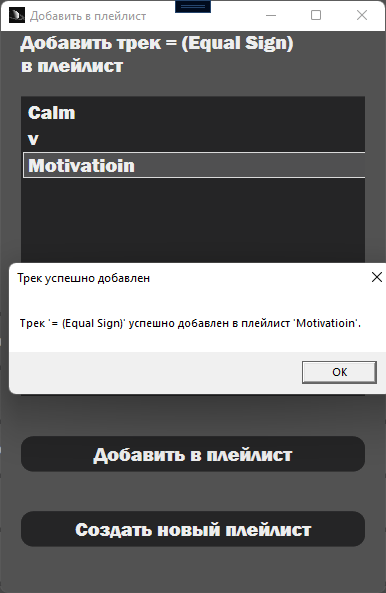


Рисунок 2.3. 7 - Сообщение, что трек добавлен в плейлист

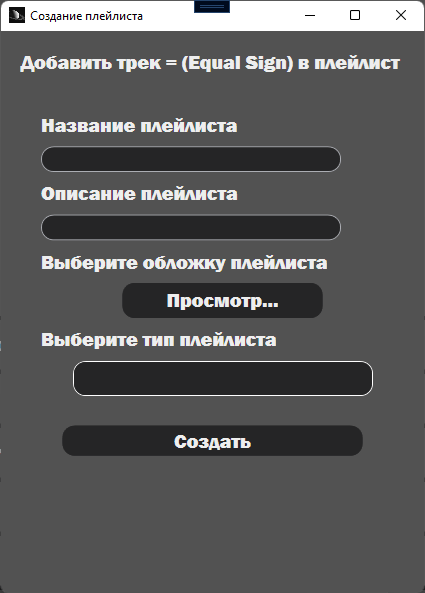


Рисунок 2.3.8 - Создание нового плейлиста и последующее добавление трека

Пользователь может также просматривать информацию об исполнителе. Для этого нужно выбрать композицию, исполнителя которой пользователь хочет посмотреть, и нажать на имя исполнителя. Результат нажатия на исполнителя представлен на рисунке 2.3.9.

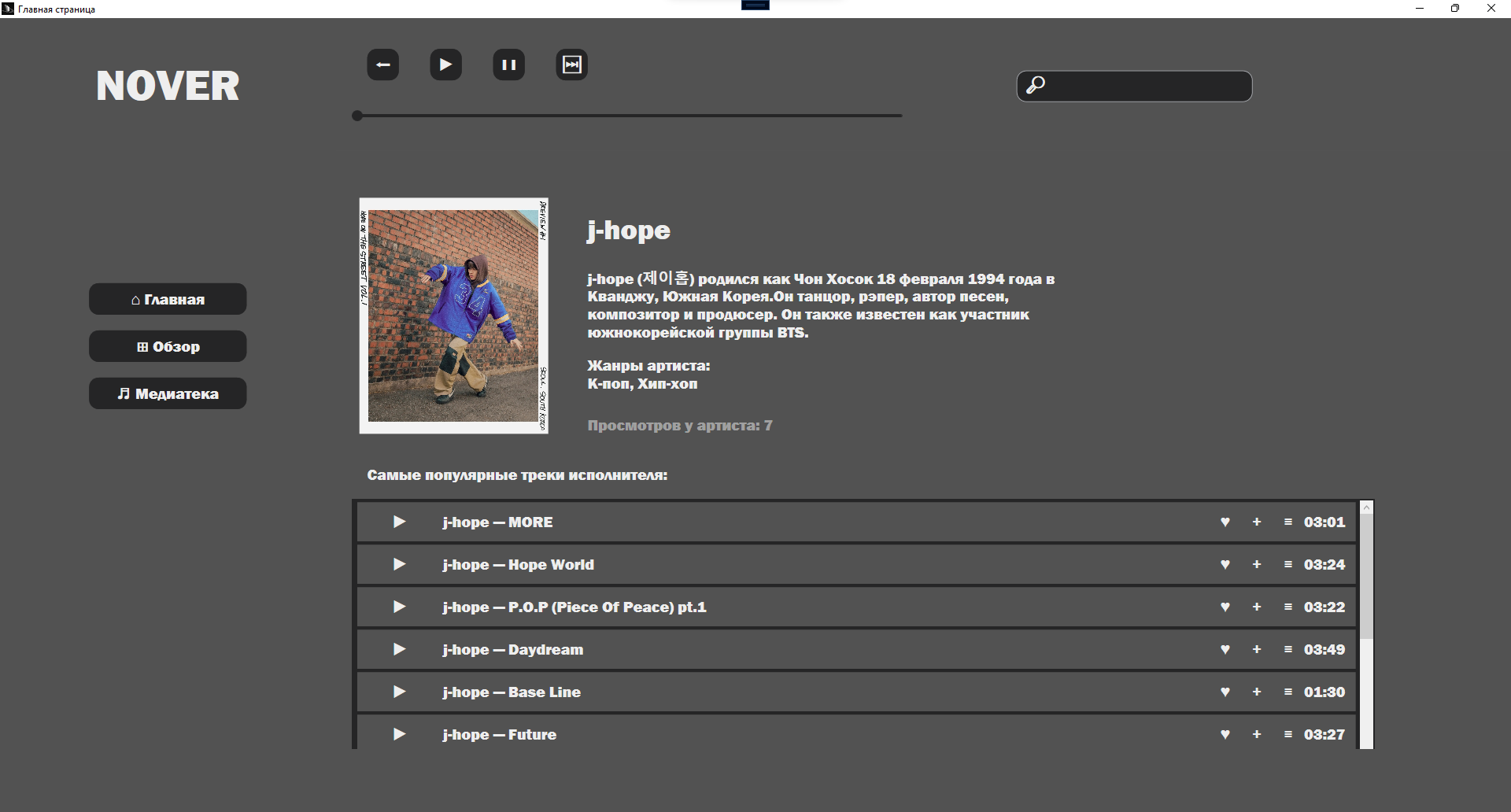


Рисунок 2.3.9 - Просмотр информации об исполнителе

На открывшейся странице можно увидеть фотографию исполнителя, его имя, биографию, наиболее популярные жанры, в которых он исполняет музыку, и количество его просмотров на данном сервисе.

На вкладке «Обзор» представлены для пользователя альбомы, плейлисты других пользователей и все треки, которые расположены в порядке популярности. Если пользователь выберет альбом и нажмет на него, то тогда откроется страница, включающая в себя информацию об альбоме, а именно: название альбома, исполнитель, выпустивший альбом, и дата выпуска альбома. Ниже представлен трек-лист данного альбома (рисунок 2.3.10).

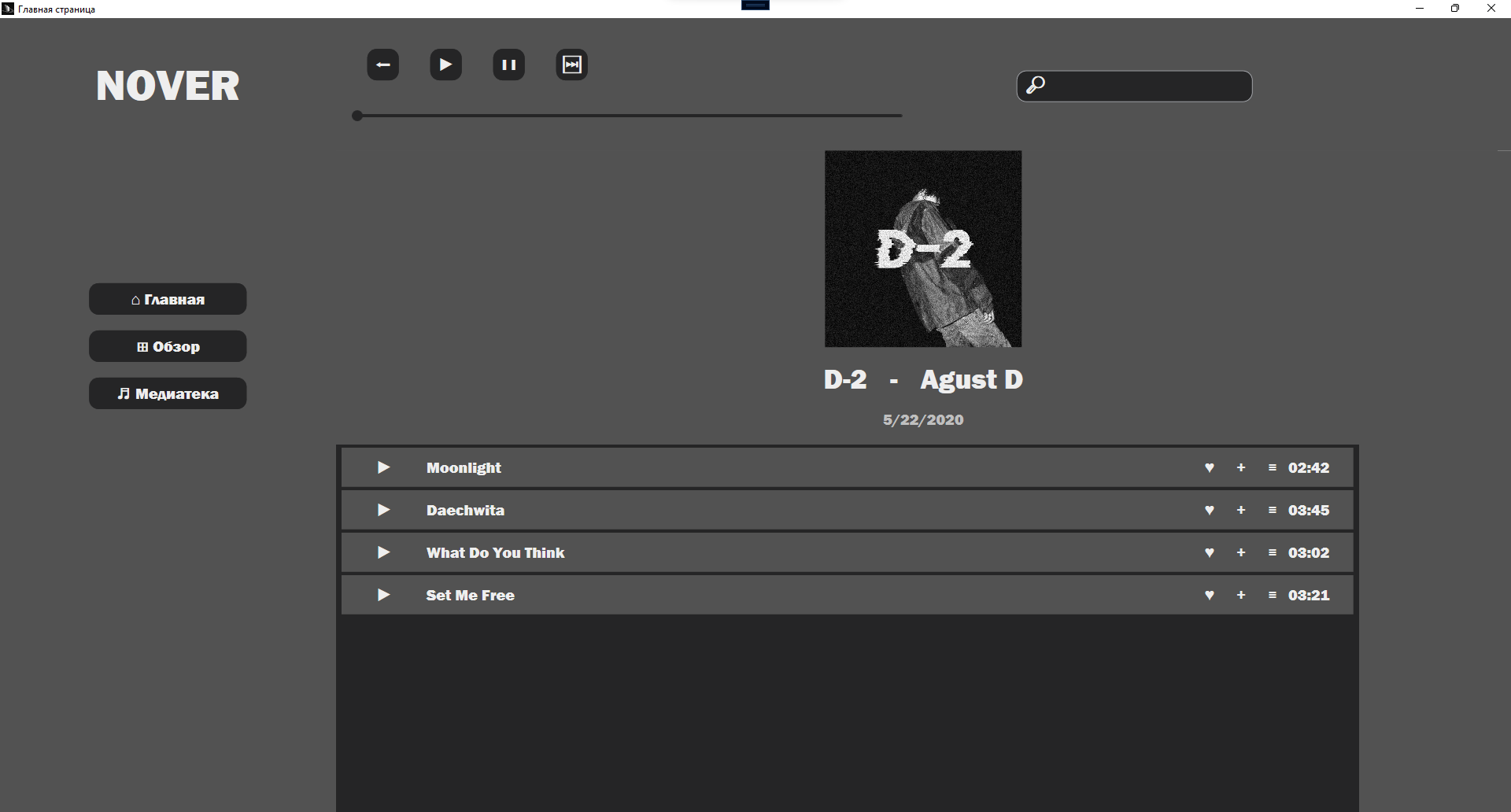


Рисунок 2.3. 10 - Информация об альбоме

Кроме этого, на вкладке «Обзор» можно выбрать плейлист других пользователей и нажать на него. Тогда откроется страница с информацией о плейлисте: название, имя пользователя, который создал этот плейлист, дата создания и описание плейлиста (рисунок 2.3.11).

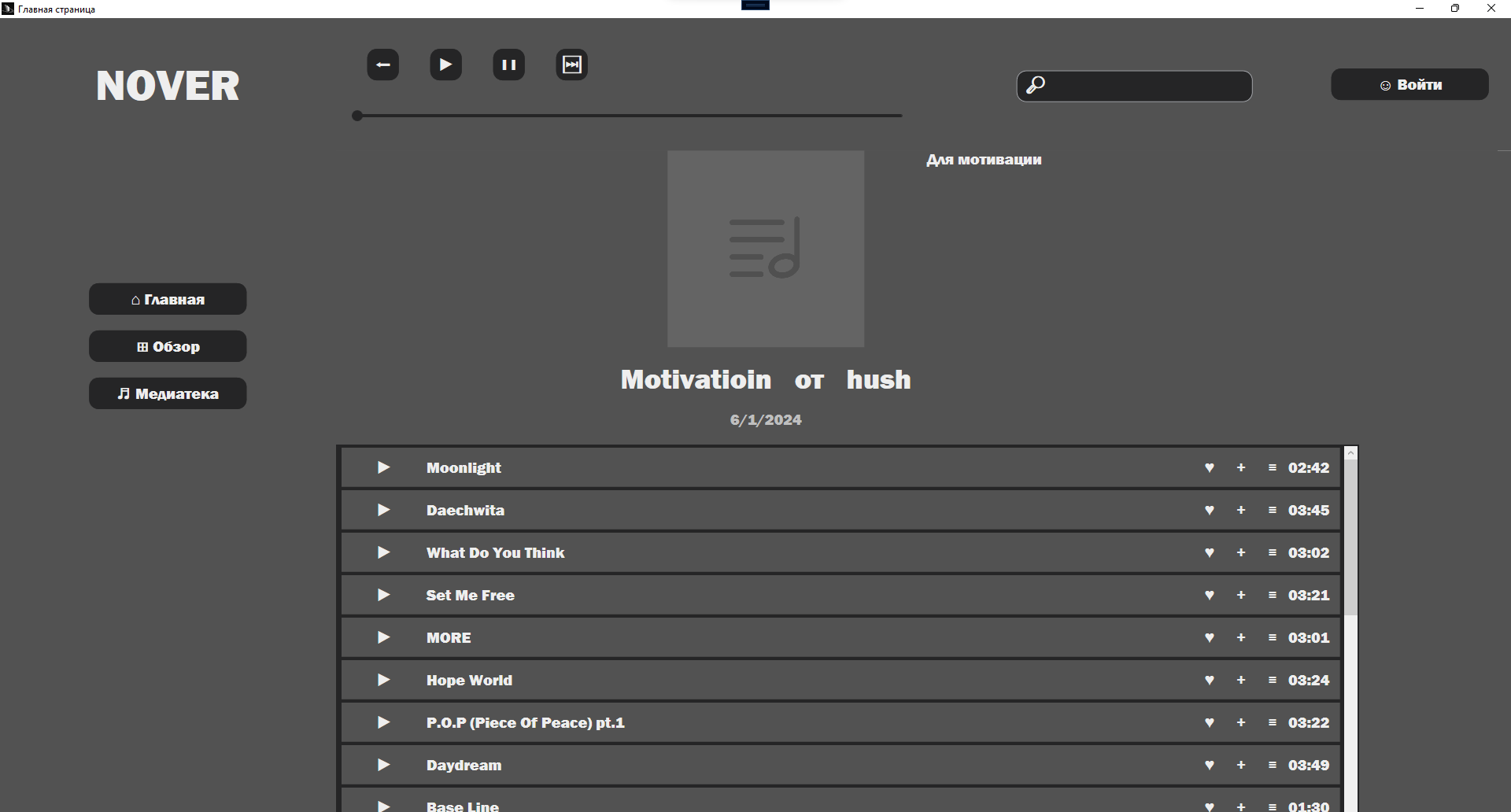


Рисунок 2.3.11 - Информация о плейлисте

Также пользователь может искать треки по названию композиции или исполнителю (рисунок 2.3.12).

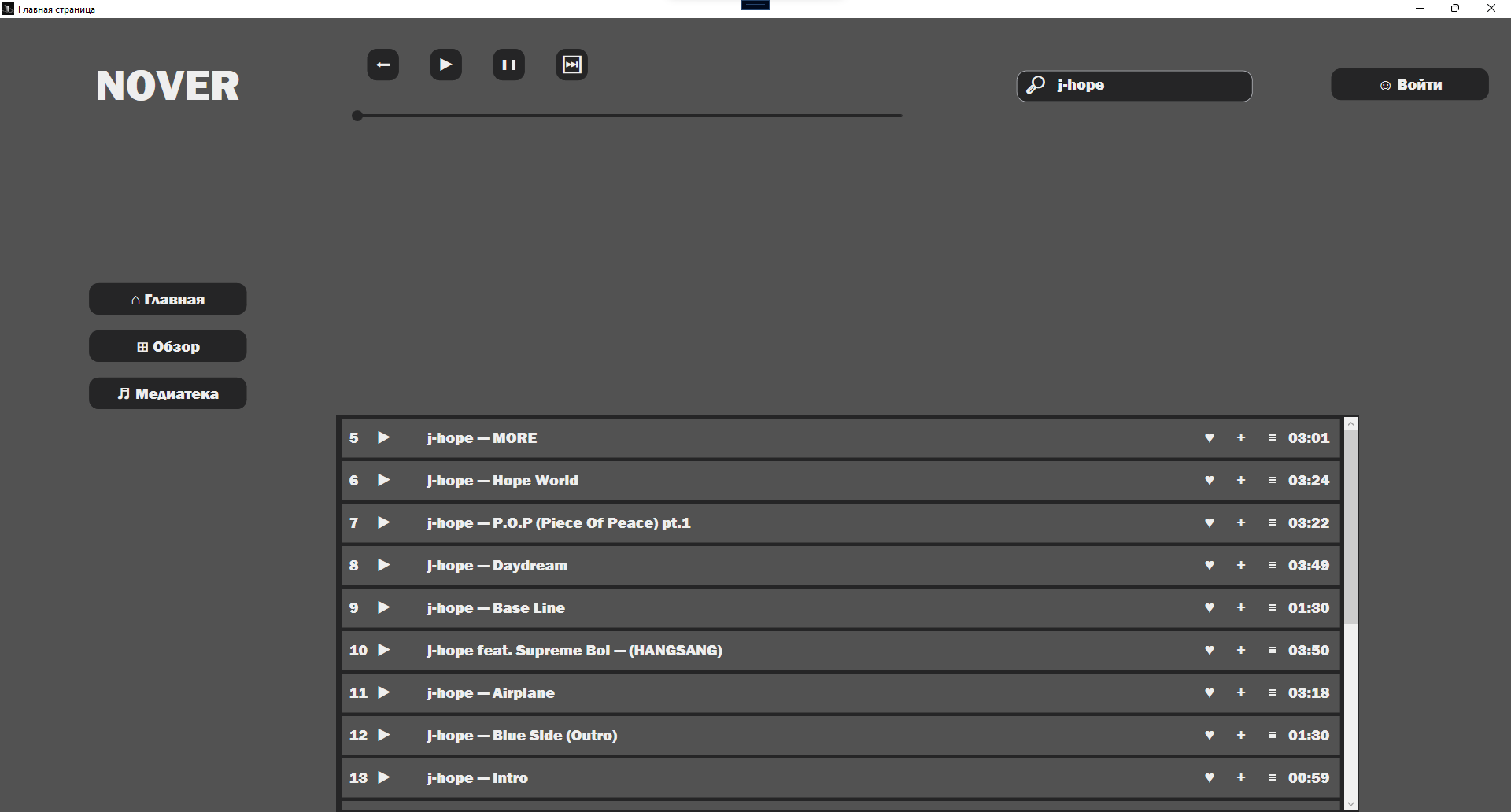


Рисунок 2.3.12 - Результаты поиска по исполнителю

Также пользователь может перематывать трек вперёд или назад, используя ползунок длительности, воспроизводить следующий трек кнопкой «⏭», возвращаться к прошлому треку, нажав на кнопку «🡨», остановить трек кнопкой «I I».

Пользователь может войти в свой аккаунт. Для этого ему необходимо ввести логин и пароль от своего профиля (рисунок 2.3.13). Нажав на кнопку «Назад» пользователь будет перенаправлен на главное меню.

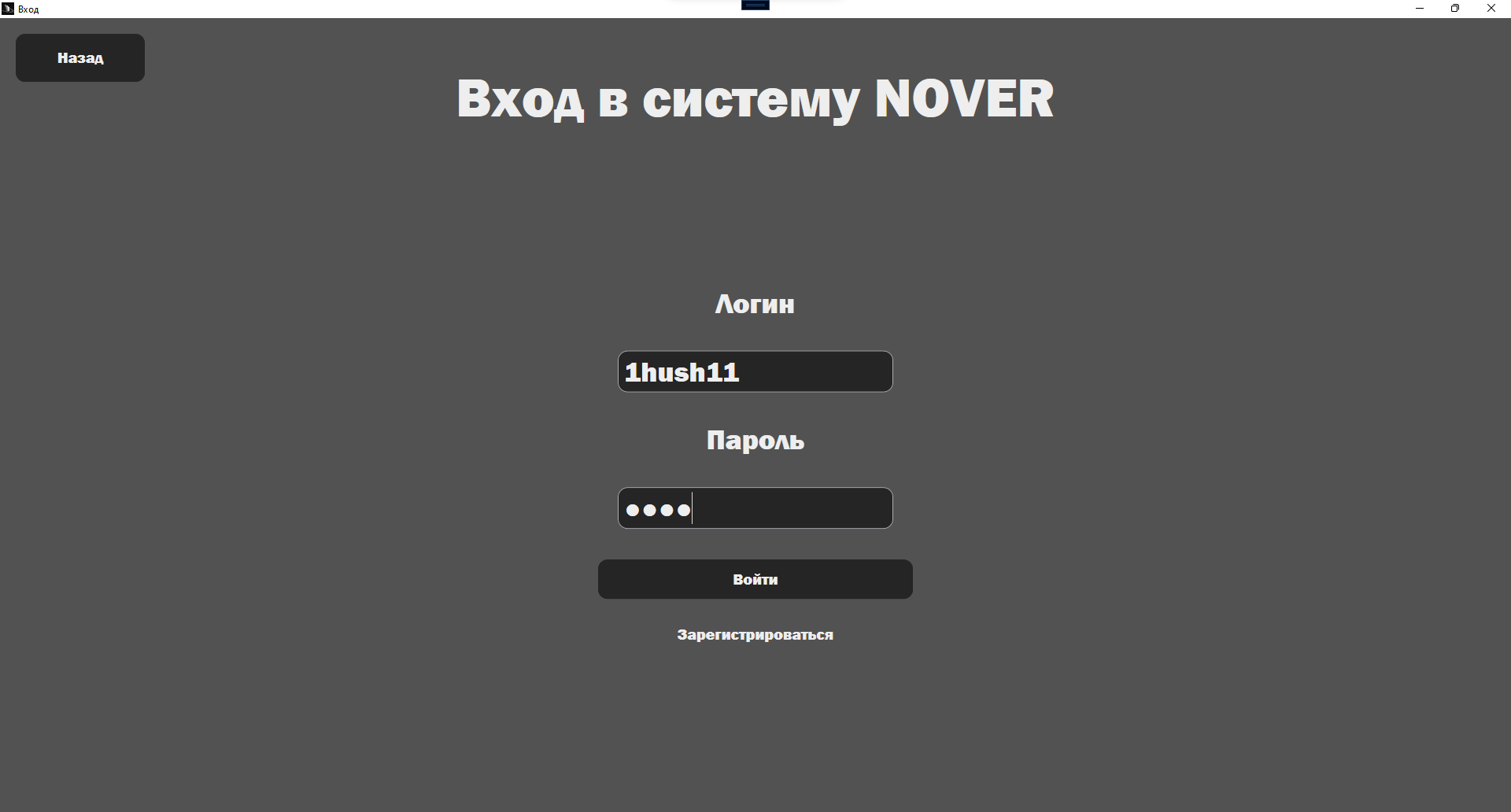


Рисунок 2.3.13 - Авторизация пользователя

Если же пользователь не зарегистрирован в системе, то тогда он может нажать на кнопку «Зарегистрироваться». И тогда откроется окно регистрации, где пользователю необходимо ввести все данные, требуемые системой, после чего нажать кнопку «Зарегистрироваться» (рисунок 2.3.14). Нажав на кнопку «Назад» пользователь будет перенаправлен на главное меню.

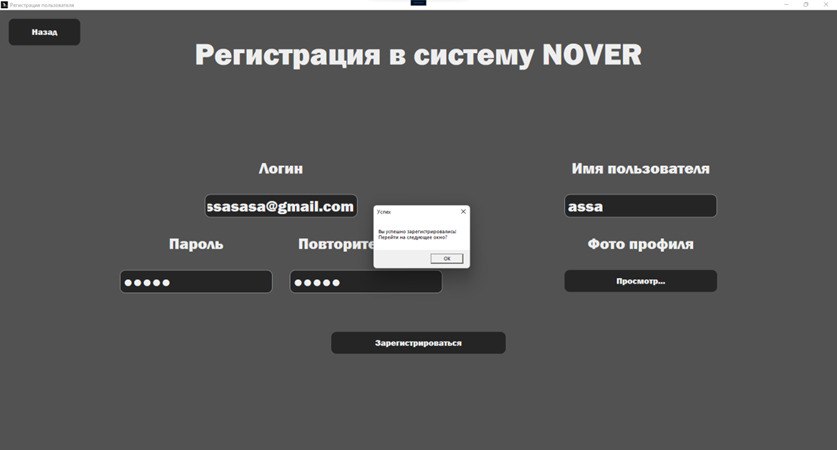


Рисунок 2.3.14 - Успешная регистрация пользователя

Пользователь может выйти из системы, нажав на кнопку «Выйти» во вкладке «Медиатека» (рисунок 2.3.3). После чего он будет переправлен на главное окно программы (рисунок 2.3.1). Также, нажав на кнопку «▶ Слушать» будут воспроизведены все треки в медиатеке пользователя.

## ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была разработана структура и алгоритм работы справочной системы для прослушивания музыкальных композиций и просмотра информации об исполнителях. В рамках проекта были изучены особенности создания клиентских приложений с удобным и интуитивно понятным пользовательским интерфейсом.

Результатом работы стало создание приложения, позволяющего пользователям:

* прослушивать музыку;
* добавлять музыкальные композиции в личную медиатеку;
* просматривать информацию об исполнителях и жанрах;
* создавать и управлять плейлистами.

Приложение написано на языке C# с использованием среды разработки Visual Studio 2022 и языка разметки XAML. Для управления данными использовалась система управления базой данных MySQL. Использовался паттерн проектирования MVVM.

Были проведены опытная эксплуатация и отладочное тестирование приложения. В ходе тестирования были устранены некоторые недостатки, в частности улучшена точность алгоритмов добавления и управления данными. После этого было написано руководство пользователя.

Приложение прошло проверку на контрольном примере, и полученные результаты полностью соответствуют ожидаемой выходной информации, что подтверждает его корректную работу и эффективность.

## Приложение А

Шаблоны выходной информации

{Исполнитель} - {Название трека} {Длительность трека}

{Исполнитель} - {Название трека} {Длительность трека}

{Исполнитель} - {Название трека} {Длительность трека}

{Исполнитель} – {Название трека} {Длительность трека}

Рисунок А.1 – выходная информация о треках

Информация о статистике

{Фотография альбома}

{Название альбома}– {Исполнитель}

{Дата создания альбома}

{Фотография альбома}

{Название альбома}– {Исполнитель}

{Дата создания альбома}

{Фотография альбома}

{Название альбома}– {Исполнитель}

{Дата создания альбома}

{Фотография альбома}

{Название альбома}– {Исполнитель}

{Дата создания альбома}

Продолжение приложения А

{Фотография плейлиста}

{Название плейлиста} от {Имя пользователя} {Описание плейлиста}

{Дата создания плейлиста}

{Фотография плейлиста}

{Название плейлиста} от {Имя пользователя} {Описание плейлиста}

{Дата создания плейлиста}

{Фотография плейлиста}

{Название плейлиста} от {Имя пользователя} {Описание плейлиста}

{Дата создания плейлиста}

{Фотография плейлиста}

{Название плейлиста} от {Имя пользователя} {Описание плейлиста}

{Дата создания плейлиста}

{Имя исполнителя}

{Фотография исполнителя} {Описание исполнителя}

Жанры артиста:

{Популярные жанры треков артиста}

Просмотров у артиста: {Просмотры исполнителя}

{Имя исполнителя}

{Фотография исполнителя} {Описание исполнителя}

Жанры артиста:

{Популярные жанры треков артиста}

Просмотров у артиста: {Просмотры исполнителя}

Продолжение приложения А

{Имя исполнителя}

{Фотография исполнителя} {Описание исполнителя}

Жанры артиста:

{Популярные жанры треков артиста}

Просмотров у артиста: {Просмотры исполнителя}

{Имя исполнителя}

{Фотография исполнителя} {Описание исполнителя}

Жанры артиста:

{Популярные жанры треков артиста}

Просмотров у артиста: {Просмотры исполнителя}

## Приложение Б

Входные данные контрольного примера

Таблица Б. - Пользователи

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID пользователя | Имя пользователя | Логин | Пароль | Фотография | Роль пользователя |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | hush | 1hush11 | hush | AVA.jpg | Пользователь |
| 2 | assasas | asa@gmail.com | qwe | kafka\_mummy.jpg | Пользователь |
| 3 | angels | angelplays@gmail.com | 123 | федя.jpg | Пользователь |

Таблица Б. - Плейлисты

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID плейлиста | ID создателя плейлиста | Название плейлиста | Фотография | Описание | Дата создания | Тип плейлиста |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 1 | Calm | BTS.jpg |  | 2024-05-10 | Приватный |
| 2 | 1 | Walking head | AVA.jpg | Плейлист для прогулок | 2024-04-28 | Публичный |

Продолжение приложения Б

Продолжение таблицы Б.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 3 | 2 | Motivatioin | Motivation.png | Для мотивации | 2024-06-01 | Приватный |

## Приложение В

Выходные данные контрольного примера

Таблица В. - Исполнители

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID исполнителя | Имя исполнителя | Фотография | Биография | Количество просмотров |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Agust D | Agust D.jpeg | Agust D - один из сценических псевдонимов Мин Юнги, южнокорейского рэпера, композитора и продюсера. Он также известен как SUGA, ведущий рэпер группы BTS. | 12 |
| 2 | Jimin (지민) | Jimin.jpg | Jimin (지민) родился как Пак Чимин в Пусане, Южная Корея, 13 октября 1995 года. Он ведущий вокалист и главный танцор южнокорейской группы BTS. | 2 |
| 3 | j-hope | J-hope.jpeg | j-hope (제이홉) родился как Чон Хосок 18 февраля 1994 года в Кванджу, Южная Корея.Он танцор, рэпер, автор песен, композитор и продюсер. Он также известен как участник южнокорейской группы BTS. | 6 |

Продолжение приложения В

Таблица В. - Альбомы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID альбома | Название альбома | ID исполнителя | Фотография | Дата выпуска |
| 1 | 2 | 3 | 4 | 5 |
| 1 | D-2 | 1 | D-2.jpg | 2020-05-22 |
| 2 | Jack In The Box | 3 | Jack In The Box.png | 2022-07-15 |
| 3 | Hope World | 3 | Hope World.jpg | 2018-03-01 |

Таблица В.3 - Треки

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ID трека | ID альбома | ID жанра | Название трека | Длительность трека | Аудио файл | Фотография | Количество прослушиваний |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 1 | Moonlight | 162 | Moonlight.mp3 | D-2.jpg | 14 |
| 7 | 3 | 2 | P.O.P (Piece Of Peace) pt.1 | 202 | P.O.P (Piece Of Peace) pt.1.mp3 | Hope World.jpg | 5 |
| 18 | 2 | 2 | What if... | 206 | What if....mp3 | Jack In The Box.png | 1 |

Продолжение приложения В

Таблица В. 4 - Жанры

|  |  |  |
| --- | --- | --- |
| ID жанра | Жанр | Описание |
| 1 | 2 | 3 |
| 1 | Хип-хоп | Хип-хоп – это музыкальный жанр, который состоит из зачитывания рэпа и ритма, задаваемого диджеем. Однако треки вовсе не строятся вокруг текста – порой они могут основываться только на мелодии. Хотя искусство правильной рифмовки и сочинения тоже очень ценится в написании любой композиции! |
| 2 | К-поп | K-pop (аббревиатура от англ. Korean pop) — музыкальный жанр, возникший в Южной Корее и вобравший в себя элементы западного электропопа, хип-хопа, танцевальной музыки и современного ритм-н-блюза. Появившись изначально как музыкальный жанр, K-pop превратился в масштабную музыкальную субкультуру с миллионами поклонников во всём мире. |

## Приложение Г

Код программы

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using Microsoft.EntityFrameworkCore;

using NOVER.Models;

using NOVER.ViewModels;

namespace NOVER.Services

{

public class ServicesTrack

{

public static List<Track> GetTracks()

{

using (NoverMusicContext db = new NoverMusicContext())

{

var singers = db.Singers.ToList();

var genres = db.Genres.ToList();

var albums = db.Albums.ToList();

var tracks = db.Tracks.Include(t => t.IdSingers).ToList();

return tracks;

}

}

public static List<Genre> GetGenres()

{

using (NoverMusicContext db = new NoverMusicContext())

{

db.Tracks.ToList();

db.Genres.ToList();

var genres = db.Genres.ToList();

return genres;

}

}

public static void UpdateListenTrack(Track track)

{

try

{

using (NoverMusicContext db = new NoverMusicContext())

{

var existingTrack = db.Tracks.FirstOrDefault(t => t.IdTrack == track.IdTrack);

if (existingTrack != null)

{

existingTrack.ListenTrack = track.ListenTrack + 1;

BrowseTrack.SelectedTrack = existingTrack;

db.SaveChanges();

}

}

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public static List<String> GetTypesPlaylists()

{

using (NoverMusicContext db = new NoverMusicContext())

{

var typesPlaylist = new List<String>(db.Playlists.Select(p => p.TypePlaylist).Distinct().ToList());

return typesPlaylist;

}

}

}

}

using NOVER.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace NOVER.Services

{

public class ServicesAlbum

{

public static List<Album> GetAlbums()

{

using (NoverMusicContext db = new NoverMusicContext())

{

var playlists = db.Playlists.ToList();

var tracks = db.Tracks.ToList();

var users = db.Users.ToList();

var singers = db.Singers.ToList();

var albums = db.Albums.ToList();

return albums;

}

}

}

}

using Microsoft.EntityFrameworkCore;

using NOVER.Models;

using NOVER.ViewModels;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace NOVER.Services

{

public class ServicesPlaylist

{

public static List<Playlist> GetPlaylists()

{

using (NoverMusicContext db = new NoverMusicContext())

{

var tracks = db.Tracks.ToList();

var users = db.Users.ToList();

var singers = db.Singers.ToList();

var playlists = db.Playlists.ToList();

return playlists;

}

}

public static void AddPlaylistMethod(Playlist NewPlaylist)

{

User currentUser = BrowseUser.CurrentUser;

if (currentUser != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var existingPlaylist = db.Playlists.Include(p => p.IdTracks)

.FirstOrDefault(p => p.NamePlaylist == NewPlaylist.NamePlaylist && p.IdCreator == currentUser.IdUser);

if (existingPlaylist == null)

{

Playlist newPlaylist = new Playlist

{

IdCreator = currentUser.IdUser,

NamePlaylist = NewPlaylist.NamePlaylist,

DescPlaylist = NewPlaylist.DescPlaylist,

PicPlaylist = NewPlaylist.PicPlaylist,

TypePlaylist = BrowseTrack.SelectedType,

DatePlaylist = DateOnly.FromDateTime(DateTime.Now)

};

db.Playlists.Add(newPlaylist);

var track = db.Tracks.FirstOrDefault(t => t.IdTrack == BrowseTrack.SelectedTrack.IdTrack);

if (track != null)

{

newPlaylist.IdTracks.Add(track);

}

db.SaveChanges();

MessageBox.Show($"Плейлист '{newPlaylist.NamePlaylist}' успешно добавлен, и трек '{track.NameTrack}' добавлен в плейлист.");

}

else

{

MessageBoxResult result = MessageBox.Show($"Плейлист с названием '{NewPlaylist.NamePlaylist}' уже существует.\nХотите удалить его и создать новый?", "Плейлист уже существует", MessageBoxButton.YesNo);

if (result == MessageBoxResult.Yes)

{

db.Playlists.Remove(existingPlaylist);

Playlist newPlaylist = new Playlist

{

IdCreator = currentUser.IdUser,

IdUser = currentUser.IdUser,

NamePlaylist = NewPlaylist.NamePlaylist,

DescPlaylist = NewPlaylist.DescPlaylist,

PicPlaylist = NewPlaylist.PicPlaylist,

TypePlaylist = BrowseTrack.SelectedType,

DatePlaylist = DateOnly.FromDateTime(DateTime.Now)

};

db.Playlists.Add(newPlaylist);

var track = db.Tracks.FirstOrDefault(t => t.IdTrack == BrowseTrack.SelectedTrack.IdTrack);

if (track != null)

{

newPlaylist.IdTracks.Add(track);

}

db.SaveChanges();

MessageBox.Show($"Старый плейлист с названием '{existingPlaylist.NamePlaylist}' был удален, и создан новый плейлист '{newPlaylist.NamePlaylist}' с треком '{track.NameTrack}'.");

}

}

}

}

else

{

MessageBox.Show("Пользователь не найден в базе данных.");

}

}

public static void AddToPlaylistMethod()

{

if (BrowseTrack.SelectedTrack != null && BrowseTrack.SelectedPlaylist != null)

{

User currentUser = BrowseUser.CurrentUser;

if (currentUser != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var playlist = db.Playlists.Include(p => p.IdTracks).FirstOrDefault(p => p.IdPlaylist == BrowseTrack.SelectedPlaylist.IdPlaylist && p.IdCreator == currentUser.IdUser);

if (playlist != null)

{

var track = db.Tracks.FirstOrDefault(t => t.IdTrack == BrowseTrack.SelectedTrack.IdTrack);

if (track != null)

{

if (!playlist.IdTracks.Any(t => t.IdTrack == track.IdTrack))

{

playlist.IdTracks.Add(track);

db.SaveChanges();

MessageBox.Show($"Трек '{track.NameTrack}' успешно добавлен в плейлист '{playlist.NamePlaylist}'.", "Трек успешно добавлен");

}

else

{

MessageBox.Show("Этот трек уже существует в выбранном плейлисте.");

}

}

else

{

MessageBox.Show("Трек не найден.");

}

}

else

{

MessageBox.Show("Плейлист не найден или у вас нет прав для его редактирования.");

}

}

}

else

{

MessageBox.Show("Пользователь не найден.");

}

}

else

{

MessageBox.Show("Выберите трек и плейлист.");

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls.Primitives;

using System.Windows;

using Microsoft.EntityFrameworkCore;

using NOVER.Models;

using NOVER.ViewModels;

namespace NOVER.Services

{

public class ServicesSinger

{

public static List<Singer> GetSingers()

{

using (NoverMusicContext db = new NoverMusicContext())

{

db.Tracks.ToList();

db.Genres.ToList();

db.Singers.ToList();

var singers = db.Singers.Include(s => s.IdTracks)

.ThenInclude(t => t.IdGenreNavigation).ToList();

return singers;

}

}

public static void UpdateSingerView(Singer singer)

{

try

{

using (NoverMusicContext db = new NoverMusicContext())

{

var existingSinger = db.Singers.FirstOrDefault(s => s.IdSinger == singer.IdSinger);

if (existingSinger != null)

{

existingSinger.ViewSinger = singer.ViewSinger + 1;

BrowseTrack.SelectedSinger = existingSinger;

db.SaveChanges();

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

using Microsoft.EntityFrameworkCore;

using NOVER.Models;

using NOVER.ViewModels;

using NOVER.Views;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace NOVER.Services

{

public class ServicesUser

{

public static List<User> GetUsers()

{

using (NoverMusicContext db = new NoverMusicContext())

{

var playlists = db.Playlists.ToList();

var tracks = db.Tracks.ToList();

var users = db.Users.ToList();

return users;

}

}

public static void SignUpMethod(User newUser, SignUpWindow signUpWindow)

{

try

{

using (NoverMusicContext db = new NoverMusicContext())

{

var password = signUpWindow.passwordTB.Password.ToString();

var passwordSecond = signUpWindow.passwordSecondTB.Password.ToString();

if (string.IsNullOrEmpty(newUser.LoginUser) ||

string.IsNullOrEmpty(newUser.UserName) ||

string.IsNullOrEmpty(password) ||

string.IsNullOrEmpty(passwordSecond))

{

MessageBox.Show("Все поля должны быть заполнены!", "Ошибка");

return;

}

if (password != passwordSecond)

{

MessageBox.Show("Пароли не совпадают!", "Ошибка");

return;

}

var existingUser = db.Users.FirstOrDefault(u => u.LoginUser == newUser.LoginUser);

if (existingUser != null)

{

MessageBox.Show("Пользователь с таким логином уже существует!", "Ошибка");

return;

}

newUser.PassUser = password;

newUser.IdUser = db.Users.Count() + 1;

newUser.RoleUser = "Пользователь";

db.Users.Add(newUser);

db.SaveChanges();

BrowseUser.CurrentUser = newUser;

MessageBoxResult result = MessageBox.Show("Вы успешно зарегистрировались!\nПерейти на следующее окно?", "Успех", MessageBoxButton.OK);

if (result == MessageBoxResult.OK)

{

BrowseTrack.OpenLibraryPageMethod();

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

public static void UpdateUser(User user)

{

try

{

using (NoverMusicContext db = new NoverMusicContext())

{

var existingUser = db.Users.FirstOrDefault(u => u.IdUser == user.IdUser);

if (existingUser != null)

{

existingUser.UserName = user.UserName;

existingUser.LoginUser = user.LoginUser;

existingUser.PassUser = user.PassUser;

existingUser.PicUser = user.PicUser;

existingUser.RoleUser = user.RoleUser;

//existingUser.IdTracks = user.IdTracks;

if (user.PlaylistIdCreatorNavigations != null && user.PlaylistIdCreatorNavigations.Count > 0)

{

existingUser.PlaylistIdCreatorNavigations = new List<Playlist>(user.PlaylistIdCreatorNavigations);

}

if (user.PlaylistIdUserNavigations != null && user.PlaylistIdUserNavigations.Count > 0)

{

existingUser.PlaylistIdUserNavigations = new List<Playlist>(user.PlaylistIdUserNavigations);

}

if (user.IdPlaylists != null && user.IdPlaylists.Count > 0)

{

existingUser.IdPlaylists = new List<Playlist>(user.IdPlaylists);

}

if (user.IdTracks != null && user.IdTracks.Count > 0)

{

existingUser.IdTracks = new List<Track>(user.IdTracks);

}

db.SaveChanges();

}

else

{

throw new InvalidOperationException($"Пользователь с идентификатором {user.IdUser} не найден в базе данных.");

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при обновлении пользователя: {ex.Message}");

}

}

public static void AddToLibraryMethod(Track track)

{

User CurrentUser = BrowseUser.CurrentUser;

if (CurrentUser != null && track != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var dbUser = db.Users.Include(u => u.IdTracks).FirstOrDefault(u => u.IdUser == CurrentUser.IdUser);

if (dbUser != null)

{

bool trackExistsInLibrary = dbUser.IdTracks.Any(t => t.IdTrack == track.IdTrack);

if (!trackExistsInLibrary)

{

AddTracksToUserLibrary(CurrentUser, new List<Track> { track });

MessageBox.Show($"Трек '{track.NameTrack}' добавлен в вашу медиатеку.");

}

else

{

MessageBoxResult result = MessageBox.Show($"Трек '{track.NameTrack}' уже присутствует в вашей медиатеке.\nХотите удалить его?", "Трек уже в медиатеке", MessageBoxButton.YesNo);

if (result == MessageBoxResult.Yes)

{

RemoveTrackFromLibrary(CurrentUser, track);

MessageBox.Show($"Трек '{track.NameTrack}' был удалён из вашей медиатеки.");

}

}

}

else

{

MessageBox.Show("Пользователь не найден в базе данных.");

}

}

}

else

{

MessageBox.Show("Не удалось добавить трек в медиатеку. Пользователь не определен или трек не выбран.");

}

}

public static void AddTracksToUserLibrary(User user, List<Track> tracks)

{

if (user != null && tracks != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var dbUser = db.Users.Include(u => u.IdTracks).FirstOrDefault(u => u.IdUser == user.IdUser);

if (dbUser != null)

{

foreach (var track in tracks)

{

var dbTrack = db.Tracks.Find(track.IdTrack);

if (dbTrack != null)

{

dbUser.IdTracks.Add(dbTrack);

}

}

db.SaveChanges();

}

else

{

MessageBox.Show("Пользователь не найден в базе данных.");

}

}

}

else

{

MessageBox.Show("Не удалось добавить треки в медиатеку. Пользователь или список треков не определены.");

}

}

public static void RemoveTrackFromLibrary(User user, Track track)

{

if (user != null && track != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var dbUser = db.Users.Include(u => u.IdTracks).FirstOrDefault(u => u.IdUser == user.IdUser);

if (dbUser != null)

{

var dbTrack = db.Tracks.Find(track.IdTrack);

if (dbTrack != null)

{

dbUser.IdTracks.Remove(dbTrack);

db.SaveChanges();

}

}

else

{

MessageBox.Show("Пользователь не найден в базе данных.");

}

}

}

else

{

MessageBox.Show("Не удалось удалить трек из медиатеки. Пользователь или трек не определены.");

}

}

public static List<Playlist> GetPlaylistsCurrentUser()

{

if (BrowseUser.CurrentUser != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var users = db.Users.ToList();

var userPlaylists = db.Playlists

.Where(p => p.IdCreator == BrowseUser.CurrentUser.IdUser)

.ToList();

var playlistsCurrentUser = new List<Playlist>(userPlaylists);

return playlistsCurrentUser;

}

}

else

{

return null;

}

}

public static List<Track> GetTracksCurrentUser()

{

if (BrowseTrack.CurrentUser != null)

{

using (NoverMusicContext db = new NoverMusicContext())

{

var userTracks = db.Tracks

.Where(t => t.IdUsers.Any(u => u.IdUser == BrowseTrack.CurrentUser.IdUser))

.ToList();

foreach (var track in userTracks)

{

db.Entry(track).Collection(t => t.IdSingers).Load();

}

return userTracks;

}

}

else

{

return null;

}

}

}

}

using DevExpress.Mvvm;

using Microsoft.EntityFrameworkCore;

using Microsoft.Win32;

using NOVER.Models;

using NOVER.Services;

using NOVER.Views;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Threading.Channels;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Threading;

namespace NOVER.ViewModels

{

public class BrowseTrack : BindableBase, INotifyPropertyChanged

{

public ICommand NavigateCommand { get; }

private List<Track> tracks;

private List<Track> allTracks;

private List<Singer> singers = ServicesSinger.GetSingers();

private List<Genre> genres = ServicesTrack.GetGenres();

private List<Album> albums = ServicesAlbum.GetAlbums();

private List<Playlist> playlists = ServicesPlaylist.GetPlaylists();

private List<Playlist> playlistsCurrentUser = ServicesUser.GetPlaylistsCurrentUser();

private List<Track> tracksCurrentUser = ServicesUser.GetTracksCurrentUser();

public static List<Track> popularTracks = new List<Track>();

public static List<String> popularGenres = new List<String>();

private static List<String> typesPlaylist = ServicesTrack.GetTypesPlaylists();

public static string PopularGenresString;

private string search;

public static User? CurrentUser

{

get { return BrowseUser.CurrentUser; }

set

{

BrowseUser.CurrentUser = value;

BrowseTrack browseTrack = new BrowseTrack();

browseTrack.NotifyPropertyChanged("CurrentUser");

}

}

private void NotifyPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

public BrowseTrack()

{

allTracks = ServicesTrack.GetTracks();

Tracks = new List<Track>(allTracks);

NewPlaylist = new Playlist();

}

public List<Track> Tracks

{

get

{

return tracks;

}

set

{

tracks = value;

NotifyPropertyChanged("Tracks");

}

}

public List<Track> TracksCurrentUser

{

get

{

return tracksCurrentUser;

}

set

{

tracksCurrentUser = value;

NotifyPropertyChanged("TracksCurrentUser");

}

}

public List<Track> PopularTracks

{

get

{

return popularTracks;

}

set

{

popularTracks = value;

NotifyPropertyChanged("PopularTracks");

}

}

public List<Singer> Singers

{

get

{

return singers;

}

set

{

singers = value;

NotifyPropertyChanged("Singers");

}

}

public List<Genre> Genres

{

get

{

return genres;

}

set

{

genres = value;

NotifyPropertyChanged("Genres");

}

}

public List<Album> Albums

{

get

{

return albums;

}

set

{

albums = value;

NotifyPropertyChanged("Albums");

}

}

public List<Playlist> Playlists

{

get

{

return playlists;

}

set

{

playlists = value;

NotifyPropertyChanged("Playlists");

}

}

public List<Playlist> PlaylistsCurrentUser

{

get

{

return playlistsCurrentUser;

}

set

{

playlistsCurrentUser = value;

NotifyPropertyChanged("PlaylistsCurrentUser");

}

}

public List<String> PopularGenres

{

get

{

return popularGenres;

}

set

{

popularGenres = value;

NotifyPropertyChanged("PopularGenres");

}

}

public List<String> TypesPlaylist

{

get

{

return typesPlaylist;

}

set

{

typesPlaylist = value;

NotifyPropertyChanged("TypesPlaylist");

}

}

// Поиск

public string Search

{

get { return search; }

set

{

search = value;

OnPropertyChanged(nameof(Search));

UpdateTrackList();

}

}

private void UpdateTrackList()

{

if (string.IsNullOrEmpty(Search))

{

Tracks = new List<Track>(allTracks);

}

else

{

Tracks = allTracks.Where(t => t.NameTrack.ToLower().Contains(Search.ToLower()) ||

t.SingersNames.ToLower().Contains(Search.ToLower())).ToList();

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

public static Track? SelectedTrack { get; set; }

public static Singer? SelectedSinger { get; set; }

public static Album? SelectedAlbum { get; set; }

public static Playlist? SelectedPlaylist { get; set; }

public Playlist NewPlaylist { get; set; }

public static String SelectedType { get; set; }

private static TimeSpan pausedPosition;

// Вывод главной страницы

private RelayCommand openHomePage;

public RelayCommand OpenHomePage

{

get

{

return openHomePage ?? new RelayCommand(obj =>

{

OpenHomePageMethod();

});

}

}

public static void OpenHomePageMethod()

{

Home home = new Home();

MainWindow.mainWindow.contentFR.NavigationService.Navigate(home, Visibility.Visible);

}

private RelayCommand openBrowsePage;

public RelayCommand OpenBrowsePage

{

get

{

return openHomePage ?? new RelayCommand(obj =>

{

OpenBrowsePageMethod();

});

}

}

public static void OpenBrowsePageMethod()

{

BrowsePage browsePage = new BrowsePage();

Home.home.infoFR.NavigationService.Navigate(browsePage, Visibility.Visible);

}

// Воспроизведение выбранного трека

private RelayCommand playTrack;

public RelayCommand PlayTrack

{

get

{

return playTrack ?? new RelayCommand(obj =>

{

if (SelectedTrack != null)

{

PlayTrackMethod(SelectedTrack, this);

}

});

}

}

private static void PlayTrackMethod(Track selectedTrack, BrowseTrack browseTrack)

{

if (selectedTrack != null && browseTrack != null)

{

string trackPath = @"C:\\Users\\HOME\\Downloads\\NOVER\_M\\NOVER\_audios\" + selectedTrack.AudioTrack;

Home.home.mediaPlayer.Open(new Uri(trackPath));

Home.home.mediaPlayer.Play();

Home.home.mediaPlayer.MediaEnded += MediaPlayer\_MediaEnded;

Home.home.selectedTrackSingerNameTB.Content = selectedTrack.SingersNames + " —";

Home.home.selectedTrackNameTrackTB.Text = " " + selectedTrack.NameTrack;

browseTrack.StartSliderUpdate();

ServicesTrack.UpdateListenTrack(SelectedTrack);

}

}

// Метод для обработки события окончания трека

private static void MediaPlayer\_MediaEnded(object sender, EventArgs e)

{

BrowseTrack browseTrack = new BrowseTrack();

browseTrack.PlayNextTrack();

}

// Воспроизведение следующего трека

private void PlayNextTrack()

{

if (Tracks != null && SelectedTrack != null)

{

int currentIndex = Tracks.FindIndex(t => t.IdTrack == SelectedTrack.IdTrack);

if (currentIndex >= 0 && currentIndex < Tracks.Count - 1)

{

SelectedTrack = Tracks[currentIndex + 1];

PlayTrackMethod(SelectedTrack, this);

Home.home.tracksLV.SelectedItem = SelectedTrack;

}

}

}

// Мтода для продолжения воспроизведения трека с момента паузы

private RelayCommand resumeTrack;

public RelayCommand ResumeTrack

{

get

{

return resumeTrack ?? new RelayCommand(obj =>

{

if (SelectedTrack != null)

{

ResumeTrackMethod();

}

});

}

}

private void ResumeTrackMethod()

{

if (Home.home.mediaPlayer.Source != null)

{

Home.home.mediaPlayer.Position = pausedPosition;

Home.home.mediaPlayer.Play();

}

}

// Поизиция слайдера - время в песне

private void StartSliderUpdate()

{

DispatcherTimer timer = new DispatcherTimer

{

Interval = TimeSpan.FromSeconds(1)

};

timer.Tick += (sender, args) =>

{

if (Home.home.mediaPlayer.Source != null && Home.home.mediaPlayer.NaturalDuration.HasTimeSpan)

{

Home.home.audioSL.Maximum = Home.home.mediaPlayer.NaturalDuration.TimeSpan.TotalSeconds;

Home.home.audioSL.Value = Home.home.mediaPlayer.Position.TotalSeconds;

}

};

timer.Start();

}

// Пауза выбранного трека

private RelayCommand stopTrack;

public RelayCommand StopTrack

{

get

{

return stopTrack ?? new RelayCommand(obj =>

{

if (SelectedTrack != null)

{

StopTrackMethod(SelectedTrack);

}

});

}

}

private static void StopTrackMethod(Track selectedTrack)

{

pausedPosition = Home.home.mediaPlayer.Position;

Home.home.mediaPlayer.Pause();

}

// Воспроизвдение прошлого трека

private RelayCommand backTrackCommand;

public RelayCommand BackTrackCommand

{

get

{

return backTrackCommand ?? new RelayCommand(obj =>

{

BackTrackMethod(SelectedTrack);

});

}

}

private void BackTrackMethod(Track track)

{

if (track != null)

{

BrowseTrack browseTrack = new BrowseTrack();

int currentIndex = browseTrack.Tracks.FindIndex(t => t.IdTrack == track.IdTrack);

if (currentIndex > 0)

{

SelectedTrack = browseTrack.Tracks[currentIndex - 1];

PlayTrackMethod(SelectedTrack, this);

Home.home.tracksLV.SelectedItem = SelectedTrack;

}

}

}

// Вызов метода с воспроизвдением следущего трека

private RelayCommand playNextTrackCommand;

public RelayCommand PlayNextTrackCommand

{

get

{

return playNextTrackCommand ?? new RelayCommand(obj =>

{

PlayNextTrack();

});

}

}

// Открытие главного окна

private RelayCommand openMainWindow;

public RelayCommand OpenMainWindow

{

get

{

return openMainWindow ?? new RelayCommand(obj =>

{

OpenMainWindowMethod(obj as Window);

});

}

}

public static void OpenMainWindowMethod(Window currentWindow)

{

MainWindow mainWindow = new MainWindow();

mainWindow.Show();

currentWindow?.Close();

}

// Открытие страницы профиля

private RelayCommand openLibraryPage;

public RelayCommand OpenLibraryPage

{

get

{

return openLibraryPage ?? new RelayCommand(obj =>

{

OpenLibraryPageMethod();

});

}

}

public static void OpenLibraryPageMethod()

{

CurrentUser = BrowseUser.CurrentUser;

if (CurrentUser != null)

{

LibraryPage libraryPage = new LibraryPage();

MainWindow.mainWindow.contentFR.NavigationService.Navigate(libraryPage, Visibility.Visible);

}

else

{

MessageBox.Show("Чтобы перейти к медиатеке, пожалуйста, авторизируйтесь.", "Внимание!");

}

}

// Открытие окна авторизации

private RelayCommand openLogInWindow;

public RelayCommand OpenLogInWindow

{

get

{

return openLogInWindow ?? new RelayCommand(obj =>

{

OpenLogInWindowMethod();

});

}

}

public static void OpenLogInWindowMethod()

{

LogInWindow logInWindow = new LogInWindow();

logInWindow.Show();

logInWindow.WindowStartupLocation = System.Windows.WindowStartupLocation.CenterScreen;

}

// Открытие страницы с исполнителем

private RelayCommand openSingerInfoPage;

public RelayCommand OpenSingerInfoPage

{

get

{

return openSingerInfoPage ?? new RelayCommand(obj =>

{

OpenSingerInfoPageMethod();

});

}

}

private static void OpenSingerInfoPageMethod()

{

if (SelectedTrack != null)

{

BrowseTrack browseTrack = new BrowseTrack();

SelectedSinger = browseTrack.Singers.FirstOrDefault(s => s.NameSinger.Equals(SelectedTrack.SingersNames, StringComparison.OrdinalIgnoreCase));

SingersInfoPage singersInfoPage = new SingersInfoPage();

Home.home.infoFR.NavigationService.Navigate(singersInfoPage, Visibility.Visible);

browseTrack.PopularTracks = SelectedSinger.IdTracks.OrderByDescending(t => t.ListenTrack).ToList();

string popularGenresString;

foreach (var track in SelectedSinger.IdTracks)

{

browseTrack.PopularGenres.Add(track.IdGenreNavigation?.NameGenre);

browseTrack.PopularGenres = browseTrack.PopularGenres.Distinct().ToList();

popularGenresString = string.Join(", ", popularGenres);

PopularGenresString = popularGenresString;

}

ServicesSinger.UpdateSingerView(SelectedSinger);

if (PopularGenresString != "")

SingersInfoPage.singersInfoPage.popularGenresTB.Text = "Жанры артиста:\n" + PopularGenresString;

SingersInfoPage.singersInfoPage.viewsSingerTB.Text = "Просмотров у артиста: " + SelectedSinger.ViewSinger;

}

}

private RelayCommand openAlbumsInfoPageCommand;

public RelayCommand OpenAlbumsInfoPageCommand

{

get

{

return openAlbumsInfoPageCommand ?? new RelayCommand(obj =>

{

OpenAlbumsInfoPageMethod();

});

}

}

private void OpenAlbumsInfoPageMethod()

{

AlbumsInfoPage albumsInfoPage = new AlbumsInfoPage();

BrowsePage.browsePage.infoFR.NavigationService.Navigate(albumsInfoPage, Visibility.Visible);

}

private RelayCommand openPlaylistsInfoPageCommand;

public RelayCommand OpenPlaylistsInfoPageCommand

{

get

{

return openPlaylistsInfoPageCommand ?? new RelayCommand(obj =>

{

//OpenPlaylistsInfoPageMethod();

});

}

}

// Добавление выбранного трека в медиатеку

private RelayCommand addToLibraryCommand;

public RelayCommand AddToLibraryCommand

{

get

{

return addToLibraryCommand ?? new RelayCommand(obj =>

{

if (CurrentUser != null && SelectedTrack != null)

{

ServicesUser.AddToLibraryMethod(SelectedTrack);

}

});

}

}

// Открытие окна добавления в плейлист

private RelayCommand openAddToPlaylistCommand;

public RelayCommand OpenAddToPlaylistCommand

{

get

{

return openAddToPlaylistCommand ?? new RelayCommand(obj =>

{

if (BrowseUser.CurrentUser != null && SelectedTrack != null)

{

OpenAddToPlaylistWindow();

}

});

}

}

private void OpenAddToPlaylistWindow()

{

if (SelectedTrack != null)

{

AddToPlaylistWindow addToPlaylistWindow = new AddToPlaylistWindow();

addToPlaylistWindow.ShowDialog();

}

}

// Открытие окна создания плейлиста

private RelayCommand openCreatePlaylistWindow;

public RelayCommand OpenCreatePlaylistWindow

{

get

{

return openCreatePlaylistWindow ?? new RelayCommand(obj =>

{

if (BrowseUser.CurrentUser != null && SelectedTrack != null)

{

OpenCreatePlaylistWindowMethod();

}

});

}

}

public void OpenCreatePlaylistWindowMethod()

{

CreatePlaylistWindow createPlaylistWindow = new CreatePlaylistWindow();

createPlaylistWindow.ShowDialog();

}

// Добавление в плейлист

private RelayCommand addToPlaylistCommand;

public RelayCommand AddToPlaylistCommand

{

get

{

return addToLibraryCommand ?? new RelayCommand(obj =>

{

ServicesPlaylist.AddToPlaylistMethod();

});

}

}

// Создание плейлиста

private RelayCommand createPlaylistCommand;

public RelayCommand CreatePlaylistCommand

{

get

{

return createPlaylistCommand ?? new RelayCommand(obj =>

{

ServicesPlaylist.AddPlaylistMethod(NewPlaylist);

});

}

}

// Выбор изображения для плейлиста

private RelayCommand openFileDialogCommand;

public RelayCommand OpenFileDialogCommand

{

get

{

return openFileDialogCommand ?? (openFileDialogCommand = new RelayCommand(obj =>

{

OpenFileDialogMethod();

}));

}

}

private void OpenFileDialogMethod()

{

try

{

OpenFileDialog ofd = new OpenFileDialog();

if (ofd.ShowDialog() == true)

{

string fileName = System.IO.Path.GetFileName(ofd.FileName);

NewPlaylist.PicPlaylist = fileName;

}

}

catch

{

MessageBox.Show("Выберите изображение");

}

}

private RelayCommand showInformationCommand;

public RelayCommand ShowInformationCommand

{

get

{

return showInformationCommand ?? new RelayCommand(obj =>

{

ShowInformationMethod();

});

}

}

private void ShowInformationMethod()

{

MessageBox.Show("assss");

}

}

}

using DevExpress.Mvvm;

using Microsoft.Win32;

using NOVER.Models;

using NOVER.Services;

using NOVER.Views;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading.Tasks.Dataflow;

using System.Windows;

using System.Windows.Input;

namespace NOVER.ViewModels

{

public class BrowseUser : BindableBase, INotifyPropertyChanged

{

public ICommand NavigateCommand { get; }

public event PropertyChangedEventHandler PropertyChanged;

private List<User> users = ServicesUser.GetUsers();

private void NotifyPropertyChanged(string propertyName)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName));

}

}

public List<User> Users

{

get

{

return users;

}

set

{

users = value;

NotifyPropertyChanged("Users");

}

}

public static User? SelectedUser { get; set; }

public static User CurrentUser { get; set; }

public User NewUser { get; set; }

public BrowseUser()

{

NewUser = new User();

}

private RelayCommand openMainWindow;

public RelayCommand OpenMainWindow

{

get

{

return openMainWindow ?? new RelayCommand(obj =>

{

OpenMainWindowMethod(obj as Window);

});

}

}

private static void OpenMainWindowMethod(Window currentWindow)

{

MainWindow mainWindow = new MainWindow();

mainWindow.Show();

currentWindow?.Close();

}

private RelayCommand openSignUpWindow;

public RelayCommand OpenSignUpWindow

{

get

{

return openSignUpWindow ?? new RelayCommand(obj =>

{

OpenSignUpWindowMethod();

});

}

}

public static void OpenSignUpWindowMethod()

{

SignUpWindow signUpWindow = new SignUpWindow();

signUpWindow.Owner = Application.Current.MainWindow;

signUpWindow.WindowStartupLocation = System.Windows.WindowStartupLocation.CenterOwner;

signUpWindow.Show();

}

private RelayCommand openFileDialogCommand;

public RelayCommand OpenFileDialogCommand

{

get

{

return openFileDialogCommand ?? (openFileDialogCommand = new RelayCommand(obj =>

{

OpenFileDialogMethod();

}));

}

}

private void OpenFileDialogMethod()

{

try

{

OpenFileDialog ofd = new OpenFileDialog();

if (ofd.ShowDialog() == true)

{

string fileName = System.IO.Path.GetFileName(ofd.FileName);

NewUser.PicUser = fileName;

}

}

catch

{

MessageBox.Show("Выберите изображение");

}

}

private RelayCommand signUpCommand;

public RelayCommand SignUpCommand

{

get

{

return signUpCommand ?? (signUpCommand = new RelayCommand(param =>

{

var window = param as SignUpWindow;

ServicesUser.SignUpMethod(NewUser, window);

}));

}

}

private RelayCommand logInCommand;

public RelayCommand LogInCommand

{

get

{

return logInCommand ?? new RelayCommand(obj =>

{

LogInMethod();

});

}

}

private void LogInMethod()

{

string login = LogInWindow.logInWindow.loginTB.Text;

string password = LogInWindow.logInWindow.passwordTB.Password.ToString();

bool isLogIn = false;

foreach (User user in users)

{

if (user.LoginUser == login && user.PassUser == password)

{

CurrentUser = user;

LogInWindow logInWindow = new LogInWindow();

OpenMainWindowMethod(logInWindow);

isLogIn = true;

}

}

if (!isLogIn)

{

MessageBox.Show("Неверный логин или пароль\nПожалуйста, повторите попытку", "Ошибка");

}

}

private RelayCommand logOutCommand;

public RelayCommand LogOutCommand

{

get

{

return logOutCommand ?? new RelayCommand(obj =>

{

LogOutMethod();

});

}

}

private void LogOutMethod()

{

CurrentUser = null;

BrowseTrack.OpenHomePageMethod();

}

}

}

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Документация Firebase Realtime Database (android): офиц. сайт – URL: https://firebase.google.com/docs/database/android/start?hl=en (дата обращения: 26.01.2024).
2. Документация Firebase Realtime Database (web) – firebase.google.com – URL: https://firebase.google.com/docs/database/web/start?hl=en (дата обращения: 26.01.2024) – Текст: электронный.
3. Документация MapKit: офиц. сайт – URL: https://yandex.ru/dev/mapkit/doc/ru/ (дата обращения: 26.01.2024) – Текст: электронный.
4. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. – Питер, 2022. – 352 с. – ISBN 978-5-4461-0772-8 – Текст: непосредственный.
5. Начало работы React: офиц. сайт – URL: https://ru.react.js.org/docs/getting-started.html (дата обращения: 26.01.2024).
6. Стефанов С. React. Быстрый старт / Стефанов С. – 2-е изд. – Питер, 2023. – 304 с. – ISBN 978-5-4461-2115-1 – Текст: непосредственный
7. Firebase Authentication по номеру телефона – firebase.google.com – URL: https://firebase.google.com/docs/auth/android/phone-auth?hl=en (дата
8. Yandex MapKit Android — нажатие на маркер – ruclient.ru – URL: https://ruclient.ru/yandex-mapkit-android-nazatie-na-marker/ (дата обращения: 26.01.2024) – Текст: электронный.
9. https://habr.com/ru/companies/joydev/articles/754800/ (дата обращения: 26.01.2024) – Текст: электронный.
10. C-sharpcorner [Электронный ресурс] / Руководство по MVVM. – Режим доступа https://www.c-sharpcorner.com/UploadFile/ptmujeeb/wpf-mvvm-pattern-a-simple-tutorial-for-absolute-beginners/, свободныйWPF Tutorial [Электронный ресурс] / Руководство по WPF. – Режим доступа https://wpf-tutorial.com/xaml/what-is-xaml/, свободный
11. GitHub [Электронный ресурс] / Хостинг Веб-Сервис. – Режим доступа https://github.com/, свободный.