

Dynamic Music Playlist Management System

A MINI PROJECT REPORT

Submitted by

Kushal Vekariya (RA2211026010164)

Yatharth Patidar (RA2211026010169)

Himavarshith Reddy (RA2211026010170)

for the course

21CSC201J

Data Structures and Algorithms

Under the Guidance of

Dr. Prithi S

Assistant Professor, Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

**FACULTY OF ENGINEERING AND TECHNOLOGY
SCHOOL OF COMPUTING
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR**

NOVEMBER 2023

**SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203**

BONAFIDE CERTIFICATE

Certified that the 21CSC201J Data Structures and Algorithms course project report titled “Dynamic Music Playlist Management System” is the bonafide work done by Kushal Vekariya (RA2211026010164), Yatharth Patidar (RA2211026010169) and Himavarshith Reddy (RA2211026010170) who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE

Dr. Prithi S

DSA– Course Faculty

Assistant Professor

Department of Computational Intelligence

SRM Institute of Science and Technology

Kattankulathur

SIGNATURE

Dr. Annie Uthra R

Head of the Department

Professor

Department of Computational Intelligence

SRM Institute of Science and Technology

Kattankulathur

PROBLEM STATEMENT

In the evolving landscape of digital music consumption, users often face challenges in efficiently organizing, rearranging, and managing their music libraries. To address this, a robust music playlist management system is needed. This system employs a doubly linked list structure, treating each song as a node, to empower users with seamless organization and navigation capabilities.

INTRODUCTION

In the digital era of music consumption, the organization and management of vast music libraries have become increasingly complex. Recognizing the need for a sophisticated solution, this project introduces a Dynamic Music Playlist Management System. The system is designed to empower users with the ability to seamlessly organize, rearrange, and manage their music collections through the implementation of a doubly linked list structure.

Drawing inspiration from the bidirectional nature of a doubly linked list, each song is treated as a node, forming a dynamic sequence that allows users to navigate through their playlists with unparalleled ease. The bidirectional capability ensures smooth traversal both forward and backward, enhancing the overall user experience.

Key features of the system include dynamic playlist sequencing, efficient organization tools, and a user-friendly interface that facilitates intuitive interactions. The bidirectional linked list structure serves as the backbone for dynamic playlist manipulation, enabling users to effortlessly insert, delete, and move songs within their playlists.

To further enrich the user experience, the system incorporates search and filtering mechanisms, cross-platform compatibility, backup and restore functionalities, and customization options. These features collectively contribute to a comprehensive music playlist management experience, catering to the diverse needs and preferences of users across various platforms.

Through this project, we aim to provide a robust solution for music enthusiasts seeking a seamless and interactive means of organizing and navigating through their music libraries. The implementation of the Dynamic Music Playlist Management System represents a significant step towards enhancing the efficiency and user-friendliness of digital music organization in the contemporary landscape.

ABSTRACT

In the ever-evolving landscape of digital music consumption, efficient organization and management of music libraries have become imperative. This project introduces a Dynamic Music Playlist Management System designed to address the challenges users face in navigating, arranging, and curating their extensive music collections. Central to this system is the implementation of a doubly linked list structure, treating each song as a node, providing users with a dynamic and bidirectional sequence for playlist management.

The bidirectional nature of the linked list allows for seamless navigation, enabling users to traverse their playlists both forward and backward effortlessly. Key features include dynamic playlist sequencing, user-friendly interfaces for intuitive interactions, and tools for efficient organization. The system facilitates easy insertion, deletion, and movement of songs within playlists, enhancing the overall user experience.

Beyond playlist manipulation, the system incorporates advanced features such as search and filtering mechanisms, cross-platform compatibility, backup and restore functionalities, and customization options. These features collectively contribute to a comprehensive solution, catering to the diverse needs of users in organizing and personalizing their music libraries.

This project aims to provide a versatile and user-friendly platform, allowing music enthusiasts to curate, organize, and navigate their playlists with unprecedented ease. The Dynamic Music Playlist Management System represents a significant advancement in the realm of digital music organization, contributing to a more seamless and enjoyable music listening experience for users across various platforms.

Data Structures Concepts Used

- 1 Insertion at the End (add_song method):
 - Description: Adds a new song node at the end of the playlist.
- 2 Deletion by Title (remove_song method):
 - Description: Removes a song node with a specified title from the playlist
- 3 Traversal and Display (display_playlist method):
 - Description: Displays the current state of the playlist, visually representing each song node.
- 4 Shuffle Playlist (shuffle_playlist method):
 - Description: Shuffles the order of songs in the playlist.
- 5 Insertion at Next Position (add_next_song method):
 - Description: Adds a new song node after the currently playing song.
- 6 Deletion of Currently Playing Song (delete_current_song method):
 - Description: Deletes the currently playing song from the playlist.

CODE IMPLEMENTATION:

```
import tkinter as tk
from tkinter import ttk, messagebox
import time
import random

class SongNode:
    def __init__(self, number, title):
        self.number = number
        self.title = title
        self.prev = None
        self.next = None

class MusicPlaylistApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Music Playlist Management")
        self.root.geometry("800x400")

        self.playlist_head = None
        self.current_song_number = 1
        self.currently_playing = None

        self.create_ui()

    def create_ui(self):
        # Canvas for visualization
        self.canvas = tk.Canvas(self.root, width=2500, height=400)
        self.canvas.pack(pady=30)

        # Entry fields for song information
        self.title_label = tk.Label(self.root, text="Title:", font=("Helvetica", 14))
        self.title_label.pack(pady=2)
        self.title_entry = ttk.Entry(self.root, style="Padded.TEntry", font=("Helvetica", 15))
        self.title_entry.pack(pady=15)

        # Entry field for song title to remove
        self.remove_title_label = tk.Label(self.root, text="Song Title to Remove:",
font=("Helvetica", 14))
        self.remove_title_label.pack(pady=2)
        self.remove_title_entry = ttk.Entry(self.root, style="Padded.TEntry", font=("Helvetica", 15))
        self.remove_title_entry.pack(pady=15)

        # Buttons for playlist management
```

```

self.button_frame = ttk.Frame(self.root)
self.button_frame.pack(pady=20)

# Insert buttons
insert_end_button = ttk.Button(self.button_frame, text="Add Song to Playlist",
command=self.add_song, style="TButton")
insert_end_button.grid(row=0, column=0, padx=10)

# Delete buttons
delete_button = ttk.Button(self.button_frame, text="Remove Song",
command=self.remove_song, style="TButton")
delete_button.grid(row=0, column=1, padx=10)

# Shuffle button
shuffle_button = ttk.Button(self.button_frame, text="Shuffle Playlist",
command=self.shuffle_playlist, style="TButton")
shuffle_button.grid(row=0, column=2, padx=10)

# Play, Next, and Previous buttons
play_button = ttk.Button(self.button_frame, text="Play", command=self.play_song,
style="TButton")
play_button.grid(row=0, column=3, padx=10)

next_button = ttk.Button(self.button_frame, text="Next", command=self.next_song,
style="TButton")
next_button.grid(row=0, column=4, padx=10)

prev_button = ttk.Button(self.button_frame, text="Previous", command=self.prev_song,
style="TButton")
prev_button.grid(row=0, column=5, padx=10)

# Delete current song and Add next song buttons
delete_current_button = ttk.Button(self.button_frame, text="Delete Current Song",
command=self.delete_current_song, style="TButton")
delete_current_button.grid(row=0, column=6, padx=10)

add_next_button = ttk.Button(self.button_frame, text="Add Next Song",
command=self.add_next_song, style="TButton")
add_next_button.grid(row=0, column=7, padx=10)

# Move current song to next and previous positions
move_next_button = ttk.Button(self.button_frame, text="Move to Next Position",
command=self.move_to_next_position, style="TButton")
move_next_button.grid(row=0, column=8, padx=10)

move_prev_button = ttk.Button(self.button_frame, text="Move to Previous Position",
command=self.move_to_prev_position, style="TButton")

```



```
move_prev_button.grid(row=0, column=9, padx=10)
```

```
def add_song(self):
```

```
    title = self.title_entry.get()
```

```
    if title:
```

```
        new_song = SongNode(self.current_song_number, title)
```

```
        self.current_song_number += 1
```

```
    if not self.playlist_head:
```

```
        self.playlist_head = new_song
```

```
    else:
```

```
        current = self.playlist_head
```

```
        while current.next:
```

```
            current = current.next
```

```
        current.next = new_song
```

```
        new_song.prev = current
```

```
    self.display_playlist()
```

```
def remove_song(self):
```

```
    title_to_remove = self.remove_title_entry.get()
```

```
    if not title_to_remove:
```

```
        messagebox.showerror("Error", "Enter the title of the song to remove.")
```

```
    return
```

```
    current = self.playlist_head
```

```
    while current:
```

```
        if current.title == title_to_remove:
```

```
            if current.prev:
```

```
                current.prev.next = current.next
```

```
            else:
```

```
                self.playlist_head = current.next
```

```
        if current.next:
```

```
            current.next.prev = current.prev
```

```
        self.display_playlist()
```

```
        return
```

```
    current = current.next
```

```
    messagebox.showerror("Error", f"Song with title '{title_to_remove}' not found in the playlist.")
```

```
def shuffle_playlist(self):
```

```
    if self.playlist_head:
```

```
        node_values = []
```

```
        current = self.playlist_head
```

```
        while current:
```

```

        node_values.append(current.title)
        current = current.next

    random.shuffle(node_values)

    current = self.playlist_head
    for value in node_values:
        current.title = value
        current = current.next

    self.display_playlist()

def delete_current_song(self):
    if self.currently_playing:
        current = self.currently_playing
        if current.prev:
            current.prev.next = current.next
        else:
            self.playlist_head = current.next
        if current.next:
            current.next.prev = current.prev
        self.currently_playing = current.next
        self.display_playlist()

def add_next_song(self):
    if self.currently_playing:
        title = self.title_entry.get()
        if title:
            new_song = SongNode(self.current_song_number, title)
            self.current_song_number += 1

            current = self.currently_playing
            new_song.next = current.next
            new_song.prev = current
            if current.next:
                current.next.prev = new_song
            current.next = new_song

            self.display_playlist()

def play_song(self):
    if self.playlist_head:
        self.currently_playing = self.playlist_head
        self.display_playlist()

def next_song(self):
    if self.currently_playing and self.currently_playing.next:

```

```

        self.currently_playing = self.currently_playing.next
        self.display_playlist()

def prev_song(self):
    if self.currently_playing and self.currently_playing.prev:
        self.currently_playing = self.currently_playing.prev
        self.display_playlist()

def move_to_next_position(self):
    if self.currently_playing and self.currently_playing.next:
        current = self.currently_playing
        next_node = current.next
        if next_node.next:
            current.next = next_node.next
            next_node.next.prev = current
            next_node.prev = current.prev
            current.prev.next = next_node
            next_node.next = current
            current.prev = next_node
        self.display_playlist()

def move_to_prev_position(self):
    if self.currently_playing and self.currently_playing.prev:
        current = self.currently_playing
        prev_node = current.prev
        if prev_node.prev:
            current.prev = prev_node.prev
            prev_node.prev.next = current
            prev_node.next = current.next
            current.next.prev = prev_node
            prev_node.prev = current
            current.next = prev_node
        self.display_playlist()

def display_playlist(self):
    self.canvas.delete("all")
    current = self.playlist_head
    temp_x = 50

    while current:
        fill_color = "yellow" if current == self.currently_playing else "#3498db"
        text_color = "black" if current == self.currently_playing else "white"

        self.canvas.create_rectangle(temp_x, 150, temp_x + 200, 200, fill=fill_color)

        self.canvas.create_text(temp_x + 30, 175, text=f"Prev: {current.prev.number}" if
current.prev else "Prev: null", font=("Helvetica", 9), fill=text_color)

```

```
        self.canvas.create_text(temp_x + 100, 175, text=f"{current.number}. {current.title}",
font=("Helvetica", 12), fill=text_color)
```

```
        self.canvas.create_text(temp_x + 170, 175, text=f"Next: {current.next.number}" if
current.next else "Next: null", font=("Helvetica", 9), fill=text_color)
```

```
    if current.next:
```

```
        self.canvas.create_line(temp_x + 200, 175, temp_x + 270, 175, arrow=tk.LAST)
```

```
        self.canvas.create_line(temp_x + 58, 150, temp_x + 58, 200, fill="black") # Line between
prev and title
```

```
        self.canvas.create_line(temp_x + 140, 150, temp_x + 140, 200, fill="black") # Line
between title and next
```

```
        temp_x += 270
```

```
        current = current.next
```

```
        self.root.update()
```

```
        time.sleep(0.2)
```

```
if __name__ == "__main__":
```

```
    root = tk.Tk()
```

```
    app = MusicPlaylistApp(root)
```

```
    root.mainloop()
```

Output:

Music Playlist Management

Prev: null | 1. Perfect | Next: null

Title:

Song Title to Remove:

Music Playlist Management

Prev: null | 1. Intenstions | Next: 3 → Prev: 1 | 3. Sunflower | Next: 4 → Prev: 3 | 4. Perfect | Next: null

Title:

Song Title to Remove:

Music Playlist Management

Prev: null

1. Perfect

Next: 3

→

Prev: 1

3. Sunflower

Next: 4

→

Prev: 3

4. Intenstions

Next: null

Title:

Intenstions

Song Title to Remove:

Add Song to Playlist

Remove Song

Shuffle Playlist

Play

Next

Previous

Delete Current Song

Add Next Song

Move to Next Position

Move to Previous Position

Conclusion:

In conclusion, the Dynamic Music Playlist Management System has been successfully developed to address the challenges associated with organizing and managing digital music libraries. This project leverages a doubly linked list data structure to create a versatile and interactive platform for users to curate, navigate, and manipulate their playlists effectively.

The implementation encompasses key features such as dynamic playlist sequencing, bidirectional navigation, and user-friendly interfaces, providing users with the ability to seamlessly add, remove, shuffle, and play songs within their playlists. The project also introduces functionalities like deleting the current song, adding the next song, and moving songs to different positions in the playlist, enhancing the overall user experience.

The visual representation of the playlist through the Tkinter Canvas offers users a clear and intuitive view of the playlist's current state, including the song being played. The project strives to balance functionality with an aesthetically pleasing interface, promoting a user-friendly and engaging experience.

The adoption of additional features such as search and filtering mechanisms, cross-platform compatibility, and backup and restore functionalities could further enhance the system's robustness and user appeal in future iterations.