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  И.С. Рубцов  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2025 |

2025

**Цель работы**

Создать приложение «медиапроигрыватель» на языке программирования C# на платформе WPF.

**Задание**

Разработать приложение, выполняющее следующие требования:

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

**Ход выполнения задания**

При запуске приложения открывается рабочие окно.(Рисунок 1)

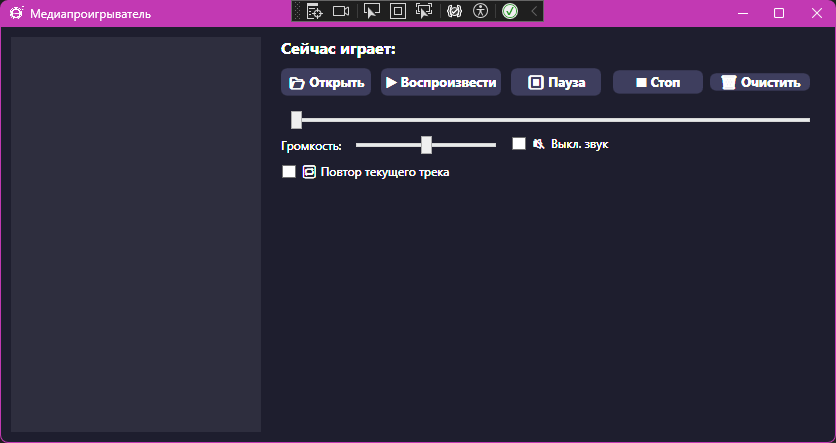


Рисунок 1 – Запуск приложения

Добавляем медиафайлы.(Рисунок 2)

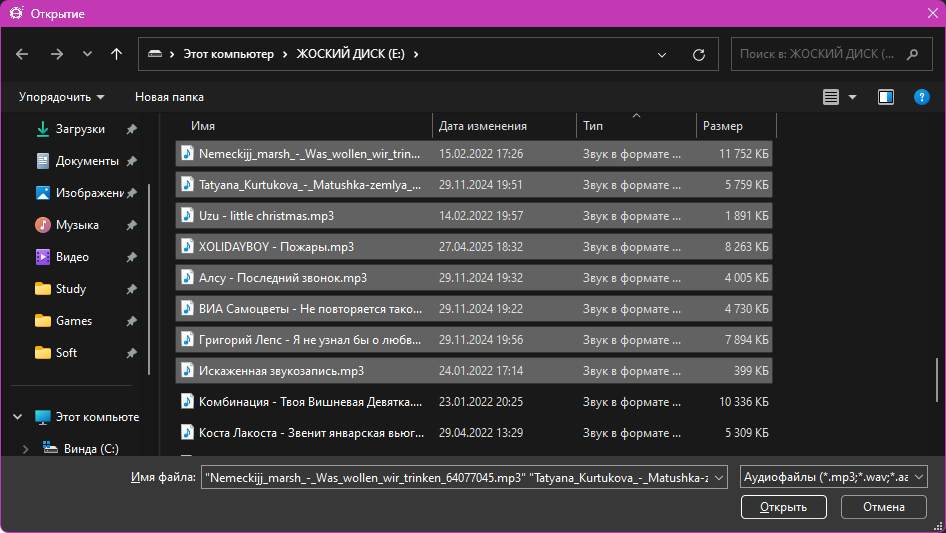


Рисунок 2 – Добавление медиафайлов

Выбираем трек из списка треков и воспроизводим.(Рисунок 3)

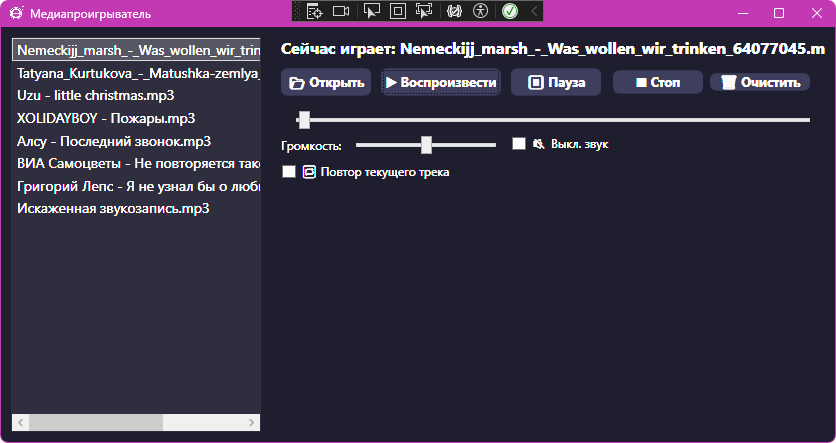


Рисунок 3 – Воспроизведение трека

Так же в функционале присутствует бегунок изменения громкости, выключения звука и повтор текущего трека.(Рисунок 4)

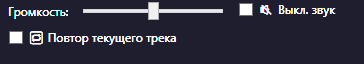


Рисунок 4 – Функционал приложения

С помощью кнопки очистка очищаем список треков.(Рисунок 5)

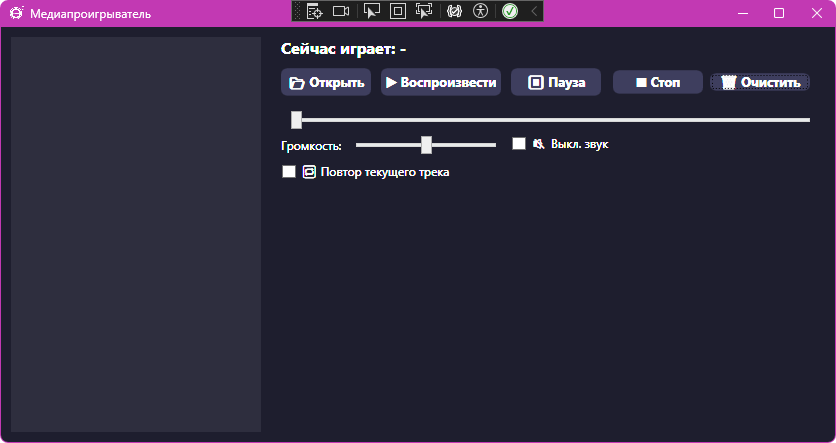


Рисунок 5- Очистка списка треков

**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.

**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 isStopRequested = false;

public MainWindow()

{

InitializeComponent();

// 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()

{

outputDevice?.Stop();

outputDevice?.Dispose();

outputDevice = null;

audioFileReader?.Dispose();

audioFileReader = null;

progressTimer.Stop();

sliderProgress.Value = 0;

}

// Обработка окончания воспроизведения

private void OnPlaybackStopped(object sender, StoppedEventArgs e)

{

Dispatcher.Invoke(() =>

{

if (isStopRequested)

{

isStopRequested = false; // сбросим флаг

return; // ничего не делаем — это было ручное нажатие "Стоп"

}

if (chkRepeat.IsChecked == true)

{

btnPlay\_Click(null, null);

}

else

{

PlayNextTrack();

}

});

}

// 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;

}

// Воспроизведение следующего трека в списке

private void PlayNextTrack()

{

if (playlist.Count == 0) return;

currentTrackIndex++;

if (currentTrackIndex >= playlist.Count)

currentTrackIndex = 0;

listBoxTracks.SelectedIndex = currentTrackIndex;

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);

}

//Выбор трека из списка

private void listBoxTracks\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (listBoxTracks.SelectedIndex >= 0)

{

currentTrackIndex = listBoxTracks.SelectedIndex;

btnPlay\_Click(null, null);

}

}

// 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

}

}

}

}