Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ярославский государственный технический университет»

Кафедра «Информационные системы и технологии»

Работа защищена

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель

Прозоров А.Н.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2025

**Разработка приложения «медиапроигрыватель» На платформе WPF**

Отчет о лабораторной работе по курсу

«Технология программирования»

ЯГТУ 09.03.02 - 004 ЛР

|  |  |
| --- | --- |
|  | Отчет выполнил  студент группы ЗЦИС-26  А.А. Чернов  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2025 |

2025

**Цель работы**

Доработать приложение созданное в 1 лабораторной работе, добавив сопряжение с базой данных.

**Задание**

Разработать приложение, выполняющее следующие требования:

* Открытие файла
* Остановка\воспроизведение\пауза
* Управление громкостью (выкл звук, задать громкость)
* Список треков\повтор текущей композиции

Сформировать и внедрить БД SQLite

* Обеспечить возможность создания и выбора плейлиста и категории плейлистов.

**Ход выполнения задания**

При запуске приложения открывается рабочее окно.

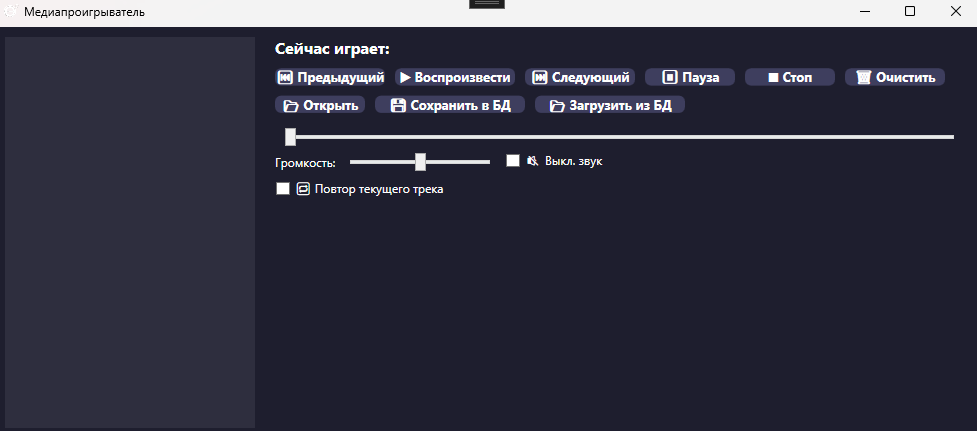


Рисунок 1 – Запуск приложения

Добавляем медиафайлы.

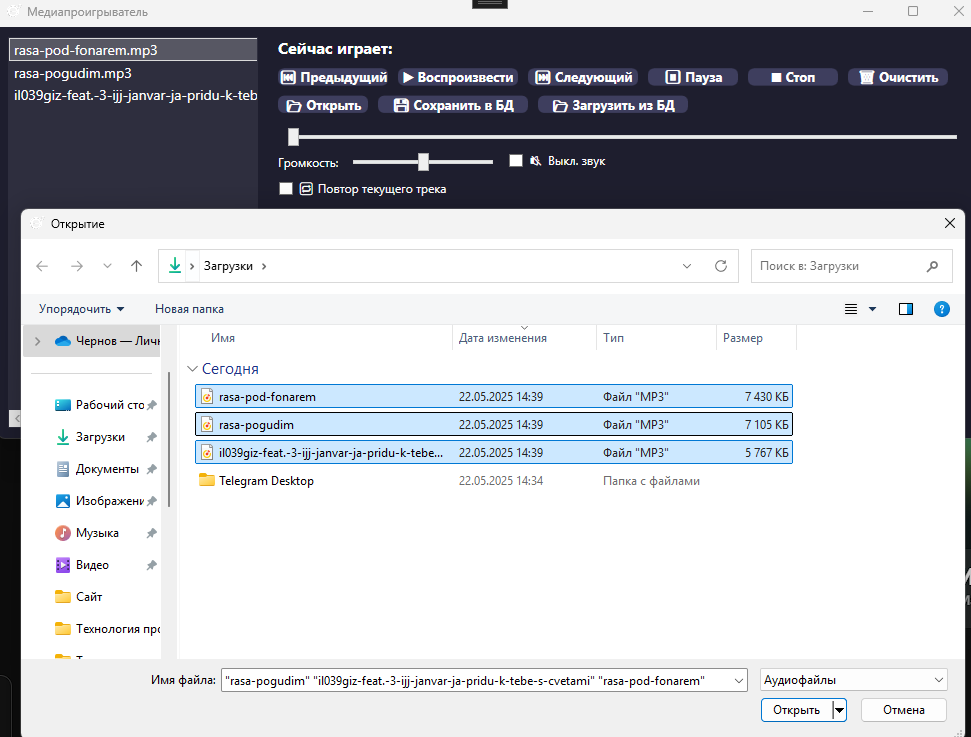


Рисунок 2 – Добавление медиафайлов

При сохранении плейлиста, дает выбор категории плейлистов и возможность создания новых.

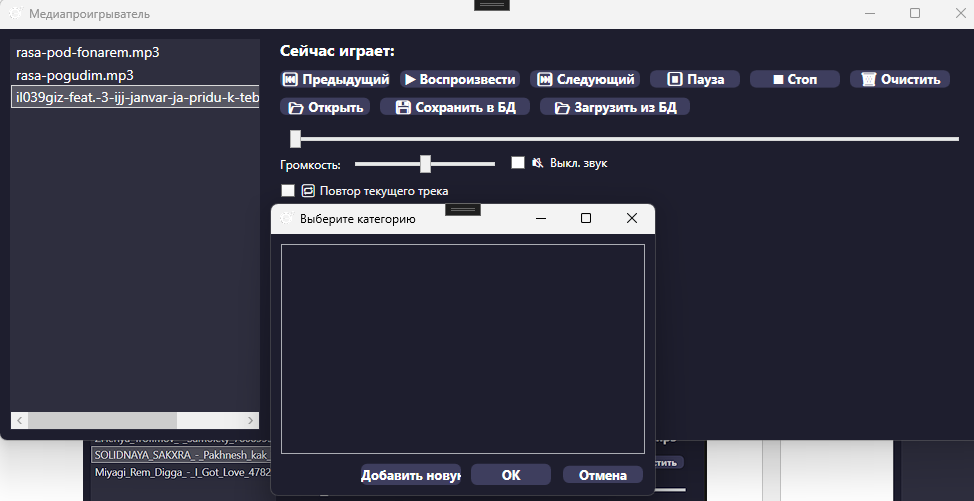


Рисунок 3 – Выбор категории

Так же в функционале присутствует возможность создания плейлиста в категории, появляется автоматически при создании категории.

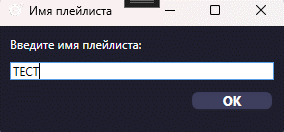


Рисунок 4 – Создание плейлиста

Пример отображения категорий плейлистов.

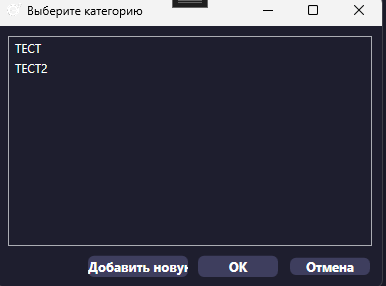


Рисунок 5- Список категорий

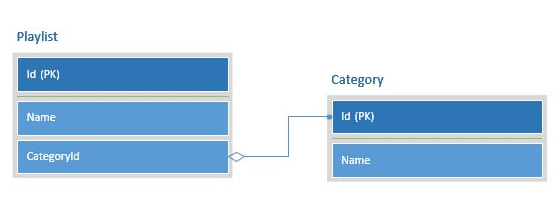


Рисунок 6 – Схема БД

**Вывод**

Было разработано приложение «медиаплеер» на языке программирования C# со следующим функционалом:

* Открытие файла
* Остановка\воспроизведение\пауза
* Управление громкостью (выкл звук, задать громкость)
* Список треков\повтор текущей композиции.
* Создание и выбор категорий плейлистов
* Создание и выбор плейлистов

**Приложение А**

**Листинг приложения**

using Microsoft.Win32;

using NAudio.Wave;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Threading;

namespace PlayerApp

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

// Плейлист со списком путей к трекам

private List<string> playlist = new List<string>();

// Поток для чтения аудиофайла

private AudioFileReader audioFileReader;

// Устройство вывода звука

private WaveOutEvent outputDevice;

// Индекс текущего воспроизводимого трека

private int currentTrackIndex = -1;

// Флаг состояния "без звука"

private bool isMuted = false;

// Последнее значение громкости (до отключения звука)

private float lastVolume = 0.5f;

// Флаг для отслеживания, перетаскивает ли пользователь ползунок прогресса

private bool isDraggingSlider = false;

// Таймер обновления положения ползунка прогресса

private DispatcherTimer progressTimer = new DispatcherTimer();

private bool isManualTrackChange = false;

private bool isStopRequested = false;

public MainWindow()

{

InitializeComponent();

DatabaseHelper.InitializeDatabase();

// Устанавливаем громкость из настроек

sliderVolume.Value = Properties.Settings.Default.Volume \* 100;

lastVolume = Properties.Settings.Default.Volume;

// Настраиваем таймер

progressTimer.Interval = TimeSpan.FromMilliseconds(500);

progressTimer.Tick += ProgressTimer\_Tick;

// Загружаем треки из предыдущей сессии

LoadPlaylistFromLastSession();

}

// Открытие и загрузка файлов

private void btnOpenFile\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog ofd = new OpenFileDialog

{

Multiselect = true,

Filter = "Аудиофайлы|\*.mp3;\*.wav;\*.aac;\*.wma"

};

if (ofd.ShowDialog() == true)

{

playlist = ofd.FileNames.ToList();

listBoxTracks.Items.Clear();

foreach (var file in playlist)

listBoxTracks.Items.Add(System.IO.Path.GetFileName(file));

listBoxTracks.SelectedIndex = 0;

currentTrackIndex = 0;

SavePlaylistToSession();

}

}

// Кнопка воспроизведения

private void btnPlay\_Click(object sender, RoutedEventArgs e)

{

if (outputDevice != null)

{

if (outputDevice.PlaybackState == PlaybackState.Playing)

{

// Защита от повторного воспроизведения

return;

}

else if (outputDevice.PlaybackState == PlaybackState.Paused)

{

// Возобновляем с паузы

outputDevice.Play();

progressTimer.Start();

return;

}

}

if (currentTrackIndex == -1 || playlist.Count == 0)

{

MessageBox.Show("Выберите трек для воспроизведения.");

return;

}

try

{

StopPlayback();

audioFileReader = new AudioFileReader(playlist[currentTrackIndex]);

outputDevice = new WaveOutEvent();

outputDevice.Init(audioFileReader);

outputDevice.PlaybackStopped += OnPlaybackStopped;

SetVolume();

outputDevice.Play();

progressTimer.Start();

lblNowPlaying.Text = "Сейчас играет: " + System.IO.Path.GetFileName(playlist[currentTrackIndex]);

SavePlaybackState();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка воспроизведения: " + ex.Message);

}

}

// Пауза

private void btnPause\_Click(object sender, RoutedEventArgs e)

{

if (outputDevice != null && outputDevice.PlaybackState == PlaybackState.Playing)

{

outputDevice.Pause();

progressTimer.Stop();

}

}

// Остановка

private void btnStop\_Click(object sender, RoutedEventArgs e)

{

if (outputDevice != null && audioFileReader != null)

{

isStopRequested = true;

outputDevice.Stop();

audioFileReader.CurrentTime = TimeSpan.Zero;

sliderProgress.Value = 0;

progressTimer.Stop();

lblNowPlaying.Text = "Сейчас играет: -";

}

}

// Очистить плейлист

private void btnClearPlaylist\_Click(object sender, RoutedEventArgs e)

{

StopPlayback();

playlist.Clear();

listBoxTracks.Items.Clear();

currentTrackIndex = -1;

lblNowPlaying.Text = "Сейчас играет: -";

// Удаляем сохранённый плейлист и последний трек

if (File.Exists("lastPlaylist.txt"))

File.Delete("lastPlaylist.txt");

Properties.Settings.Default.LastTrack = "";

Properties.Settings.Default.Save();

}

// Изменение громкости

private void sliderVolume\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

lastVolume = (float)(sliderVolume.Value / 100.0);

Properties.Settings.Default.Volume = lastVolume;

Properties.Settings.Default.Save();

if (!isMuted)

SetVolume();

}

// Включение/отключение звука

private void chkMute\_Checked(object sender, RoutedEventArgs e)

{

isMuted = chkMute.IsChecked == true;

SetVolume();

}

// Применение текущей громкости к потоку

private void SetVolume()

{

if (audioFileReader != null)

audioFileReader.Volume = isMuted ? 0.0f : lastVolume;

}

// Смена выбранного трека

private void listBoxTracks\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (listBoxTracks.SelectedIndex >= 0)

currentTrackIndex = listBoxTracks.SelectedIndex;

}

// Остановка текущего воспроизведения и освобождение ресурсов

private void StopPlayback()

{

// Отписываемся от события, чтобы избежать рекурсии

if (outputDevice != null)

{

outputDevice.PlaybackStopped -= OnPlaybackStopped;

}

outputDevice?.Stop();

outputDevice?.Dispose();

outputDevice = null;

audioFileReader?.Dispose();

audioFileReader = null;

progressTimer.Stop();

sliderProgress.Value = 0;

// Снова подписываемся на событие

if (outputDevice != null)

{

outputDevice.PlaybackStopped += OnPlaybackStopped;

}

}

// Обработка окончания воспроизведения

private void OnPlaybackStopped(object sender, StoppedEventArgs e)

{

Dispatcher.Invoke(() =>

{

// Если остановка была вызвана вручную (кнопкой Стоп или переключением трека) - ничего не делаем

if (isStopRequested || isManualTrackChange)

{

isStopRequested = false;

return;

}

// Автоматический переход

if (chkRepeat.IsChecked == true)

{

RestartCurrentTrack();

}

else

{

btnNext\_Click(null, null);

}

});

}

// Обновление ползунка прогресса

private void ProgressTimer\_Tick(object sender, EventArgs e)

{

if (audioFileReader != null && !isDraggingSlider)

{

sliderProgress.Maximum = audioFileReader.TotalTime.TotalSeconds;

sliderProgress.Value = audioFileReader.CurrentTime.TotalSeconds;

}

}

// Изменение прогресса пользователем

private void sliderProgress\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

if (isDraggingSlider && audioFileReader != null)

{

audioFileReader.CurrentTime = TimeSpan.FromSeconds(sliderProgress.Value);

}

}

// Начало перетаскивания ползунка

private void sliderProgress\_PreviewMouseDown(object sender, MouseButtonEventArgs e)

{

isDraggingSlider = true;

}

private void sliderProgress\_PreviewMouseUp(object sender, MouseButtonEventArgs e)

{

if (audioFileReader != null)

{

var slider = sender as Slider;

audioFileReader.CurrentTime = TimeSpan.FromSeconds(slider.Value);

}

isDraggingSlider = false;

}

private void btnPrevious\_Click(object sender, RoutedEventArgs e)

{

if (playlist.Count == 0) return;

// Получаем индекс предыдущего трека

int newIndex = currentTrackIndex - 1;

if (newIndex < 0) newIndex = playlist.Count - 1;

// Переключаемся только если это другой трек

if (newIndex != currentTrackIndex)

{

ChangeTrack(newIndex);

}

else

{

// Если это тот же трек (в списке 1 трек), просто перезапускаем

RestartCurrentTrack();

}

}

// Воспроизведение следующего трека

private void btnNext\_Click(object sender, RoutedEventArgs e)

{

if (playlist.Count == 0) return;

// Получаем индекс следующего трека

int newIndex = currentTrackIndex + 1;

if (newIndex >= playlist.Count) newIndex = 0;

ChangeTrack(newIndex);

}

private void ChangeTrack(int newIndex)

{

// Сохраняем состояние воспроизведения

bool wasPlaying = (outputDevice != null && outputDevice.PlaybackState == PlaybackState.Playing);

StopPlayback();

currentTrackIndex = newIndex;

listBoxTracks.SelectedIndex = newIndex;

if (wasPlaying)

{

btnPlay\_Click(null, null);

}

}

private void RestartCurrentTrack()

{

if (outputDevice != null && outputDevice.PlaybackState == PlaybackState.Playing)

{

// Просто перематываем в начало и продолжаем воспроизведение

audioFileReader.CurrentTime = TimeSpan.Zero;

}

else

{

// Запускаем трек заново

StopPlayback();

btnPlay\_Click(null, null);

}

}

// Сохранение текущего трека в настройки

private void SavePlaybackState()

{

if (currentTrackIndex >= 0 && currentTrackIndex < playlist.Count)

{

Properties.Settings.Default.LastTrack = playlist[currentTrackIndex];

Properties.Settings.Default.Save();

}

}

// Сохранение плейлиста между сессиями в файл

private void SavePlaylistToSession()

{

File.WriteAllLines("lastPlaylist.txt", playlist);

}

// Загрузка плейлиста и восстановление последнего трека при запуске

private void LoadPlaylistFromLastSession()

{

try

{

if (File.Exists("lastPlaylist.txt"))

{

playlist = File.ReadAllLines("lastPlaylist.txt").ToList();

listBoxTracks.Items.Clear();

foreach (var file in playlist)

listBoxTracks.Items.Add(System.IO.Path.GetFileName(file));

string lastTrack = Properties.Settings.Default.LastTrack;

int index = playlist.IndexOf(lastTrack);

if (index >= 0)

{

currentTrackIndex = index;

listBoxTracks.SelectedIndex = index;

}

}

}

catch

{

// Защита от ошибок загрузки

}

}

// Сохранение в БД

private void btnSaveToDb\_Click(object sender, RoutedEventArgs e)

{

if (playlist.Count == 0)

{

MessageBox.Show("Плейлист пуст. Нечего сохранять.");

return;

}

// Сначала выбираем категорию

var categories = DatabaseHelper.GetAllCategories();

var categoryDialog = new CategorySelectorDialog(categories);

if (categoryDialog.ShowDialog() == true)

{

int categoryId;

if (categoryDialog.AddNewCategorySelected)

{

// Диалог для ввода имени новой категории

var inputDialog = new InputDialog("Введите название новой категории:");

if (inputDialog.ShowDialog() == true && !string.IsNullOrWhiteSpace(inputDialog.InputText))

{

categoryId = DatabaseHelper.AddCategory(inputDialog.InputText);

}

else

{

return; // Пользователь отменил ввод

}

}

else

{

categoryId = categoryDialog.SelectedCategory?.Id ?? 0;

if (categoryId == 0) return;

}

// Затем вводим имя плейлиста

var nameDialog = new PlaylistNameDialog();

if (nameDialog.ShowDialog() == true && !string.IsNullOrWhiteSpace(nameDialog.PlaylistName))

{

try

{

DatabaseHelper.SavePlaylist(nameDialog.PlaylistName, categoryId, playlist);

MessageBox.Show("Плейлист успешно сохранён!");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении: {ex.Message}");

}

}

}

}

// Загрузка из БД

private void btnLoadFromDb\_Click(object sender, RoutedEventArgs e)

{

try

{

// Сначала выбираем категорию

var categories = DatabaseHelper.GetAllCategories();

if (categories.Count == 0)

{

MessageBox.Show("Нет доступных категорий.");

return;

}

var categoryDialog = new CategorySelectorDialog(categories);

if (categoryDialog.ShowDialog() == true && categoryDialog.SelectedCategory != null)

{

// Затем выбираем плейлист из выбранной категории

var playlists = DatabaseHelper.GetPlaylistsByCategory(categoryDialog.SelectedCategory.Id);

if (playlists.Count == 0)

{

MessageBox.Show("В выбранной категории нет плейлистов.");

return;

}

var playlistDialog = new PlaylistSelectorDialog(playlists);

if (playlistDialog.ShowDialog() == true && playlistDialog.SelectedPlaylist != null)

{

// Загружаем выбранный плейлист

playlist = playlistDialog.SelectedPlaylist.Tracks;

listBoxTracks.Items.Clear();

foreach (var file in playlist)

listBoxTracks.Items.Add(Path.GetFileName(file));

if (playlist.Count > 0)

{

listBoxTracks.SelectedIndex = 0;

currentTrackIndex = 0;

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при загрузке: {ex.Message}");

}

}

}

}

**Приложение Б**

**Листинг Playlist.cs**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PlayerApp

{

public class Playlist

{

public int Id { get; set; }

public string Name { get; set; }

public int CategoryId { get; set; } // Внешний ключ

public List<string> Tracks { get; set; } = new List<string>();

[JsonIgnore] // Игнорируем при сериализации в JSON

public Category Category { get; set; }

}

}

**Приложение В**

**Листинг Category.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PlayerApp

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**Приложение Г**

**Листинг CategorySelectorDialog.xaml.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace PlayerApp

{

/// <summary>

/// Логика взаимодействия для CategorySelectorDialog.xaml

/// </summary>

public partial class CategorySelectorDialog : Window

{

public Category SelectedCategory { get; private set; }

public bool AddNewCategorySelected { get; private set; }

public CategorySelectorDialog(List<Category> categories)

{

InitializeComponent();

listBoxCategories.ItemsSource = categories;

}

private void OkButton\_Click(object sender, RoutedEventArgs e)

{

SelectedCategory = listBoxCategories.SelectedItem as Category;

DialogResult = true;

}

private void AddNewButton\_Click(object sender, RoutedEventArgs e)

{

AddNewCategorySelected = true;

DialogResult = true;

}

private void CancelButton\_Click(object sender, RoutedEventArgs e)

{

DialogResult = false;

Close();

}

}

}

**Приложение Д**

**Листинг DataBaseHelper.cs**

using Newtonsoft.Json;

using PlayerApp;

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.IO;

public static class DatabaseHelper

{

private static string databaseFile = "playlists.db";

private static string connectionString = $"Data Source={databaseFile};Version=3;";

public static void InitializeDatabase()

{

if (!File.Exists(databaseFile))

{

SQLiteConnection.CreateFile(databaseFile);

}

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

// Создаём таблицу Categories, если её нет

string createCategories = @"

CREATE TABLE IF NOT EXISTS Categories (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Name TEXT NOT NULL UNIQUE

)";

new SQLiteCommand(createCategories, connection).ExecuteNonQuery();

// Создаём таблицу Playlists с внешним ключом

string createPlaylists = @"

CREATE TABLE IF NOT EXISTS Playlists (

Id INTEGER PRIMARY KEY AUTOINCREMENT,

Name TEXT NOT NULL,

CategoryId INTEGER NOT NULL,

PlaylistData TEXT NOT NULL,

FOREIGN KEY(CategoryId) REFERENCES Categories(Id)

)";

new SQLiteCommand(createPlaylists, connection).ExecuteNonQuery();

}

}

public static void SavePlaylist(string name, int categoryId, List<string> tracks)

{

string json = JsonConvert.SerializeObject(tracks);

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

string sql = @"INSERT INTO Playlists (Name, CategoryId, PlaylistData)

VALUES (@name, @categoryId, @data)";

var command = new SQLiteCommand(sql, connection);

command.Parameters.AddWithValue("@name", name);

command.Parameters.AddWithValue("@categoryId", categoryId);

command.Parameters.AddWithValue("@data", json);

command.ExecuteNonQuery();

}

}

public static List<Playlist> GetPlaylistsByCategory(int categoryId)

{

var playlists = new List<Playlist>();

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

string sql = @"SELECT Id, Name, PlaylistData FROM Playlists

WHERE CategoryId = @categoryId ORDER BY Name";

var command = new SQLiteCommand(sql, connection);

command.Parameters.AddWithValue("@categoryId", categoryId);

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

playlists.Add(new Playlist

{

Id = reader.GetInt32(0),

Name = reader.GetString(1),

CategoryId = categoryId,

Tracks = JsonConvert.DeserializeObject<List<string>>(reader.GetString(2))

});

}

}

}

return playlists;

}

public static List<Category> GetAllCategories()

{

var categories = new List<Category>();

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

string sql = "SELECT Id, Name FROM Categories ORDER BY Name";

var command = new SQLiteCommand(sql, connection);

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

categories.Add(new Category

{

Id = reader.GetInt32(0),

Name = reader.GetString(1)

});

}

}

}

return categories;

}

public static int AddCategory(string name)

{

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

string sql = "INSERT INTO Categories (Name) VALUES (@name); SELECT last\_insert\_rowid();";

var command = new SQLiteCommand(sql, connection);

command.Parameters.AddWithValue("@name", name);

return Convert.ToInt32(command.ExecuteScalar());

}

}

public static void DeletePlaylist(int id)

{

using (var connection = new SQLiteConnection(connectionString))

{

connection.Open();

string sql = "DELETE FROM Playlists WHERE Id = @id";

var command = new SQLiteCommand(sql, connection);

command.Parameters.AddWithValue("@id", id);

command.ExecuteNonQuery();

}

}

}

**Приложение Е**

**Листинг InputDialog.xaml.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace PlayerApp

{

/// <summary>

/// Логика взаимодействия для InputDialog.xaml

/// </summary>

public partial class InputDialog : Window

{

public string InputText => txtInput.Text;

public InputDialog(string prompt)

{

InitializeComponent();

lblPrompt.Content = prompt;

}

private void OkButton\_Click(object sender, RoutedEventArgs e)

{

DialogResult = true;

}

}

}

**Приложение Ж**

**Листинг PlaylistNameDialog.xaml.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace PlayerApp

{

/// <summary>

/// Логика взаимодействия для PlaylistNameDialog.xaml

/// </summary>

// PlaylistNameDialog.xaml.cs

public partial class PlaylistNameDialog : Window

{

public string PlaylistName { get; private set; }

public PlaylistNameDialog()

{

InitializeComponent();

}

private void OkButton\_Click(object sender, RoutedEventArgs e)

{

PlaylistName = txtName.Text;

DialogResult = true;

}

}

}

**Приложение З**

**Листинг PlaylistSelectorDialog.xaml.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace PlayerApp

{

/// <summary>

/// Логика взаимодействия для PlaylistSelectorDialog.xaml

/// </summary>

public partial class PlaylistSelectorDialog : Window

{

public Playlist SelectedPlaylist { get; private set; }

public PlaylistSelectorDialog(List<Playlist> playlists)

{

InitializeComponent();

listBoxPlaylists.ItemsSource = playlists

.OrderBy(p => p.Category?.Name)

.ThenBy(p => p.Name);

listBoxPlaylists.DisplayMemberPath = "Name";

}

private void OkButton\_Click(object sender, RoutedEventArgs e)

{

SelectedPlaylist = listBoxPlaylists.SelectedItem as Playlist;

DialogResult = true;

}

private void DeleteButton\_Click(object sender, RoutedEventArgs e)

{

var playlist = listBoxPlaylists.SelectedItem as Playlist;

if (playlist != null)

{

if (MessageBox.Show($"Удалить плейлист '{playlist.Name}'?", "Подтверждение",

MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

DatabaseHelper.DeletePlaylist(playlist.Id);

((List<Playlist>)listBoxPlaylists.ItemsSource).Remove(playlist);

listBoxPlaylists.Items.Refresh();

}

}

}

private void CancelButton\_Click(object sender, RoutedEventArgs e)

{

DialogResult = false;

}

}

}