**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH (UEH)**

**TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ**

🖎🕮✍



**ĐỒ ÁN MÔN HỌC**

**ĐỀ TÀI**

**ỨNG DỤNG LINKEDLIST**

**TRONG QUẢN LÝ PLAYLIST NHẠC**

**Học Phần: Cấu Trúc Dữ Liệu & Giải Thuật**

**Danh Sách Nhóm**:

1. NGUYỄN PHAN TUẤN ĐỨC (nhóm trưởng)
2. NGUYỄN HOÀNG BẢO
3. HUỲNH KIM NGUYÊN
4. VÕ ĐỨC HƯNG

**Chuyên Ngành**: CÔNG NGHỆ THÔNG TIN

**Khóa**: K49

**Giảng Viên**: TS. Đặng Ngọc Hoàng Thành

**Tp. Hồ Chí Minh, Ngày 18 tháng 04 năm 2024**

# MỤC LỤC

[MỤC LỤC 2](#_Toc164242983)

[CHƯƠNG 1. LINKEDLIST 3](#_Toc164242984)

[1.1. Các Khái Niệm Liên Quan 3](#_Toc164242985)

[1.2. Cấu Trúc và Cài Đặt LinkedList 4](#_Toc164242986)

[1.3. Các Thuật Toán Trên LinkedList 7](#_Toc164242987)

[a) Thuật Toán Tìm Nút 7](#_Toc164242988)

[b) Thuật Toán Chèn Nút 8](#_Toc164242989)

[c) Thuật Toán Xóa Nút 10](#_Toc164242990)

[d) Thuật Toán In Thông Tin Nút 11](#_Toc164242991)

[CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP 13](#_Toc164242992)

[2.1. Phân Tích Bài Toán Quản Lý Playlist Nhạc Bằng LinkedList (Doubly LinkedList) 13](#_Toc164242993)

[2.1.1. Các chức năng trong Playlist Nhạc 13](#_Toc164242994)

[2.1.2. Các tài nguyên cần thiết 13](#_Toc164242995)

[2.2. Sơ Đồ Lớp 14](#_Toc164242996)

[2.3. Cài Đặt Lớp 18](#_Toc164242997)

[CHƯƠNG 3. THIẾT KẾ GIAO DIỆN 33](#_Toc164242998)

[3.1. Giao Diện Menu Chính 33](#_Toc164242999)

[3.2. Chi Tiết Chức Năng 33](#_Toc164243000)

[CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ 36](#_Toc164243001)

[4.1. Các Kết Quả Nhận Được 36](#_Toc164243002)

[4.1.1. Giao diện khi khởi động chương trình 36](#_Toc164243003)

[4.1.2. Giao diện khi thêm bài hát vào Playlist 37](#_Toc164243004)

[4.1.3. Giao diện khi phát/dừng bài hát tại mục Playlist 38](#_Toc164243005)

[4.1.4. Giao diện khi phát/dừng bài hát tại mục Home 39](#_Toc164243006)

[4.1.5. Giao diện khi xóa bài hát 40](#_Toc164243007)

[4.1.6. Giao diện khi thêm bài hát trùng 40](#_Toc164243008)

[4.1.7. Các tính năng khác 41](#_Toc164243009)

[4.2. Một Số Tồn Tại 41](#_Toc164243010)

[4.3. Hướng Phát Triển 42](#_Toc164243011)

[PHỤ LỤC 43](#_Toc164243012)

[TÀI LIỆU THAM KHẢO 48](#_Toc164243013)

# CHƯƠNG 1. LINKEDLIST

## 1.1. Các Khái Niệm Liên Quan

LinkedList là một cấu trúc dữ liệu trong lập trình, trong đó các phần tử được tổ chức dưới dạng các "node" (nút) nối với nhau theo một thứ tự nhất định. Mỗi node chứa dữ liệu và một tham chiếu (pointer) đến node tiếp theo trong chuỗi.

* Một số khái niệm liên quan đến LinkedList:

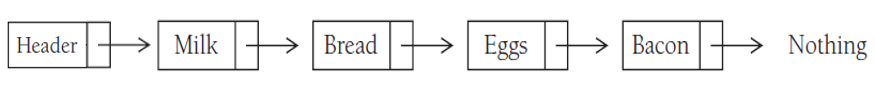
**Node (Nút)**: Là một phần tử cơ bản của LinkedList, mỗi node chứa dữ liệu và một tham chiếu đến node tiếp theo (và/hoặc node trước đó, trong trường hợp của Doubly LinkedList).

**Head (Đầu)**: Là node đầu tiên trong LinkedList. Nó là điểm bắt đầu khi bạn muốn truy cập đến các node khác trong danh sách.

**Tail (Đuôi)**: Là node cuối cùng trong LinkedList. Thông thường, nếu LinkedList có tham chiếu đến tail, việc thêm một phần tử mới vào LinkedList sẽ nhanh chóng hơn.

* Các loại liên kết LinkedList:

**Singular LinkedList**: Là loại LinkedList chỉ duyệt các phần tử theo chiều về trước, là phần liên kết (link) chứa thông tin của nút tiếp theo.



**Doubly LinkedList**: Là một loại LinkedList mà mỗi node không chỉ có tham chiếu đến node tiếp theo mà còn có tham chiếu đến node trước đó hay nói cách khác là phần liên kết chứa thông tin của nút trước (flink) và nút sau (blink).

Diagram

Description automatically generated

**Circular LinkedList**: Là một loại LinkedList trong đó tham chiếu của node cuối cùng trỏ đến node đầu tiên, tạo thành một vòng lặp.

A diagram of a number

Description automatically generated

* Các phương thức tác động cơ bản lên LinkedList:

**Traversal (Duyệt)**: Là quá trình đi qua từng node trong LinkedList, thường bắt đầu từ head cho đến khi gặp node cuối cùng.

**Insertion (Chèn)**: Thêm một node mới vào LinkedList. Có thể chèn vào đầu danh sách, cuối danh sách hoặc ở vị trí bất kỳ trong danh sách.

**Deletion (Xóa)**: Loại bỏ một node khỏi LinkedList. Tương tự như chèn, việc xóa cũng có thể diễn ra ở đầu, cuối hoặc ở vị trí bất kỳ trong danh sách.

**Search (Tìm kiếm)**: Tìm kiếm một node có chứa giá trị cụ thể trong LinkedList.

**Size (Kích thước)**: Số lượng node có trong LinkedList.

**Empty LinkedList**: LinkedList không chứa bất kỳ node nào.

**Sparse LinkedList**: LinkedList mà số lượng node trong đó ít so với dung lượng được cấp phát.

## 1.2. Cấu Trúc và Cài Đặt LinkedList

Một LinkedList phải gồm các nút (node)

Mỗi node bao gồm thông tin (element) và liên kết (link)

Có 3 loại LinkedList:

* Với Singular LinkedList:

Định nghĩa:

public class Node

{

    public object element;

    public Node link;

    public Node()

    {

        element = null;

        link = null;

    }

    public Node(object element)

    {

        this.element = element;

        link = null;

    }

}

public class LinkedList{

    public Node header;

    public LinkedList(){

        header = new Node("Header");

    }

. . .

}

Cách cài đặt:

    static void Main()

    {  
        LinkedList list = new LinkedList();

        list.Insert("First", "Header");

        list.Insert("Second", "First");

        list.Insert("Third", "Second");

        list.Print();

        Console.WriteLine("===");

        list.Remove("Second");

        list.Print();

        Console.ReadLine();

    }

* Với Doubly LinkedList:

Định nghĩa:

public class Node2{

    public object element;

    public Node2 flink, blink;

    public Node2(){

        element = null;

        flink = blink = null;

    }

    public Node2(object element){

        this.element = element;

        flink = blink = null;

    }

}

public class DoubleLinkedList{

    public Node2 header;

    public DoubleLinkedList(){

        header = new Node2("Header");

    }

. . .

}

Cách cài đặt:

    static void Main()

    {  
        DoubleLinkedList dlist = new DoubleLinkedList();

        dlist.Insert("First", "Header");

        dlist.Insert("Second", "First");

        dlist.Insert("Third", "Second");

        dlist.Print();

        Console.WriteLine("---");

        dlist.Remove("Second");

        dlist.Print();

        Console.ReadLine();

    }

Với Circular LinkedList:

Định nghĩa:

public class Node2Circular

{

    public object element;

    public Node2Circular next;

    public Node2Circular()

    {

        element = null;

        next = null;

    }

    public Node2Circular(object element)

    {

        this.element = element;

        next = null;

    }

}

public class CircularLinkedList

{

    public Node2Circular header;

    public CircularLinkedList()

    {

        header = new Node2Circular("Header");

        header.next = header; // Liên kết header với chính nó tạo thành vòng

    }

Cách cài đặt:

static void Main()

{

    CircularLinkedList clist = new CircularLinkedList();

    clist.Insert("First", "Header");

    clist.Insert("Second", "First");

    clist.Insert("Third", "Second");

    clist.Print();

    Console.WriteLine("---");

    clist.Remove("Second");

    clist.Print();

    Console.ReadLine();

}

## 1.3. Các Thuật Toán Trên LinkedList

### a) Thuật Toán Tìm Nút

Với Singular LinkedList:

public class Node2Circular

{

    public object element;

    public Node2Circular next;

    public Node2Circular()

    {

        element = null;

        next = null;

    }

    public Node2Circular(object element)

    {

        this.element = element;

        next = null;

    }

}

Với Doubly LinkedList:

private Node2 Find(object element){

   Node2 current = new Node2();

   current = header;

   while(current.element!=element){

       current = current.flink;

   }

   return current;

}

Với Circular LinkedList:

private Node2Circular Find(object element)

{

    Node2Circular current = header.next; // Bắt đầu từ phần tử sau header

    while (current.element != element && current != header)

    {

        current = current.next;

    }

    if (current != header) // Kiểm tra xem phần tử đã được tìm thấy chưa

    {

        return current;

    }

    else

    {

        return null;

    }

}

### b) Thuật Toán Chèn Nút

Với Singular LinkedList:

Ví dụ muốn thêm nút Cookies vào giữa danh sách liên kết sau:

A picture containing diagram

Description automatically generated

Ta có thể theo mẫu code:

public void Insert(object newelement, object afterelement){

   Node current = new Node();

   Node newnode = new Node(newelement);

   current = Find(afterelement);

   newnode.link = current.link;

   current.link = newnode;

}

Với Doubly LinkedList:

Ví dụ muốn thêm nút Raymond vào giữa danh sách liên kết sau:

A diagram of a person's name

Description automatically generated

Ta có thể theo mẫu code:

public void Insert(object newelement, object afterelement){

  Node2 current = new Node2();

  Node2 newnode = new Node2(newelement);

  current = Find(afterelement);

  newnode.flink = current.flink;

  newnode.blink = current;

  current.flink = newnode;

}

Với Circular LinkedList:

Ví dụ muốn thêm nút 30 vào giữa danh sách liên kết sau:

A diagram of a block diagram

Description automatically generated

Ta có thể theo mẫu code:

public void Insert(object newElement, object afterElement)

{

    Node2Circular current = Find(afterElement);

    if (current != null)

    {

        Node2Circular newNode = new Node2Circular(newElement);

        newNode.next = current.next;

        newNode.next.blink = newNode; // Liên kết node mới với node sau nó

        current.next = newNode;

        newNode.blink = current;

    }

}

### c) Thuật Toán Xóa Nút

Với Singular LinkedList:

public Node FindPrev(object element){

  Node current = header;

  while(current.link!=null && current.link.element!=element)

    current = current.link;

  return current;

}

public void Remove(object element) {

    Node2 current = Find(element);

    if (current.flink != null) {

       current.blink.flink = current.flink;

       current.flink.blink = current.blink;

       current.flink = null;

       current.blink = null;

    }

}

Với Doubly LinedList:

public void Remove(object element) {

    Node2 current = Find(element);

    if (current.flink != null) {

       current.blink.flink = current.flink;

       current.flink.blink = current.blink;

       current.flink = null;

       current.blink = null;

    }

}

Với Circurlar LinedList:

public void Remove(object element)

{

    Node2Circular current = Find(element);

    if (current != null && current != header

    {

        current.blink.next = current.next;

        current.next.blink = current.blink;

        current.next = null;

        current.blink = null;

    }

}

### d) Thuật Toán In Thông Tin Nút

Với Singular LinkedList:

public void Print(){

   Node current = new Node();

   current = header;

   while(current.link!=null){

       Console.WriteLine(current.link.element);

       current = current.link;

   }

}

Với Doubly LinedList:

private Node2 FindLast() {

   Node2 current = new Node2();

   current = header;

   while(!(current.flink == null))

       current = current.flink;

   return current;

}

public void Print() {

   Node2 current = new Node2();

   current = FindLast();

   while (!(current.blink == null)) {

       Console.WriteLine(current.element);

       current = current.blink;

   }

}

Với Circurlar LinedList:

private Node2Circular FindLast()

{

    Node2Circular current = header;

    while (current.next != header) // Duyệt qua các nút cho đến khi gặp lại header

        current = current.next;

    return current;

}

public void Print()

{

    Node2Circular current = FindLast().next; // Bắt đầu từ nút sau header

    while (current != header)

    {

        Console.WriteLine(current.element);

        current = current.next;

    }

}

\*Lưu ý: Circular LinkedList tương tự như Singular LinkedList, khác ở chỗ CircularLinkedList với nút cuối cùng sẽ liên kết với nút đầu tiên còn Singular LinkedList thì lại không có bất kỳ liên kết

# CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP

## 2.1. Phân Tích Bài Toán Quản Lý Playlist Nhạc Bằng LinkedList (Doubly LinkedList)

Ứng dụng Quản lý Playlist Nhạc sử dụng Doubly LinkedList để tổ chức và quản lý danh sách các bài hát. Trong mô hình này, mỗi bài hát được biểu diễn bằng một nút trong danh sách liên kết, chứa các thông tin như tên bài hát, nghệ sĩ, album, thời lượng và hình ảnh. Doubly LinkedList cho phép một bài hát có thể được truy cập và điều hướng dễ dàng, cho phép người dùng chọn và chơi bài hát theo mong muốn.

Ứng dụng cung cấp một giao diện người dùng thân thiện, được xây dựng trên nền tảng WinForms. Giao diện này cho phép người dùng thêm, xóa và chọn các bài hát từ danh sách playlist. Mỗi khi bài hát được chọn, ứng dụng sẽ chơi nó bằng cách sử dụng các thư viện phát nhạc có sẵn trong môi trường WinForms, như Windows Media Player hoặc NAudio.

Người dùng có thể điều khiển việc phát nhạc thông qua các nút điều khiển trên giao diện, bao gồm chơi, dừng, tạm dừng, chuyển sang bài hát tiếp theo và quay lại bài hát trước đó. Các chức năng này được kích hoạt bằng cách tương tác với Doubly LinkedList, thay đổi vị trí của bài hát hiện tại và bài hát tiếp theo.

Tóm lại, chương trình này cung cấp một cách thuận tiện và linh hoạt để quản lý và thưởng thức các bài hát trong danh sách playlist, thông qua việc kết hợp giữa Doubly LinkedList và giao diện người dùng trực quan.

### 2.1.1. Các chức năng trong Playlist Nhạc

Ban đầu nhóm em quyết định làm vừa đủ chức năng theo yêu cầu của đề bài, tuy nhiên sau đó vì có thời gian rảnh nên nhóm em quyết định thêm nhiều chức năng khác để hoàn thiện hơn về mặt tính năng như một Media Music Player hoàn chỉnh gồm:

* Nút thêm file nhạc từ máy tính vào playlist
* Phát nhạc khi click vào bất kỳ bài hát nào
* Nút phát, dừng bài hát
* Nút chuyển bài hát tiếp theo hoặc lùi lại bài hát cũ
* Nút lặp lại bài hát đang phát
* Thanh âm lượng để điều chỉnh theo nhu cầu
* Thanh thời gian bài hát có thể điều chỉnh nhu cầu nghe nhạc tại bất kỳ thời điểm nào của bài hát
* Hiển thị thông tin bài hát đang phát

### 2.1.2. Các tài nguyên cần thiết

Các Framework cần cài đặt

* Framework NAudio
* Framework Guna.UI2.WinForms
* Framework TagLib
* Framework System.Security.AccessControl
* Framework System.Security.Principal.Windows
* Framework Microsoft.Win32.Registry

Các thư viện cần khai báo:

* using System.Windows.Forms
* using System.Drawing
* using NAudio.Wave
* using System

## 2.2. Sơ Đồ Lớp

Nhóm đã chia chương trình Music Player thành 6 lớp với mỗi lớp là một chức năng khác nhau bao gồm:

* Lớp Song:
* Đại diện cho một đối tượng bài hát với các thuộc tính như tên tệp, tiêu đề, nghệ sĩ, album, thể loại, thời lượng và hình ảnh.
* Cung cấp các hàm khởi tạo để khởi tạo một đối tượng bài hát với các thuộc tính này.
* Lớp LinkedList:
* Quản lý một danh sách liên kết các đối tượng Bài hát.
* Cho phép thêm một bài hát mới vào cuối danh sách.
* Cho phép xóa một bài hát khỏi danh sách.
* Cung cấp một phương thức để kiểm tra sự trùng lặp của bài hát dựa trên tên tệp.
* Chứa một phương thức để lấy thông tin bài hát như tiêu đề, nghệ sĩ, thể loại, album, thời lượng và hình ảnh từ siêu dữ liệu tệp.
* Lớp Player:
* Xử lý việc phát lại các bài hát bằng cách sử dụng thư viện NAudio.
* Quản lý thiết bị đầu ra âm thanh và bài hát đang phát hiện tại.
* Cung cấp các phương thức để phát, dừng, tạm dừng, tiếp tục, chuyển sang bài hát kế tiếp và quay lại bài hát trước đó.
* Lớp MainDisplay:
* Đại diện cho giao diện người dùng chính của ứng dụng nghe nhạc.
* Chứa các điều khiển để nhập bài hát, hiển thị thông tin bài hát, điều khiển phát lại (phát, tạm dừng, dừng, bài kế tiếp, bài trước), điều chỉnh âm lượng và kích hoạt chế độ lặp lại.
* Sử dụng một Timer để cập nhật tiến độ phát lại.
* Tích hợp với các lớp Player và LinkedList để quản lý các bài hát và phát lại chúng.
* Lớp SongComponent:
* Đại diện cho một thành phần trực quan để hiển thị các bài hát cá nhân trong danh sách phát.
* Chứa các điều khiển để phát một bài hát và xóa nó khỏi danh sách phát.
* Cung cấp các trình xử lý sự kiện cho các tương tác chuột như di chuột và nhấp chuột vào thành phần bài hát.
* Lớp Program:
* Là lớp chứa phương thức Main(), là điểm khởi đầu chính của ứng dụng.
* Phương thức Main() được đánh dấu bằng thuộc tính [STAThread], chỉ định môi trường luồng đa luồng (multithreaded apartment) cho ứng dụng Windows Forms, cần thiết cho việc sử dụng các thành phần như hộp thoại chọn tệp hoặc hộp thoại chọn thư mục trong môi trường đa luồng.
* Phương thức Main() khởi tạo ứng dụng bằng cách kích hoạt việc sử dụng các phong cách trực quan (visual styles) và thiết lập các cài đặt về hiển thị văn bản mặc định cho ứng dụng.
* Cuối cùng, phương thức Main() khởi chạy một phiên thể hiện của lớp MainDisplay, đại diện cho giao diện người dùng chính của ứng dụng nghe nhạc. Điều này bắt đầu vòng lặp chính của ứng dụng và chạy giao diện người dùng cho đến khi ứng dụng được đóng lại.

A screenshot of a computer

Description automatically generated

Ngoài ra còn 2 class mặc định có sẵn trong winform giúp chúng em có thể chứa các dữ liệu về hình ảnh và cài đặt cần thiết cho chương trình chúng em gồm: Lớp Resources và Lớp Settings

A screenshot of a computer

Description automatically generated

## 2.3. Cài Đặt Lớp

* Lớp Song:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace MusicPlayer

{

    public class Song

    {

        public string filename {  get; set; }

        public string title { get; set; }

        public string artist { get; set; }

        public string album { get; set; }

        public string genre { get; set; }

        public TimeSpan duration { get; set; }

        public Image image { get; set; }

        public Song next { get; set; }

        public Song prev { get; set; }

        public Song()

        {

            filename = null;

            title = null;

            artist = null;

            album = null;

            genre = null;

            duration = TimeSpan.Zero;

            image = null;

            next = null;

            prev = null;

        }

        public Song(string filename, string title, string artist, string album, string genre, TimeSpan duration, Image image)

        {

            this.filename = filename;

            this.title = title;

            this.artist = artist;

            this.album = album;

            this.genre = genre;

            this.duration = duration;

            this.image = image;

            next = null;

            prev = null;

        }

    }

}

* Lớp LinkedList:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using TagLib.Mpeg;

using NAudio.Wave;

using System.IO;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Xml.Linq;

using TagLib.IFD.Entries;

namespace MusicPlayer

{

    internal class LinkedList

    {

        public Song head {  get; set; }

        public void Add(Song newSong)

        {

            if (head == null)

            {

                head = newSong;

            }

            else

            {

                Song current = head;

                while (current.next != null)

                {

                    current = current.next;

                }

                current.next = newSong;

                newSong.prev = current;

            }

        }

        public void Delete(Song delSong)

        {

            if (head == delSong)

            {

                head = delSong.next;

            }

            if (delSong.prev != null)

                delSong.prev.next = delSong.next;

            if (delSong.next != null)

                delSong.next.prev = delSong.prev;

            delSong.prev = null;

            delSong.next = null;

        }

        public bool CheckDuplicate(Song newSong)

        {

            Song current = head;

            while (current != null)

            {

                if (current.filename == newSong.filename)

                {

                    return true;

                }

                current = current.next;

            }

            return false;

        }

        public void GetSongInfo(Song song)

        {

            TagLib.File file = TagLib.File.Create(song.filename);

            song.title = file.Tag.Title ?? "Unknown";

            song.artist = file.Tag.FirstPerformer ?? "Unknown";

            song.genre = file.Tag.FirstGenre ?? "Unknown";

            song.album = file.Tag.Album ?? "Unknown";

            var audioFile = new AudioFileReader(song.filename);

            song.duration = audioFile.TotalTime;

            if (file.Tag.Pictures.Length > 0)

            {

                using (MemoryStream ms = new MemoryStream(file.Tag.Pictures[0].Data.Data))

                {

                    song.image = Image.FromStream(ms);

                }

            }

            else

            {

                song.image = Properties.Resources.\_default;

            }

        }

        public List<Song> ToList(LinkedList ll)

        {

            List<Song> list = new List<Song>();

            Song current = ll.head;

            while(current != null)

            {

                list.Add(current);

                current = current.next;

            }

            return list;

        }

    }

}

* Lớp Player:

using NAudio.Wave;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace MusicPlayer

{

    public class Player

    {

        public WaveOutEvent outputDevice = new WaveOutEvent();

        public AudioFileReader audioFile;

        public Song currentSong { get; set; }

        public MainDisplay mainDisplay;

        public void PlaySong(Song song)

        {

            if (audioFile != null)

            {

                outputDevice.Stop();

            }

            audioFile = new AudioFileReader(song.filename);

            outputDevice.Init(audioFile);

            outputDevice.Play();

        }

        public void Stop()

        {

            outputDevice.Stop();

            currentSong = null;

            audioFile = null;

            outputDevice = null;

        }

        public void Pause()

        {

            outputDevice.Pause();

        }

        public void Continue()

        {

            outputDevice.Play();

        }

        public void Next()

        {

            if(currentSong.next != null)

            {

                currentSong = currentSong.next;

                PlaySong(currentSong);

            }

        }

        public void Previous()

        {

            if (currentSong.prev != null)

            {

                currentSong = currentSong.prev;

                PlaySong(currentSong);

            }

        }

    }

}

* Lớp MainDisplay:

using NAudio.Wave;

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Text;

using System.Linq;

using System.Runtime.Remoting.Metadata.W3cXsd2001;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml.Serialization;

using TagLib.Mpeg;

using TagLib.Riff;

namespace MusicPlayer

{

    public partial class MainDisplay : Form

    {

        public MainDisplay()

        {

            InitializeComponent();

            timer.Interval = 1000;

            timer.Tick += timer1\_Tick;

        }

        private LinkedList ll = new LinkedList();

        public Player player = new Player();

        public Timer timer = new Timer();

        public bool isMuted = false;

        public bool isLoop = false;

        public bool isRandom = false;

        public Random random = new Random();

        public void SetSongInfo()

        {

            lblTitle.Text = player.currentSong.title;

            lblArtist.Text = player.currentSong.artist;

            ptbSongImage.Image = player.currentSong.image;

            if (lblTitle.Size.Width > 150)

                lblTitle.Text = lblTitle.Text.Substring(0, 20) + "...";

            if (lblArtist.Size.Width > 100)

                lblArtist.Text = lblArtist.Text.Substring(0, 15) + "...";

            ptbDetail.Image = player.currentSong.image;

            DetailTitle.Text = "Title: " + player.currentSong.title;

            DetailArtist.Text = "Artist: " + player.currentSong.artist;

            DetailAlbum.Text = "Album: " + player.currentSong.album;

            DetailGenre.Text = "Genre: " + player.currentSong.genre;

            DetailLength.Text = "Length: " + player.currentSong.duration.ToString("m':'ss");

            lblEnd.Text = player.currentSong.duration.ToString("m':'ss");

            MainTitle.Text = player.currentSong.title;

            MainArtist.Text = player.currentSong.artist;

            if (MainTitle.Size.Width > 500)

                MainTitle.Text = MainTitle.Text.Substring(0, 23) + "...";

            ptbMain.Image = player.currentSong.image;

            if (MainTitle.Visible == true)

            {

                Color[] dominantColors = GetDominantColors(player.currentSong.image);

                guna2CustomGradientPanel1.FillColor = dominantColors[0];

                guna2CustomGradientPanel1.FillColor2 = dominantColors[1];

                guna2CustomGradientPanel1.FillColor3 = dominantColors[2];

                guna2CustomGradientPanel1.FillColor4 = dominantColors[3];

            }

        }

        public void Play()

        {

            player.PlaySong(player.currentSong);

            timer.Start();

            btnPlay.Image = Properties.Resources.icons8\_pause\_50;

            SetSongInfo();

            player.outputDevice.Volume = tbVolume.Value / 100f;

        }

        public static Color[] GetDominantColors(Image image)

        {

            using (Bitmap bitmap = new Bitmap(image))

            {

                Bitmap resizedBitmap = new Bitmap(bitmap, new Size(100, 100));

                var colors = new System.Collections.Generic.List<Color>();

                for (int y = 0; y < resizedBitmap.Height; y++)

                {

                    for (int x = 0; x < resizedBitmap.Width; x++)

                    {

                        Color pixelColor = resizedBitmap.GetPixel(x, y);

                        colors.Add(pixelColor);

                    }

                }

                var dominantColors = colors.GroupBy(c => c)

                                           .OrderByDescending(g => g.Count())

                                           .Take(4)

                                           .Select(g => g.Key)

                                           .ToArray();

                return dominantColors;

            }

        }

        #region Event

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

        {

            if (player.outputDevice != null && player.audioFile != null)

            {

                double progressPercentage = tbDuration.Value;

                double timeToSeek = (progressPercentage / 100) \* player.audioFile.TotalTime.TotalSeconds;

                player.audioFile.CurrentTime = TimeSpan.FromSeconds(timeToSeek);

            }

            else

            {

                tbDuration.Value = 0;

            }

        }

        private void tbVolume\_Scroll(object sender, ScrollEventArgs e)

        {

            if (player.outputDevice != null && isMuted == false)

            {

                player.outputDevice.Volume = (float)tbVolume.Value / 100;

            }

        }

        private void timer1\_Tick(object sender, EventArgs e)

        {

            if (player.outputDevice != null && player.audioFile != null)

            {

                double progressPercentage = (player.audioFile.CurrentTime.TotalSeconds / player.audioFile.TotalTime.TotalSeconds) \* 100;

                tbDuration.Value = (int)progressPercentage;

            }

            if (player.outputDevice.PlaybackState == PlaybackState.Stopped || player.audioFile.CurrentTime >= player.audioFile.TotalTime)

            {

                timer.Stop();

                if (!isLoop)

                {

                    if (player.currentSong.next != null)

                    {

                        player.currentSong = player.currentSong.next;

                        player.PlaySong(player.currentSong);

                        timer.Start();

                        SetSongInfo();

                    }

                    else

                    {

                        player.currentSong = ll.head;

                        player.PlaySong(player.currentSong);

                        timer.Start();

                        SetSongInfo();

                        player.Pause();

                        btnPlay.Image = Properties.Resources.icons8\_play\_50;

                    }

                }

                else

                {

                    player.PlaySong(player.currentSong);

                    timer.Start();

                    SetSongInfo();

                }

            }

            TimeSpan currentTime = player.audioFile.CurrentTime;

            lblStart.Text = currentTime.ToString(@"m\:ss");

        }

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

        {

            if (player.outputDevice.PlaybackState == PlaybackState.Playing)

            {

                player.Pause();

                btnPlay.Image = Properties.Resources.icons8\_play\_50;

            }

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

            {

                player.Continue();

                btnPlay.Image = Properties.Resources.icons8\_pause\_50;

            }

        }

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

        {

            if (isRandom)

            {

                List<Song> list = ll.ToList(ll);

                if(list.Count == 1)

                {

                    return;

                }

                int r = random.Next(0, list.Count);

                while (list[r] == player.currentSong)

                {

                    r = random.Next(0, list.Count);

                }

                player.currentSong = list[r];

                player.PlaySong(player.currentSong);

                SetSongInfo();

            }

            else if (player.currentSong != null)

            {

                player.Next();

                SetSongInfo();

            }

        }

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

        {

            if (isRandom)

            {

                List<Song> list = ll.ToList(ll);

                if (list.Count == 1)

                {

                    return;

                }

                int r = random.Next(0, list.Count);

                while (list[r] == player.currentSong)

                {

                    r = random.Next(0, list.Count);

                }

                player.currentSong = list[r];

                player.PlaySong(player.currentSong);

                SetSongInfo();

            }

            else if (player.currentSong != null)

            {

                player.Previous();

                SetSongInfo();

            }

        }

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

        {

            if (isMuted == false)

            {

                player.outputDevice.Volume = 0;

                btnVolumn.Image = Properties.Resources.icons8\_mute\_50;

                isMuted = true;

            }

            else

            {

                player.outputDevice.Volume = (float)tbVolume.Value / 100;

                btnVolumn.Image = Properties.Resources.icons8\_voice\_50;

                isMuted = false;

            }

        }

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

        {

            if (!isLoop)

            {

                btnLoop.Image = Properties.Resources.icons8\_loop\_64\_on;

                isLoop = true;

            }

            else

            {

                btnLoop.Image = Properties.Resources.icons8\_loop\_64;

                isLoop = false;

            }

        }

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

        {

            if (!isRandom)

            {

                btnRandom.Image = Properties.Resources.icons8\_shuffle\_64\_on;

                isRandom = true;

            }

            else

            {

                btnRandom.Image = Properties.Resources.icons8\_shuffle\_64;

                isRandom = false;

            }

        }

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

        {

            OpenFileDialog ofd = new OpenFileDialog();

            ofd.Filter = "Audio Files|\*.mp3;\*.wav;\*.aac|All Files|\*.\*";

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

            {

                Song song = new Song();

                SongComponent songComponent = new SongComponent(this);

                song.filename = ofd.FileName;

                if (ll.CheckDuplicate(song))

                {

                    MessageBox.Show("The song is already in the list", "Duplicate Song", MessageBoxButtons.OK, MessageBoxIcon.Error);

                }

                else

                {

                    ll.GetSongInfo(song);

                    ll.Add(song);

                    songComponent.song = song;

                    songComponent.SetSongInfo(song);

                    flpSong.Controls.Add(songComponent);

                }

            }

        }

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

        {

            btnHome.FillColor = Color.Black;

            btnPlaylist.FillColor = Color.Transparent;

            flpSong.Visible = false;

            btnImport.Visible = false;

            guna2TextBox1.Visible = false;

            label1.Visible = false;

            label2.Visible = false;

            panel2.Visible = false;

            guna2CustomGradientPanel1.Width = 893;

            MainTitle.Visible = true;

            MainArtist.Visible = true;

            ptbMain.Visible = true;

            if (player.currentSong != null)

            {

                Color[] dominantColors = GetDominantColors(player.currentSong.image);

                guna2CustomGradientPanel1.FillColor = dominantColors[0];

                guna2CustomGradientPanel1.FillColor2 = dominantColors[1];

                guna2CustomGradientPanel1.FillColor3 = dominantColors[2];

                guna2CustomGradientPanel1.FillColor4 = dominantColors[3];

            }

        }

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

        {

            btnHome.FillColor = Color.Transparent;

            btnPlaylist.FillColor = Color.Black;

            flpSong.Visible = true;

            btnImport.Visible = true;

            guna2TextBox1.Visible = true;

            label1.Visible = true;

            label2.Visible = true;

            panel2.Visible = true;

            guna2CustomGradientPanel1.Width = 670;

            MainTitle.Visible = false;

            MainArtist.Visible = false;

            ptbMain.Visible = false;

            if (player.currentSong != null)

            {

                guna2CustomGradientPanel1.FillColor = Color.FromArgb(18, 18, 18);

                guna2CustomGradientPanel1.FillColor2 = Color.FromArgb(18, 18, 18);

                guna2CustomGradientPanel1.FillColor3 = Color.FromArgb(18, 18, 18);

                guna2CustomGradientPanel1.FillColor4 = Color.FromArgb(18, 18, 18);

            }

        }

        #endregion

    }

}

* Lớp SongComponent:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MusicPlayer

{

    public partial class SongComponent : UserControl

    {

        public SongComponent(MainDisplay mainDisplay)

        {

            InitializeComponent();

            this.mainDisplay = mainDisplay;

        }

        private MainDisplay mainDisplay;

        private LinkedList ll = new LinkedList();

        public Song song { get; set; }

        public void SetSongInfo(Song song)

        {

            lblTitle.Text = song.title;

            lblArtist.Text = song.artist;

            lblDuration.Text = song.duration.ToString("m':'ss");

            ptbSongImage.Image = song.image;

        }

    #region Event

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

        {

            if (mainDisplay.player.currentSong != song)

            {

                mainDisplay.player.currentSong = song;

                mainDisplay.Play();

            }

        }

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

        {

            if(mainDisplay.player.currentSong == song)

            {

                MessageBox.Show("The song is playing, can't delete", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

            }

            else

            {

                DialogResult result = MessageBox.Show("Delete this song?", "Delete song", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

                if(result == DialogResult.Yes)

                {

                    ll.Delete(song);

                    var flowLayoutPanel = Parent as FlowLayoutPanel;

                    flowLayoutPanel?.Controls.Remove(this);

                }

            }

        }

        private void SongComponent\_MouseEnter(object sender, EventArgs e)

        {

            pnItem.FillColor = Color.FromArgb(50, 50, 50);

            btnPlay.Image = Properties.Resources.icons8\_play\_50;

            btnDelete.Image = Properties.Resources.icons8\_bin\_48;

    }

        private void SongComponent\_MouseLeave(object sender, EventArgs e)

        {

            pnItem.FillColor = Color.Transparent;

            btnPlay.Image = null;

            btnDelete.Image = null;

        }

        private void SongComponent\_Load(object sender, EventArgs e)

        {

            if (lblTitle.Size.Width > 200)

                lblTitle.Text = lblTitle.Text.Substring(0, 30) + "...";

            if (lblArtist.Size.Width > 150)

                lblArtist.Text = lblArtist.Text.Substring(0, 20) + "...";

        }

        #endregion

    }

}

* Lớp Program:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace MusicPlayer

{

    internal static class Program

    {

        /// <summary>

        /// The main entry point for the application.

        /// </summary>

        [STAThread]

        static void Main()

        {

            Application.EnableVisualStyles();

            Application.SetCompatibleTextRenderingDefault(false);

            Application.Run(new MainDisplay());

        }

    }

}

# CHƯƠNG 3. THIẾT KẾ GIAO DIỆN

## 3.1. Giao Diện Menu Chính

A screenshot of a video player

Description automatically generated

## 3.2. Chi Tiết Chức Năng

A white house on a black background

Description automatically generatedNút Home: màn hình to hiển thị bài hát đang được phát kèm theo thông tin cơ bản của một bài hát gồm tên bài hát và nhạc sĩ sáng tác

A black background with white text

Description automatically generatedNút Playlist: màn hình chia thành 2 khung: 1 khung gồm danh sách bài hát được thêm vào, 1 khung gồm các thông tin chi tiết của bài hát gồm tên bài hát, tác giả, thể loại, album,…

Nút Import: giúp người dùng thêm bài hát bất kỳ từ đã có sẵn trên hệ thống người dùng là các file mp3, mp4

A white recycle bin on a black background

Description automatically generatedA white arrow pointing up

Description automatically generatedNút Delete: nằm trên dòng bài hát mục Playlist, giúp người dùng sau khi thêm bài hát vào có thể xóa bài hát theo ý muốn nếu có nhu cầu

Nút PlaySong: giúp phát hoặc dừng phát bài hát đang được nghe

\*Lưu ý: Với nút PlaySong tương tự được hiển thị trong mỗi bài hát tại danh sách các bài hát thì có chức năng khi click vào bài hát nào sẽ phát bài hát đó

A white arrow on a black background

Description automatically generated

Nút Previous: giúp người dùng khi ấn vào sẽ quay lại bài hát trước đó theo thứ tự trong danh sách ban đầu

A white arrow on a black background

Description automatically generated

Nút Next: giúp người dùng khi ấn vào sẽ tiến đến bài hát tiếp theo theo thứ tự trong danh sách ban đầu



Nút Random: giúp người dùng khi ấn vào hệ thống sẽ ngẫu nhiên phát các bài hát theo thứ tự bất kỳ không giống danh sách (lưu ý: bài hát có thể lặp lại)

A white x with arrows

Description automatically generated

Nút Loop: giúp người dùng khi ấn vào sẽ tự động lặp đi lặp lại bài hát đang được phát



Thanh Volumn: giúp người dùng tăng giảm âm lượng của bài hát theo nhu cầu

A green line with a black background

Description automatically generated

Thanh DurationSong: giúp người dùng có thể tua, chỉ định vị trí, hiển thị thời gian của bài hát đang được phát



Thanh Scroll: giúp người dùng có thể quản lý được số lượng bài hát của mình và xem được các bài hát mình thêm vào danh sách nếu có quá nhiều bài hát được thêm vào

# CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ

## 4.1. Các Kết Quả Nhận Được

### 4.1.1. Giao diện khi khởi động chương trình

A screenshot of a video player

Description automatically generated

### 4.1.2. Giao diện khi thêm bài hát vào Playlist

A screenshot of a computer

Description automatically generated

A screenshot of a music player

Description automatically generated

### 4.1.3. Giao diện khi phát/dừng bài hát tại mục Playlist

A screenshot of a music player

Description automatically generated

A screenshot of a music player

Description automatically generated

### 4.1.4. Giao diện khi phát/dừng bài hát tại mục Home

A screenshot of a video

Description automatically generated

A person sitting on a chair

Description automatically generated

### 4.1.5. Giao diện khi xóa bài hát

A screenshot of a video player

Description automatically generated

### 4.1.6. Giao diện khi thêm bài hát trùng

A screenshot of a video player

Description automatically generated

### 4.1.7. Các tính năng khác

Tính năng lặp lại bài hát với nút Loop

Tính năng phát ngẫu nhiên bài hát với nút Random

Tính năng tự động phát bài hát kế tiếp khi bài hát đang phát chạy xong

Tính năng chuyển tới bài hát tiếp theo với nút Next

Tính năng lùi lại bài hát trước đó với nút Previous

Tính năng xem số lượng bài hát trong danh sách Playlist với thanh Scroll

* Đặc biệt khi ấn vào bất kỳ biểu tượng nào sẽ hiện lại viền tròn màu đen bao quanh giúp người dùng dễ dàng nhận biết tính năng đã được hiện thực hay chưa

## 4.2. Một Số Tồn Tại

* Nhóm em chưa phát triển được tính năng tìm kiếm bài hát trong danh sách để người dùng dễ dàng hơn trong việc tìm được bài hát mình cần

A black rectangular frame with blue border

Description automatically generated

* Nhóm em chưa thiết kế được nút Setting và Account sẽ gồm những tính năng gì giúp người dùng hiện thực hóa trải nghiệm

A screenshot of a computer

Description automatically generated

* Với nút Random, nhóm chúng em chưa thiết kế được sao cho các bài hát khi phát ngẫu nhiên sẽ không bị phát lại lần thứ 2
* Khi phóng to hay thu nhỏ màn hình, vị trí các thiết kế sẽ không thay đổi nên nhóm em đã ẩn đi khả năng Maximum màn hình để giúp người dùng có trải nghiệm tốt nhất

A screenshot of a video player

Description automatically generated

* Khi kéo thanh thời gian của bài hát thì vẫn bị tình trạng delay, không mượt mà

## 4.3. Hướng Phát Triển

* Thêm các tính năng như thanh Search để người dùng dễ dàng hơn trong việc tìm được bài hát mình cần
* Thêm tính năng Account để người dùng có thể đăng nhập thông tin và lưu lại những hoạt động mình cần mà không cần phải làm lại với những lần thực hiện sau đó
* Thêm tính năng Setting giúp người dùng
* Thêm mục Favourite giúp người dùng có thể thêm những bài hát yêu thích vào danh sách riêng để nghe
* Cập nhật vị trí mặc định của các thiết kế sao cho khi điều chỉnh kích thước chung của màn hình sẽ không bị thay đổi
* Chuyển đổi các nút Thoát mặc định của Windows sang các nút có thể hiện thực trong Winform qua nút Logout
* Có thể thêm một lúc nhiều file bài hát mà không phải chờ đợi thêm lần lượt từng bài

# PHỤ LỤC

- Link Toàn Bộ Mã Nguồn Trên GitHub

[GitHub - tundut/MusicPlayer](https://github.com/tundut/MusicPlayer?fbclid=IwZXh0bgNhZW0CMTAAAR3KnhwJmgNc_3Z9VE55aCUUo5-ImIQE4sNm8BxAv_BgytFBM6GL7KNgj-U_aem_AXM2NJ798634VynxJbUuElWhETzaelweNyjZnebc0keeG8bKFEj7CewjEFMJqDWvvar1f3pMxDeiLIoB92fyL0rr" \o "Github mã nguồn quản lý Playlist nhạc bằng Doubly LinkedList)

- Hướng Dẫn Cách Cài Đặt Để Chạy

**Với Github:**

**Bước 1**: Copy link dưới đây và lên Google hoặc bất kì trình duyệt Internet nào để cài đặt

[Github quản lý Playlist nhạc bằng Doubly LinkedList](https://github.com/tundut/MusicPlayer?fbclid=IwZXh0bgNhZW0CMTAAAR3KnhwJmgNc_3Z9VE55aCUUo5-ImIQE4sNm8BxAv_BgytFBM6GL7KNgj-U_aem_AXM2NJ798634VynxJbUuElWhETzaelweNyjZnebc0keeG8bKFEj7CewjEFMJqDWvvar1f3pMxDeiLIoB92fyL0rr)

**Bước 2**: Tải Github Desktop để có thể kéo được code trên server về máy mình (trước hết bản thân nên có một tài khoản Github trước và tiến hành đăng nhập các bước ở ứng dụng ấy).

Link tải Github Desktop: <https://desktop.github.com/>

A screenshot of a computer

Description automatically generated

\*Lưu ý: Khi download về phải đăng nhập tài khoản trước hết

**Bước 3**: Tại trang của link ở Bước 1, click “Code” sau đó click “Open with Gihub Desktop”

A screenshot of a computer

Description automatically generated

**Bước 4**: Sau đó sẽ ra giao diện bên dưới, nên tạo một folder mới để lưu trữ file code, sau đó nhấn “Clone”.

A screenshot of a computer

Description automatically generated

**Bước 5**: Sau đó click “Show in Explorer”.

A screenshot of a computer

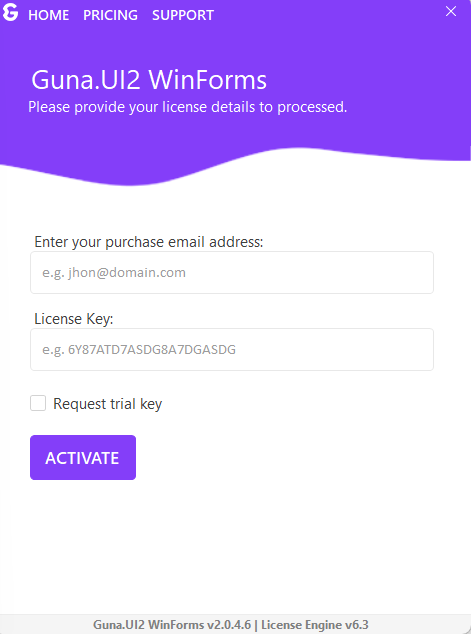
Description automatically generated

**Bước 6**: Sau đó chọn tệp bên dưới là đã có tất cả code của bài.

A screenshot of a computer

Description automatically generated

**Bước 7:** Khi mở chương trình lên và chạy đôi lúc sẽ bị Framework Guna tự động hiện lên tab riêng với mục đích quảng cáo mua thiết kế và sản phẩm của họ, do chúng em dùng free thư viện nên sẽ bỏ qua mục này mà sử dụng những gì sẵn có của Framework. Nếu có hiện, hãy đợi 10s để tắt đi và chương trình sẽ hoạt động như bình thường



- Phân Công Công Việc

|  |  |
| --- | --- |
| **Thành Viên** | **Nhiệm Vụ** |
| Nguyễn Phan Tuấn Đức | * Viết thuật toán cho class LinkedList * Viết thuật toán cho class MainDisplay * Viết thuật toán cho class Player * Định hướng bài làm |
| Nguyễn Hoàng Bảo | * Hỗ trợ code các thuật toán và design giao diện * Test cách cài đặt Github và sử dụng * Tìm hiểu các tư liệu mẫu có sẵn trên mạng để tham khảo cách thiết kế |
| Huỳnh Kim Nguyên | * Làm toàn bộ báo cáo file word * Thiết kế giao diện MainDisplay.Design * Tìm kiếm các nguồn tư liệu hướng dẫn sử dụng framework, icon cho nút và thuật toán của framework |
| Võ Đức Hưng | * Viết thuật toán cho class Song * Viết thuật toán cho class SongComponent * Làm canva để trình chiếu thuyết trình |

**Lưu ý**: nhiệm vụ cần phân công rõ ràng, ko có kiểu là “cùng nhau” làm cái này !

# TÀI LIỆU THAM KHẢO

1. *Chat Blackbox: AI Code Generation, Code Chat, Code Search - Blackbox*. (n.d.). Retrieved April 17, 2024, from https://www.blackbox.ai/

2. *How Kteam - Free Education | How Kteam*. (n.d.). Retrieved April 17, 2024, from https://howkteam.vn/

3. *GitHub*. (n.d.). Retrieved April 17, 2024, from https://github.com/

4. *Demos | Guna Framework - .NET UI/UX Controls and Components for Developers of Desktop, Reporting, Data Visualization*. (n.d.). Retrieved April 17, 2024, from https://gunaui.com/demos/

5. *UI Design Music Player Application - Winform C# Guna Framework - YouTube*. (n.d.). Retrieved April 17, 2024, from https://www.youtube.com/watch?v=ZH8X5zlCj6A

6. *(1) N H C music player C# winform using NAudio lib (demo) - YouTube*. (n.d.). Retrieved April 17, 2024, from https://www.youtube.com/watch?v=JAPLeKB2F54

7. *(1) Music Player in C# Visual Studio By Rohit Programming Zone - YouTube*. (n.d.). Retrieved April 17, 2024, from https://www.youtube.com/watch?v=QJkFfKDhz5o

8. *Implementing taglib# into a c# music player - Stack Overflow*. (n.d.). Retrieved April 17, 2024, from https://stackoverflow.com/questions/20341212/implementing-taglib-into-a-c-sharp-music-player