МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Цифровой музыкальный плеер “Allegro”»

Исполнитель

студент 2 курса группы 6 Козак Олег Дмитриевич

(Ф.И.О.)

Руководитель работы ассистент Чистякова Ю.А.

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

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2023

Содержание

[Введение 3](#_Toc135206283)

[1. Анализ прототипов и литературных источников 4](#_Toc135206284)

[1.1. Анализ прототипов 4](#_Toc135206285)

[1.1.1. Spotify 4](#_Toc135206286)

[1.1.2. Яндекс.Музыка. 5](#_Toc135206287)

[1.1.3. Apple Music. 6](#_Toc135206288)

[1.2. Постановка задачи 7](#_Toc135206289)

[2 Анализ требований к программному средству и разработка функциональных требований 8](#_Toc135206290)

[2.1 Описание инструментов для разработки программного средства 8](#_Toc135206291)

[2.2 Описание функциональности программного средства 9](#_Toc135206292)

[3 Проектирование программного средства 10](#_Toc135206293)

[3.1 Модель базы данных 10](#_Toc135206294)

[3.2 Структура проекта 12](#_Toc135206295)

[3.3 Структура классов программного средства 13](#_Toc135206296)

[3.4 Работа с базой данных 14](#_Toc135206297)

[3.5 Аутентификация пользователей в системе 15](#_Toc135206298)

[3.6 Окна приложения 15](#_Toc135206299)

[4 Реализация программного средства 17](#_Toc135206300)

[4.1 Реализация MVVM и других шаблонов 17](#_Toc135206301)

[4.2 Добавление песни в избранные 27](#_Toc135206302)

[4.3 Реализация библиотеки CustomControls 27](#_Toc135206303)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 30](#_Toc135206304)

[6 Методика использования программного средства 33](#_Toc135206305)

[6.1 Регистрация и авторизация 33](#_Toc135206306)

[6.2 Домашнее представление 34](#_Toc135206307)

[6.3 Администрирование 36](#_Toc135206308)

[6.4 Смена пользователя 38](#_Toc135206309)

[Заключение 39](#_Toc135206310)

[Список литературы 40](#_Toc135206311)

[Приложение А 41](#_Toc135206312)

# Введение

В современном мире музыка занимает важное место в жизни людей.

Желание всегда иметь возможность слушать любимые композиции, привело к

появлению большого количества приложений для поиска и потребления

музыкального контента. В данном курсовом проекте будет рассмотрено создание

программного средства «Цифровой музыкальный плеер Allegro», которое

позволяет слушать и формировать свои плейлисты из доступной музыки, которую

добавляет администратор. В данной работе будет рассмотрен весь процесс

разработки приложения, от анализа требований до тестирования и релиза, а также

будут описаны основные особенности и преимущества разработанного

приложения.

Цель данного курсового проекта - создать приложение, которое будет

предоставлять пользователям удобный и функциональный инструмент для

прослушивания музыки. В рамках проекта будут рассмотрены все этапы

разработки программного средства, начиная от анализа требований и выбора

технологий, заканчивая тестированием и документированием результатов.

Программное средство Цифровой музыкальный плеер Allegro» будет

предоставлять пользователям множество возможностей, таких как создание

индивидуальных плейлистов, прослушивание и поиск новых композиций. Кроме

того, приложение будет иметь удобный и интуитивно понятный интерфейс,

который позволит пользователям быстро и легко находить необходимую

информацию.

Главная задача данного курсового проектирования – это разработка

программного средства, которое реализует все вышеперечисленные функции и

решает поставленные задачи. Язык разработки проекта – C#. При выполнении

курсового проекта будут использованы принципы и приемы ООП, база данных MS

SQL Server, и технология Windows Presentation Foundation (WPF).

# 1. Анализ прототипов и литературных источников

В настоящее время существует большое количество программных продуктов для прослушивания музыки, а программируемые системы облегчают работу с базами данных. Использование новейших технологий является логичным выбором для создания приложений на современной технологической основе.

Для того чтобы окончательно определиться с постановкой задачи курсового проекта, необходимо проанализировать прототипы программных средств выбранной темы.

# 1.1. Анализ прототипов

При изучении программ-прототипов было выявлено несколько схожих по функциональности программ. Все программные средства представлены в качестве приложений для прослушивания музыки. Результат анализа представлен ниже.

# 1.1.1. Spotify

В качестве одного из аналогов возьмем программу «Spotify». «Spotify» - [стриминговый сервис](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BC%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0#%D0%A1%D1%82%D1%80%D0%B8%D0%BC%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D1%8B%D0%B5_%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D1%8B), позволяющий легально прослушивать музыкальные композиции, [аудиокниги](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D0%BE%D0%BA%D0%BD%D0%B8%D0%B3%D0%B0) и [подкасты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BA%D0%B0%D1%81%D1%82%D0%B8%D0%BD%D0%B3), не скачивая их на устройство. Доступен в виде [веб-сайта](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B0%D0%B9%D1%82), [приложений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B1%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) для всех операционных систем, смартфонов, смарт-устройств и медиасистем автомобилей.

Интерфейс программы представлен на рисунке 1.1.

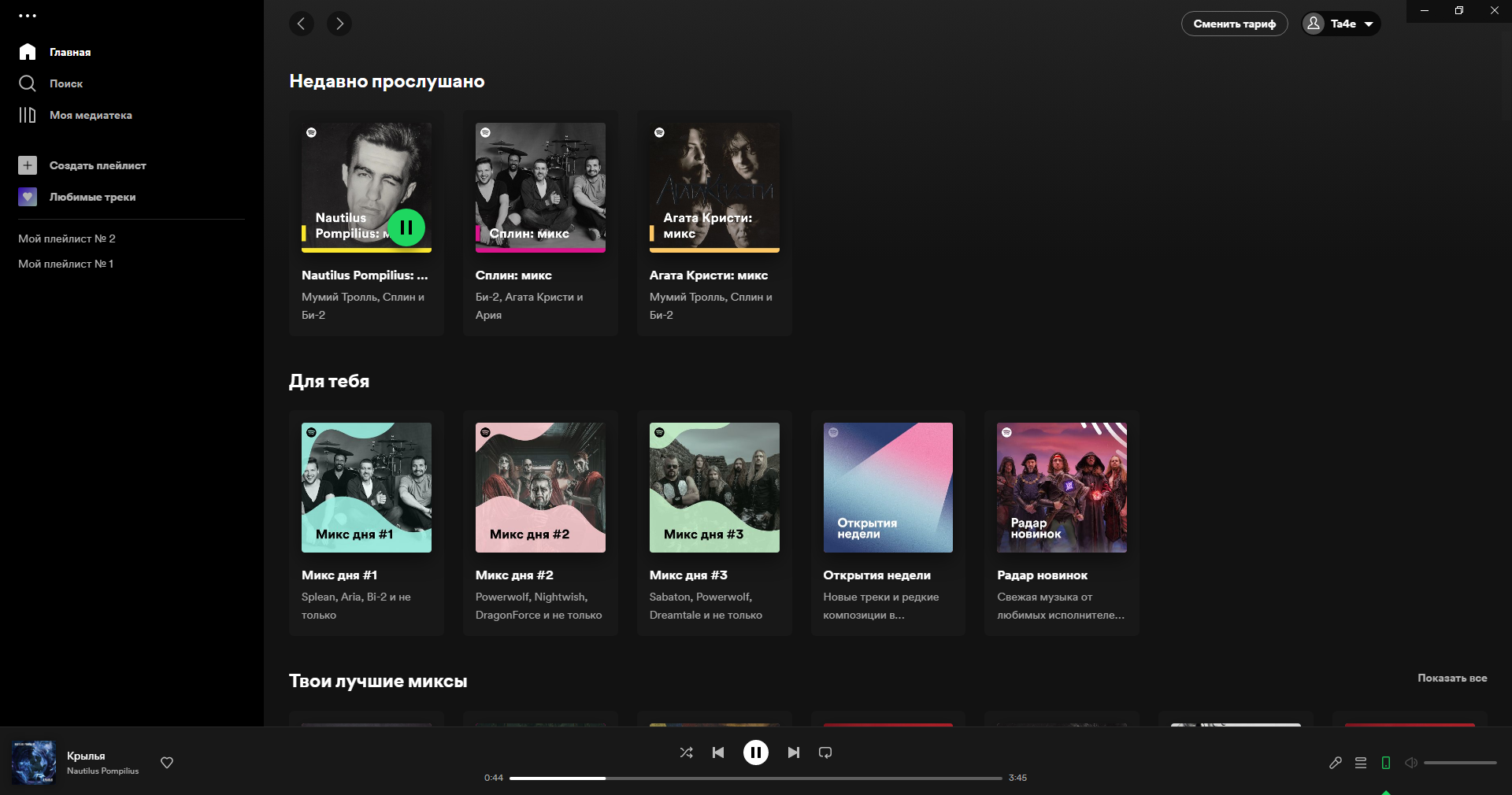


Рисунок 1.1 – Интерфейс «Spotify»

Изучив программное средство Spotify, можно выделить его основные достоинства и недостатки.

Достоинства:

* Огромный каталог музыки с более чем 70 миллионами треков доступен для потоковой передачи.
* Высокое качество звука с возможностью стриминга в формате Lossless.
* Персонализированные рекомендации на основе прослушиваемых пользователем треков и их предпочтений.
* Удобный интерфейс с возможностью создания и редактирования плейлистов.

Недостатки:

* Наличие рекламы в бесплатной версии сервиса, что может быть раздражающим для пользователей.
* Ограничения в использовании для пользователей из некоторых стран.
* Высокая стоимость подписки на премиум-планы, особенно для семейного использования.
* Отсутствие возможности покупки музыки для хранения на устройствах, что ограничивает пользователей, которые хотят иметь физический доступ к своей музыке.

# 1.1.2. Яндекс.Музыка.

Следующая программа для анализа – «Яндекс.Музыка».  
Яндекс.Музыка – ещё один популярный стриминговый сервис музыки. Функционал схож с «Spotify».

Интерфейс программы представлен на рисунке 1.2.

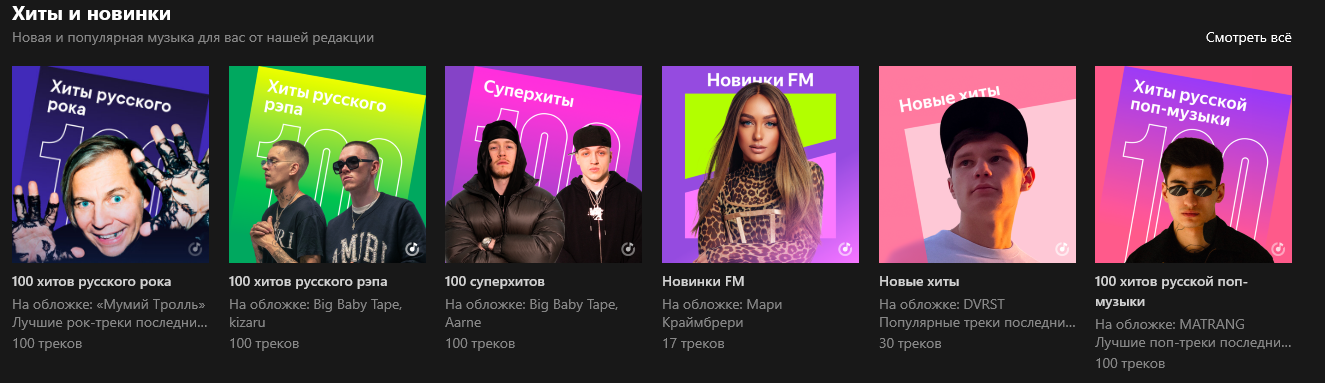


Рисунок 1.2 – Яндекс.Музыка

Достоинства:

* Огромная библиотека музыки, включая русскую и зарубежную музыку, подкасты и радио.
* Персонализированные рекомендации на основе прослушиваемых пользователем треков, жанров и настроений.
* Отсутствие рекламы для пользователей, которые оплачивают премиум-подписку.
* Совместимость с голосовыми помощниками, такими как Алиса и Siri.

Недостатки:

* Ограничения на использование для пользователей за пределами России и стран СНГ.
* Отсутствие функции Lossless, которая доступна в некоторых других музыкальных сервисах.
* Низкое качество звука в бесплатной версии приложения.
* Некоторые функции, такие как создание собственных плейлистов и использование функции "Только Wi-Fi", доступны только для пользователей с премиум-подпиской.

# 1.1.3. Apple Music.

Apple Music - это музыкальный стриминговый сервис, предлагаемый компанией Apple. Сервис позволяет пользователю доступ к огромной библиотеке музыки, включая тысячи альбомов и плейлистов, а также оригинальные программы и шоу, созданные специально для Apple Music Оригинальный контент: Apple Music также предлагает оригинальный контент, включая концерты, интервью и документальные фильмы о музыкантах.

Интерфейс программы представлен на рисунке 1.3.

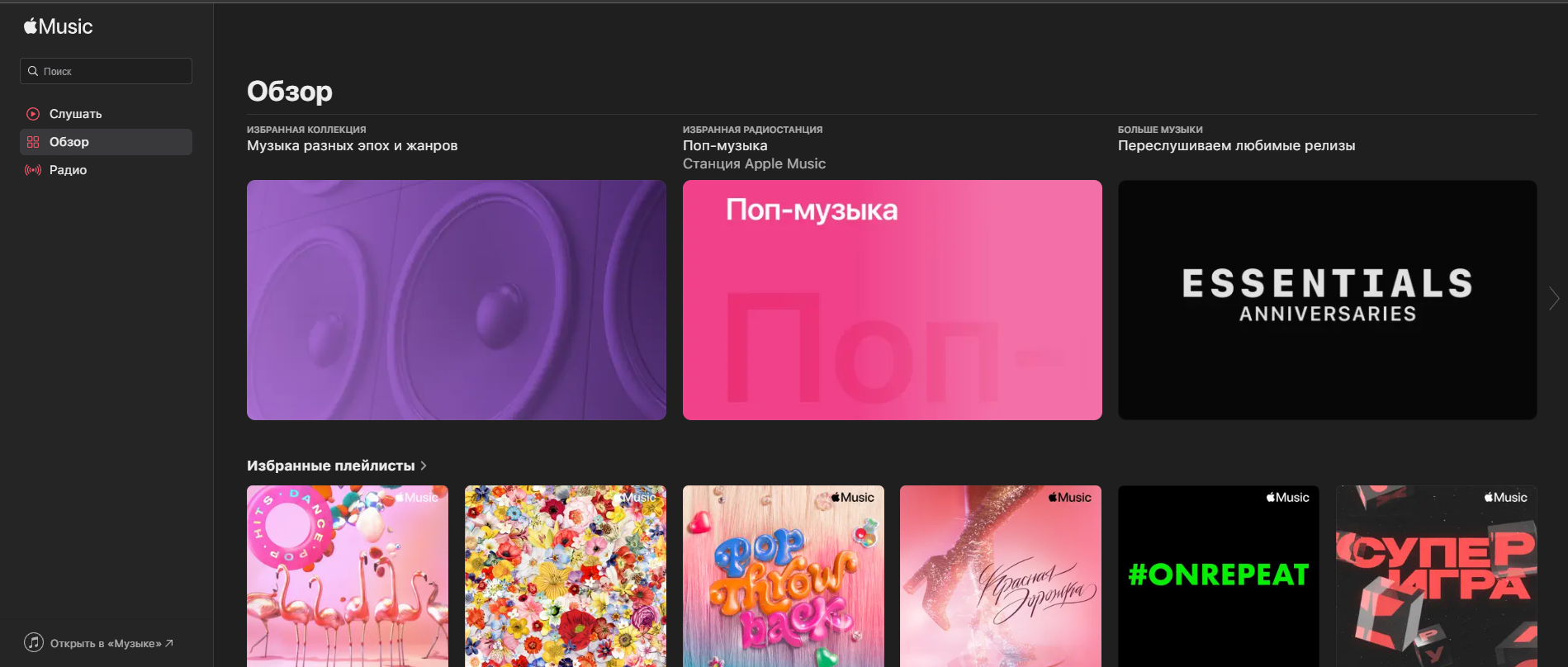


Рисунок 1.3 – Apple Music

Достоинства:

* Большая библиотека музыки.
* Оригинальные программы и шоу: сервис предлагает эксклюзивный контент, созданный специально для Apple Music, включая интересные интервью и радиошоу.
* Высокое качество звука: Apple Music предлагает потоковое воспроизведение музыки высокого качества, включая формат Lossless и пространственный звук Dolby Atmos.

Недостатки:

* Невозможность использования на устройствах без операционной системы Apple: в отличие от других сервисов, таких как Spotify, Apple Music не поддерживает использование на всех устройствах, что ограничивает доступность сервиса.
* Ограниченная бесплатная версия: в отличие от других сервисов, таких как Spotify, Apple Music не предлагает полноценную бесплатную версию с ограниченным доступом к музыке.
* Высокая цена: в сравнении с другими сервисами, Apple Music предлагает довольно высокую цену за подписку, что может сделать его менее доступным для некоторых пользователей.

# 1.2. Постановка задачи

Проанализировав прототипы, были выделены основная задача и функциональные требования проекта.

Основной задачей курсового проекта является разработка десктопного приложения, позволяющего выполнять авторизацию и регистрацию пользователей, пользователю в лице администратора добавлять музыку с информацией о ней с возможностью удаления и редактирования. Пользователю просматривать всю доступную музыку, добавлять в избранные, создавать и редактировать плейлисты.

Функциональные требования описаны в главе 2.

# 2 Анализ требований к программному средству и разработка функциональных требований

Прежде чем приступить непосредственно к этапу разработки проекта прежде необходимо сформулировать функциональные требования к программному средству.

2.1 Описание инструментов для разработки программного средства

В ходе разработки данного программного средства используются следующие инструменты:

* объектно-ориентированный язык программирования C#;
* платформа для кроссплатформенной разработки с открытым исходным
* кодом .NET Core;
* технология WPF;
* расширяемый язык разметки XAML;
* система управления базами данных Microsoft SQL Server;
* шаблон проектирования MVVM;
* фреймворк для работы с базой данных Entity Framework 6;
* интегрированная среда разработки Microsoft Visual Studio 2022.

В качестве интерфейса прикладного программирования был выбран обширный API-интерфейс – Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом. В основе графической технологии WPF лежит мощная инфраструктура, основанная на DirectX. Это является одним из основных отличий WPF от более ранней технологии создания пользовательских интерфейсов – Windows Forms.

C# — это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Core. Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Core предоставляет доступ ко многим службам операционной системы и другим полезным, правильным классам, что существенно ускоряет цикл разработки.

XAML — это декларативный язык разметки. С точки зрения модели программирования .NET Core язык XAML упрощает создание пользовательского интерфейса для приложения .NET Core. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов. Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке.

MVVM — шаблон проектирования архитектуры приложения, который позволяет отделить логику приложения от визуальной части, что упрощает тестирование и поддержку приложения.

2.2 Описание функциональности программного средства

На основе анализа были составлены следующие функциональные требования для программного средства для клиента:

* Выполнять регистрацию и авторизацию;
* Поиск композиций
* Формирование плейлистов;
* Редактирование пелйлистов
* Прослушивание композиции;
* Добавление композиций в избранные

Функции администратора:

* Поддерживать работу c базой данных;
* Добавление и удаление композиций
* Редактирование композиций

В графическом материале приведена схема Use-case, описывающая функциональность программного средства для каждого из пользователей.

Из данной схемы видно, что клиент может выполнять операции по поиску песен, их добавление и удаление из списка избранных, а также формировать плейлисты на свой вкус и редактировать их. Администратор может изменять песни, добавлять и удалять композиции.

# 3 Проектирование программного средства

3.1 Модель базы данных

Для создания программного средства была разработана база данных AllegroBD, состоящая из 8 таблиц. Описания таблиц базы данных представлены в таблицах 3.1 – 3.8:

Таблица 3.1 – Таблица Admin

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор админа |
| Name | nvarchar(MAX) | Содержит имя админа |
| Password | nvarchar(MAX) | Содержит пароль админа |

Таблица AdminList предназначена для хранения песен.

Таблица 3.2 – Таблица AdminList

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор админа |
| IdSong | int | Содержит идентификатор песни |

В таблице Users хранится информация о пользователях.

Таблица 3.3 – Таблица Users

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор пользователя |
| UserName | nvarchar(MAX) | Содержит имя пользователя |
| Password | nvarchar(MAX) | Содержит пароль пользователя |
| Email | nvarchar(MAX) | Содержит электронную почту пользователя |

Таблица UserList предназначена для хранения избранных песен.

Таблица 3.4 – Таблица UserList

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор списка |
| List\_Name | nvarchar(MAX) | Содержит название списка |

Таблица UsersListMusic является дополнительной таблицей для связи UserList с песнями.

Таблица 3.5 – Таблица UsersListMusic

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор таблицы |
| UserListId | int | Содержит идентификатор списка избранных |
| SoungId | int | Содержит идентификатор песни |

Таблица PlayList хранит информацию о плейлистах пользователя.

Таблица 3.6 – Таблица PlayList

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор плейлиста |
| UserID | int | Содержит идентификатор пользователя |
| PlayList\_Name | nvarchar(MAX) | Содержит название плейлиста |
| ImagePath | nvarchar(MAX) | Содержит путь к картинке |
| Author | nvarchar(MAX) | Содержит имя автора плейлиста |

Таблица PlayListItem является дополнительной таблицей для связи PlayList с песнями.

Таблица 3.7 – Таблица PlayListItem

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Id PlayListItem |
| PlayListId | int | ID плейлиста |
| SoungId | int | ID песни |

Таблица Songs хранит всю информацию о композиции.

Таблица 3.8 – Таблица Songs

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Id | int | Содержит идентификатор песни |
| Name | nvarchar(MAX) | Содержит название песни |
| Title | nvarchar(MAX) | Содержит описание песни |
| Artist | nvarchar(MAX) | Содержит название автора песни |
| ImagePath | nvarchar(MAX) | Содержит путь к картинке |
| SongsPath | nvarchar(MAX) | Содержит путь к песне |
| Duration | nvarchar(MAX) | Содержит длительность песне |
| AdminListId | int | Содержит идентификатор общего списка |

Диаграмма базы данных, отображающая связи между таблицами, представлена в графическом материале.

3.2 Структура проекта

Программное средство выполнено одним проектам структура которого показана на рисунке 3.2

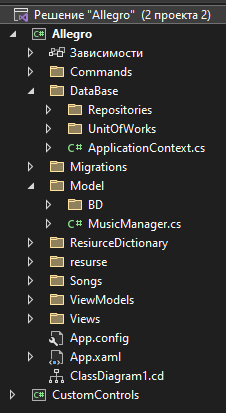


Рисунок 3.2 – Структура проекта

Описание хранимых в папках проекта элементах представлено в таблице 3.9.

Таблица 3.9 – Описание структуры проекта

|  |  |
| --- | --- |
| Элемент | Описание |
| Папка «Command» | Содержит команды |
| Папка «DataBase» | Содержит в себе реализацию паттернов репозиторий и UnitOfWorks, а также контекст БД |
| Папка «Migration» | Содержит миграцию, создаётся автоматически |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| Элемент | Описание |
| Папка «Model» | Содержит модели для БД, а также менеджера |
| Папка «ResiurceDictionary» | Содержит словари ресурсов, а также темы для приложения |
| Папка «resurse» | Содержит картинки |
| Папка «Songs» | Содержит песни |
| Папка «ViewModels» | Содержит ViewModels для MVVM паттерна |
| Папка «Views» | Содержит представления, которые отвечают за визуальную часть для MVVM паттерна |
| App.config | Содержит строки конфигурации приложения |
| App.xaml | Содержит стартовую точку, с которой начинается выполнение приложения |

## **3.3 Структура классов программного средства**

Структура классов программного средства представлена в виде диаграммы классов.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Основными элементами являются классы и связи между ними. Классы характеризуются при помощи атрибутов и операций.

Атрибуты описывают свойства объектов класса. Большинство объектов в классе получают свою индивидуальность из-за различий в их атрибутах и взаимосвязи с другими объектами. Однако, возможны объекты с идентичными значениями атрибутов и взаимосвязей. Т.е. индивидуальность объектов определяется самим фактом их существования, а не различиями в их свойствах. Имя атрибута должно быть уникально в пределах класса. За именем атрибута может следовать его тип и значение по умолчанию.

В данном пункте будут рассматриваться только классы модели приложения без структуры модели представления.

Модель приложения представлена 8 основными классами.

Класс Admin определят админа, который имеет свой идентификатор, имя и пароль.

Класс AdminList определяет главный список музыки, имеет ссылку на Songs.

Класс PlayList определяет плейлист пользователя. Имеет ссылки на User, содержит уникальный идентификатор, название, путь к картинке и автора.

Класс PlayListItem определяет элемент плейлиста, имеет ссылку на плейлист и на песню, содержит уникальный идентификатор.

Класс Songs определяет саму песню, имеет ID, название, описание, автора, картинку, путь к песне и длительность.

Класс UserList определяет список избранных песен, имеет ссылку на User. Содержит ID, и название.

Класс UserListMusic определяет элемент UserList. Содержит ID, имеет ссылку на UserList и на пользователя.

Класс Users определяет пользователя. Имеет ссылку на UserList. Имеет следующие поля: ID, имя пользователя, пароль, электронную почту.

На основании данных классов была создана база данных и ее таблицы. База данных с таблицами создавалась с помощью Entity Framework 6.

Диаграмма классов представлена в графической части.

3.4 Работа с базой данных

Для доступа к базе данных используется Entity Framework 6. Этот подход предоставляет ряд существенных преимуществ: нам не нужно беспокоиться о коде доступа к данным и не нужно знать деталей работы СУБД SQL Server и синтаксиса языка запросов T-SQL, вместо этого мы работаем с таблицами базы данных как с классами C#, с полями этих таблиц - как со свойствами классов, а синтаксис SQL-запросов заменен на  [LINQ](https://professorweb.ru/my/LINQ/base/level1/info_linq.php). Entity Framework берет на себя обязанности по преобразованию кода C# в SQL-инструкции.

Существуют 3 подхода по проектированию базы данных:

* Database-First;
* Model-First;
* Code-First.

В своей работе я использовал подход Code-First, при данном подходе рабочий процесс в Entity Framework предлагает разработчикам начать с определения модели данных в виде классов (сущностей) в коде приложения. Затем на основе этих классов Entity Framework создаст соответствующую базу данных.

Подход Code-First позволяет гибко определять модель данных в коде и автоматически создавать соответствующую базу данных. Он широко используется в современной разработке приложений, обеспечивая удобство и гибкость в работе с данными.

3.5 Аутентификация пользователей в системе

Программное средство подразумевает использование несколькими пользователями с разными правами. Для этого необходима регистрация и аутентификация пользователей в системе. В системе может зарегистрироваться только пользователь, администратор добавляется напрямую. Данные, введенные пользователями, должны проходить валидацию, а пароль – хешироваться для безопасности.

3.6 Окна приложения

Набор окон приложения доступных каждому пользователю, будет отличаться, т. к. отличаются их возможности в системе. Для этого необходимо разработать схему переходов для каждого пользователя. Запуск ПО будет начинаться с окна логина, откуда пользователь может перейти к регистрации, или войти и перейти к основному меню. Далее он может посмотреть главный список музыки, посмотреть список избранных треков, открыть страничку с плейлистами, а также посмотреть содержимое плейлистов. Администратор, в свою очередь, может добавлять и удалять музыку из общего списка, а также редактировать её. Схемы пользователей представлены на рисунках 3.3-3.4.

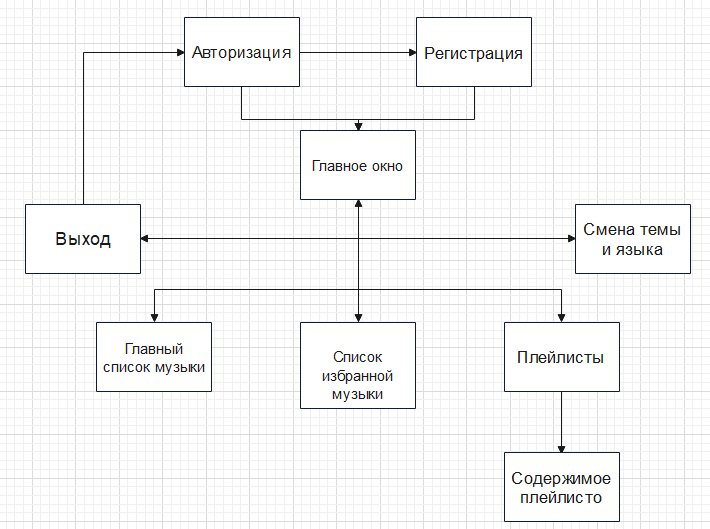


Рисунок 3.3 – Схема окон пользователя

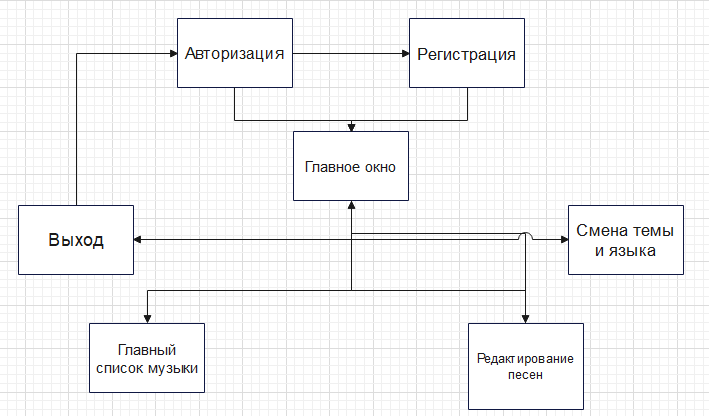


Рисунок 3.4 – Схема окон администратора

Диаграмма последовательностей для добавления в плейлист приведена в графическом материале.

# 4 Реализация программного средства

4.1 Реализация MVVM и других шаблонов

Паттерн MVVM реализуется через базовый класс ViewModelBase (листинг 4.1), наследующий интерфейс INotifyPropertyChanged. Данный интерфейс используется для уведомления представления об изменениях свойств объекта. Реализация классом интерфейса предполагает генерацию события PropertyChanged каждый раз, когда значение свойства объекта изменяется. Такое поведение позволяет привязкам данных отслеживать состояние объекта и обновлять данные пользовательского интерфейса при изменении значения связанного свойства.На основе класса ViewModelBase сделаны все ViewModel, присутствующие в данном проекте.

|  |
| --- |
| public abstract class ViewModelBase : INotifyPropertyChanged  {  public event PropertyChangedEventHandler PropertyChanged;  protected void OnPropertyChanged(string propertyName)  {  PropertyChangedEventHandler handler = PropertyChanged;  if (handler != null)  {  handler(this, new PropertyChangedEventArgs(propertyName));  }  }  #region Fields  protected static ApplicationContext db = new();  protected static Users? CurrentUser = new();  protected static Admin? CurrentAdmin = new();  public static MainView? MainView = new MainView();  private static Frame? MainFrame;  private static bool isAdmin = false;  private static bool isUser = false;  private static List<Songs> songs = new(db.GetSongList());  public bool IsAdmin { get { return isAdmin; } set { isAdmin = value; OnPropertyChanged(nameof(IsAdmin)); } }  public bool IsUser { get { return isUser; } set { isUser = value; OnPropertyChanged(nameof(IsUser)); } }  protected static void ShowMainWindow()  {  MainView view = new MainView();  MainView = view;  MainFrame = view.MainFrame;  view.Show();  }  protected static void CloseMainView()  {  MusicManager.Instance.Pause();  MainView.Close();  }  protected static void ShowPage(Page page)  {  MainFrame.Content = page;  }  protected void ShowWindow(Window window)  {  window.Show();  }  } |

Листинг 4.1 – структура класса ViewModelBase

Примером реализации потомка данного класса может служить класс AddListMusicViewModel (листинг 4.2).

|  |
| --- |
| public class AddListMusicViewModel : ViewModelBase  {  private readonly Users users;  private List<PlayList> playList;  public AddListMusicViewModel()  {  users = CurrentUser;  playList = MusicManager.Instance.unitOfWork.PlayListRepositories.Get(users).ToList();  }  public List<PlayList> PlayList  {  get  {  return MusicManager.Instance.unitOfWork.PlayListRepositories.Get(users).ToList();  }  set  {  playList = value;  OnPropertyChanged("PlayList");  }  }  #region  public Frame CurrentFrame  {  set  {  MainView.MainFrame.Content = value;  OnPropertyChanged(nameof(CurrentFrame));  }  }  #endregion  #region Выбранный элемент  private PlayList \_selectedMusic;  public PlayList SelectedPlayList  {  get { return \_selectedMusic; }  set  {  if (\_selectedMusic != value)  {  \_selectedMusic = value;  OnPropertyChanged("SelectedPlayList");  }  }  }  #endregion  #region Oткрытие нового окна  private DelegateCommand openNewWindows;  public ICommand IOpenNewWindows  {  get  {  if (openNewWindows == null)  {  openNewWindows = new DelegateCommand(OpenNewWindow);  }  return openNewWindows;  }  }  private static void OpenNewWindow()  {  var newWindow = new NewPlayListView();  var newWindowViewModel = new NewPlayListViewModel(newWindow);  newWindow.DataContext = newWindowViewModel;  newWindow.ShowDialog();  }  #endregion  #region Открытие фрейма со списком песен  private DelegateCommand openNewFrameListMusicAdd;  public ICommand OpenNewFrameListMusicAdd  {  get  {  if (openNewFrameListMusicAdd == null)  {  openNewFrameListMusicAdd = new DelegateCommand(OpenNewFrameListMusicAddMe);  }  return openNewFrameListMusicAdd;  }  }  private void OpenNewFrameListMusicAddMe()  {  if (SelectedPlayList == null)  {  return;  }  var newContext = new AddMusicInPlayListViewModel(SelectedPlayList);  CurrentFrame = new Frame() { Content = new AddMusicInPlayListView() { DataContext = newContext } };  }  #endregion  #region удаление плей листа  #region Удаление  private DelegateCommand deleteCommand;  public ICommand Delete  {  get  {  if (deleteCommand == null)  {  deleteCommand = new DelegateCommand(deleteMusic);  }  return deleteCommand;  }  }  private void deleteMusic()  {  if (SelectedPlayList == null) { return; }  MusicManager.Instance.unitOfWork.PlayListRepositories.Delete(SelectedPlayList.Id);  MusicManager.Instance.unitOfWork.Save();  PlayList = MusicManager.Instance.unitOfWork.PlayListRepositories.Get(users).ToList();  SelectedPlayList = null;  }  public bool CanDeleteItem  {  get  {  return SelectedPlayList != null;  }  }  #endregion  #endregion  } |

Листинг 4.2 – структура класса AddListMusicViewModel

В приложении используется паттерн Command который позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. В WPF команды представлены интерфейсом ICommand. Также есть классы которые позволяет делегировать командную логику методам, передаваемым в качестве параметров, пример представлен на листинге 4.3

|  |
| --- |
| public class DelegateCommand : ICommand  {  #region Constructors  /// <summary>  /// Constructor  /// </summary>  public DelegateCommand(Action executeMethod)  : this(executeMethod, null, false)  {  }  /// <summary>  /// Constructor  /// </summary>  public DelegateCommand(Action executeMethod, Func<bool> canExecuteMethod)  : this(executeMethod, canExecuteMethod, false)  {  }  /// <summary>  /// Constructor  /// </summary>  public DelegateCommand(Action executeMethod, Func<bool> canExecuteMethod, bool isAutomaticRequeryDisabled)  {  if (executeMethod == null)  {  throw new ArgumentNullException("executeMethod");  }  \_executeMethod = executeMethod;  \_canExecuteMethod = canExecuteMethod;  \_isAutomaticRequeryDisabled = isAutomaticRequeryDisabled;  }  #endregion  #region Public Methods  /// <summary>  /// Method to determine if the command can be executed  /// </summary>  public bool CanExecute()  {  if (\_canExecuteMethod != null)  {  return \_canExecuteMethod();  }  return true;  }  /// <summary>  /// Execution of the command  /// </summary>  public void Execute()  {  if (\_executeMethod != null)  {  \_executeMethod();  }  }  void ICommand.Execute(object parameter)  {  Execute();  }  #endregion  #region Data  private readonly Action \_executeMethod = null;  private readonly Func<bool> \_canExecuteMethod = null;  private bool \_isAutomaticRequeryDisabled = false;  private List<WeakReference> \_canExecuteChangedHandlers;  #endregion  } |

Листинг 4.3 – пример класса DelegateCommand

В приложении, для удобства использования общих ресурсов реализован паттерн «Одиночка», благодаря чему мы получаем объект контекста только 1 раз, а после при обращении просто его возвращает. Для решения этой проблемы паттерн Singleton возлагает контроль над созданием единственного объекта на сам класс. Доступ к этому объекту осуществляется через статическую функцию-член класса, которая возвращает указатель или ссылку на него. Этот объект будет создан только при первом обращении к методу, а все последующие вызовы просто возвращают его адрес. Для обеспечения уникальности объекта, конструкторы и оператор присваивания объявляются закрытыми.

|  |
| --- |
| public class MusicManager  {  public static MusicManager Instance { get; set; }  public UnitOfWork unitOfWork = new UnitOfWork();  private WindowsMediaPlayer player = new WindowsMediaPlayer(); //private  public Songs selectSongs = new Songs();  private int currentIndex = -1;  public List<Songs> SongsList = new List<Songs>();  static MusicManager()  {  Instance = new MusicManager();  }  /// <summary>  public void PlaySong()  {  if (selectSongs == null)  {  return;  }  currentIndex = SongsList.IndexOf(selectSongs);  player.URL = selectSongs.SongsPath;  player.controls.play();  }  public void Pause()  {  player.controls.pause();  }  public void PlayingAfterPause()  {  player.controls.play();  }  public void PlayNext()  {  if (SongsList.Count == 0)  {  return;  }  if (currentIndex < SongsList.Count - 1)  {  currentIndex++;  }  else  {  currentIndex = 0;  }  selectSongs = SongsList[currentIndex];  PlaySong();  }  public void PlayPrevious()  {  if (SongsList.Count == 0)  {  return;  }  if (currentIndex > 0)  {  currentIndex--;  }  else  {  currentIndex = SongsList.Count - 1;  }  selectSongs = SongsList[currentIndex];  PlaySong();  }  public bool IsPlaying()  {  return player.playState == WMPPlayState.wmppsPlaying;  }  public double GetCurrentPosition()  {  return player.controls.currentPosition;  }  public double GetSongDuration()  {  return player.currentMedia.duration;  }  public void SetPosition(double position)  {  player.controls.currentPosition = position;  }  } |

Листинг 4.4 – класс MusicManager

В данном проекте, есть реализация паттернов Репозиторий (Repository) и Единица работы (Unit of Work), они предоставляют следующие преимущества и функциональность:

* Разделение ответственности: Паттерн Репозиторий позволяет отделить код, отвечающий за доступ к данным, от бизнес-логики и пользовательского интерфейса. Репозиторий предоставляет унифицированный интерфейс для работы с данными и скрывает детали доступа к базе данных. Это упрощает разработку и поддержку приложения, позволяет легко изменять и заменять источник данных.
* Абстракция доступа к данным: Репозиторий предоставляет абстракцию для работы с данными, скрывая конкретные детали реализации доступа к базе данных через Entity Framework. Это позволяет сосредоточиться на бизнес-логике и использовать унифицированный интерфейс для выполнения операций с данными, таких как создание, чтение, обновление и удаление.
* Централизованное управление транзакциями: Паттерн Единица работы (Unit of Work) позволяет объединить несколько операций с данными в рамках одной транзакции базы данных. Это гарантирует согласованность данных и обеспечивает целостность базы данных

Улучшение тестирования и поддержки кода: Репозиторий и Единица работы обеспечивают лучшую тестируемость кода, поскольку можно легко создавать заглушки или имитировать реализацию репозитория и единицы работы для юнит-тестирования. Кроме того, паттерны позволяют упростить поддержку и изменение кода, поскольку логика доступа к данным вынесена в отдельные компоненты.

Пример реализации паттерна Репозиторий для модели User представлен на листинге 4.5, а на листинге 4.6 реализация паттерна Unit of Work

|  |
| --- |
| public class UsersRepositories : IRepository<Users>  {  private ApplicationContext db;  public UsersRepositories(ApplicationContext context)  {  this.db = context;  }  public IEnumerable<Users> GetAll()  {  return db.Users;  }  public Users Get(int id)  {  return db.Users.Find(id);  }  public void Create(Users user)  {  UserList userList = new UserList();  userList.User = user;  userList.List\_Name = "";  db.Users.Add(user);  MusicManager.Instance.unitOfWork.Save();  MusicManager.Instance.unitOfWork.UserListRepositories.Create(userList);    }  public void Update(Users userList)  {  db.Entry(userList).State = EntityState.Modified;  }  public void Delete(int id)  {  Users userList = db.Users.Find(id);  if (userList != null)  db.Users.Remove(userList);  }  public bool AuthenticateUser(NetworkCredential credential)  {  bool validUser;  return true;  }  } |

Листинг 4.5 – класс UsersRepositories

|  |
| --- |
| public class UnitOfWork : IDisposable  {  private ApplicationContext db = new ApplicationContext();  private AdminListRepositories adminListRepositories;  private AdminRepository adminRepository;  private PlayListItemRepositories playlistRepository;  private PlayListRepositories playListRepositories;  private SongsRepositories songsRepositories;  private UserListMusicRepositories userListMusicRepositories;  private UserListRepositories userListRepositories;  private UsersRepositories usersRepositories;  public AdminListRepositories AdminListRepositories  {  get  {  if (adminListRepositories == null)  adminListRepositories = new AdminListRepositories(db);  return adminListRepositories;  }  }  public PlayListItemRepositories PlayListItemRepositories  {  get  {  if (playlistRepository == null)  playlistRepository = new PlayListItemRepositories(db);  return playlistRepository;  }  }  public AdminRepository AdminRepository  {  get  {  if (adminRepository == null)  adminRepository = new AdminRepository(db);  return adminRepository;  }  }  public PlayListRepositories PlayListRepositories  {  get  {  if (playListRepositories == null)  playListRepositories = new PlayListRepositories(db);  return playListRepositories;  }  }  public SongsRepositories SongsRepositories  {  get  {  if (songsRepositories == null)  songsRepositories = new SongsRepositories(db);  return songsRepositories;  }  }  public UserListMusicRepositories UserListMusicRepositories  {  get  {  if (userListMusicRepositories == null)  userListMusicRepositories = new UserListMusicRepositories(db);  return userListMusicRepositories;  }  }  public UserListRepositories UserListRepositories  {  get  {  if (userListRepositories == null)  userListRepositories = new UserListRepositories(db);  return userListRepositories;  }  }  public UsersRepositories UsersRepositories  {  get  {  if (usersRepositories == null)  usersRepositories = new UsersRepositories(db);  return usersRepositories;  }  }  public void Save()  {  db.SaveChanges();  }  public void Dispose()  {  db.Dispose();  }  } |

Листинг 4.6 – класс UnitOfWork

4.2 Добавление песни в избранные

Добавление музыки в избранное происходит следующим образом: авторизованный пользователь выбирает понравившуюся ему песню и наживает на иконку «Сердечко», после этого песня добавляется в список избранных.

На листинге 4.7 находится код, реализующий добавление в избранные.

|  |
| --- |
| private DelegateCommand addMusicInUserList;  public ICommand AddMusicInUserList  { get  {  if (addMusicInUserList == null)  {  addMusicInUserList = new DelegateCommand(AddSongToUserList);  }  return addMusicInUserList;  }  }  private void AddSongToUserList()  {  if (MusicManager.Instance.unitOfWork.UserListRepositories.AddSongs(SelectedSongs, CurrentUser))  {  MessageBox.Show(Application.Current.FindResource("gut").ToString());  MusicManager.Instance.unitOfWork.Save();  songsList = MusicManager.Instance.unitOfWork.AdminListRepositories.GetAllSongs().ToList();  OnPropertyChanged(nameof(ListMusicShow));  }} |

Листинг 4.7 – Реализующая добавление в избранные

4.3 Реализация библиотеки CustomControls

Для реализации функции, когда пароль заменяется на звездочки при вводе в окно, был разработан Custom Controls «BindablePasswordBox», он расположен в отдельной библиотеке, которую после добавил в зависимости моего проета. Это даёт следующие преимущества:

* Переиспользование кода: Вынос CustomControls в библиотеку позволяет легко повторно использовать этот код в разных проектах без необходимости копирования и вставки в каждый проект.
* Разделение ответственности: Путем выноса CustomControls в отдельную библиотеку явно определяются границы ответственности между различными компонентами проекта. Библиотека CustomControls может быть отделена от остальной логики проекта, что облегчает понимание кода и его сопровождение.
* Вынос CustomControls в отдельную библиотеку способствует созданию модульной архитектуры вашего проекта. Это позволяет легко добавлять и изменять функциональность CustomControls без необходимости вносить изменения в другие части проекта.

Листинг «BindablePasswordBox» 4.8

|  |
| --- |
| <ResourceDictionary  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:local="clr-namespace:CustomControls.Controls">  <Style TargetType="{x:Type local:BindablePasswordBox}">  <Setter Property="Template">  <Setter.Value>  <ControlTemplate TargetType="{x:Type local:BindablePasswordBox}">  <Border Background="{TemplateBinding Background}"  BorderBrush="{TemplateBinding BorderBrush}"  BorderThickness="{TemplateBinding BorderThickness}">  <StackPanel>  <TextBlock Name="textBlock"  Foreground="LightGray"/>  <PasswordBox Name="passwordBox">  <PasswordBox.Style>  <Style TargetType="{x:Type PasswordBox}">  <Setter Property="Background" Value="Transparent"/>  <Setter Property="BorderThickness" Value="0 0 0 2"/>  <Setter Property="BorderBrush" Value="LightGray"/>  <Setter Property="Foreground" Value="LightGray"/>  <Setter Property="FontSize" Value="16"/>  <Setter Property="CaretBrush" Value="LightGray"/>  <Setter Property="Padding" Value="5"/>  </Style>  </PasswordBox.Style>  </PasswordBox>  <TextBox Name="errorTextBox"  IsEnabled="False"  FontWeight="Bold"  Foreground="Red"  Visibility="Collapsed"  Background="Transparent"  BorderThickness="0"  Text="{Binding MessageError, RelativeSource={RelativeSource AncestorType={x:Type local:BindablePasswordBox}}}" />  </StackPanel>  </Border>  </ControlTemplate>  </Setter.Value>  </Setter>  </Style> </ResourceDictionary> |

Листинг 4.8 – XAML BindablePasswordBox

# 5 Тестирование, проверка работоспособности и анализ полученных результатов

В курсовом проекте задействуется обработка ошибок, таким образом, что пользователь будет уведомлен о неудачном выполнении запроса к базе данных, или недоступности данных в формах.

При запуске приложения открывается представление с формой для авторизации, куда следует ввести свой логин и пароль. Если логин или пароль введены неверно, то в TextBox будет выведена информация о произошедшей ошибке (рис. 5.1).

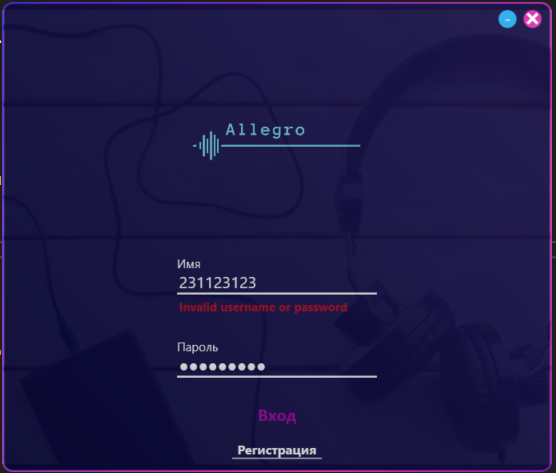


Рисунок 5.1 – При авторизации введены ошибочные данные

Далее рассмотрим представление для регистрации. Оно имеет 6 полей для ввода: Имя, оно же логин, пароль и email. В приложении предусмотрена обработка следующих ошибок:

Если пользователь оставил поле незаполненным, он увидит сообщение с рисунка 5.2.

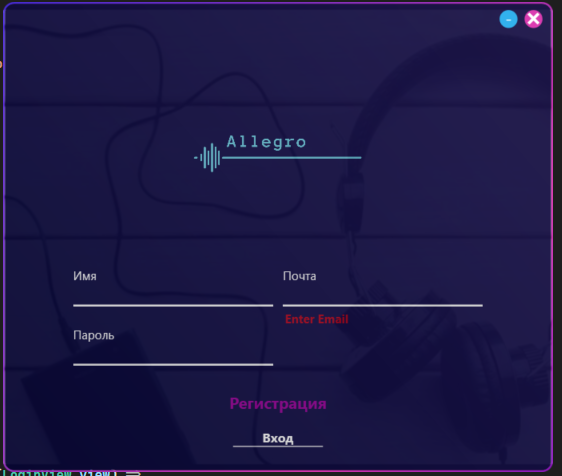


Рисунок 5.2 – При регистрации есть пустые поля

Если пользователь попробует занять логин, который уже существует в базе данных, он увидит сообщение с рисунка 5.3.

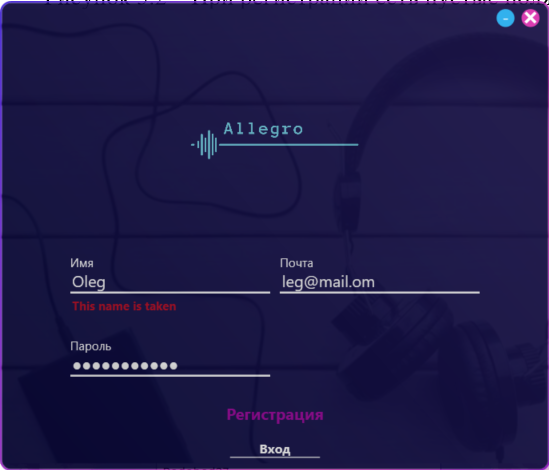


Рисунок 5.3 – При регистрации выбран уже занятый логин

Если пользователь при регистрации введет пароли, которые является слабым, приложение ему подскажет сообщением с рисунка 5.4

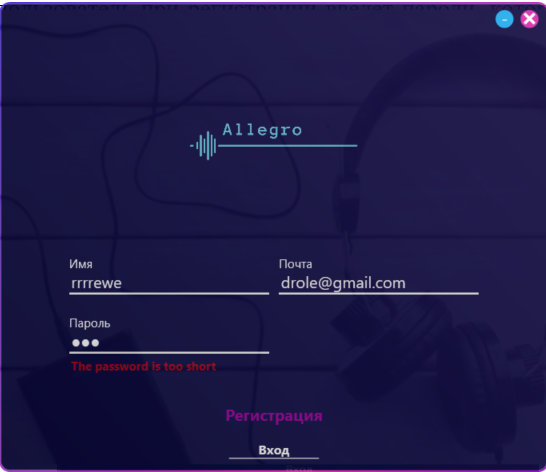


Рисунок 5.4 – При регистрации пользователь ввел слабый пароль

Если пользователь при регистрации введет некорректную почту, приложение ему подскажет сообщением, продемонстрировано это на рисунка 5.5.

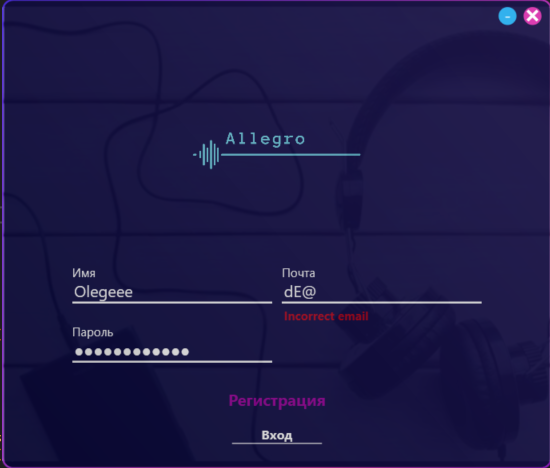


Рисунок 5.5 – При регистрации пользователь ввел некорректную почту

# 6 Методика использования программного средства

6.1 Регистрация и авторизация

При запуске приложения запускается окно авторизации. Если у пользователя еще нет аккаунта, ему следует нажать на кнопку «Регистрация», которая его перенаправит на представление с формой для регистрации. Представление авторизации представлено на рисунке 6.1.

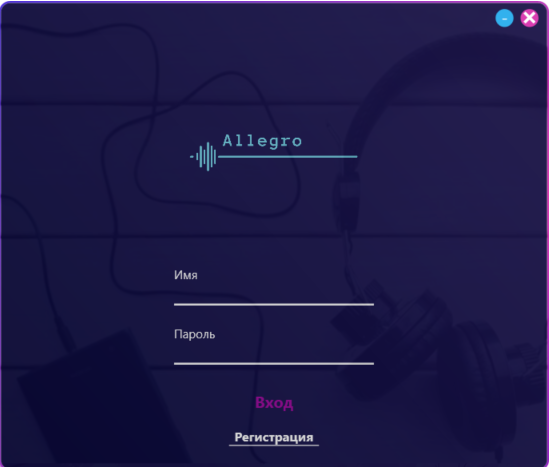


Рисунок 6.1 – Представление авторизации

Если была нажата кнопка «Регистрация», в открывшемся представлении следует ввести все данные о регистрируемом пользователе. В случае, если кнопка была нажата ошибочно, можно нажать на кнопку «Вход» и вернуться к представлению авторизации. Представление регистрации изображено на рисунке 6.2.

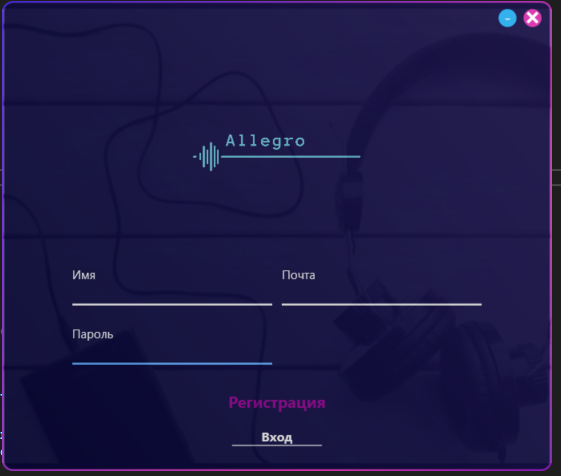


Рисунок 6.2 ‒ Представление регистрации

## **6.2** Домашнее представление

При успешной авторизации пользователя приложение переключит представление с авторизации на домашнее. На рисунке 6.3. показано то, что новый пользователь увидит после своей авторизации.

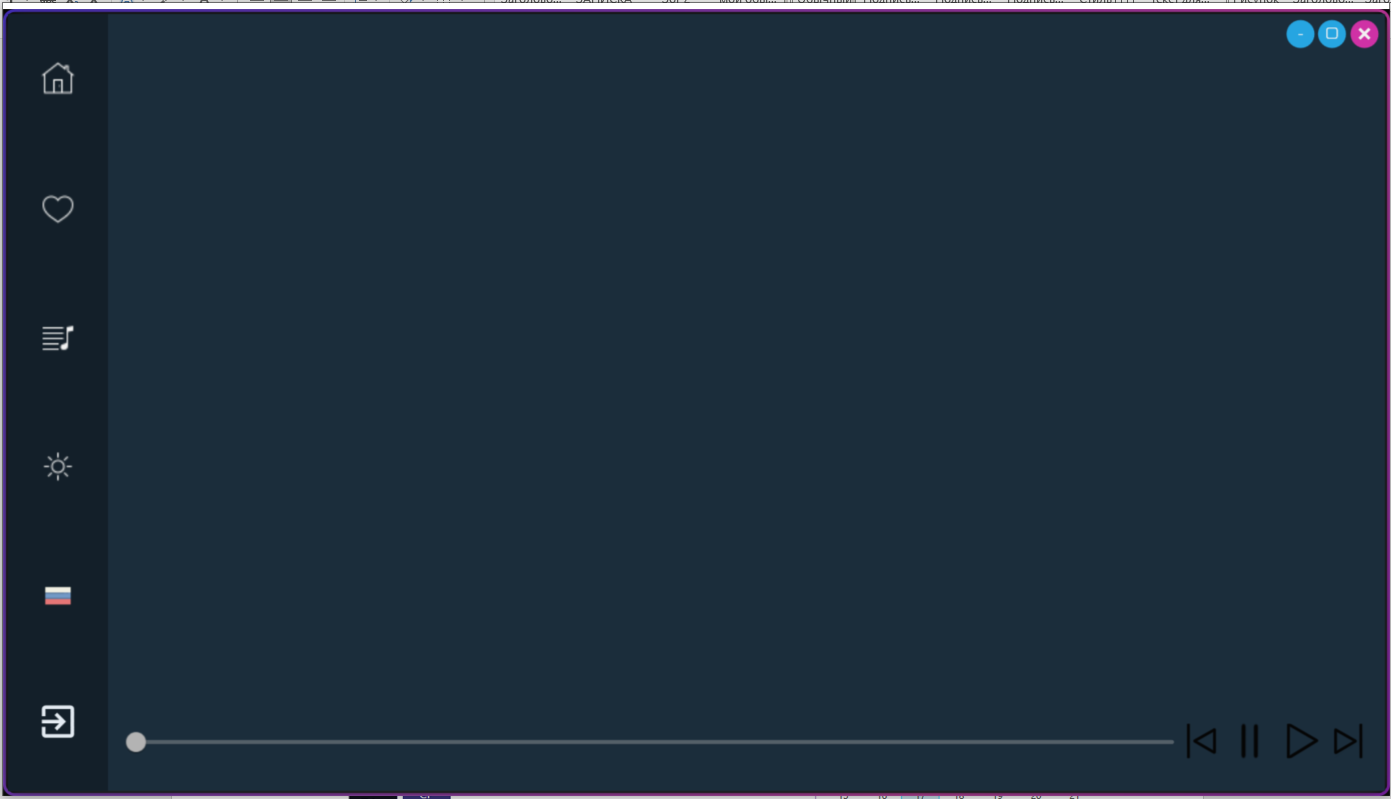


Рисунок 6.3 ‒ Домашнее представление после авторизации

Если пользователь желает посмотреть список всех песен, то он должен нажать на первую кнопку, то, что увидит пользователь, продемонстрировано на рисунке 6.4.

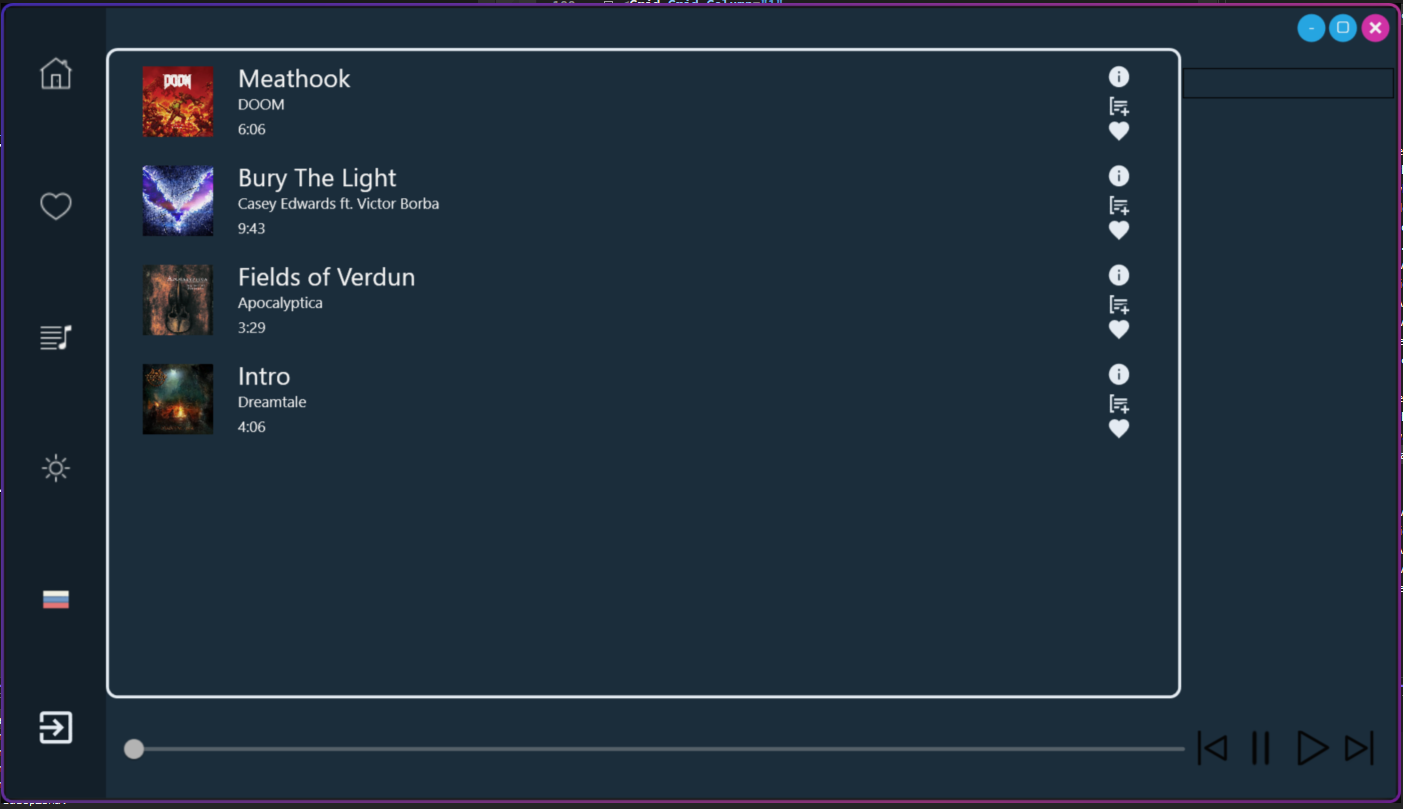


Рисунок 6.4 ‒ Главный список

Для того, чтобы добавить песню в избранные, её нужно выбрать, дальше, можно нажать на кнопку в виде сердца, эту кнопку можно нажать у любых элементов, но добавится только выбранный.

Чтобы увидеть её в списке избранных, нужно в меню выбрать вторую кнопку, нажав на данную кнопку, у нас откроется список избранной музыки. Это можно увидеть на рисунке 6.5.

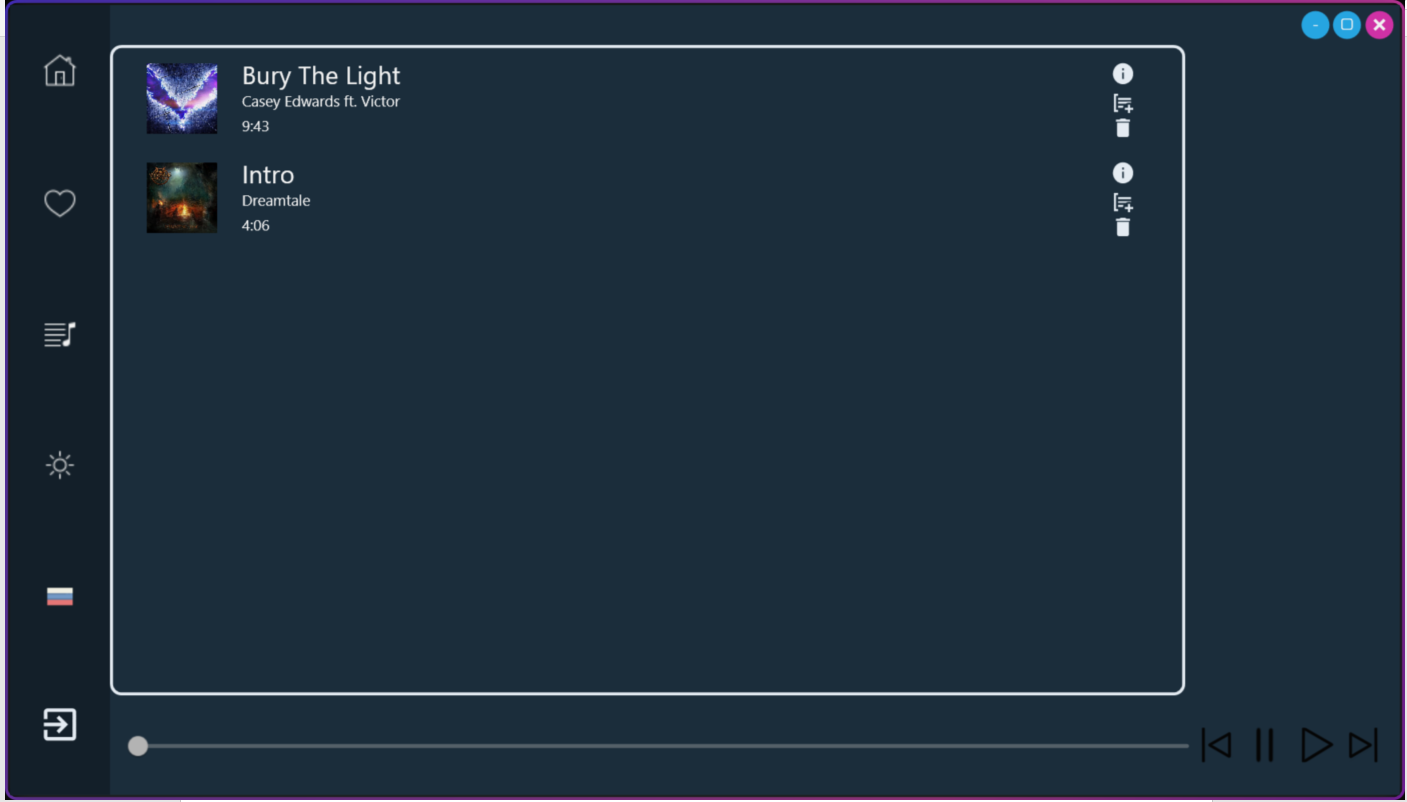


Рисунок 6.5 ‒ Избранные композиции

Чтобы увидеть свои плейлисты нужно нажать на третью кнопку. На данной странице мы можем взаимодействовать с плейлистами, а также можем создавать и удалять их.

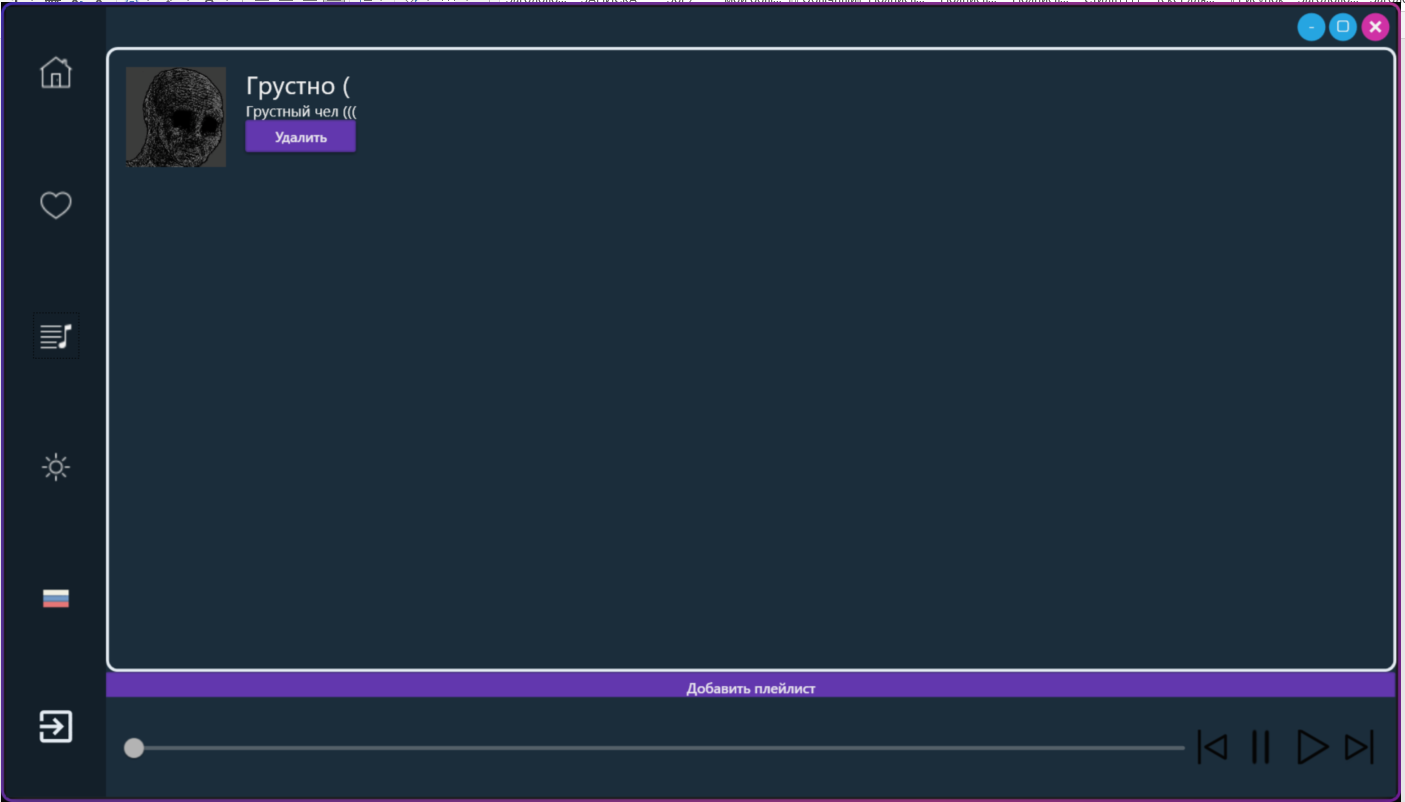


Рисунок 6.6 ‒ Мои плейлисты

Чтобы посмотреть содержимое плейлиста нужно два раза нажать на него левой кнопкой мыши, после этого мы увидим содержимое плейлиста. Это изображено на рисунке 6.7.

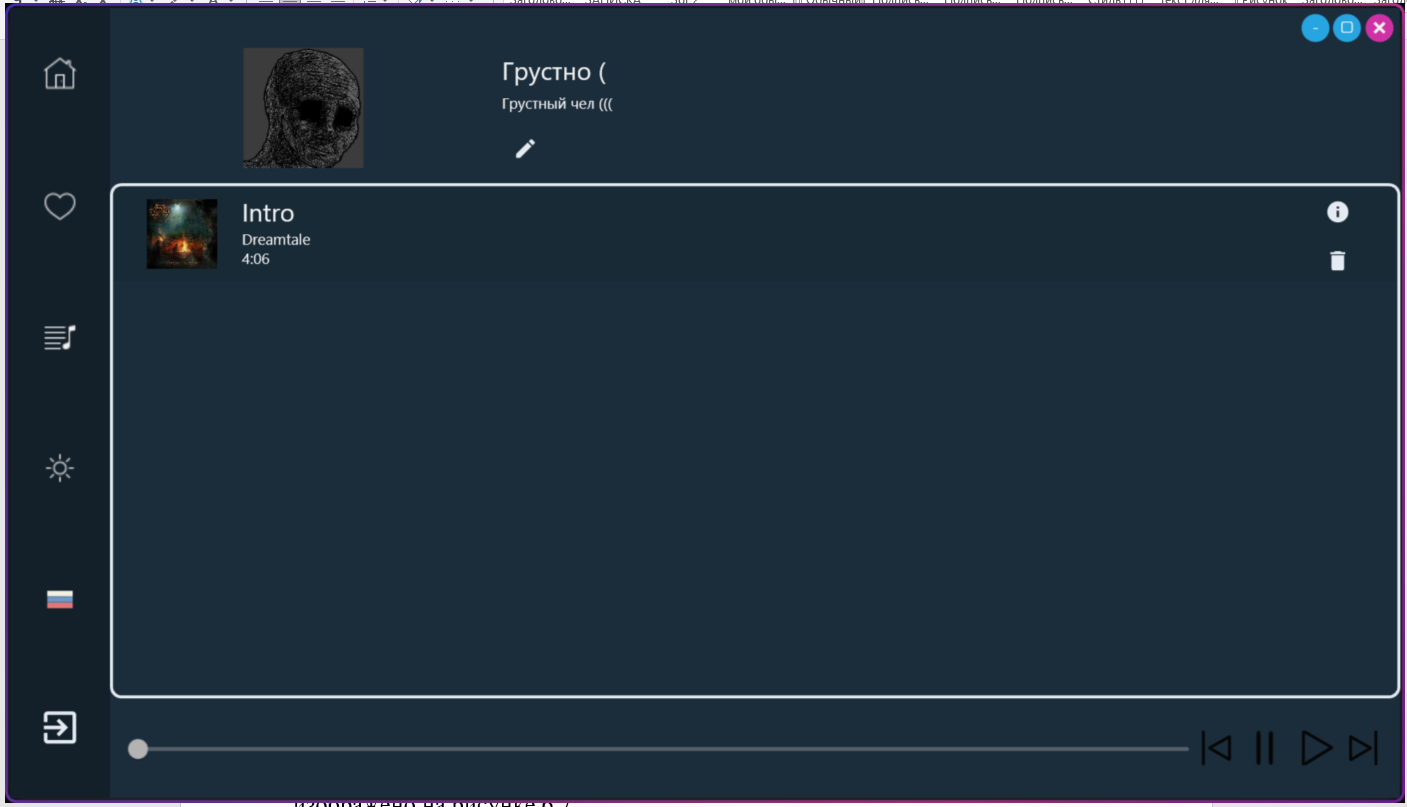


Рисунок 6.7 – Плейлист

## **6.3 Администрирование**

При успешной авторизации администратора приложение переключит представление с авторизации на домашнее. В домашнем представлении для администратора доступны такие кнопки как «Главный список», «Добавить песню», а также у админа появляется возможность редактировать песни.

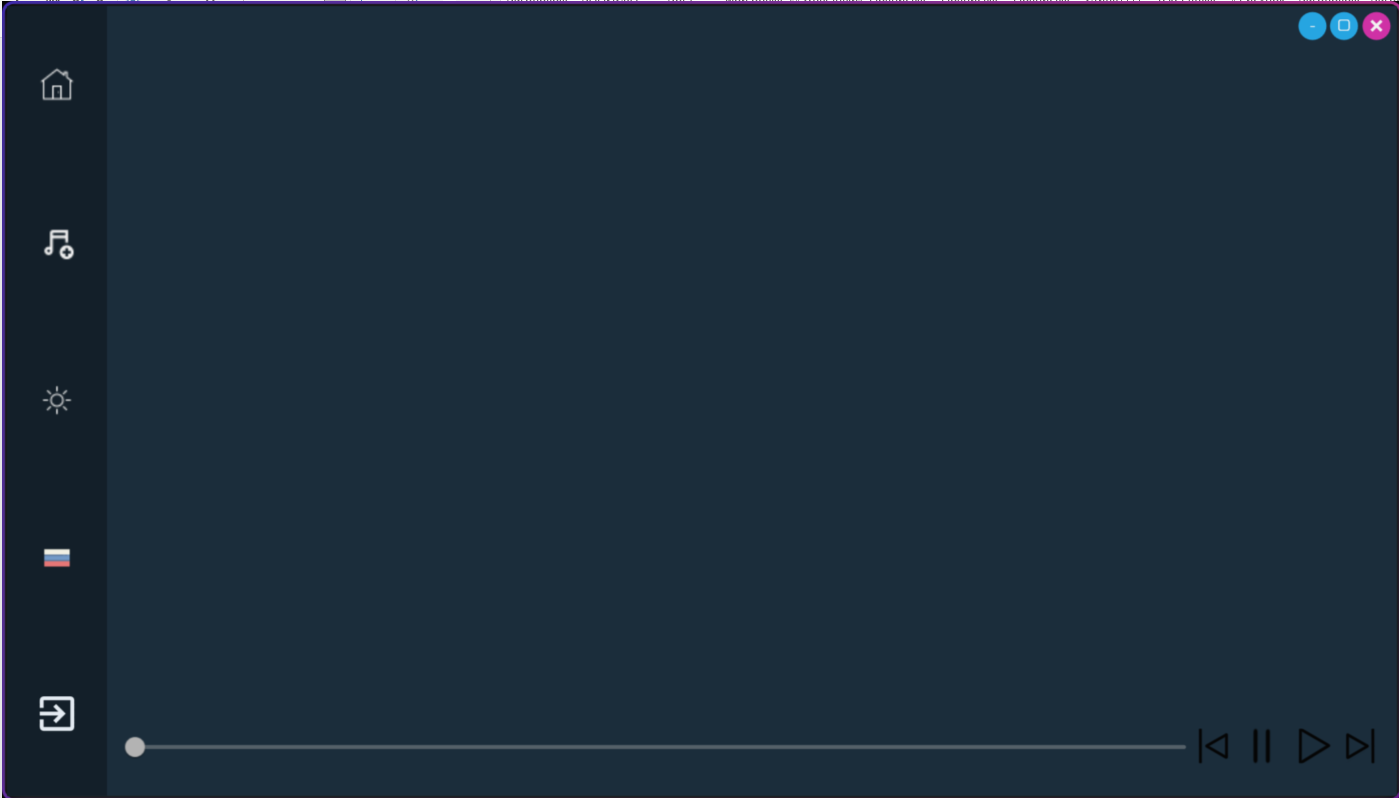


Рисунок 6.8 ‒ домашнее представление

При нажатии на кнопку «Главный список» откроется представление, в котором можно увидеть все композиции, также их можно редактировать и удалять. Это изображено на рисунке 6.9.

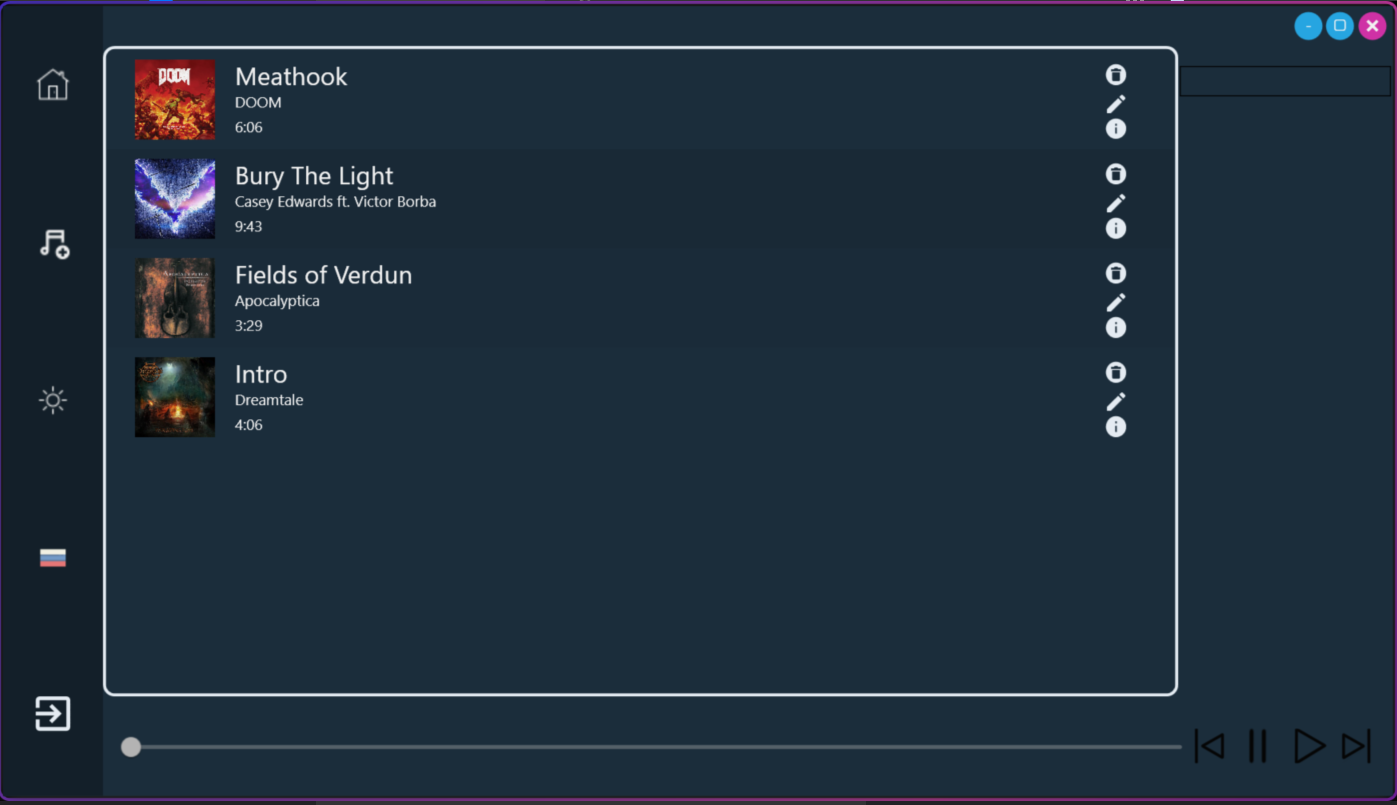


Рисунок 6.9 – «Главный список»

При нажатии на кнопку «Добавить песню» откроется представление, в котором есть поля для заполнения данных о песне, а также кнопки, для выбора картинки и самой песни. Это изображено на рисунке 6.10.

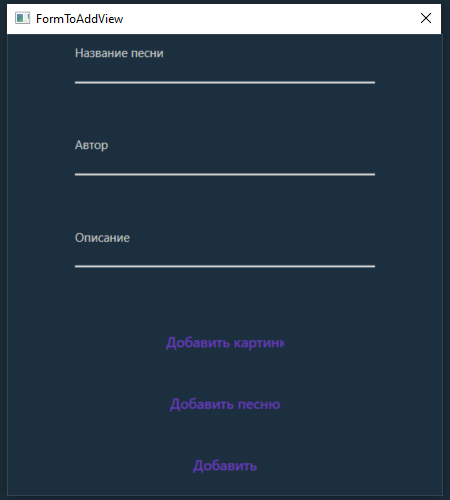


Рисунок 6.10 – Главный список

При нажатии на кнопку «Редактировать песню» откроется окно, в котором есть поля для заполнения новых данных о песне, а также кнопки, для выбора картинки и отмены изменений. Это изображено на рисунке 6.11.

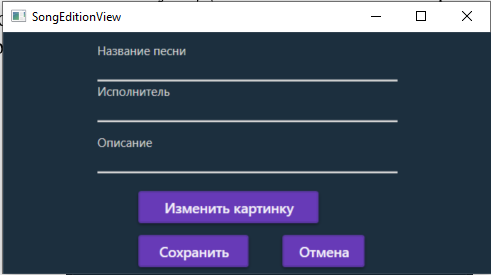


Рисунок 6.11 – Редактирование песни

## **6.4 Смена пользователя**

При желании сменить пользователя или выйти из аккаунта, пользователю стоит нажать на кнопку «Выход» в меню.

# Заключение

При выполнении данного курсового проекта было разработано программное средство для цифрового музыкального плеер, которое позволяет пользователям прослушивать, формировать свои плейлисты, а также добавлять музыку в избранные. Это позволяет пользователю всегда иметь доступ к нужной ему музыке, а также распределять и объединять песни в плейлисты. Для разработки приложения, которое полностью бы удовлетворяло запросы пользователя, был проведен анализ аналогов для выявления сильных и слабых сторон других программных средств и разработки функциональных требований, проектирование и разработка приложения.

Исходя из анализа аналогов, были выделены требования к программному обеспечению:

– удобный и понятный интерфейс для пользователей с любой компьютерной грамотностью;

– разделение бизнес-логики приложения на клиентскую часть и часть администратора.

Приложение было протестировано, а также проверено на полноту выполняемых операций.

Разработанное приложение удовлетворяет всем требованиям, предъявленным в задании.

Также в процессе выполнения данного курсового проекта были закреплены навыки в программировании на объектно-ориентированном языке C#, усовершенствованы навыки в работе с платформой для кроссплатформенной разработки с открытым исходным кодом .NET Core, ознакомление с созданием десктопных приложений на расширяемом языке разметки WPF, использовании фреймворка для работы с базой данных Entity Framework Core 6.1.0, работа с современными паттернами MVVM, Command, Репозиторий и Unit of Work, а также Одиночка, проектирование базы данных и реализация их в системе управления базами данных Microsoft SQL Server 2019.

# Список литературы

* docs.microsoft.com Сайт о программировании [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com>.
* Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
* METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com>.
* Руководство по WPF Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com/sharp/wpf/#google_vignette>.
* Code First Сайт о программировании [Электронный ресурс] / Режим доступа:<https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first>

# Приложение А

