Anexa Documentatie

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: SQLManager.cs \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace SQLiteManager

{

public class SQLManager

{

private SQLManager()

{

if (!File.Exists("ProjectDatabase.db"))

\_sqlConnection = new SQLiteConnection("Data Source=ProjectDatabase.db;Version=3;New=True;Compress=True;");

else

\_sqlConnection = new SQLiteConnection("Data Source=ProjectDatabase.db;Version=3;Compess=True;");

try

{

\_sqlConnection.Open();

}

catch

{

MessageBox.Show("Unable to open database");

}

CreateTables();

// Adaugare de playlist-uri implicite in baza de date

AddPlaylist("Toate melodiile");

AddPlaylist("Favorite");

AddPlaylist("Melodii Recente");

}

public void DisposeDatabase()

{

\_sqlConnection.Close();

}

public void CreateTables()

{

SQLiteCommand cmd;

cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "CREATE TABLE if NOT EXISTS Melodie (" +

"melodieID INTEGER PRIMARY KEY AUTOINCREMENT," +

"Sursa\_Video TEXT NOT NULL," +

"Nume\_Melodie TEXT NOT NULL," +

"Duration INTEGER NOT NULL," +

"UNIQUE(Sursa\_Video)" +

")";

cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE if NOT EXISTS Playlist\_uri (" +

"playlistID INTEGER PRIMARY KEY AUTOINCREMENT," +

"Nume\_Playlist TEXT NOT NULL," +

"UNIQUE(Nume\_Playlist)" +

")";

cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE if NOT EXISTS Playlist (" +

"playlistID INTEGER NOT NULL," +

"melodieID INTEGER NOT NULL," +

"FOREIGN KEY(melodieID) REFERENCES Melodie(melodieID), " +

"FOREIGN KEY(playlistID) REFERENCES Playlist\_uri(playlistID)" +

")";

cmd.ExecuteNonQuery();

cmd.CommandText = "CREATE TABLE if NOT EXISTS PlaylistRecente (" +

"id INTEGER PRIMARY KEY AUTOINCREMENT," +

"playlistID INTEGER NOT NULL," +

"FOREIGN KEY(playlistID) REFERENCES Playlist\_uri(playlistID)" +

")";

cmd.ExecuteNonQuery();

}

// Stergere melodii dintr-un playlist in functie de id-ul melodiei

// Este apelat din optiunea "Sterge melodie" din evenimentul de click-dreapta pe o melodie

public void DeleteSongFromPlaylist(string playlistName, int songID)

{

int playlistID = -1;

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

SQLiteDataReader reader;

cmd.CommandText = "SELECT playlistID FROM Playlist\_uri WHERE Nume\_Playlist=\'" + playlistName + "\'";

reader = cmd.ExecuteReader();

if (reader.Read())

playlistID = reader.GetInt32(0);

reader.Close();

if (songID != -1 && playlistID != -1)

{

cmd.CommandText = "DELETE FROM Playlist WHERE playlistID=" + playlistID + " AND " + "melodieID=" + songID;

cmd.ExecuteNonQuery();

}

}

// Sterge melodiile din lista de melodii si din playlist-urile care o detin

public void DeleteSong(int songID)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "DELETE FROM Melodie WHERE melodieID=" + songID;

cmd.ExecuteNonQuery();

cmd.CommandText = "DELETE FROM Playlist WHERE melodieID=" + songID;

cmd.ExecuteNonQuery();

}

// Stergerea unui playlist in functie de id

public void DeletePlaylist(int playlistID)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "DELETE FROM Playlist\_uri WHERE playlistID=" + playlistID;

cmd.ExecuteNonQuery();

cmd.CommandText = "DELETE FROM Playlist WHERE playlistID=" + playlistID;

cmd.ExecuteNonQuery();

}

// Procedura de adaugare a unei melodii in baza de date

// Este aplicata din panoul de YouTube cand se alege una din cele doua optiuni

public void AddMelodie(string url, string name, int duration)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

try

{

cmd.CommandText = "INSERT INTO Melodie " +

"(Sursa\_Video, Nume\_Melodie, Duration) VALUES " +

"(\'" + url + "\', \'" + name + "\', " + duration + ")";

cmd.ExecuteNonQuery();

}

catch

{

MessageBox.Show("Operatia de adaugare a melodiei nu a putut fi efectuata");

}

}

// Adauga melodia intr-un playlist

public void AddToPlaylist(int playListID, int melodieID)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

try

{

cmd.CommandText = "INSERT INTO Playlist " +

"(playlistID, melodieID) VALUES " +

"(" + playListID + ", " + melodieID + ")";

cmd.ExecuteNonQuery();

}

catch

{

MessageBox.Show("Operatia de adaugare in playlist a esuat");

}

}

public void AddToPlaylist(string playlistName, string melodie)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "SELECT playlistID FROM Playlist\_uri WHERE Nume\_Playlist=\'" + playlistName + "\'";

int listID = -1, songID = -1;

SQLiteDataReader reader = cmd.ExecuteReader();

if (reader.Read())

listID = reader.GetInt32(0);

reader.Close();

cmd.CommandText = "SELECT melodieID FROM Melodie WHERE Nume\_Melodie=\'" + melodie + "\'";

reader = cmd.ExecuteReader();

if (reader.Read())

songID = reader.GetInt32(0);

if (listID != -1 && songID != -1)

AddToPlaylist(listID, songID);

}

public string GetSongURL(string songName)

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

SQLiteDataReader reader;

cmd.CommandText = "SELECT Sursa\_Video FROM Melodie WHERE Nume\_Melodie=\'" + songName + "\'";

reader = cmd.ExecuteReader();

string rez = "";

if (reader.Read())

rez = reader.GetString(0);

return rez;

}

// Metoda ce se va interoga baza de date in functie de widlcard-ul de la operatia de cautare din panoul de cautare

public Dictionary<string, string> SearchSongsByName(string wildcard)

{

Dictionary<string, string> rez = new Dictionary<string, string>();

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

SQLiteDataReader reader;

cmd.CommandText = "SELECT Nume\_Melodie FROM Melodie WHERE Nume\_Melodie LIKE \'%" + wildcard + "%\'";

reader = cmd.ExecuteReader();

while (reader.Read())

{

string temp = reader.GetString(0);

rez[temp] = GetSongStats(temp);

}

return rez;

}

// Interogheaza baza de date pentru numele melodiilor dintr-un playlist

public List<string> GetListSongs(int listID)

{

List<string> songs = new List<string>();

SQLiteDataReader reader;

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

if (listID != -1)

cmd.CommandText = "SELECT melodieID FROM Playlist WHERE playlistID=" + listID;

else

cmd.CommandText = "SELECT melodieID FROM Melodie";

reader = cmd.ExecuteReader();

List<int> aux = new List<int>();

while (reader.Read())

{

aux.Add(reader.GetInt32(0));

}

reader.Close();

foreach (int elem in aux)

{

cmd.CommandText = "SELECT Nume\_Melodie FROM Melodie WHERE melodieID=" + elem;

reader = cmd.ExecuteReader();

if (reader.Read())

{

songs.Add(reader.GetString(0));

}

reader.Close();

}

return songs;

}

// Interogheaza baza de date pentru melodii in momentul in care se acceseaza un playlist

public Dictionary<string, string> GetSongsFromPlayList(string listName)

{

Dictionary<string, string> rez = new Dictionary<string, string>();

foreach (string song in (List<string>)GetListSongs(listName))

{

rez[song] = GetSongStats(song);

}

return rez;

}

// Metoda ce interogheaza baza de date pentru toate playlist-urile in momentul in care se afiseaza panoul pentru playlist-uri

public Dictionary<string, string> GetPlaylists()

{

Dictionary<string, string> rez = new Dictionary<string, string>();

List<string> playLists = new List<string>();

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "SELECT Nume\_Playlist FROM Playlist\_uri";

SQLiteDataReader reader = cmd.ExecuteReader();

while (reader.Read())

{

playLists.Add(reader.GetString(0));

}

reader.Close();

foreach (string playList in playLists)

{

rez[playList] = GetPlayListStats(playList);

}

return rez;

}

// Proces prin care se obtine numele melodiilor si detalii despre acestea

public Dictionary<string, string> GetSongs()

{

Dictionary<string, string> rez = new Dictionary<string, string>();

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

cmd.CommandText = "SELECT Nume\_Melodie FROM Melodie";

SQLiteDataReader reader = cmd.ExecuteReader();

string song;

while (reader.Read())

{

song = reader.GetString(0);

rez[song] = GetSongStats(song);

}

return rez;

}

// Introducere in baza de date a unui nou playlist

public void AddPlaylist(string newList)

{

if (!CheckPlaylist(newList))

{

SQLiteCommand cmd = \_sqlConnection.CreateCommand();

try

{

cmd.CommandText = "INSERT INTO Playlist\_uri " +

"(Nume\_Playlist) VALUES " +

"(\'" + newList + "\')";

cmd.ExecuteNonQuery();

}

catch

{

MessageBox.Show("Operatia de adaugare a noului playlist a esuat");

}

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: MusicPlayer.cs \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace MediaPlayer

{

public class MusicPlayer

{

// Se va obtine handler-ul pentru redarea melodiei

// se vor genera 10 canale pentru equalizer

// Se va seta volumul cu valoarea de la slider-ul de volum din interfata grafica

private void PlayFunc(string fileName, int volume)

{

Stop();

if (WasapiOut.IsSupportedOnCurrentPlatform)

{

\_soundOut = new WasapiOut();

}

else

{

\_soundOut = new DirectSoundOut();

}

var source = CodecFactory.Instance.GetCodec(fileName);

source = new LoopStream(source) { EnableLoop = false };

((LoopStream)source).StreamFinished += (s, args) => Stop();

\_eq = Equalizer.Create10BandEqualizer(source.ToSampleSource());

\_soundOut.Initialize(\_eq.ToWaveSource(16));

\_soundOut.Play();

ChangeVolume = (float)volume;

\_isReady = true;

}

// Este necesar pentru a afisa cu acuratete progresul melodiei in interfata grafica

// Va fi apelata de evenimentul tick al elementului TimerSong

public string monitorTime(int duration, int percentage)

{

if (\_eq != null && \_isReady)

{

int currentTime = duration \* percentage / 100;

if (currentTime % 60 > 9)

return currentTime / 60 + ":" + currentTime % 60 + "/";

else

return currentTime / 60 + ":0" + currentTime % 60 + "/";

}

return "";

}

// Procedura de oprire si eliminare a handler-lor audio

public void Stop()

{

if (\_soundOut != null)

{

try

{

if (\_soundOut != null)

{

\_soundOut.Stop();

\_soundOut.Dispose();

}

if (\_eq != null)

\_eq.Dispose();

\_soundOut = null;

\_eq = null;

}

catch (Exception err)

{

Console.WriteLine(err);

}

}

}

// Se va apela in momentul in care se apasa buton de play/pause din interfata grafica

public bool Play\_Pause\_Click()

{

if (\_soundOut != null)

if (\_soundOut.PlaybackState == PlaybackState.Playing)

{

\_soundOut.Pause();

}

else if (\_soundOut.PlaybackState == PlaybackState.Paused)

{

\_soundOut.Play();

return true;

}

return false;

}

// Pentru utilizarea utilitarului youtube-dl vom apela functii de sistem

// Din acest motiv am creat aceasta metoda ce va simplifica acest proces prin intermediul argumentelor

private void ExecCommand(string command, string args)

{

var processInfo = new ProcessStartInfo(command, args);

processInfo.CreateNoWindow = true;

processInfo.UseShellExecute = false;

processInfo.RedirectStandardError = true;

processInfo.RedirectStandardOutput = true;

Process process;

try

{

process = Process.Start(processInfo);

}

catch

{

MessageBox.Show("Lipseste una din executabilele dependenta.");

return;

}

process.OutputDataReceived += (object send, DataReceivedEventArgs ev) =>

Console.WriteLine("output>>" + ev.Data);

process.BeginOutputReadLine();

process.ErrorDataReceived += (object send, DataReceivedEventArgs ev) =>

Console.WriteLine("error>>" + ev.Data);

process.BeginErrorReadLine();

process.WaitForExit();

Console.WriteLine("ExitCode: {0}", process.ExitCode);

process.Close();

}

// Metoda de a obtine titlul unui videoclip

public string GetName(string url)

{

string line = "";

var processInfo = new ProcessStartInfo("youtube-dl.exe", "-e " + ParseLink(url));

processInfo.CreateNoWindow = true;

processInfo.UseShellExecute = false;

processInfo.RedirectStandardError = true;

processInfo.RedirectStandardOutput = true;

var process = Process.Start(processInfo);

while (!process.StandardOutput.EndOfStream)

{

line = process.StandardOutput.ReadLine();

}

process.WaitForExit();

Console.WriteLine("ExitCode: {0}", process.ExitCode);

process.Close();

return line;

}

// Metoda de a obtine durata unui videoclip

public int GetDuration(string url)

{

string line = "";

var processInfo = new ProcessStartInfo("youtube-dl.exe", "--get-duration " + ParseLink(url));

processInfo.CreateNoWindow = true;

processInfo.UseShellExecute = false;

processInfo.RedirectStandardError = true;

processInfo.RedirectStandardOutput = true;

var process = Process.Start(processInfo);

while (!process.StandardOutput.EndOfStream)

{

line = process.StandardOutput.ReadLine();

}

process.WaitForExit();

Console.WriteLine("ExitCode: {0}", process.ExitCode);

process.Close();

if (line != "")

{

int duration = 0;

string[] time = line.Split(':');

for (int i = time.Length - 1; i >= 0; --i)

duration += (int)Math.Pow(60, time.Length - i - 1) \* Convert.ToInt32(time[i]);

return duration;

}

else

return 0;

}

// Metoda creata pentru a simplifica procedeul de downloadare a unei melodii

// Melodia va fi in format audio

// In plus in acelasi director vom salva si thumbnail-ul corespunzator melodiei

public void DownloadProcedure(string link, int volume)

{

// Oprim si eliminam melodia din prezent pentru a evita accesarea fisierului audio dupa ce acesta este sters in pasurile urmatoare

if (\_soundOut != null && \_soundOut.PlaybackState != PlaybackState.Stopped)

Stop();

ExecCommand("cmd.exe", "/c mkdir Samples");

ExecCommand("cmd.exe", " /c del /Q \"Samples\\\*\"");

ExecCommand("youtube-dl.exe", "-f best " + ParseLink(link) + " -x --audio-format \"wav\" -o \"Samples\\audio.wav\"");

DownloadThumbnail(ParseLink(link), "https://i.ytimg.com/vi/%s/mqdefault.jpg");

try

{

PlayFunc("Samples\\audio.wav", volume);

}

catch

{

MessageBox.Show("Redarea melodiei nu s-a facut cu succes. Incearca din nou");

}

}

// Metoda de obtinere a thumbnail-ului

public bool DownloadThumbnail(string vid\_id, string url)

{

string path = "Samples/audio.jpg",

id;

try

{

id = vid\_id.Split('=')[1];

}

catch

{

return false;

}

if (File.Exists(@path))

File.Delete(@path);

WebClient client = new WebClient();

client.DownloadFileAsync(new Uri(url.Replace("%s", id)), @path);

return true;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: ViewItem.cs \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace IPAplicatie

{

class ViewManager

{

// Sterge elemntele grafice din vector

public void CleanupItems()

{

if (\_items != null && \_items.Length > 0)

foreach (LayoutItem item in \_items)

{

item.Dispose();

}

\_items = null;

}

// Metoda ce genereaza si afiseaza in panou elementele grafice in functie de layout

public void CreateRoutine(Dictionary<string, string> playLists)

{

int height = 0,

width = 0,

type = 0;

string check = \_mainForm.CheckView(); // Se stocheaza numele panoului vizibil din prezent

// Se determina ce tip de layout se va folosi

// In cazul in care ne dorim sa afisam panoul acasa sau panoul playlist vom afisa elementele intr-un layout de tip tabel

// In cazul in care ne dorim sa afisam melodiile dintr-un playlist vom afisa elemntele intr-un layout de tip lista

if (check == "acasa" || check == "")

{

height = 165;

width = \_panel.Width / Layout.GridItem.NumberItems;

type = 1;

\_panel.Height = (height \* (playLists.Keys.Count / Layout.GridItem.NumberItems + 1));

}

else if (check == "playlistsList")

{

height = 200;

width = \_panel.Width / Layout.GridItem.NumberItems;

type = 1;

\_panel.Height = (height \* (playLists.Keys.Count / Layout.GridItem.NumberItems + 1));

}

else if (check == "playlist" || check == "search")

{

height = 70;

width = \_panel.Width;

\_panel.Height = height \* playLists.Keys.Count;

}

if (\_items == null)

\_items = new LayoutItem[playLists.Keys.Count];

else

{

LayoutItem[] temp = new LayoutItem[\_items.Length + playLists.Keys.Count];

for (int i = 0; i < \_items.Length; ++i)

temp[i + playLists.Keys.Count] = \_items[i];

\_items = temp;

}

for (int i = 0; i < playLists.Keys.Count; ++i)

{

\_items[i] = \_factory.GetItem(type, width, height, i, playLists.Keys.ElementAt(i), playLists.Values.ElementAt(i), global::IPAplicatie.Properties.Resources.music);

\_items[i].SetEvents(new EventHandler(EnterEvent), new EventHandler(LeaveEvent), new MouseEventHandler(RightClickEvent), new EventHandler(DoubleClickEvent));

\_panel.Controls.Add(\_items[i].Panel);

}

}

// In majoritatea cazurilor de afisare a elemtelor grafica dintr-o lista se apeleaza mai intai procedura de eliminare a elementelor pentru inlocuirea acestora cu noile elemente

// Aceasta metoda a fost creata pentru simplificarea apelulilor

public void CreatePlaylists(Dictionary<string, string> playLists)

{

CleanupItems();

CreateRoutine(playLists);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: MainForm.cs \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

namespace IPAplicatie

{

public class MainForm

{

public void RefreshRecentPlaylists(string playlist)

{

\_sqlManager.InsertToRecent(playlist);

}

// Afisarea melodiilor dintr-un playlist in interfata grafica

public void DisplayPlayList(string playList)

{

SetView(panelPlaylist);

\_viewManager.SetPanel = panelPlaylistSongs;

if (playList != "" && playList != "Toate melodiile")

{

labelPlaylistSongsTitle.Text = playList;

\_viewManager.CreatePlaylists(\_sqlManager.GetSongsFromPlayList(playList));

}

else

{

labelPlaylistSongsTitle.Text = "Toate melodiile";

\_viewManager.CreatePlaylists(\_sqlManager.GetSongs());

}

}

// Modifica si asigura ca singurul panou vizibil este cel pasat ca argument

// Si returneaza daca panelul exista in dictionarul de paneluri

public bool SetView(Panel panel = null)

{

bool ret = false;

foreach (KeyValuePair<string, Panel> item in Views)

{

if (item.Value == panel)

ret = true;

if (item.Value != panel)

item.Value.Visible = false;

else

item.Value.Visible = true;

}

if (panel != null && panel != panelPlaylistsList)

\_viewManager.SetPanel = panel;

return ret;

}

// Se foloseste pentru a verificarea panoului vizibil

public string CheckView()

{

foreach (KeyValuePair<string, Panel> item in Views)

{

if (item.Value.Visible)

return item.Key;

}

return "";

}

// Panoul acasa are in componenta sa doua panouri care au rolul de a afisa separat melodiile si playlist-urile recente

// Restul layout-urilor au doar o zona de afisare a listelor deci se face de fiecare data dispose la inceputul crearii elementelor

// Pentru panoul acasa trebuie luat in calcul acest lucru asa ca vom apela metoda de dispose doar o singura data la inceputul metodei

public void ShowAcasa()

{

\_viewManager.CleanupItems();

SetView(panelAcasa);

\_viewManager.SetPanel = panelAcasaMelodii;

\_viewManager.CreateRoutine(\_sqlManager.GetRecentSongs());

\_viewManager.SetPanel = panelAcasaPlaylisturi;

\_viewManager.CreateRoutine(\_sqlManager.GetRecentPlaylists());

}

// Aceasta metoda este apelata in evenimentul de click-dreapta elementele corespondente melodiilor din layout

// Afiseaza optiunile din meniul pentru melodii la coordonatele mouse-ului

public void ShowSongsContext()

{

SongContextMenuStrip.Show(new Point(MousePosition.X, MousePosition.Y));

}

// Aceasta metoda este apelata in evenimentul de click-dreapta elementele corespondente playlist-urilor din layout

// Afiseaza optiunile din meniul pentru playlist-uri la coordonatele mouse-ului

public void ShowPlaylistContext()

{

if (\_viewManager.GetSelectedPlaylist != "Toate melodiile" && \_viewManager.GetSelectedPlaylist != "Favorite" && \_viewManager.GetSelectedPlaylist != "Melodii Recente")

PlaylistContextMenuStrip.Show(new Point(MousePosition.X, MousePosition.Y));

}

public string AddSongToDatabase(string link)

{

string songTitle = "";

if (link != "")

{

songTitle = \_player.GetName(link);

if (songTitle == "")

throw new Exception("Link invalid, este necesar un link catre un videoclip YouTube");

if (songTitle.IndexOf("-") != -1)

songTitle = songTitle.Substring(0, songTitle.IndexOf("-")).Trim(' ') + " - " + songTitle.Substring(songTitle.IndexOf("-") + 1).Trim(' ');

if (!\_sqlManager.CheckMelodie(link))

\_sqlManager.AddMelodie(link, songTitle, \_player.GetDuration(link));

}

return songTitle;

}

// Aceasta metoda este apelata de fiecare data cand se doreste redarea unei melodii

// Trebuie tratata perioada in care se deruleaza procedura de download

// Daca aceasta tratare nu este efectuata exista riscul sa accesam un fisier inexistent

// Se evita aceasta accesare ilegala prin setarea flagului isReady din '\_player' pe valoarea false

public void SetMedia(string title)

{

if (title != "")

{

\_player.Ready = false;

\_sqlManager.UpdateRecentPlaylist(title);

if (CheckView() == "playlist" && labelPlaylistSongsTitle.Text == "Cele mai recente")

{

SetView(panelPlaylist);

\_viewManager.SetPanel = panelPlaylistSongs;

\_viewManager.CreatePlaylists(\_sqlManager.GetRecentSongs());

}

labelSongName.Text = title.Substring(title.IndexOf("-") + 1).Trim(' ');

labelArtistName.Text = \_sqlManager.GetSongStats(title);

\_currentSongDuration = \_sqlManager.GetSongDuration(title);

pictureMediaPlay.Image = IPAplicatie.Properties.Resources.pause;

if (currentOperation != null && currentOperation.IsAlive)

{

//MessageBox.Show(currentOperation.IsAlive.ToString());

currentOperation.Abort();

}

currentOperation = new Thread(() => \_player.DownloadProcedure(\_sqlManager.GetSongURL(title), volumeValue));

currentOperation.Start();

timerThumbnail.Start();

}

}

// Configurarea slider-urilor cu valorile din fisierul equalizer.data

// si returneaza daca s-a citit corect din fisierul equalizer.data

public bool CheckEqualizer(string path)

{

try

{

if (File.Exists(@path))

{

string settings = File.ReadAllText(@path);

return ProcessEqualizer(settings);

}

else

return false;

}

catch

{

MessageBox.Show("Eroare la parsarea datelor de equalizer");

ClearEqualizer();

SaveEqualizer();

return false;

}

}

public bool ProcessEqualizer(string data)

{

string[] values = data.Split(' ');

for (int i = 0; i < 10; i++)

{

int value = Int32.Parse(values[i]);

if (value < 0 || value > 40)

throw new Exception("error");

((TrackBar)(panelEqualizer.Controls.Find("trackBarEq" + (i + 1), false)[0])).Value = value;

}

return true;

}

// Metoda pentru resetarea valorilor din panoul equalizer

private void ClearEqualizer()

{

for (int i = 0; i < 10; i++)

{

((TrackBar)(panelEqualizer.Controls.Find("trackBarEq" + (i + 1), false)[0])).Value = 5;

}

}

// Metoda pentru salvarea configurarilor din panoul pentru equalizer

private void SaveEqualizer()

{

try

{

string text = "";

for (int i = 0; i < 10; i++)

{

text += ((TrackBar)(panelEqualizer.Controls.Find("trackBarEq" + (i + 1), false)[0])).Value.ToString() + " ";

}

File.WriteAllText(@EqualizerPath, text);

}

catch

{

MessageBox.Show("Eroare la scrierea datelor de equalizer");

return;

}

}

}

}