Universitatea Tehnică “Gheorghe Asachi”, Iași
Facultatea de Automatică și Calculatoare

Music Player

Autori: Timofte Constantin, Noroc Sorin, Costandache Cosmin

Grupa: 1307A

Cuprins

1. Titlu sugestiv

Music Player – Aplicație interactivă potrivită pentru rularea fișierelor de tip mp3, mp4, wav si flac.

2. Documentul Specificațiilor Cerințelor (SRS)

1. Introduction

1.1 Purpose

Scopul acestui document este de a descrie specificațiile de cerințe software (SRS) pentru aplicația MPV Music Player. Aplicația oferă o interfață grafică pentru redarea fișierelor audio utilizând biblioteca MPV.

1.2 Scope

Aplicația permite utilizatorilor să încarce și să redea fișiere audio, să controleze redarea (pauză, următor, anterior), să seteze viteza de redare și volumul, și să vadă timpul curent și total al piesei. Fișierele audio sunt încărcate local și gestionate printr-o listă de redare.

1.3 Definitions, Acronyms, and Abbreviations

MPV - Media Player based on MPlayer2

SRS - Software Requirements Specification

2. Overall Description

2.1 Product Perspective

Aplicația este un sistem desktop independent care utilizează wrapper-ul pentru biblioteca MPV pentru redarea media.

2.2 Product Functions

- Adăugarea melodiilor locale într-o listă de redare
- Redarea fișierelor audio selectate
- Controlul redării: play, pause, următor, anterior
- Setarea vitezei și volumului
- Afișarea timpului curent și total
- Scurtături și suport pentru help integrat

2.3 User Characteristics

Utilizatorii sunt persoane care doresc să redea fișiere audio pe sistemele Windows, fără cerințe tehnice avansate.

2.4 Constraints

- Aplicația rulează doar pe Windows
- Dependență de biblioteca MPV (DLL)
- Folderul 'local' trebuie să conțină fișiere valide
- Formatele suportate: .mp3, .wav, .flac

3. Specific Requirements

3.1 Functional Requirements

- Utilizatorul poate adăuga fișiere audio în lista de redare.
- Utilizatorul poate reda o melodie selectată.
- Utilizatorul poate pune pauză sau relua redarea.
- Utilizatorul poate naviga la următoarea sau melodia precedentă.
- Utilizatorul poate ajusta volumul.
- Utilizatorul poate modifica viteza de redare.
- Aplicația afișează timpul curent și timpul total al melodiei.
- Utilizatorul poate accesa un fișier de tip help (.chm).

3.2 Non-Functional Requirements

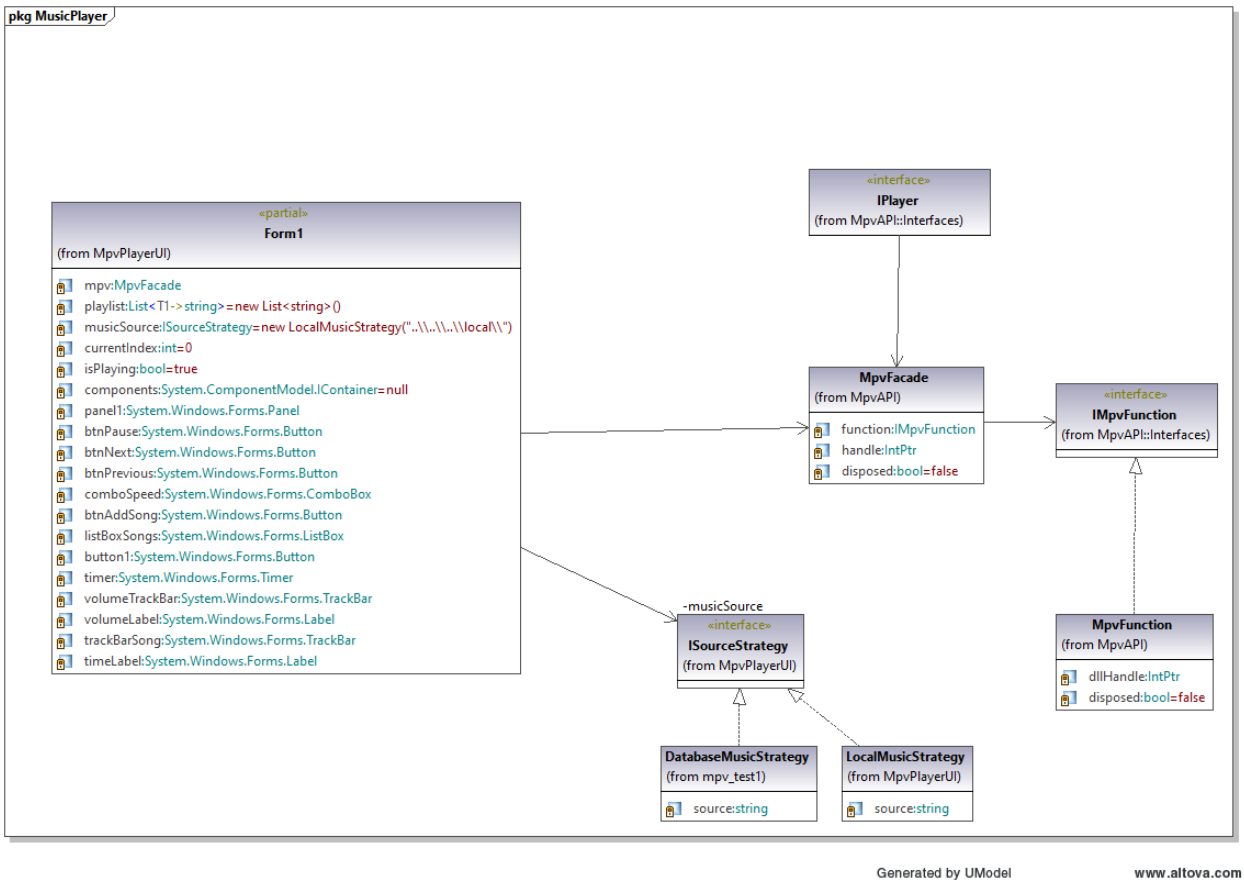
- Interfața aplicației trebuie să fie intuitivă și ușor de utilizat.
- Aplicația trebuie să răspundă comenzilor în mai puțin de 1 secundă.
- Aplicația trebuie să fie compatibilă cu Windows 10 și versiuni ulterioare.
- Aplicația trebuie să utilizeze eficient resursele sistemului.

3.3 External Interface Requirements

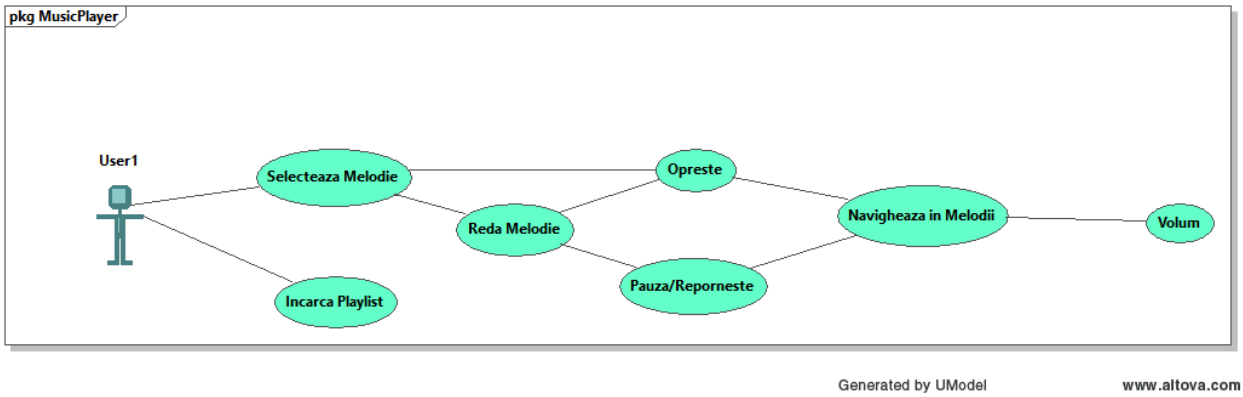
- Interfața cu utilizatorul este bazată pe Windows Forms.
- Biblioteca MPV este integrată printr-un wrapper .NET cu delegate pentru funcții native.
- Interacțiunea utilizatorului se face prin controale: butoane, listBox, trackBar, comboBox.

3. Diagrame

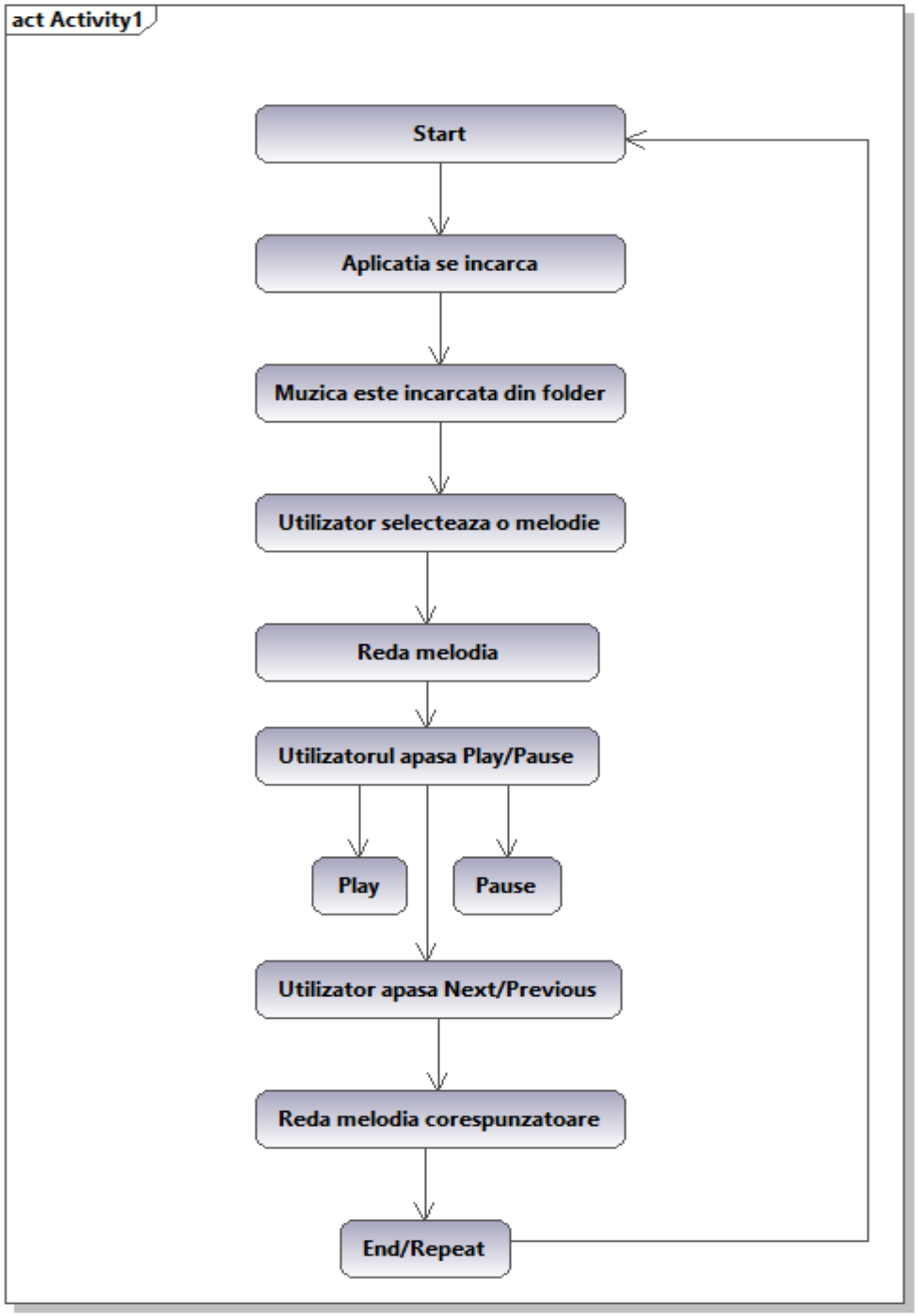
1. Diagrama UML



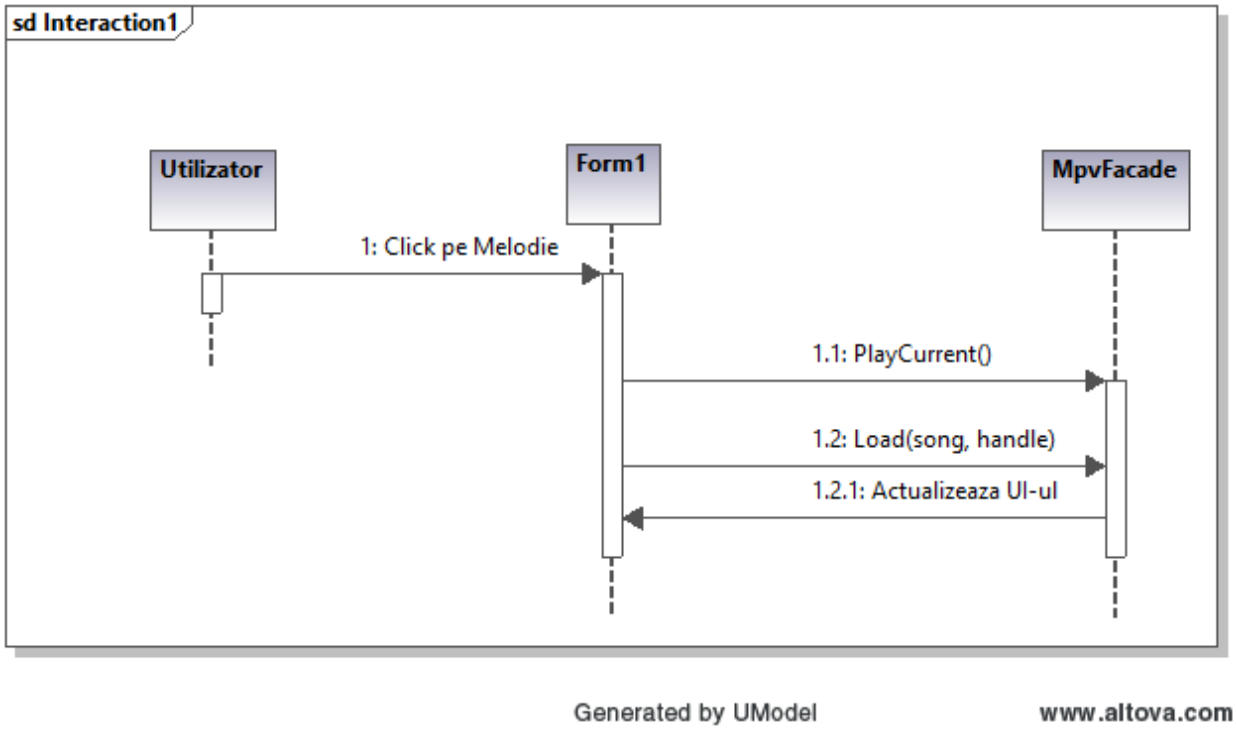
2. Diagrama de Use-Case-uri



3. Diagrama de Activități



4. Diagrama de Secvențe



4. Descriere generală

Aplicația Music Player este un program desktop dezvoltat în limbajul C#, având ca scop redarea fișierelor audio într-un mod eficient, intuitiv și plăcut pentru utilizator. Aceasta oferă o interfață grafică simplă și ușor de utilizat, adresându-se în special utilizatorilor care își doresc un player muzical rapid, stabil și fără funcționalități inutile sau complicate. Redarea fișierelor audio este realizată prin intermediul motorului extern MPV, integrat în aplicație, ceea ce permite un control precis și performant asupra procesului de playback.

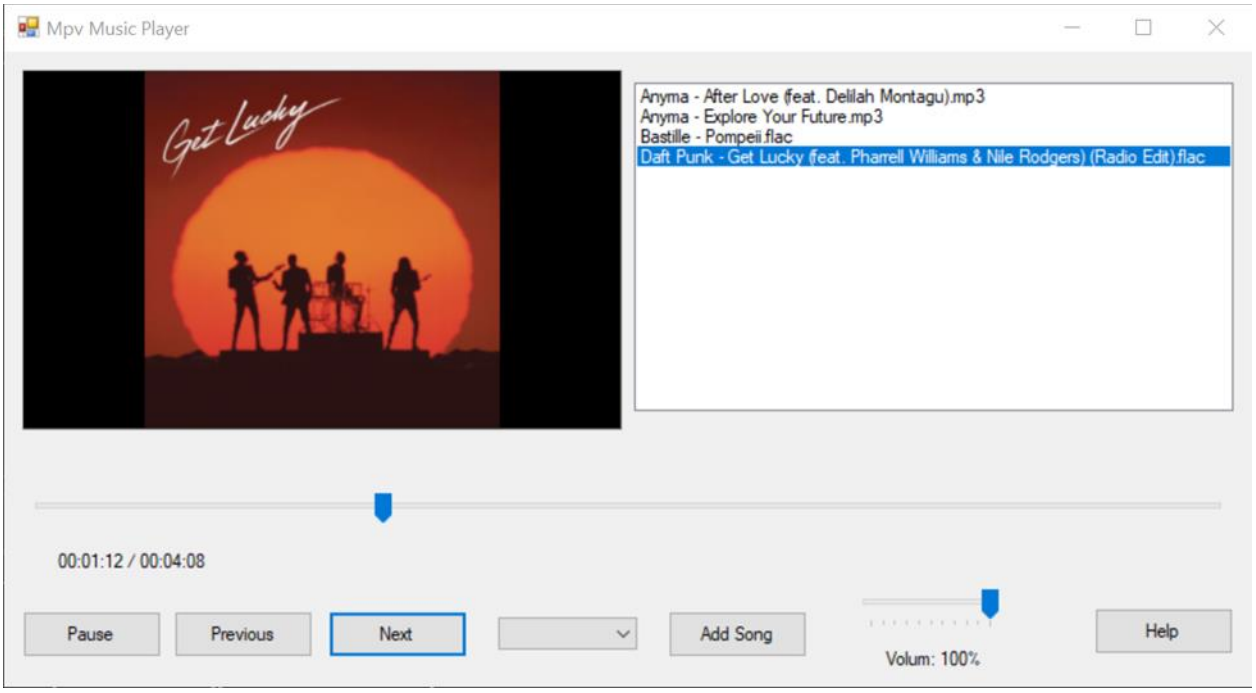
Aplicația suportă funcționalități de bază precum: încărcarea și redarea melodiilor, pauză, oprire, trecerea la melodia următoare sau anterioară, controlul volumului și al timpului de redare. În plus, utilizatorul poate vizualiza informații despre melodia curentă și poate interacționa cu playerul printr-o interfață intuitivă și modernă.

Aplicația este structurată modular, separând clar logica de business, interfața grafică și comunicația cu motorul de redare MPV. Acest lucru facilitează mentenanța și dezvoltarea ulterioară a programului. De asemenea, s-a pus accent pe scrierea de cod clar, bine organizat și documentat, iar pentru părțile critice ale aplicației au fost realizate teste unitare, pentru a asigura funcționarea corectă a principalelor componente.

Aplicația Music Player este potrivită pentru orice utilizator de Windows care dorește o experiență muzicală plăcută, fără reclame, fără consum excesiv de resurse, și fără funcții redundante. Deși este un proiect educațional, aplicația reflectă bune practici de programare și oferă o bază solidă pentru extindere – spre exemplu, în viitor, pot fi adăugate funcționalități precum playlisturi, egalizator audio, salvarea preferințelor sau integrarea cu servicii online.

5. Modul de utilizare

Interfața principală



Interfața aplicației conține o zonă video, un playlist, și un set de butoane de control al redării. Fiecare componentă are un rol specific explicat mai jos:

Lista de melodii

Ce este?

Este caseta din mijlocul aplicației în care vezi toate melodiile pe care le-ai adăugat.

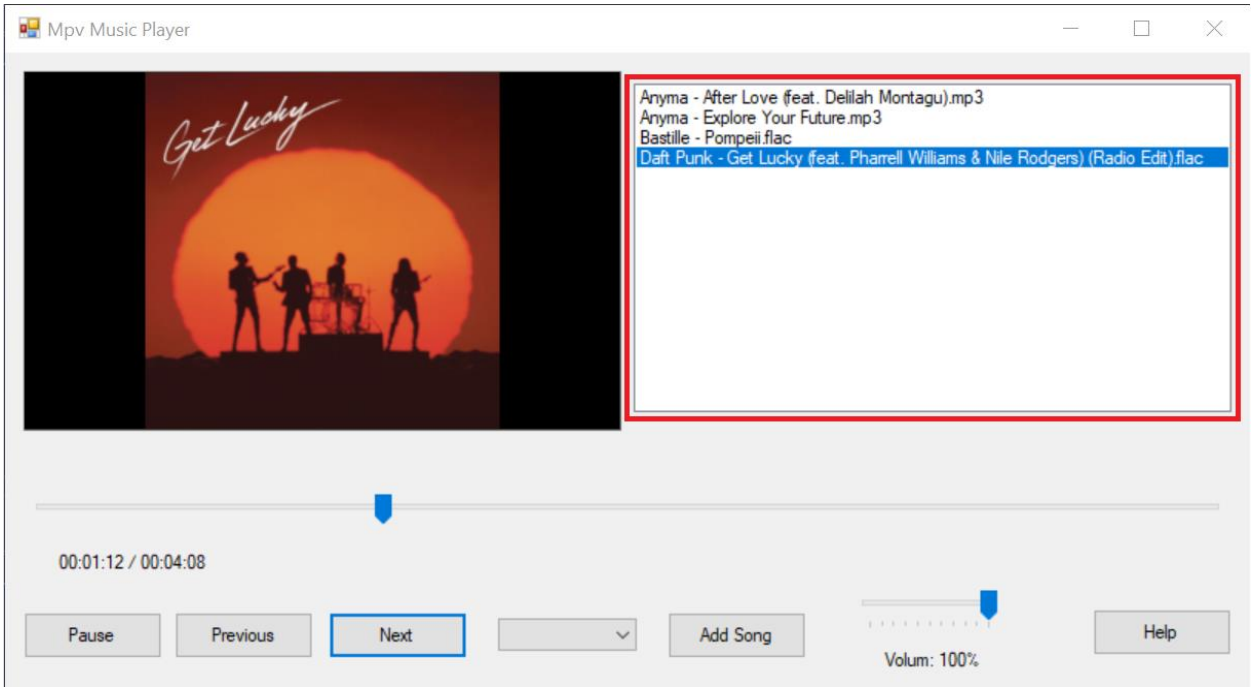
Cum o folosești?

După ce adaugi melodii, ele apar aici.

Dacă dai click pe o melodie din listă, playerul o va porni automat.

Important:

Melodiile sunt afișate cu numele fișierului. Nu poți edita lista direct, dar poți adăuga mai multe melodii cu butonul „Add Song”.



Panou video

Ce este?

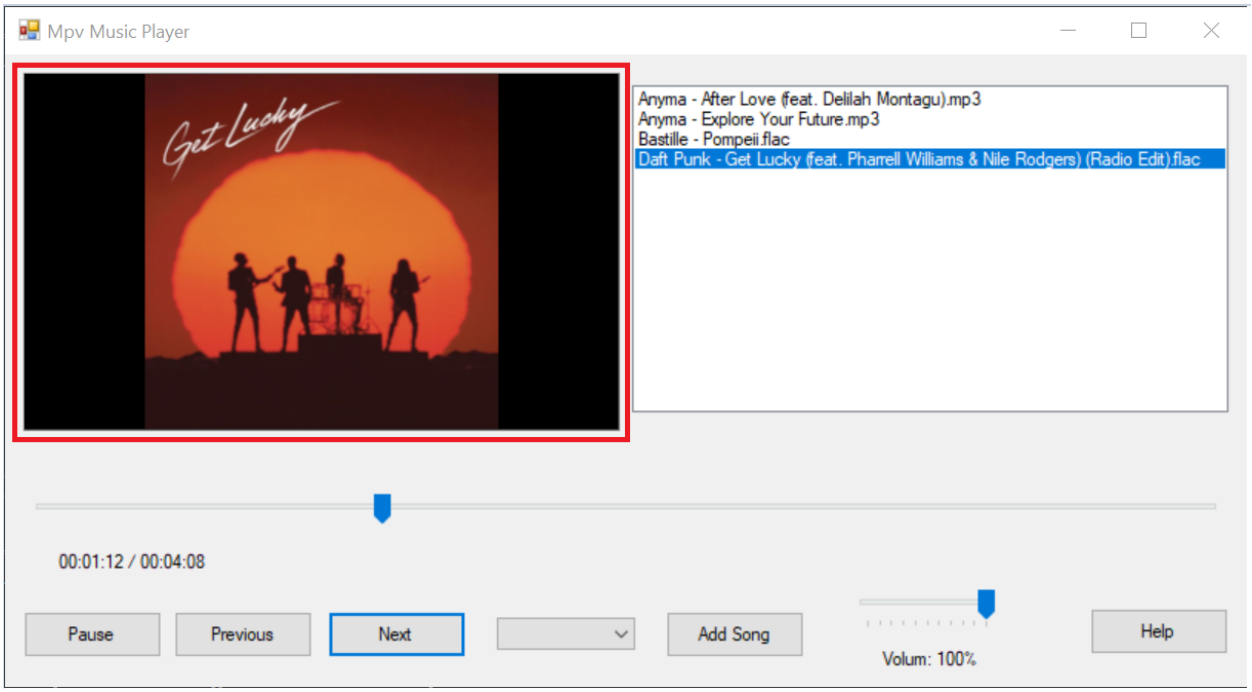
Este locul unde se afișează videoclipuri sau imagini ale melodiei (dacă există).

Unde e?

Sus în aplicație, deasupra playlistului.

Pentru utilizatori noi:

Nu toate melodiile au videoclip sau imagine. Dacă apare ceva aici, e bonus!



▶——●——■ Bara de progres

Ce este?

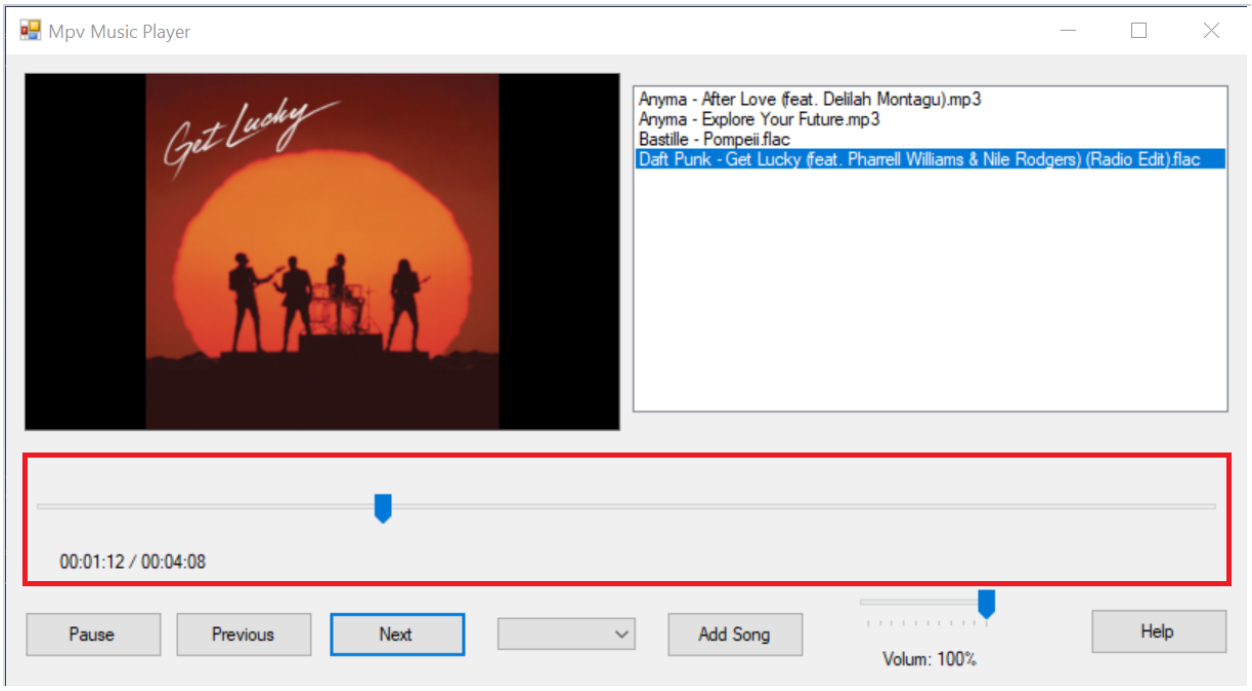
Este locul de unde se controlează timpul melodiilor.

Cum îl folosești?

Ții apăsat pe butonul albastru și tragi în stânga atunci când dorești să asculți o parte mai de la începutul melodiei, respectiv în dreapta atunci când dorești să asculți o parte mai de la finalul melodiei.

Sfaturi utile:

Se poate apăsa de asemenea și pe linie, iar butonul albastru se duce în zona respectivă, modificând și partea din melodie care este redată. Timpul la care se află utilizatorul este afișat sub buton.



□▶ Pause/Play

Ce face?

Pune pauză la melodia care cântă.

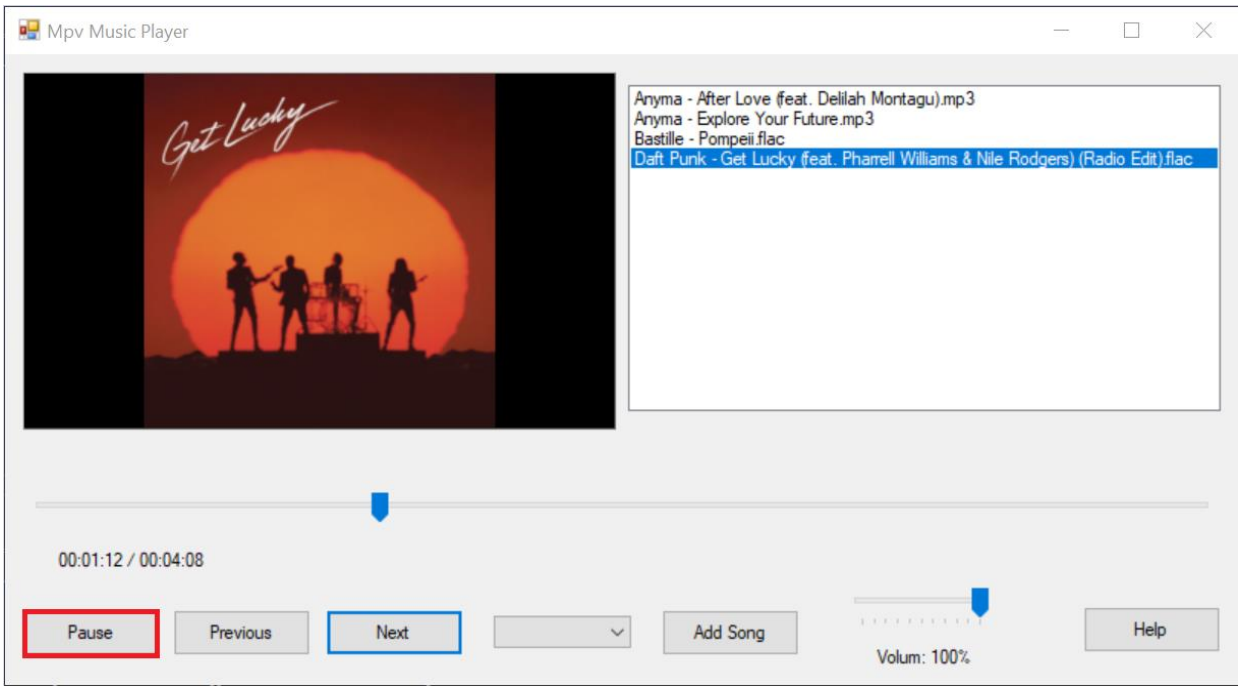
Dacă este deja pusă pe pauză, o pornește din nou.

Cum îl folosești?

Dai click pe el o dată ca să oprești temporar melodia. Dai click din nou ca să o reiei de unde a rămas.

Pentru tine ca utilizator nou:

Imaginează-ți că e un buton de „Stop temporar”. Nu oprește de tot melodia, doar o pune pe pauză.



⏮ Previous

Ce face?

Te duce la melodia anterioară din listă și începe să o cânte.

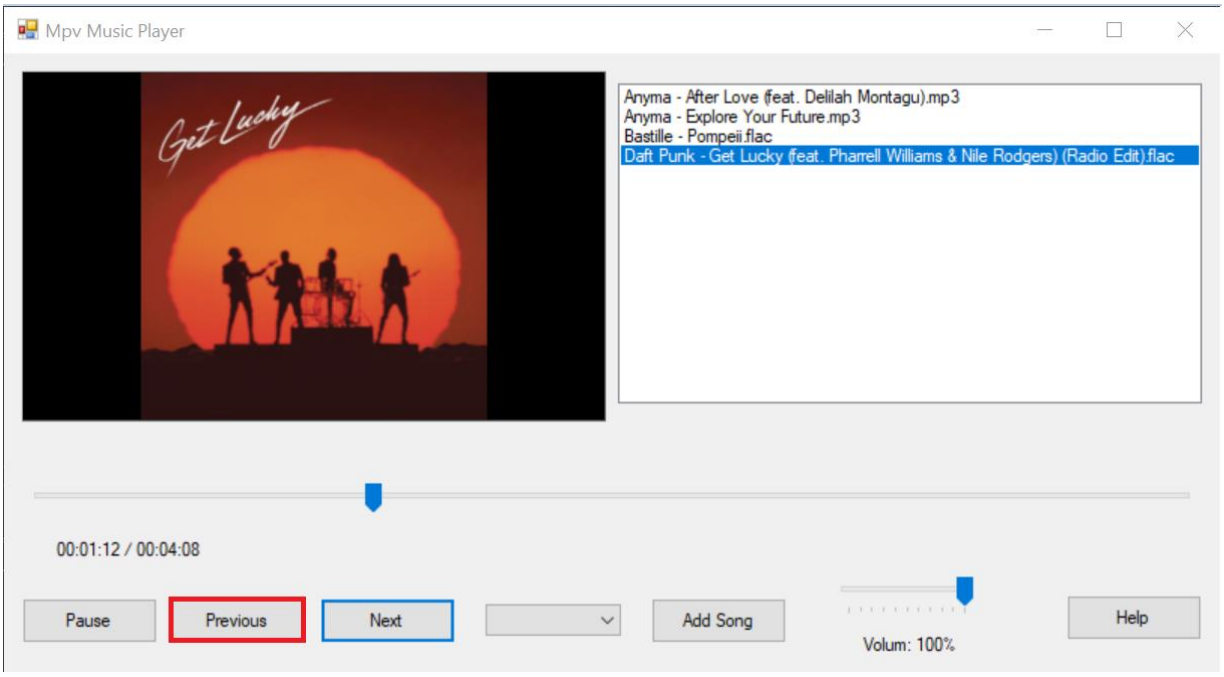
Cum îl folosești?

Dai click o dată și imediat va porni melodia de dinainte.

Detalii utile:

Dacă ești la prima melodie din listă, va reveni la ultima (ca o buclă).

Foarte util dacă ai trecut prea repede la o melodie și vrei să te întorci.



□ Next

Ce face?

Trece la următoarea melodie din listă.

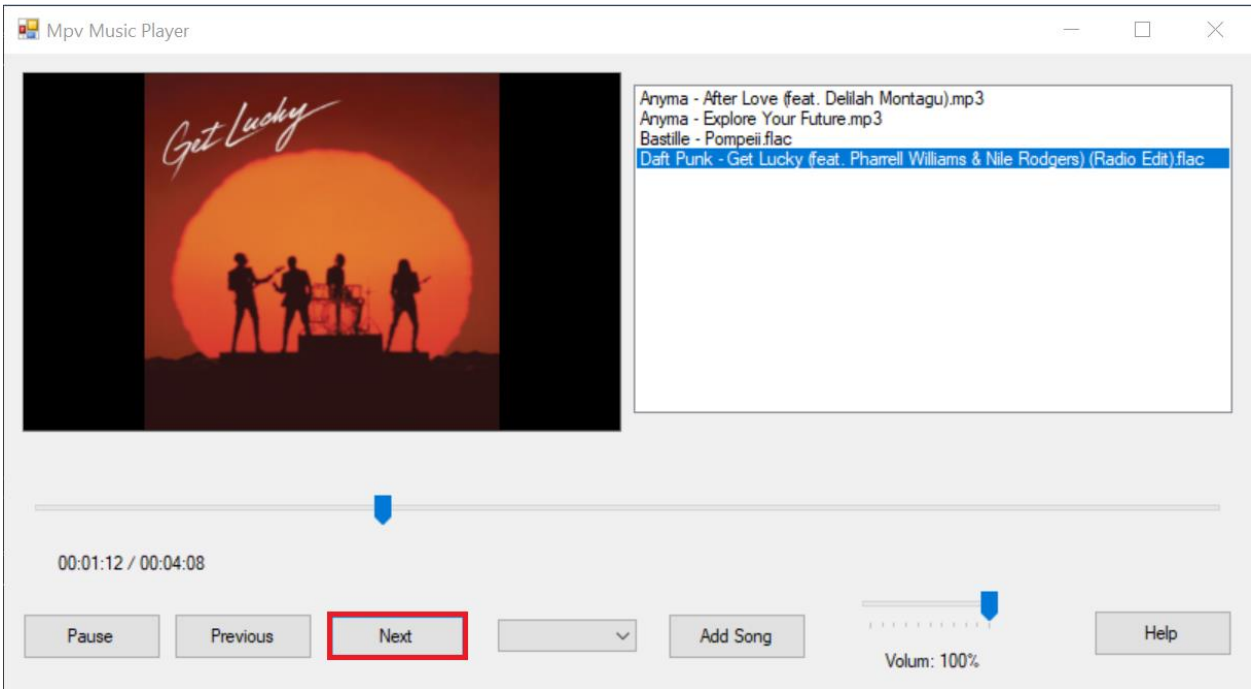
Cum îl folosești?

Dai click pe el și următoarea melodie din playlist va începe automat.

Ce trebuie să știi:

Dacă ești la ultima melodie, va reveni la prima.

Ideal când vrei să sari peste o melodie.



⏩ Viteză redare (ex: 1.0x)

Ce face?

Îți permite să alegi cât de repede (sau încet) vrei să cânte melodia.

Cum o folosești?

Selectezi o viteză din lista derulantă (dropdown).

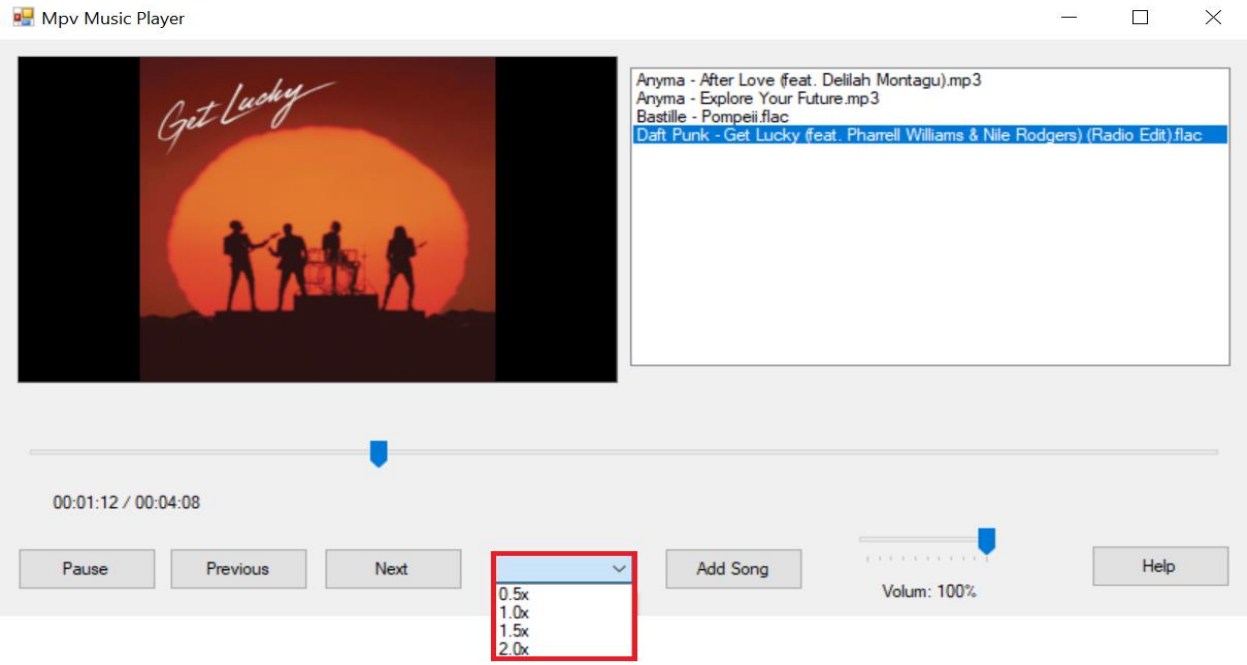
Exemplu:

- 1.0x = normal
- 0.5x = mai lent
- 1.5x = mai rapid
- 2.0x = foarte rapid

Pentru utilizatori noi:

Poți folosi asta ca să înveți versuri mai ușor (redare lentă) sau să testezi mixaje (redare rapidă).

•



➕ Add Song

Ce face?

Îți dă posibilitatea să adaugi melodii noi în player.

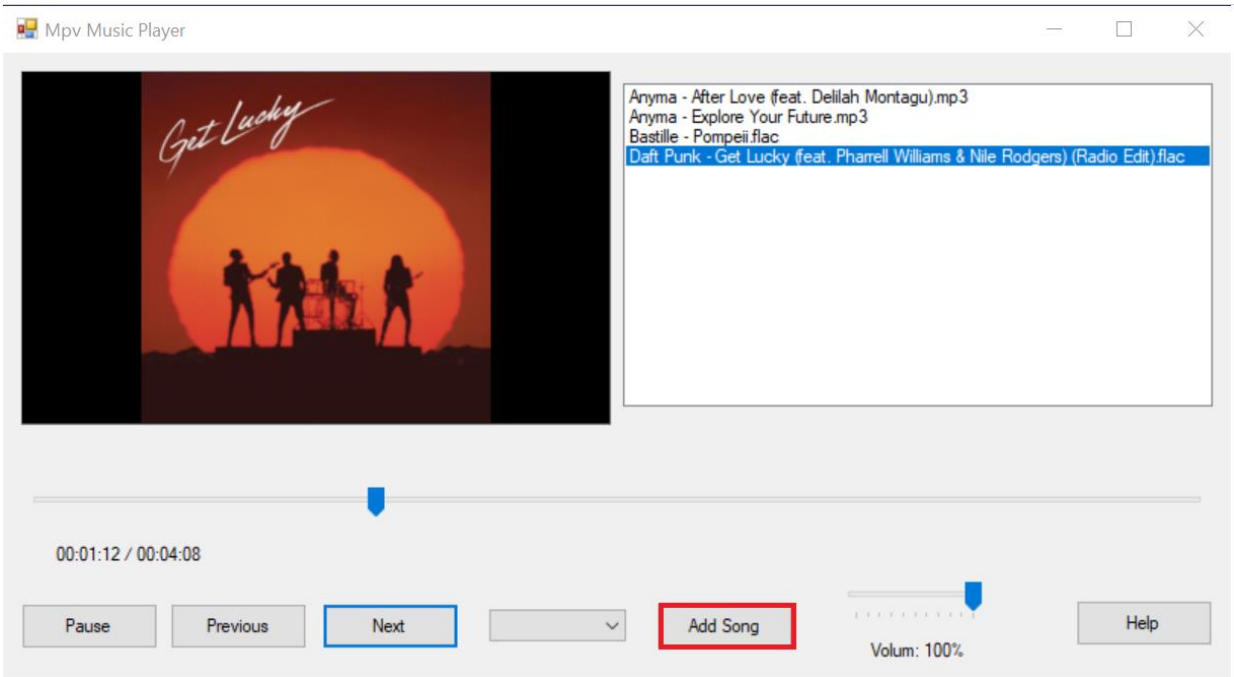
Cum îl folosești?

Când dai click, se deschide o fereastră unde alegi fișiere MP3 de pe calculator. Melodiile selectate vor apărea în listă și vor putea fi redate imediat.

Sfaturi utile:

Nu poți adăuga aceeași melodie de două ori – aplicația te va anunța.

După ce adaugi prima melodie, aceasta începe automat.



Bara de volum

Ce este?

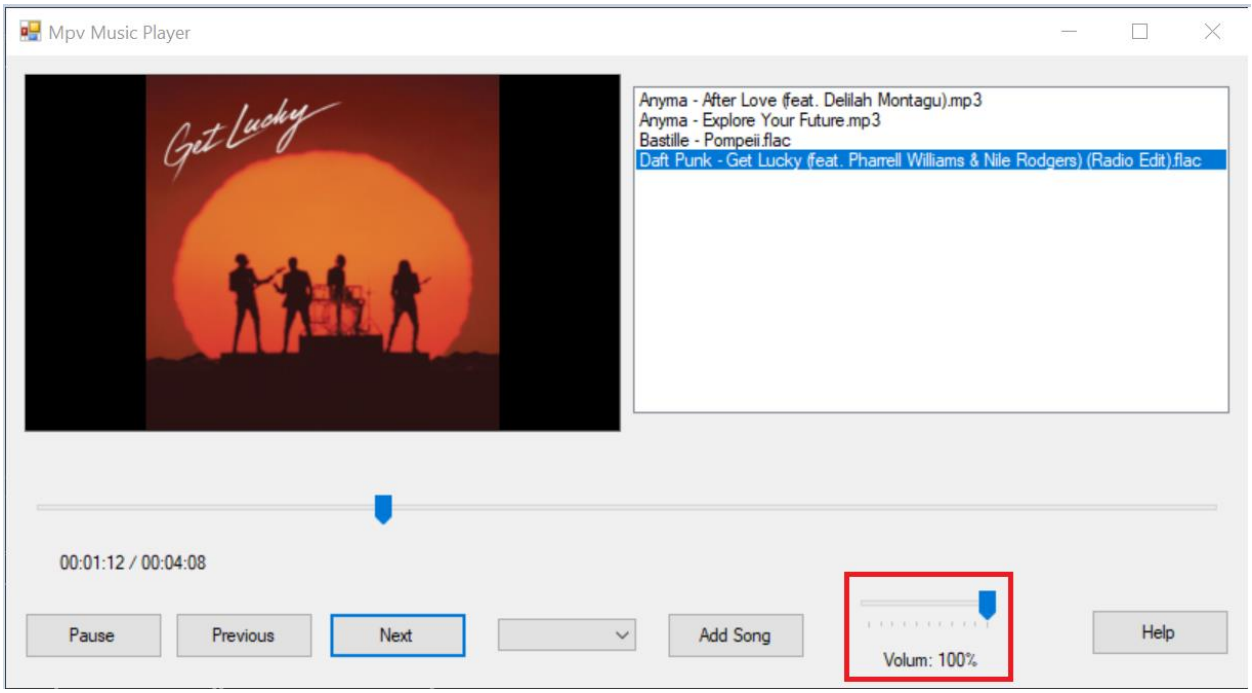
Este locul de unde se controlează volumul melodiilor.

Cum îl folosești?

Ții apăsat pe butonul albastru și tragi în stânga atunci când dorești ca volumul să fie mai mic, respectiv în dreapta atunci când dorești ca volumul să fie mai mare.

Sfaturi utile:

Se poate apăsa de asemenea și pe linie, iar butonul albastru se duce în zona respectivă, modificând și volumul (3 valori: 0%, 50% și 100%). Procentajul volumul este afișat sub buton.



❓ Help

Ce face?

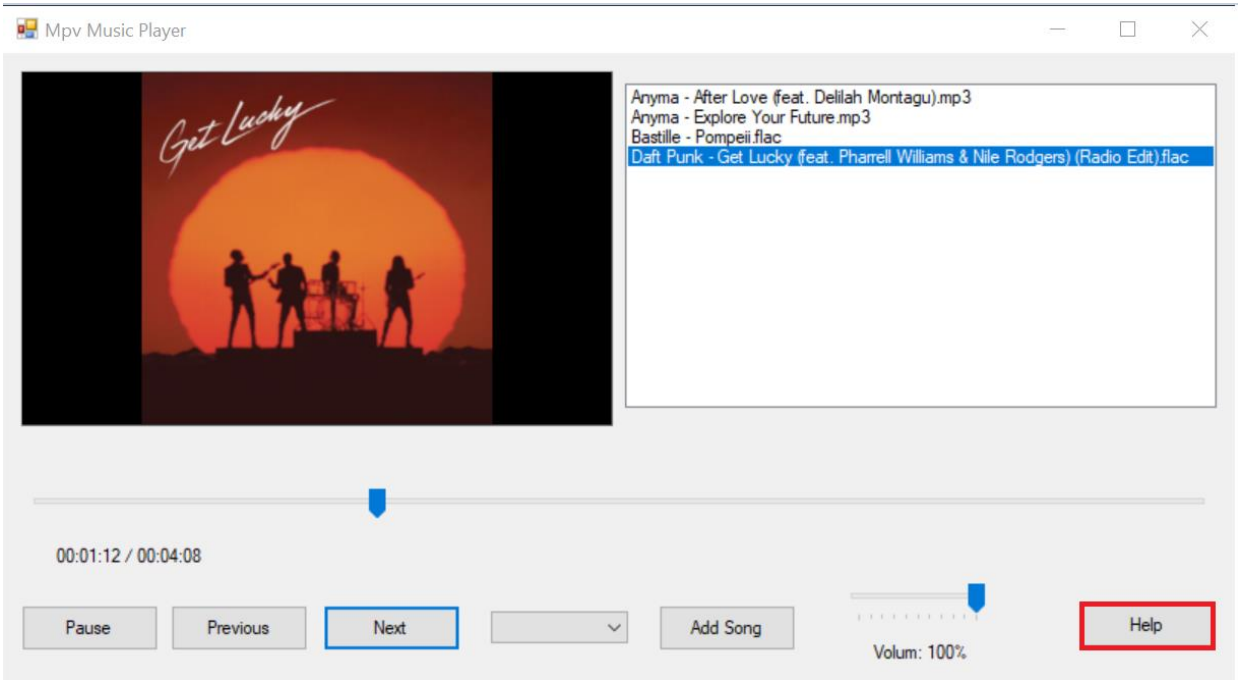
Deschide acest ghid (fișierul de ajutor .chm), ca să afli ce face fiecare funcție.

Cum îl folosești?

Dai click pe el și se deschide ghidul.

Dacă nu merge:

Dacă fișierul de ajutor lipsește, aplicația îți va spune asta și nu se va bloca.



6. Anexă

Resursă repository: https://github.com/SorinNoroc/MusicPlayer

Form1.cs:

```
// File: Form1.cs

// Functionalitate: Interfața grafică principală pentru playerul audio MPV.

// Testare unități: Interfața este testabilă vizual și poate fi testată prin acțiuni simulate (UI automation).

// Sablon utilizat: Observer Pattern (pentru notificarea schimbării stării playerului)


using System;

using System.Collections.Generic;

using System.Data.Common;

using System.Data.SqlTypes;

using System.Linq;

using System.Security.Cryptography;

using System.Windows.Forms;

using MpvAPI;


namespace MpvPlayerUI

{

    public partial class Form1 : Form

    {

        private MpvFacade mpv;

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

        private ISourceStrategy musicSource = new LocalMusicStrategy("..\\..\\..\\local\\");

        public int currentIndex = 0;

        public int CurrentIndex() { return  currentIndex; }
```

```
        public List<string> Playlist() { return playlist; }

        public Form1()
        {
            InitializeComponent();
            this.Load += Form1_Load;
            this.FormClosing += Form1_FormClosing;

            // Adaugare muzica din folderul "local" in interfata
            foreach (var song in musicSource.LoadMusic())
            {
                addMusic(song);
            }
            ;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                string dllPath = "mpv-1.dll";
                // API de acces al functiilor MPV
                mpv = new MpvFacade(dllPath);
                mpv.Initialize();
                // Timer-ul ce raspunde de actualizarea barei de timp a
melodiei
                timer.Start();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Eroare la încărcarea melodiei: " +
ex.Message);
```

```
    }

}

private void addMusic(string music)
{
    playlist.Add(music);
    listBoxSongs.Items.Add(System.IO.Path.GetFileName(music));
}

private void btnPause_Click(object sender, EventArgs e)
{
    if (mpv == null || playlist.Count == 0)
    {
        return;
    }
    try
    {
        if (mpv.isPaused())
        {
            mpv.Resume();
            btnPause.Text = "Pause";
        }
        else
        {
            mpv.Pause();
            btnPause.Text = "Play";
        }
    }
    catch (Exception ex)
    {
```

```
            MessageBox.Show("Eroare la pauză/reluare: " + ex.Message);
        }
    }


    private void playNext()
    {
        if (playlist.Count == 0)
        {
            return;
        }
        // Urmatorul index sau primul in caz ca am ajuns la final
        currentIndex = (currentIndex + 1) % playlist.Count;
        PlayCurrent();
    }

    private void btnNext_Click(object sender, EventArgs e)
    {
        playNext();
    }

    private void btnPrevious_Click(object sender, EventArgs e)
    {
        if (playlist.Count == 0)
        {
            return;
        }
        currentIndex = (currentIndex - 1 + playlist.Count) %
playlist.Count;
        PlayCurrent();
    }
```

```
        private void comboSpeed_SelectedIndexChanged(object sender, EventArgs
e)
        {
            if (mpv == null)
            {

                return;
            }

            try
            {
                string selected =
comboSpeed.SelectedItem.ToString().Replace("x", "");
                if (float.TryParse(selected, out float speed))
                {
                    mpv.SetSpeed(speed);
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Eroare la schimbarea vitezei: " +
ex.Message);
            }
        }

        private void btnAddSong_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Multiselect = true;
            ofd.Filter = "Fișiere audio
(*.mp3;*.wav;*.flac;*.mp4)|*.mp3;*.wav;*.flac;*.mp4";
            if (ofd.ShowDialog() == DialogResult.OK)
            {
```

```
                int i;
                for (i = 0; i < ofd.FileNames.Length; i++)
                {
                    if (playlist.Contains(ofd.FileNames[i]))
                    {
                        continue;

                    }
                    bool success = musicSource.SaveMusic(ofd.FileNames[i]);
                    if (success)
                    {
                        addMusic(ofd.FileNames[i]);
                    }
                }
            }
        }

        private void listBoxSongs_SelectedIndexChanged(object sender,
EventArgs e)
        {
            if (listBoxSongs.SelectedIndex >= 0)
            {
                currentIndex = listBoxSongs.SelectedIndex;
                PlayCurrent();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Citire fisier help
            string helpFile = System.IO.Path.Combine(Application.StartupPath,
"MusicPlayerHelper.chm");
```

```
            if (System.IO.File.Exists(helpFile))
            {
                Help.ShowHelp(this, helpFile);
            }
            else
            {
                MessageBox.Show("Fișierul de help nu a fost găsit!",
"Eroare", MessageBoxButtons.OK, MessageBoxIcon.Error);

            }
        }

        private void volumeTrackBar_Scroll(object sender, EventArgs e)
        {
            double volume = volumeTrackBar.Value;
            mpv.SetVolume((int)volume * 10);
            volumeLabel.Text = $"Volum: {volume * 10}%";
        }

        /**
         * Timer ce citeste date despre melodie (durata trecuta si durata
totala)
         * Actualizeaza trackBar-ul cantecului si label-ul cu timpul
         */
        private void timer_Tick(object sender, EventArgs e)
        {
            string currentString, totalString;
            int current = (int)mpv.GetTime();
            int total = (int)mpv.GetDuration();

            if (current >= 0 && total > 0)
            {
```

```
            if (current >= total - 1)
            {
                playNext();
            }
            currentString = Utilities.secondsToTime(current);
            totalString = Utilities.secondsToTime(total);
            trackBarSong.Maximum = total;
            trackBarSong.Value = Math.Min(current, trackBarSong.Maximum);


            timeLabel.Text = $"{currentString} / {totalString}";
        }
    }

    private void trackBarSong_Scroll(object sender, EventArgs e)
    {
        mpv.SetTime(trackBarSong.Value);
    }

    /**
     * Porneste cantecul de la indexul curent
     */
    private void PlayCurrent()
    {
        if (playlist.Count == 0 || mpv == null)
        {
            return;
        }

        try
        {
```

```
                string songPath = playlist[currentIndex];
                mpv.Load(songPath, panel1.Handle);
                listBoxSongs.SelectedIndex = currentIndex;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Eroare la redarea melodiei: " + ex.Message);
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)

        {
            mpv?.Dispose();
        }
    }
}
```

MpvFacade.cs:

```
// File: Mpv.cs (din ClassLibrary1)
// Functionalitate: Wrapper peste apeluri native MPV
// Testare: Testare integrată indirect prin controller
// Sablon: Wrapper

using System;
using System.Reflection;
using System.Runtime.InteropServices;
using System.Text;
using Microsoft.SqlServer.Server;
```

```
using MpvAPI.Interfaces;


namespace MpvAPI
{
    public class MpvFacade : IDisposable
    {
        /**
         * Initiere API de accesare a functiilor din MPV, ce sunt stringuri,
de ex. mpv_get_property
         */
        public IMpvFunction Function
        {
            get => function;
            set


            {
                if(value is null) throw new ArgumentNullException("value is
null");
                function = value;
            }
        }


        /**
         * Pointer-ul ce contine contextul/legatura cu MPV
         */
        public IntPtr Handle
        {
            get => handle;
            private set
            {
                if (value == IntPtr.Zero) throw new
ArgumentException("Invalid handle", nameof(handle));
```

```
                handle = value;
            }
        }
        private IMpvFunction function;
        private IntPtr handle;
        private bool disposed = false;
        public MpvFacade(string dllPath)
        {
            if (dllPath == null) throw new ArgumentNullException("Null dll
path");
            if (dllPath.Trim().Length == 0) throw new
ArgumentException("Empty dll path");
            Function = new MpvFunction(dllPath);
        }

        public void Initialize()
        {

            try
            {
                Handle = Function.Create();
                if (Handle == IntPtr.Zero)
                    throw new Exception("Eșec la creare context MPV.");
                if (Function.Initialize(Handle) != 0)
                    throw new Exception("Eșec la inițializarea funcțiilor
MPV.");
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException("Eroare la inițializare
MPV.", ex);
            }
        }
```

```
        /**
         * Verificare cu MPV daca melodia actuala este pe pauza
         */
        public bool isPaused()
        {
            try
            {
                if (Handle == IntPtr.Zero) return true;
                IntPtr lpBuffer;
                Function.GetProperty(Handle, "pause", 1, out lpBuffer);
                string paused = Marshal.PtrToStringAnsi(lpBuffer);
                Function.Free(lpBuffer);
                return paused == "yes";
            }
            catch (Exception ex)
            {


                throw new InvalidOperationException("Nu s-a putut obține
starea de pauză.", ex);
            }
        }


        public void Pause()
        {
            try
            {
                if (Handle == IntPtr.Zero) return;
                SetPropertyString("pause", "yes");
            }
            catch (Exception ex)
```

```
            {
                throw new InvalidOperationException("Nu s-a putut pune
pauză.", ex);
            }
        }

        public void Resume()
        {
            try
            {
                if (Handle == IntPtr.Zero) return;
                SetPropertyString("pause", "no");
            }
            catch (Exception ex)
            {
                throw new InvalidOperationException("Nu s-a putut relua
redarea.", ex);
            }
        }

        public void Load(string path, IntPtr windowHandle)
        {
            try
            {
                if (Handle != IntPtr.Zero)
                    Function.TerminateDestroy(Handle);
                Handle = Function.Create();
                if (Handle == IntPtr.Zero) return;
                Function.Initialize(Handle);
                SetPropertyString("keep-open", "yes");
                var windowId = windowHandle.ToInt64();
```

```
                Function.SetOption(Handle, GetUtf8Bytes("wid"), 4, ref
windowId);

                Command("loadfile", path);

            }

            catch (Exception ex)

            {

                throw new InvalidOperationException("Nu s-a putut încărca
fișierul media.", ex);

            }

        }


        public void Command(params string[] args)

        {

            IntPtr[] byteArrayPointers = null;

            IntPtr mainPtr = IntPtr.Zero;


            try

            {

                mainPtr = AllocateUtf8IntPtrArrayWithSentinel(args, out
byteArrayPointers);

                Function.Command(Handle, mainPtr);

            }


            catch (Exception ex)

            {

                throw new InvalidOperationException("Eroare la trimiterea
comenzii către MPV.", ex);

            }

            finally

            {

                if (byteArrayPointers != null)

                {

                    foreach (var ptr in byteArrayPointers)
```

```
                    Marshal.FreeHGlobal(ptr);
                }
                if (mainPtr != IntPtr.Zero)
                    Marshal.FreeHGlobal(mainPtr);
            }
        }

        public void SetProperty(string name, byte[] data, int format = 9)
        {
            if (data.Length >= 1)
            {
                var dataLength = data.Length;
                var dataPtr = Marshal.AllocCoTaskMem(dataLength);
                Marshal.Copy(data, 0, dataPtr, dataLength);
                Function.SetProperty(Handle, GetUtf8Bytes(name), format, ref
data);

            }
        }

        public void SetPropertyString(string name, string data)
        {
            if((name != null && name.Trim().Length>0)&&(data != null &&
data.Length > 0))
            {
                var bytes = GetUtf8Bytes(data);
                SetProperty(name, bytes, 1);
            }
        }

        public void SetPropertyLong(string name, long data)
        {
```

```
            if ((name != null && name.Trim().Length > 0))
            {
                var bytes = BitConverter.GetBytes(data);
                SetProperty(name, bytes, 4);
            }
        }

        public IntPtr GetPropertyString(string name)
        {
            if (Handle == IntPtr.Zero) return IntPtr.Zero;
            var bytes = IntPtr.Zero;
            bytes = (IntPtr)Function.GetPropertyString(Handle, name, 4, out
bytes);
            return bytes;
        }
        public static IntPtr AllocateUtf8IntPtrArrayWithSentinel(string[]
arr, out IntPtr[] byteArrayPointers)
        {
            int numberOfStrings = arr.Length + 1; // add extra element for
extra null pointer last (sentinel)
            byteArrayPointers = new IntPtr[numberOfStrings];
            IntPtr rootPointer = Marshal.AllocCoTaskMem(IntPtr.Size *
numberOfStrings);
            for (int index = 0; index < arr.Length; index++)
            {
                var bytes = GetUtf8Bytes(arr[index]);
                IntPtr unmanagedPointer = Marshal.AllocHGlobal(bytes.Length);
                Marshal.Copy(bytes, 0, unmanagedPointer, bytes.Length);
                byteArrayPointers[index] = unmanagedPointer;
            }
            Marshal.Copy(byteArrayPointers, 0, rootPointer, numberOfStrings);
            return rootPointer;
        }
```

```
        private static byte[] GetUtf8Bytes(string s)
        {
            return Encoding.UTF8.GetBytes(s + "\0");
        }

        public void SetSpeed(float speed)
        {
            if (Handle == IntPtr.Zero)
            {
                return;
            }
            string speedStr =
speed.ToString(System.Globalization.CultureInfo.InvariantCulture);
            Function.SetPropertyString(Handle, "speed", speedStr);
        }

        public void SetTime(double seconds)
        {
            if (Handle == IntPtr.Zero)
                return;

            string timeStr =
seconds.ToString(System.Globalization.CultureInfo.InvariantCulture);
            Function.SetPropertyString(Handle, "time-pos", timeStr);

        }

        public double GetTime()
        {
            if (Handle == IntPtr.Zero)
                return -1;
```

```
    double time = 0;
    Function.GetPropertyDouble(Handle, "time-pos", 5, ref time);

    if (time == 0)
    {
        return -1;
    }
    return time;
}

public double GetDuration()
{
    if (Handle == IntPtr.Zero)
        return -1;

    double duration = 0;
    Function.GetPropertyDouble(Handle, "duration", 5, ref duration);

    if (duration == 0)
    {
        return -1;
    }
    return (double)duration;
}


public void SetVolume(double volume)
{
    if (Handle == IntPtr.Zero)
    {
        return;
```

```
            }

            string volumeStr =
volume.ToString(System.Globalization.CultureInfo.InvariantCulture);
            Function.SetPropertyString(Handle, "volume", volumeStr);
        }

        public double GetVolume()
        {
            if (Handle == IntPtr.Zero) return -1;

            IntPtr result = IntPtr.Zero;
            Function.GetPropertyString(Handle, "volume", 4, out result);
            if (result == IntPtr.Zero) return -1;

            string volStr = Marshal.PtrToStringAnsi(result);
            Function.Free(result);

            if (double.TryParse(volStr, out double volume))
                return volume;

            return -1;
        }

        /**
         * Curatare resurse MPV
         */

        public void Dispose()
        {
            Dispose(true);
```

```
        }
        protected virtual void Dispose(bool disposing)
        {
            if (!disposed)
            {
                Function.TerminateDestroy(Handle);
                if (disposing && Function is IDisposable disposableFunctions)
                {
                    disposableFunctions.Dispose();
                }
                disposed = true;
            }

        }
        ~MpvFacade()
        {
            Dispose(false);
        }
    }
}
```

Tabel cu rezultatele testelor:

Nr. caz test	Metoda testata	Rezultat asteptat	Rezultat obtinut	Starea testului (Pass/ Failed/ Not run)
1	Test_AddSong_IncreasesListCount	true	true	Passed
2	Test_AddSameSongTwice_NotWorking	true	true	Passed
3	Test_NextSong_ChangesSelection	true	true	Passed
4	Test_SpeedChange1_SetsCorrectSpeed	true	true	Passed
5	Test_SpeedChange2_SetsCorrectSpeed	true	true	Passed
6	Test_SpeedChange3_SetsCorrectSpeed	true	true	Passed
7	Test_SpeedChange4_SetsCorrectSpeed	true	true	Passed
8	Test_SpeedChangeAfterPause_SetsCorrectSpeed	true	true	Passed
9	Test_Pause_TogglesPlayFlag	false	false	Passed
10	Test_AddMultipleSongs_IncreasesCountCorrectly	true	true	Passed
11	Test_PreviousSong_ChangesSelection	true	true	Passed
12	Test_PreviousSong_AtStart_RemainsAtZero	true	true	Passed
13	Test_PreviousSong_AtFirstSong_GoesToLastSong	true	true	Passed
14	Test_NextSong_AtEnd_GoesToFirstSong	true	true	Passed
15	Test_NextSong_AtStart_RemainsAtZero	true	true	Passed
16	Test_SelectInvalidSpeed_DefaultsToNormal	true	true	Passed
17	Test_CurrentSong_ReturnsCorrectSong	true	true	Passed
18	Test_PlayingWithoutSong_StaysFalse	false	false	Passed
19	Test_SpeedComboBox_InvalidInput_DoesNotChangeSpeed	true	true	Passed
20	Test_VolumeScroll_UpdatesVolume	true	true	Passed

7. Contribuția echipei

- **Timofte Constantin**
  - Implementare interfață grafică (UI) în Form1.cs pentru playerul audio MPV.
  - Gestionare playlist (adăugare melodii, selecție, redare, următoare/anterior).
  - Control butoane UI: Play/Pause, Next, Previous.
  - Control volume și viteză de redare.
  - Legătură cu fișierul de help (MusicPlayerHelper.chm).
  - Folosirea patternului Observer pentru actualizări în UI.
  - Scriere teste unitare în MusicPlayerTest.cs
  - Creare clasă wrapper Form1TestWrapper pentru testarea logicii interne a interfeței.
  - Testare funcționalități principale: adăugare melodii, navigare playlist, viteza de redare, volum, butonul de pauză etc.
  - Testare comportamente la margine: melodii duplicate, selectare invalidă, capetele playlistului.
  - Integrare cu biblioteca MPV prin utilizarea clasei MpvFacade în UI.
  - Apeluri directe pentru redare, pauză, modificare timp și volum.
  - Inițializare MPV și legare de panou video din UI.

- **Noroc Sorin**
  - A implementat un șablon de tip Strategy pentru gestionarea surselor de muzică, respectând principiile de extensibilitate și modularitate.
  - A creat LocalMusicStrategy – permite încărcarea fișierelor muzicale dintr-un director local și salvarea acestora cu tratarea erorilor și gestionarea duplicatelor.
  - A create DatabaseMusicStrategy – structurată pentru a permite integrarea cu o bază de date, oferind o arhitectură scalabilă.
  - A extras funcționalitatea de conversie a secundelor în format de timp (hh:mm:ss) într-o clasă auxiliară Utilities, pentru a îmbunătăți reutilizarea codului.
  - A tratat excepțiile în mod responsabil, cu mesaje informative pentru utilizator, asigurând o experiență robustă și clară.
  - A menținut o separare clară a responsabilităților, în conformitate cu principiile OOP (Programare Orientată pe Obiect).

- A pregătit codul pentru ușurința mentenanței și adăugarea de funcționalități viitoare.

- A adăugat comentarii explicative și a păstrat o structură clară a codului, contribuind astfel la lizibilitate și colaborare eficientă.
- A contribuit, conform mențiunilor, și la îmbunătățirea interfeței aplicației, posibil prin ajustări vizuale sau funcționale (nu sunt detalii concrete în fișierele trimise).
- Actualizare progres melodie prin timer_Tick.

- **Costandache Cosmin**

- Dezvoltare infrastructură pentru integrarea cu MPV (bibliotecă nativă de redare media)
- Definirea tuturor delegatelor care corespund funcțiilor native exportate de mpv-1.dll.
- Asigurarea semnăturilor corecte cu UnmanagedFunctionPointer pentru interoperabilitatea între C# și DLL-ul nativ
- Implementare Wrapper peste funcționalitățile MPV, folosind design pattern-ul Wrapper/Adapter.
- Inițializare context MPV (Handle), gestionare erori.
- Proprietăți clare pentru Function și Handle, validări riguroase la setare.
- Separare clară a logicii legate de funcțiile MPV într-un mediu controlat.
- Testabilitate indirectă asigurată prin expunerea funcțiilor într-un mod centralizat, utilizate ulterior de UI.