**Міністерство освіти та науки України**

**Український державний університет науки і технологій**

Факультет: Комп’ютерні технології і системи

Кафедра: Комп’ютерні інформаційні технології

Спеціальність: 121 Інженерія програмного забезпечення

**ЗВІТ**

з виробничої практики

на \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва бази практики) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студента (ки) групи: ПЗ2011(931)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр групи)

Кулик Сергій Вадимович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

Початок практики “8” травня 2023 р.

Закінчення практики “4” червня 2023 р.

Керівники практики:

Від бази практики\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (посада, ім’я, прізвище)

Від кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_доцент, к.т.н.\_Андрющенко Вадим Олександрович\_\_\_\_\_\_

(підпис) (посада, ім’я, прізвище )

Звіт захищено з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

“\_\_\_\_” \_\_\_\_\_\_\_\_ 20 \_\_\_ р.

м. Дніпро

# **Опис предметної області та розробка специфікацій**

## **Тема проекту**

Створення віконного додатку музичного плеєра на Windows Forms з використанням мови програмування C# та бібліотеки NAudio.Wave.

1. **Постановка задачі**

Розробити програму, яка дозволить користувачу відтворювати музику з їхньої музичної бібліотеки. Програма повинна мати графічний інтерфейс користувача, який надає зручний спосіб управління відтворенням музики.

1. **Вимоги до програми**

**Функціональні вимоги:**

1. Додавання та редагування музичних треків:
   * Користувач може додавати нові музичні треки до бібліотеки, вказуючи їхні заголовки, виконавців та іншу інформацію.
   * Користувач може редагувати існуючі музичні треки, змінюючи їхню інформацію.
2. Відтворення музики:
   * Користувач може вибрати музичний трек зі списку і почати його відтворення.
   * Під час відтворення треку, відображаються його назва, виконавець та обкладинка альбому.
   * Користувач може призупинити або продовжити відтворення треку.
   * Користувач може переміщувати позицію в треку за допомогою програвального слайдера.
3. Збереження та завантаження музичної бібліотеки:
   * Користувач може зберегти поточну музичну бібліотеку в файл.
   * Користувач може завантажити збережену музичну бібліотеку з файлу.
4. Завершення роботи:
   * При завершенні роботи програми користувачу буде запропоновано зберегти зміни в музичній бібліотеці, якщо вони були внесені і не збережені.

***Нефункціональні вимоги:***

1. Реалізація програми на мові програмування C#
2. Використання бібліотеки NAudio.
3. Забезпечення коректності та ефективності роботи програми.
4. Забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача.

**Проектування програми**

**Визначення вимог до програми:**

1. Музичний плеєр повинен забезпечувати відтворення аудіофайлів у форматі MP3.
2. Користувач повинен мати змогу додавати, редагувати і видаляти музичні треки з бібліотеки плеєра.
3. Плеєр повинен відображати інформацію про поточний відтворюваний трек, таку як назва треку, виконавець і обкладинка альбому.
4. Користувач повинен мати змогу переміщувати позицію відтворення за допомогою ползунка.
5. Плеєр повинен зберігати стан бібліотеки після закриття і відновлювати його при наступному запуску.

**Вибір технологій:**

1. Мова програмування: C#.
2. Користувальницький інтерфейс: Windows Forms.
3. Бібліотека для відтворення аудіо: NAudio.
4. Збереження даних: XML-серіалізація.

**Реалізація програми:**

1. Створення класу MusicPlayerMain, який буде відповідати за головне вікно програми.
2. Додавання необхідних елементів керування, таких як кнопки, ползунок, списки виконавців і треків.
3. Реалізація функцій для додавання, редагування і видалення виконавців і треків.
4. Налаштування обробників подій для кнопок управління відтворенням.
5. Додавання функцій для відображення інформації про поточний трек та обкладинку альбому.
6. Збереження та відновлення бібліотеки.

**Тестування та налагодження:**

Після завершення розробки програми важливо провести тестування, щоб переконатися в правильному функціонуванні всіх компонентів програми. Під час тестування необхідно перевірити, чи працюють всі методи корректно, чи відображається програма корректно і виконує свій функіонал правильно.

**Компонентна структура програми**

1. Namespace "MusicPlayer":
   * Клас "MusicPlayerMain" - основний клас програми, що успадковує клас "Form" з простору імен "System.Windows.Forms".
2. Компоненти програми:
   * Audio control members:
     + AudioFileReader - клас для читання аудіофайлів.
     + IWavePlayer - інтерфейс для відтворення аудіо.
   * CurrentSong - структура для зберігання інформації про поточну пісню.
   * Playback threading:
     + abortThread - флаг для припинення потоку відтворення.
     + pauseThread - флаг для паузи потоку відтворення.
     + threadLock - об'єкт для блокування потоків.
     + IncrementValueCallback - делегат для зміни значень.
   * Binded data:
     + Artist - клас, що представляє виконавця музики.
     + Song - клас, що представляє пісню.
   * changes\_not\_saved - флаг, що вказує на наявність незбережених змін.

**Текст розробленого додатку**

***Файл Artist.cs***

using System;

using System.Collections.Generic;

namespace MusicPlayer

{

/// <summary>

/// Represents an artist with a name and list of songs

/// </summary>

[Serializable]

public class Artist

{

public string Name { get; set; }

private List<Song> songs = new List<Song>();

public List<Song> Songs { get { return songs; } }

public Artist()

{

Name = "Artist Name";

}

public void AddSong(Song newSong)

{

songs.Add(newSong);

}

}

}

***Файл Song.cs***

using System;

namespace MusicPlayer

{

/// <summary>

/// Represents a song

/// </summary>

[Serializable]

public class Song

{

public string Artist { get; set; }

public string Title { get; set; }

public string Mp3Path { get; set; }

public string ArtworkPath { get; set; }

}

}

***Файл Program.cs***

using System;

using System.Windows.Forms;

namespace MusicPlayer

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MusicPlayerMain());

}

}

}

***Файл MusicPlayerMain.cs***

using NAudio.Wave;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.IO;

using System.Threading;

using System.Windows.Forms;

using System.Xml.Serialization;

namespace MusicPlayer

{

public partial class MusicPlayerMain : Form

{

// Audio control members

private AudioFileReader audioFile = null;

private IWavePlayer player = new WaveOut();

// To keep track of current song's information

private struct CurrentSong

{

public string Title;

public string Artist;

}

private CurrentSong currentSong = new CurrentSong();

// Playback threading

private bool abortThread = false;

private bool pauseThread = false;

private object threadLock = new object();

delegate void IncrementValueCallback();

// Binded data

List<Artist> artists = new List<Artist>();

BindingList<Artist> bindingArtists;

BindingList<Song> bindingSongs;

private bool changes\_not\_saved = false;

public MusicPlayerMain()

{

InitializeComponent();

CenterContent();

StartPosition = FormStartPosition.CenterScreen;

bindingArtists = new BindingList<Artist>(artists);

bindingArtists.RaiseListChangedEvents = true;

bindingArtists.AllowNew = true;

bindingArtists.AllowEdit = true;

artistsListBox.DataSource = bindingArtists;

artistsListBox.DisplayMember = "Name";

welcomeLabel.Text = "Welcome!\n\n" + welcomeLabel.Text;

ChangeVisibility(false);

}

#region Helper Methods

/// <summary>

/// changes whether the welcome screen or playback controls are visible

/// </summary>

/// <param name="playback\_is\_visible"></param>

private void ChangeVisibility(bool playback\_is\_visible)

{

playbackPanel.Visible = playback\_is\_visible;

titleLabel.Visible = playback\_is\_visible;

artistLabel.Visible = playback\_is\_visible;

albumArt.Visible = playback\_is\_visible;

welcomeLabel.Visible = !playback\_is\_visible;

}

/// <summary>

/// Repositions controls to the center of the form

/// </summary>

private void CenterContent()

{

albumArt.Left = (ClientSize.Width - albumArt.Width) / 2;

titleLabel.Left = (ClientSize.Width - titleLabel.Width) / 2;

artistLabel.Left = (ClientSize.Width - artistLabel.Width) / 2;

playbackPanel.Left = (ClientSize.Width - playbackPanel.Width) / 2;

playPause.Left = (playbackPanel.Width - playPause.Width) / 2;

musicTrackBar.Left = (playbackPanel.Width - musicTrackBar.Width) / 2;

welcomeLabel.Left = (ClientSize.Width - welcomeLabel.Width) / 2;

}

/// <summary>

/// Resets all playback controls to default

/// </summary>

private void ResetPlaybackPanel()

{

TimeSpan songLength = audioFile.TotalTime;

int songSeconds = songLength.Minutes \* 60 + songLength.Seconds;

musicTrackBar.Maximum = songSeconds;

musicTrackBar.Value = 0;

string seconds = songLength.Seconds.ToString();

if (songLength.Seconds < 10)

seconds = "0" + seconds;

totalTime.Text = songLength.Minutes.ToString() + ":" + seconds;

elapsedTime.Text = "0:00";

}

/// <summary>

/// Used to update playback controls when the user clicks or drags the slider

/// </summary>

/// <param name="e"></param>

private void PlaybackHandler(MouseEventArgs e)

{

// Jumps to arbitrary point on trackbar the user clicks

if (e != null)

{

double dblValue = ((double)e.X / (double)musicTrackBar.Width) \* (musicTrackBar.Maximum - musicTrackBar.Minimum);

musicTrackBar.Value = Convert.ToInt32(dblValue);

}

audioFile.CurrentTime = TimeSpan.FromSeconds(musicTrackBar.Value);

// Hacky fix for NAudio's IWavePlayer that would stop playback and dispose of the AudioFileReader

if (musicTrackBar.Value == musicTrackBar.Maximum)

{

playPause.Text = "PLAY";

PauseThread();

player.Pause();

}

if (playPause.Text == "PAUSE")

{

player.Play();

ResumeThread();

}

UpdateElapsedTime();

}

/// <summary>

/// Updates the sliders position to the song's position and updates the elapsed time

/// </summary>

private void IncrementPlaybackValues()

{

musicTrackBar.Value = (int)audioFile.CurrentTime.TotalSeconds;

UpdateElapsedTime();

if (musicTrackBar.Value == musicTrackBar.Maximum)

playPause.Text = "PLAY";

}

/// <summary>

/// Updates playback control to display correct elapsed time in song

/// </summary>

private void UpdateElapsedTime()

{

TimeSpan songPosition = audioFile.CurrentTime;

string seconds = songPosition.Seconds.ToString();

if (songPosition.Seconds < 10)

seconds = "0" + seconds;

elapsedTime.Text = songPosition.Minutes.ToString() + ":" + seconds;

}

/// <summary>

/// Playback threads run this in order to update playback controls

/// </summary>

private void ThreadUpdatePlayback()

{

while (!abortThread)

{

if (pauseThread)

{

// If the thread is paused, wait to be resumed

lock (threadLock)

{

Monitor.Wait(threadLock);

}

}

Thread.Sleep(1000);

try

{

// Attempt to update playback controls

IncrementValueCallback d = new IncrementValueCallback(IncrementPlaybackValues);

Invoke(d, new object[] { });

}

catch (Exception e)

{

break;

}

}

}

/// <summary>

/// Starts a new thread that handles playback controls

/// </summary>

private void StartThread()

{

abortThread = false;

lock (threadLock)

{

Thread thread = new Thread(new ThreadStart(ThreadUpdatePlayback));

thread.IsBackground = true;

thread.Start();

}

}

private void PauseThread()

{

pauseThread = true;

}

private void ResumeThread()

{

pauseThread = false;

lock (threadLock)

{

Monitor.PulseAll(threadLock);

}

}

private void AbortThread()

{

abortThread = true;

}

#endregion Helper Methods

#region Event Handlers

private void musicTrackBar\_MouseDown(object sender, MouseEventArgs e)

{

PlaybackHandler(e);

}

private void musicTrackBar\_Scroll(object sender, EventArgs e)

{

PlaybackHandler(null);

}

private void playPause\_Click(object sender, EventArgs e)

{

if (playPause.Text == "PLAY")

{

// Revert song to beginning if the end was reached and the user hit play

if (musicTrackBar.Value == musicTrackBar.Maximum)

{

audioFile.CurrentTime = TimeSpan.FromSeconds(0);

elapsedTime.Text = "0:00";

musicTrackBar.Value = 0;

}

player.Play();

ResumeThread();

playPause.Text = "PAUSE";

}

else

{

PauseThread();

player.Pause();

playPause.Text = "PLAY";

}

}

private void editSongButton\_Click(object sender, EventArgs e)

{

Song selected = (Song)songsListBox.SelectedItem;

// Check if the song to be editted is the currently playing song

bool updating\_currently\_playing = currentSong.Title != null && currentSong.Title == selected.Title && currentSong.Artist == selected.Artist;

// Open an edit song form

SongForm form = new SongForm(bindingArtists, selected);

form.StartPosition = FormStartPosition.CenterParent;

form.Text = "Edit Song";

if (form.ShowDialog() == DialogResult.OK)

{

// If this is true, we must update the currently playing display information with the new editted info

if (updating\_currently\_playing)

{

UpdateSongDisplayInfo(form.Song);

CenterContent();

}

// Fire this event to update the songs list with the editted song

artistsListBox\_SelectedIndexChanged(artistsListBox, null);

changes\_not\_saved = true;

}

}

private void newSongButton\_Click(object sender, EventArgs e)

{

// Open a new song form

SongForm form = new SongForm(bindingArtists);

form.StartPosition = FormStartPosition.CenterParent;

form.Text = "New Song";

if (form.ShowDialog() == DialogResult.OK)

{

// Enable the edit song button if not already, and fire event to update songs list with new song

editSongButton.Enabled = true;

artistsListBox\_SelectedIndexChanged(artistsListBox, null);

changes\_not\_saved = true;

}

}

private void artistsListBox\_SelectedIndexChanged(object sender, EventArgs e)

{

Artist selectedArtist = (Artist)artistsListBox.SelectedItem;

if (selectedArtist != null)

{

if (selectedArtist.Songs.Count == 0)

{

bindingArtists.Remove(selectedArtist);

return;

}

bindingSongs = new BindingList<Song>(selectedArtist.Songs);

bindingSongs.RaiseListChangedEvents = true;

bindingSongs.AllowEdit = true;

bindingSongs.AllowNew = true;

// Bind songs list box to new binding list of an artist's songs

songsListBox.DataSource = bindingSongs;

songsListBox.DisplayMember = "Title";

songsListBox.Invalidate();

}

}

private void songsListBox\_DoubleClick(object sender, EventArgs e)

{

Song s = (Song)songsListBox.SelectedItem;

if (s != null)

{

// Stop current song

playPause.Text = "PLAY";

player.Stop();

// Try to update the MP3 source

try

{

audioFile = new AudioFileReader(s.Mp3Path);

}

catch (Exception ex)

{

// If there's a problem with the MP3 file, display an error message

string message = "MP3 could not be played. Check file path:\n\n" + s.Mp3Path;

string caption = "Error";

MessageBox.Show(message, caption, MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Update the song's display information

UpdateSongDisplayInfo(s);

// Reset playback controls

ResetPlaybackPanel();

ChangeVisibility(true);

CenterContent();

// Stop any existing threads then start a new one

AbortThread();

StartThread();

// Update player with new song and start playing

currentSong.Title = s.Title;

currentSong.Artist = s.Artist;

playPause.Text = "PAUSE";

player.Init(audioFile);

player.Play();

}

}

private void UpdateSongDisplayInfo(Song s)

{

try

{

albumArt.BackgroundImage = Image.FromFile(s.ArtworkPath);

}

catch (Exception e)

{

albumArt.BackgroundImage = Properties.Resources.no\_album\_art;

}

titleLabel.Text = s.Title;

artistLabel.Text = s.Artist;

}

/// <summary>

/// Saves the current application music library to a file

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void saveMusicLibraryToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

XmlSerializer s = new XmlSerializer(artists.GetType());

using (TextWriter w = new StreamWriter(saveFileDialog.FileName))

{

s.Serialize(w, artists);

changes\_not\_saved = true;

}

}

}

/// <summary>

/// Loads a saved music library from storage

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void openMusicLibraryToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (changes\_not\_saved)

{

string message = "You have made changes and are attempting to load a different library.\nAll changes will be lost.";

string caption = "Unsaved Work";

DialogResult d = MessageBox.Show(message, caption, MessageBoxButtons.OKCancel, MessageBoxIcon.Information);

if (d == DialogResult.Cancel)

return;

}

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

XmlSerializer s = new XmlSerializer(artists.GetType());

using (TextReader r = new StreamReader(openFileDialog.FileName))

{

List<Artist> newArtists = (List<Artist>)s.Deserialize(r);

player.Stop();

ChangeVisibility(false);

CenterContent();

bindingArtists.Clear();

foreach (Artist a in newArtists)

bindingArtists.Add(a);

if (bindingArtists.Count > 0)

editSongButton.Enabled = true;

artistsListBox\_SelectedIndexChanged(sender, null);

changes\_not\_saved = false;

}

}

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Close();

}

private void MusicPlayer\_FormClosing(object sender, FormClosingEventArgs e)

{

if (changes\_not\_saved)

{

string message = "You have made changes without saving.\nWould you like to save your work and exit?";

string caption = "Unsaved Work";

DialogResult d = MessageBox.Show(message, caption, MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information);

if (d == DialogResult.Yes)

{

saveMusicLibraryToolStripMenuItem\_Click(sender, null);

}

else if (d == DialogResult.Cancel)

{

e.Cancel = true;

}

}

if (e.Cancel == false)

CleanUp();

}

/// <summary>

/// Aborts any active threads, stops and disposes the audio file and player

/// </summary>

private void CleanUp()

{

AbortThread();

player.Stop();

try { audioFile.Dispose(); }

catch (Exception e) { Console.WriteLine("Exception whiling disposing AudioFileReader: " + e.Message); }

player.Dispose();

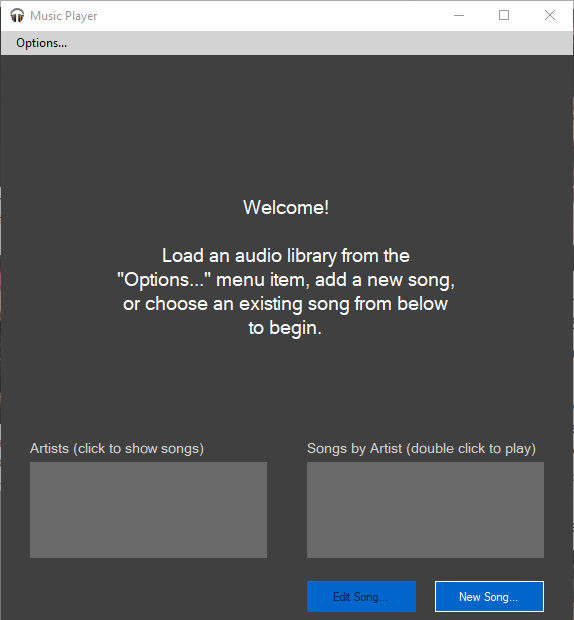
}

#endregion Event Handlers

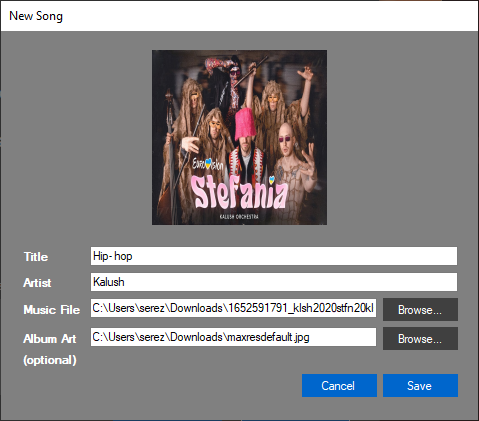
}

}

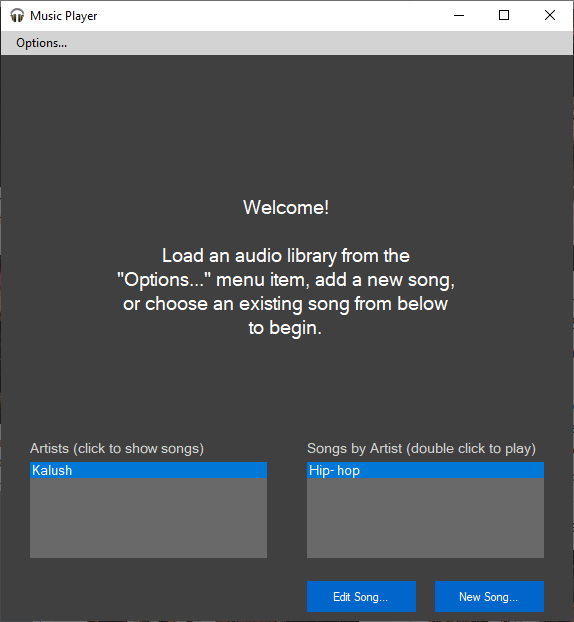
**Результат роботи програми**

****

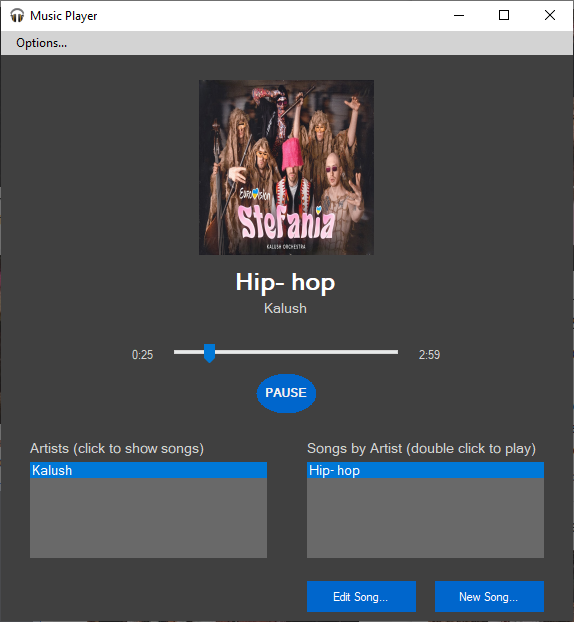
*Рис. 1 – Головне вікно програми*

**

*Рис. 2 – Вікно додавання композиції*

**

*Рис. 3 – Головне вікно після додавання композиції*

**

*Рис. 4 – Головне вікно під час відтворення композиції*

**Висновок**

У розробці музичного плеєра було використано багато технічних засобів, таких як бібліотека NAudio для роботи з аудіо, використання потоків для керування відтворенням музики, та багато інших. Результатом розробки є функціональний музичний плеєр, який дозволяє користувачам відтворювати свою улюблену музику, переглядати і редагувати інформацію про виконавців та пісні, а також зберігати свою музичну бібліотеку.