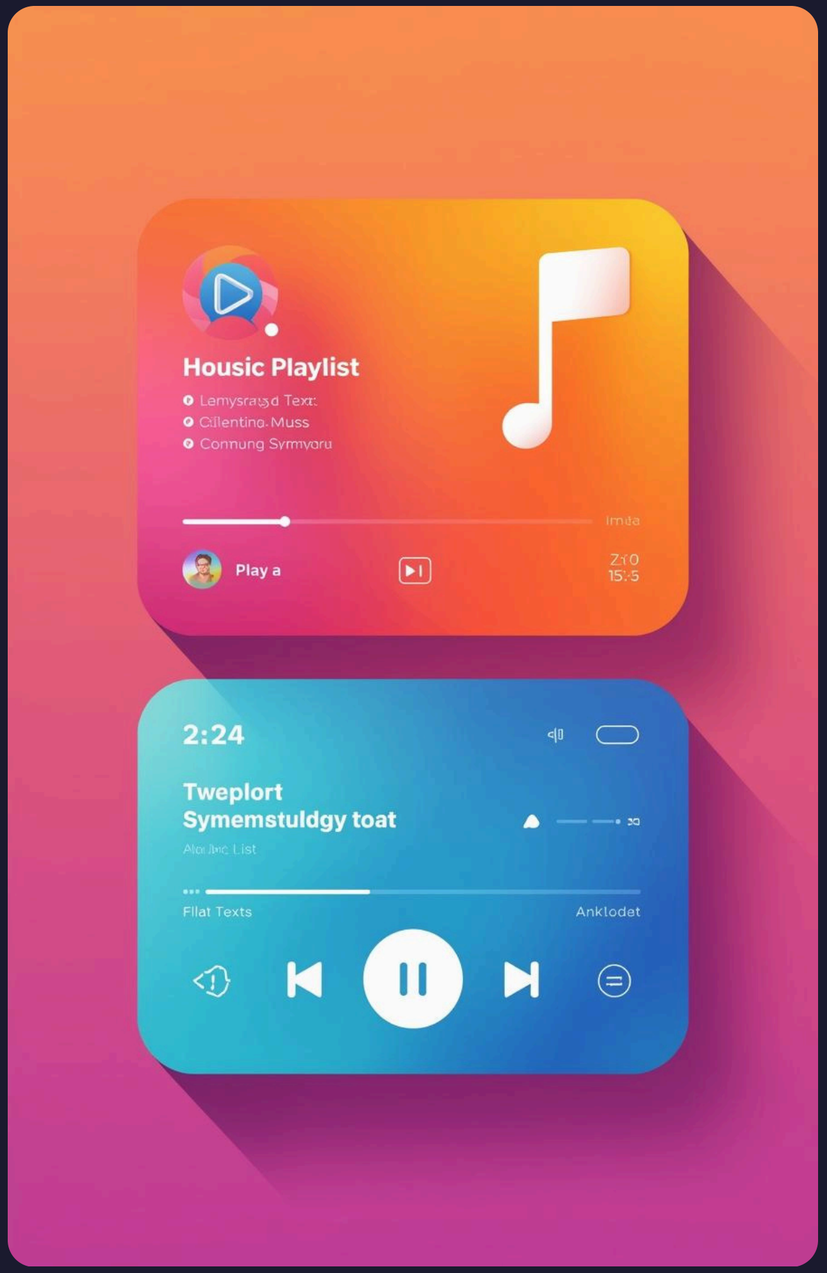
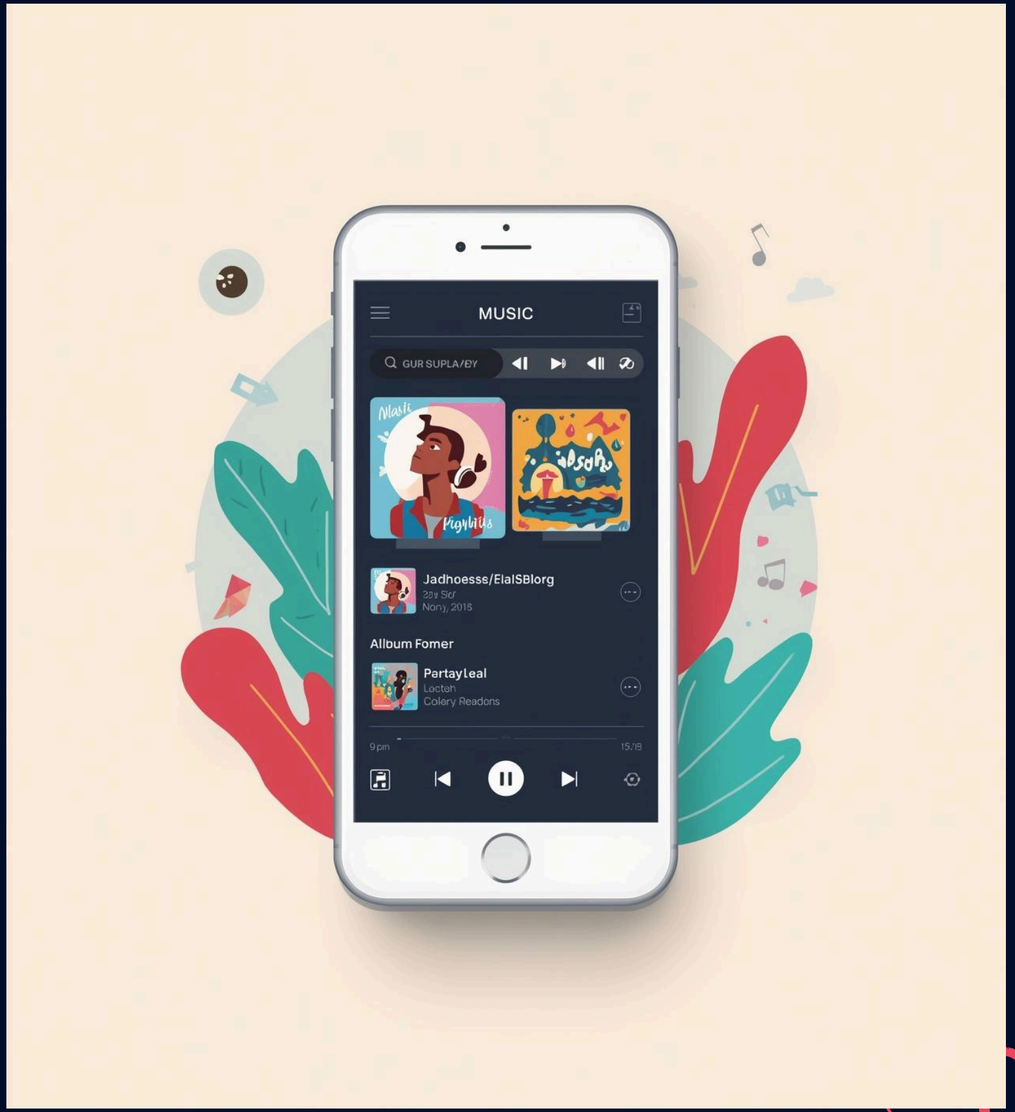
Music Playlist Simulator



What is Music Playlist Simulator?

This project simulates a real-world music playlistusing a Doubly Linked List (DLL). It demonstrateshow dynamic data structures can manage songsefficiently through forward and backward navigation.



Introduction

This project implements a console-based Music Playlist System using a Doubly Linked List.
It highlights:

- How linked lists manage dynamic data
- Real-world application of DLL in multimedia systems
- Efficient navigation using next and previous pointers
- Menu-driven interface for smooth user interaction

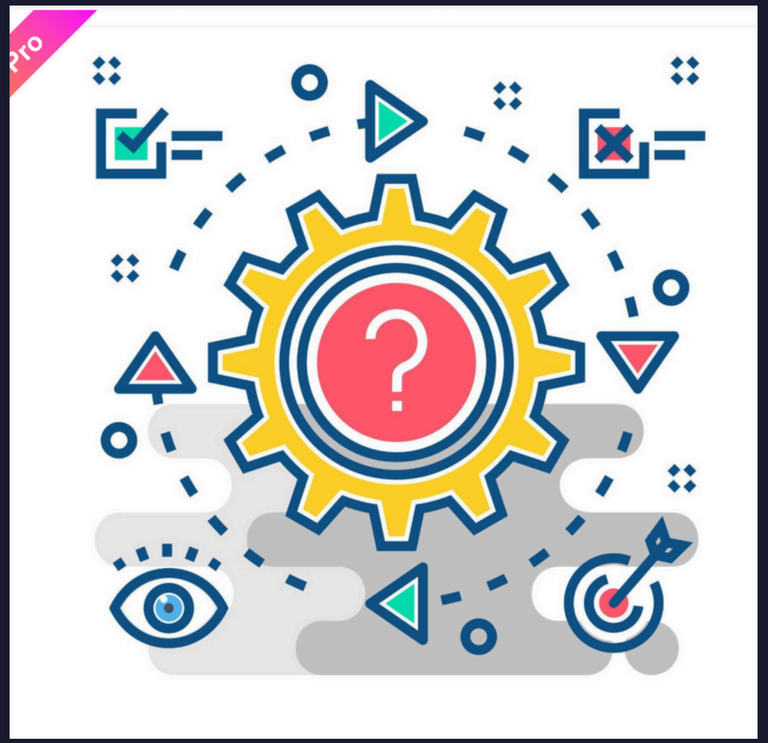
The project is simple, interactive, and demonstrates core DS concepts.

Problem Statement

The goal is to design a playlist system that:

- Stores songs dynamically
- Allows adding and removing songs easily
- Supports forward and backward song navigation
- Provides searching functionality
- Simulates real music app playlist features

The solution must use a Doubly Linked List for storing songs.



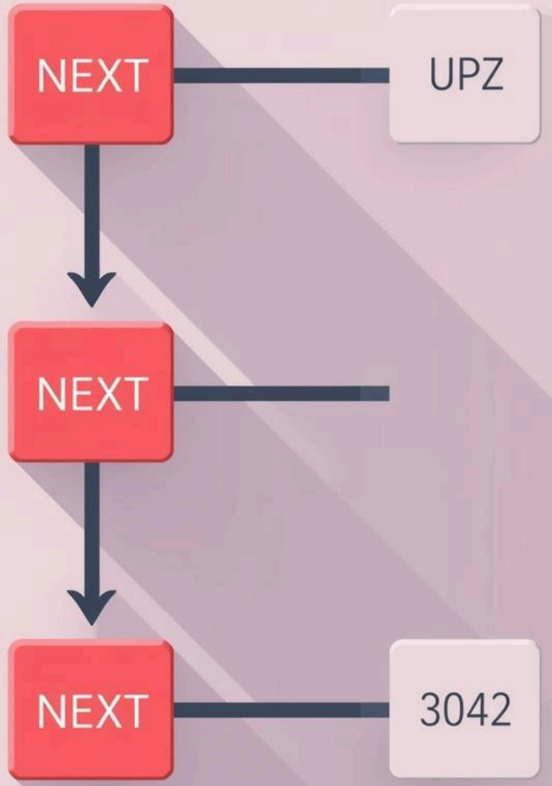
Understanding the Doubly Linked List

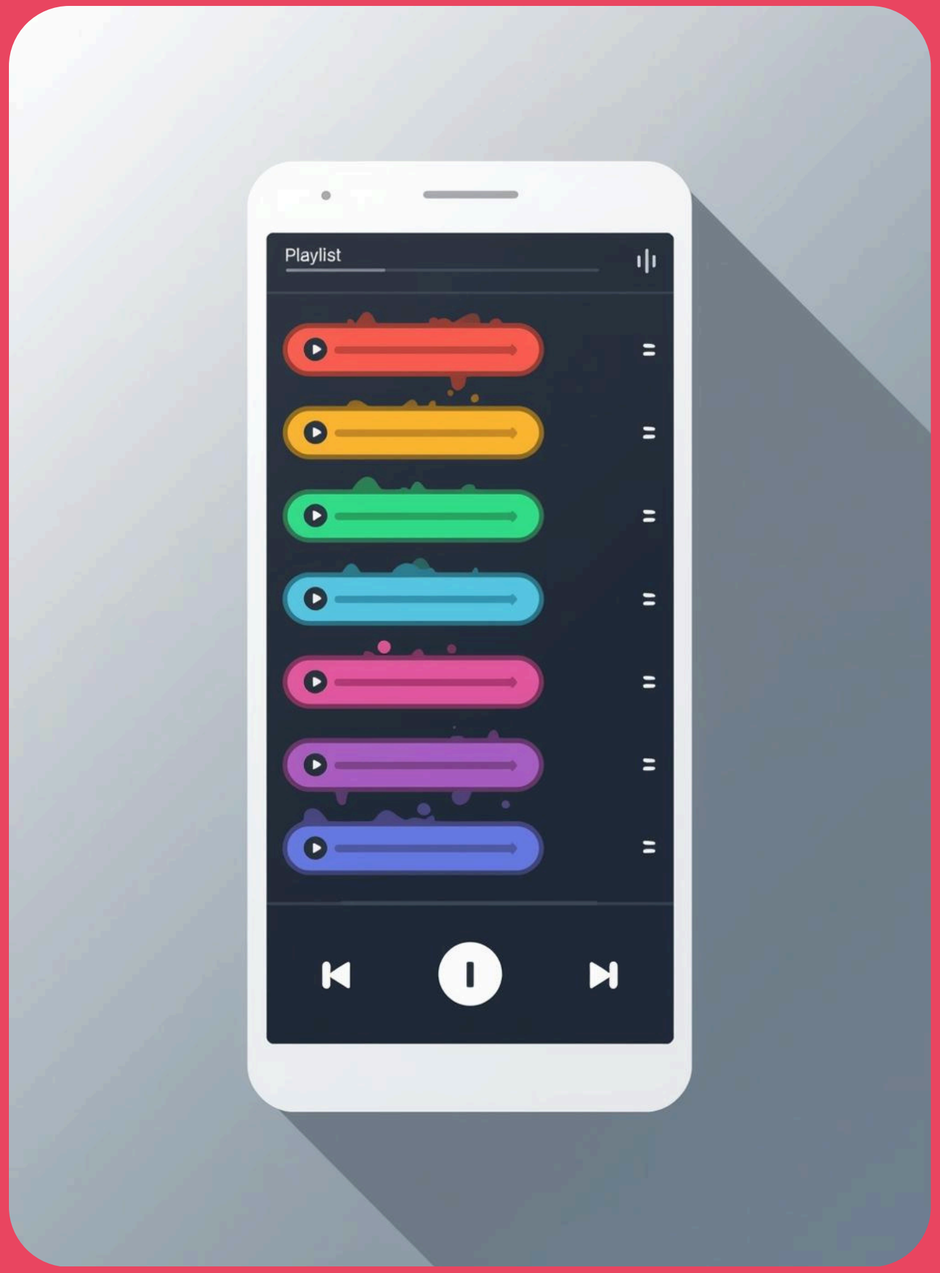
A Doubly Linked List is a dynamic data structure where each node contains:

- Data (song name)
- Pointer to next node
- Pointer to previous node

Key features:

- Supports bidirectional traversal
- Efficient insertion and deletion
- Ideal for playlists and navigation-based systems





WHY DLL FOR PLAYLIST SIMULATION?

A playlist requires:

- Next song → next pointer
- Previous song → prev pointer
- Fast insertion (adding songs anywhere)
- Fast deletion (removing songs easily)

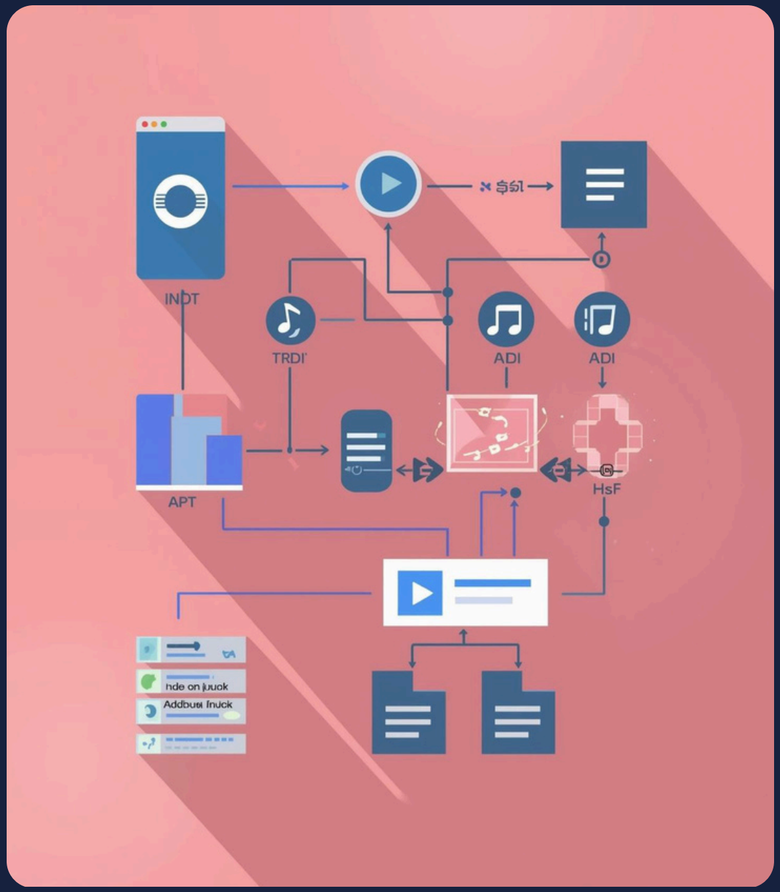
DLL provides:

- Forward and backward movement
- Minimal shifting of elements
- Efficient node-level modification

Hence, DLL is the best structure for playlist simulation.

System Architecture Overview

The system consists of three layers:
1. User Interaction Layer
• Menu-driven interface
• Takes user input such as song names
2. Core Playlist (DLL) Module
• Stores songs in DLL nodes
• Functions: addSong, deleteSong, searchSong, displayForward, displayBackward
3. Output Layer
• Displays playlist results
• Shows search and deletion outcomes
This layered design ensures modularity and clarity.



Operations Implemented

Key Functions for Playlist Management

1.Add Song

- Adds a new song to the playlist
- Creates a new DLL node

2.Remove Song

- Removes a selected song
- Adjusts prev and next pointers

3.Display Playlist (Forward)

- Traverses from head to tail

4.Display Playlist (Backward)

- Traverses from tail to head

5.Search Song

- Finds a song and displays its position

These operations closely resemble behavior in real music apps.

Doubly Linked List Song Management

Code Structure

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
char song[100];
struct Node* next;
struct Node* prev;
} Node;
Node* head = NULL;
Node* tail = NULL;
Node* createNode(char song[]) {
Node* newNode =
(Node*)malloc(sizeof(Node));
strcpy(newNode->song, song);
newNode->next = NULL;
newNode->prev = NULL;
return newNode;
}
void addSong() {
char song[100];
printf("Enter song name to add: ");
scanf(" %[^\n]", song);
Node* newNode = createNode(song);
if (head == NULL) {
head = tail = newNode;
printf("Song added to playlist.\n");
return;
}
tail->next = newNode;
newNode->prev = tail;
tail = newNode;
printf("Song added to playlist.\n");
}
void deleteSong() {
char target[100];
printf("Enter song name to delete: ");
scanf(" %[^\n]", target);
if (head == NULL) {
printf("Playlist is empty!\n");
return;
}
Node* temp = head;
while (temp != NULL && strcmp(temp->song, target) != 0) {
temp = temp->next;
}
if (temp == NULL) {
printf("Song not found!\n");
return;
}
if (temp == head && temp == tail) {
head = tail = NULL;
}
else if (temp == head) {
head = head->next;
head->prev = NULL;
}
else if (temp == tail) {
tail = tail->prev;
tail->next = NULL;
}
else {
temp->prev->next = temp->next;
temp->next->prev = temp->prev;
}
free(temp);
printf("Song deleted successfully.\n");
}
void displayForward() {
if (head == NULL) {
printf("Playlist is empty!\n");
return;
}
Node* temp = head;
printf("\nPlaylist (Forward):\n");
while (temp != NULL) {
printf("-> %s\n", temp->song);
temp = temp->next;
}
}
void displayBackward() {
if (tail == NULL) {
printf("Playlist is empty!\n");
return;
}
Node* temp = tail;
printf("\nPlaylist (Backward):\n");
while (temp != NULL) {
printf("-> %s\n", temp->song);
temp = temp->prev;
}
}
void searchSong() {
char target[100];
printf("Enter song name to search: ");
scanf(" %[^\n]", target);
Node* temp = head;
int pos = 1;
while (temp != NULL) {
if (strcmp(temp->song, target) == 0) {
printf("Song found at position: %d\n", pos);
return;
}
temp = temp->next;
pos++;
}
printf("Song not found!\n");
}
int main() {
int choice;
while (1) {
printf("\n\n===== MUSIC PLAYLIST MENU =====\n");
printf("1. Add Song\n");
printf("2. Delete Song\n");
printf("3. Display Playlist (Forward)\n");
printf("4. Display Playlist (Backward)\n");

printf("5. Search Song\n");

printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
addSong();
break;
case 2:
deleteSong();
break;
case 3:
displayForward();
break;
case 4:
displayBackward();
break
case 5:
searchSong();
break;
case 6:
printf("Exiting program...\n");
exit(0);
default:
printf("Invalid choice! Try again.\n");
}
}
return 0;
}
```

Sample Output

Output

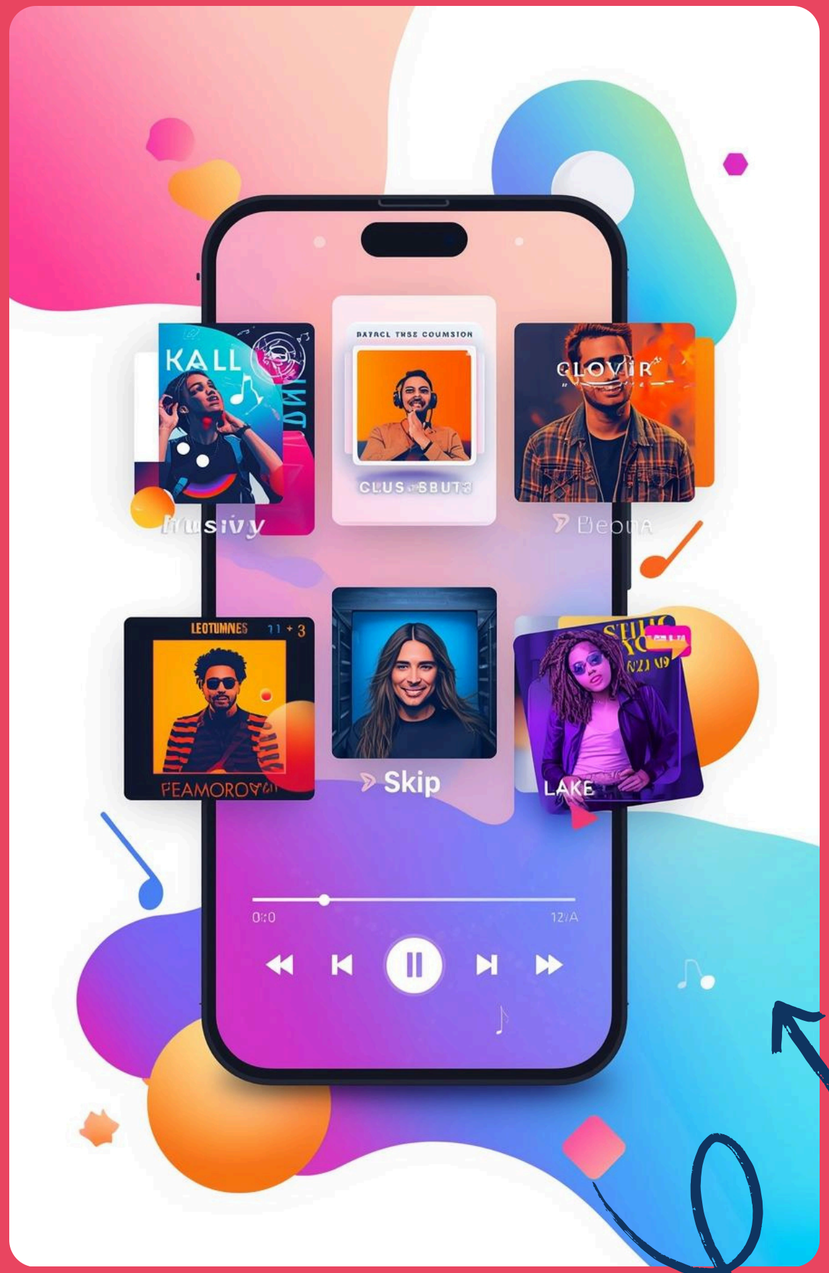
```
===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 1
Enter song name to add: Perfect
Song added to playlist.


===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 1
Enter song name to add: Memories
Song added to playlist.
```

Output

```
===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 3

Playlist (Forward):
-> Perfect
-> Memories


===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 4

Playlist (Backward):
-> Memories
-> Perfect
```

Output

```
===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 5
Enter song name to search: Perfect
Song found at position: 1


===== MUSIC PLAYLIST MENU =====
1. Add Song
2. Delete Song
3. Display Playlist (Forward)
4. Display Playlist (Backward)
5. Search Song
6. Exit
Enter your choice: 2
Enter song name to delete: Memories
Song deleted successfully.
```



Future Enhancements

Add a Possible upgrades:

- Add shuffle mode
- Add repeat mode
- Save playlist to file and reload it
- GUI-based player interface
- Song categorization (artist, album)
- Sorting songs alphabetically


THANK YOU