**MUSIC PLAYLIST MANAGER**

MINOR PROJECT REPORT

By

**R ASHWIN (RA2211026010387)**

**K B KRISHNA PRAVIN (RA2211026010364)**

**KAVIN BHARATHI(RA2211026010380)**

Under the guidance of

**Dr.A.Maheshwari**

*In partial fulfilment for the Course*

of

**21CSC201J – DATA STRUCTURES AND ALGORITHM**

**in CINTEL**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

**(Under Section 3 of UGC Act, 1956)**

## BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC201J – DATA STRUCTURES AND ALGORITHM** entitled in

"**MUSIC PLAYLIST MANAGER**" is the bonafide work of

**R ASHWIN (RA2211026010387),**

**K B KRISHNA PRAVIN(RA2211026010364)**

**and KAVIN BHARATHI(RA2211026010380)**

who carried out the work under my supervision.

### PROJECT GUIDE HEAD OF THE DEPARTMENT

Dr.A.Maheshwari Dr Annie Uthra

**Assistant Professor Professor & Head**

**Department of Computational Intelligence Department of Computational Intelligence**

SRM Institute of Science and Technology SRM Institute of Science and Technology

Kattankulathur Kattankulathur

# ABSTRACT

The Music Playlist Manager with DSA is a C-based software application that leverages data structures and algorithms to provide users with an efficient tool for managing music playlists and creating queues and other methodologies.

It offers advanced functionalities to organize, shuffle, remove and manipulate music collections in a playlists while optimizing performance.

Additionally, a priority queue is employed to manage a queue of songs to be played next, ensuring a seamless listening experience. The application employs sorting algorithms to enable various playback options, including sorting by artist, title.

## ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C.**

**MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our my Course project Faculty Dr.Maivizhi R**, Associate Professor , Department of Computational Intelligence,** forherassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr.Annie Uthra, Professor and Head, Department of Computational Intelligence** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who

directly and indirectly contributed to the successful completion of our project.

## INTRODUCTION

A Music Playlist Manager implemented in C, leveraging Data Structures and Algorithms (DSA) concepts, serves as a powerful tool for music enthusiasts to efficiently organize and manage their song collections in a playlist. This program demonstrates the application of DSA principles for an enhanced user experience in a compact, simple and efficient manner.

The key features of the Music Playlist Manager using DSA concepts are as follows:

**1. Data Structures:**

Linked Lists: Playlists and song queues are implemented as linked lists. This data structure allows for dynamic memory allocation and easy insertion, removal, and reordering of songs in the playlists.

Arrays: To facilitate the Fisher-Yates shuffle algorithm for playlist shuffling, arrays are employed to store references to songs.

**2. Song and Playlist Management:**

Create Playlists: Users can create and name playlists, and each playlist is represented as a linked list of songs.

Add and Remove Songs: Songs are added to or removed from playlists with efficient operations on linked lists, allowing for easy customization.

Queue System: A song queue is implemented as a separate linked list to manage the order of songs to play next, enhancing the playback experience.

**3. Shuffle Algorithm:**

Fisher-Yates Shuffle: The shuffle function utilizes the Fisher-Yates shuffle algorithm, implemented using arrays, to randomize the order of songs within a playlist efficiently.

**4 Display:**

View Queue: The program offers a simple display of the song queue through a linked list traversal.

**5. Menu-Driven Interface:**

Menu System: A menu-driven interface, utilizing a switch-case construct, enables users to create, edit, shuffle, remove, and play playlists seamlessly.

**6. Efficiency and Optimization:**

Time Complexity: DSA concepts are applied to optimize time complexities, ensuring efficient playlist and queue operations.

Memory Management: Efficient memory allocation and deallocation techniques are employed for song and playlist creation.

**7. User Interaction:**

User Input Handling: The program captures user choices through input functions, with error handling to ensure robust user interaction.

**8. Scalability and Robustness:**

Playlist Limit: The program sets a limit of 99 characters, enhancing robustness and resource management.

**9. Playback:**

Playback of songs is achieved by playing songs in a chosen order, playing the most recently added songs first, or shuffling the playlist as per the user's choice.

In summary, the Music Playlist Manager in C demonstrates how DSA concepts, including linked lists, arrays, and algorithmic efficiency, can be applied to create a user-friendly and efficient tool for managing music playlists. The program efficiently handles data structures and algorithms to provide a seamless user experience, allowing users to create, customize, shuffle, and enjoy their music collections with ease. It also incorporates error handling and memory management for robust performance

## Hardware and Software Requirements

**Hardware Requirements:**

A computer or device capable of running C programs. This could be a desktop computer, laptop, or a microcontroller with a C compiler.

**Software Requirements:**

A C Compiler: You need a C compiler to compile and execute C programs. Common C compilers include:

GCC (GNU Compiler Collection): Available for Linux, macOS, and Windows. You can install it on your system.

Microsoft Visual C/C++: Available for Windows.

Clang: Another open-source C/C++ compiler available for multiple platforms.

An Integrated Development Environment (IDE) (optional): While not strictly necessary, an IDE can make the development process easier. Some popular C/C++ IDEs include Visual Studio, Code::Blocks, CLion, and more.

An Operating System: The program should work on various operating systems like Linux, macOS, and Windows. The choice of the operating system depends on your preference and the C compiler you choose.

Standard C Libraries: The program relies on standard C libraries for functions like file I/O, string manipulation, and dynamic memory allocation. These libraries are typically included with the C compiler.

## PROPOSED SYSTEM

Proposed system is an is a simple command-line music playlist management system. that allows users to create playlists, insert songs, shuffle playlists, remove songs, and play songs in different orders. It also includes a new feature to add songs to a queue to play next and view the queue. Here are some advantages of the proposed system:

Organized Playlist Management: Users can create and manage playlists efficiently. The program supports multiple playlists, enabling users to categorize their music.

Dynamic Playlist Manipulation: Users can easily insert and remove songs from the playlist. The program allows for dynamic changes to the playlist content.

Shuffle Feature: The system provides a shuffle feature, allowing users to enjoy their music in a randomized order, providing a fresh listening experience.

Queue Functionality: The addition of a queue feature is advantageous. Users can prioritize and manage the order in which songs are played, ensuring a seamless and customized listening experience.

User-Friendly Menu: The menu-driven interface makes it easy for users to interact with the program, even without deep technical knowledge.

Playlist Play Options: Users have the flexibility to choose different play orders - from playing the latest added songs to the first added songs or a shuffled playlist.

Error Handling: The program includes error handling by checking for valid input and providing appropriate feedback when users enter invalid choices or if the playlist is empty.

Memory Management: The program appropriately manages memory by using dynamic allocation for song and playlist structures and freeing memory when removing songs.

Modular Design: The program uses a modular design with functions for various operations, making it easier to maintain and extend.

Randomization: The shuffle algorithm used is based on the Fisher-Yates shuffle, a well-known and effective algorithm for randomization.

Limitations Handling: The program handles limitations gracefully, such as the maximum number of playlists (10).

Data Persistence: Although not implemented in the code, the system can be extended to include file I/O for storing and loading playlists, providing data persistence.

Clean Code: The code is structured, well-commented, and easy to understand, which is beneficial for maintenance and future development.

Customizable: Users can personalize their music experience by creating and managing multiple playlists and controlling the order of songs.

Queue View: Users can easily view the songs in the queue, helping them plan their upcoming listening experience.

All the manual difficulties in managing a Bank have been rectified by implementing computerization.

# IMPLEMENTATION AND OUTPUT

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

//GROUP 14-387,380,364

//ADDING COMMENTS FOR USER UNDERSTANDING

// Define a structure for a song

struct Song {

char title[100];

char artist[100];

struct Song \*next;

};

// Define a structure for a playlist

struct Playlist {

char name[100];

struct Song \*head;

struct Song \*queue; // Add a queue pointer for the queue of songs to play next

};

// Function to create a new playlist

struct Playlist \*createPlaylist(const char \*name) {

struct Playlist \*newPlaylist = (struct Playlist \*)malloc(sizeof(struct Playlist));

if (newPlaylist) {

strcpy(newPlaylist->name, name);

newPlaylist->head = NULL;

newPlaylist->queue = NULL; // Initialize the queue member

}

return newPlaylist;

}

struct Song \*createSong(const char \*title, const char \*artist) {

struct Song \*newSong = (struct Song \*)malloc(sizeof(struct Song));

if (newSong) {

strcpy(newSong->title, title);

strcpy(newSong->artist, artist);

newSong->next = NULL;

}

return newSong;

}

// Function to insert a song into a playlist

void insertSong(struct Playlist \*playlist, const char \*title, const char \*artist) {

struct Song \*newSong = createSong(title, artist);

if (newSong) {

if (playlist->head == NULL) {

playlist->head = newSong;

} else {

struct Song \*current = playlist->head;

while (current->next) {

current = current->next;

}

current->next = newSong;

}

}

}

// Function to add a song to the queue to play next

void addToQueue(struct Playlist \*playlist, const char \*title, const char \*artist) {

struct Song \*newSong = createSong(title, artist);

if (newSong) {

if (playlist->queue == NULL) {

playlist->queue = newSong;

}

else {

struct Song \*current = playlist->queue;

while (current->next) {

current = current->next;

}

current->next = newSong;

}

}

}

// Function to view the queue

void viewQueue(struct Playlist \*playlist) {

if (playlist->queue == NULL) {

printf("Queue is empty.\n");

return;

}

struct Song \*current = playlist->queue;

printf("Queue:\n");

while (current) {

printf("%s - %s\n", current->title, current->artist);

current = current->next;

}

}

// Function to shuffle a playlist

void shufflePlaylist(struct Playlist \*playlist) {

struct Song \*songs[100];

int count = 0;

struct Song \*current = playlist->head;

while (current) {

songs[count++] = current;

current = current->next;

}

srand(time(NULL));

for (int i = count - 1; i > 0; i--) {

int j = rand() % (i + 1);

struct Song \*temp = songs[i];

songs[i] = songs[j];

songs[j] = temp;

}

playlist->head = songs[0];

for (int i = 0; i < count - 1; i++) {

songs[i]->next = songs[i + 1];

}

songs[count - 1]->next = NULL;

}

// Function to remove a song from a playlist

void removeSong(struct Playlist \*playlist, const char \*title) {

struct Song \*current = playlist->head;

struct Song \*prev = NULL;

while (current) {

if (strcmp(current->title, title) == 0) {

if (prev == NULL) {

playlist->head = current->next;

} else {

prev->next = current->next;

}

free(current);

printf("%s removed from the playlist.\n", title);

return;

}

prev = current;

current = current->next;

}

printf("%s not found in the playlist.\n", title);

}

// Function to play songs in different orders

void playPlaylist(struct Playlist \*playlist, int order) {

if (playlist->head == NULL) {

printf("Playlist is empty.\n");

return;

}

if (order == 0) {

// added first

struct Song \*current = playlist->head;

while (current) {

printf("Playing: %s - %s\n", current->title, current->artist);

current = current->next;

}

} else if (order == 1) {

// latest added

struct Song \*prev = NULL;

struct Song \*current = playlist->head;

struct Song \*next = NULL;

// Reverse the order of songs in the playlist

while (current) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

// Now, 'prev' points to the first song, which was the last one added

current = prev;

while (current) {

printf("Playing: %s - %s\n", current->title, current->artist);

current = current->next;

}

} else if (order == 2) {

// Shuffle

shufflePlaylist(playlist);

playPlaylist(playlist, 0); // Play shuffled playlist using order 0 (latest added)

}

}

int main() {

int choice;

struct Playlist playlists[10];

int playlistCount = 0;

int currentPlaylist = -1;

while (1) {

printf("\nMenu:\n");

printf("1. Create a new playlist\n");

printf("2. Insert a song into the current playlist\n");

printf("3. Shuffle the current playlist\n");

printf("4. Remove a song from the current playlist\n");

printf("5. Play the current playlist (Added First)\n");

printf("6. Play the current playlist (Latest Added)\n");

printf("7. Play the current playlist (Shuffle)\n");

printf("8. Add a song to the queue to play next\n"); // New option

printf("9. View the queue\n"); // New option

printf("10. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

// Inside the main function

// Inside the main function

if (choice == 1) {

if (playlistCount < 10) {

playlistCount++;

currentPlaylist = playlistCount - 1;

printf("Enter the name of the new playlist: ");

scanf("%s", playlists[currentPlaylist].name);

// Use a pointer to store the result of createPlaylist

struct Playlist \*newPlaylist = createPlaylist(playlists[currentPlaylist].name);

// Assign the newPlaylist pointer to the current playlist

playlists[currentPlaylist] = \*newPlaylist;

// Remember to free the newPlaylist pointer when it's no longer needed

free(newPlaylist);

} else {

printf("You have reached the maximum limit of playlists.\n");

}

}

else if (choice == 2) {

if (currentPlaylist != -1) {

char title[100], artist[100];

printf("Enter the song title: ");

scanf("%s", title);

printf("Enter the artist name: ");

scanf("%s", artist);

insertSong(&playlists[currentPlaylist], title, artist);

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 3) {

if (currentPlaylist != -1) {

shufflePlaylist(&playlists[currentPlaylist]);

printf("Playlist shuffled.\n");

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 4) {

if (currentPlaylist != -1) {

char title[100];

printf("Enter the song title to remove: ");

scanf("%s", title);

removeSong(&playlists[currentPlaylist], title);

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 5) {

if (currentPlaylist != -1) {

playPlaylist(&playlists[currentPlaylist], 0); // Play in added first order

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 6) {

if (currentPlaylist != -1) {

playPlaylist(&playlists[currentPlaylist], 1); // Play in latest added order

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 7) {

if (currentPlaylist != -1) {

playPlaylist(&playlists[currentPlaylist], 2); // Play shuffled playlist

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 8) {

if (currentPlaylist != -1) {

char title[100], artist[100];

printf("Enter the song title: ");

scanf("%99s", title);

printf("Enter the artist name: ");

scanf("%99s", artist);

addToQueue(&playlists[currentPlaylist], title, artist);

printf("Song added to the queue.\n");

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 9) {

if (currentPlaylist != -1) {

viewQueue(&playlists[currentPlaylist]);

} else {

printf("Please create a playlist first.\n");

}

} else if (choice == 10) {

exit(0);

} else {

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

**OUTPUT:**

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 1

Enter the name of the new playlist: a

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 2

Enter the song title: b

Enter the artist name: b

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 2

Enter the song title: c

Enter the artist name: c

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 2

Enter the song title: d

Enter the artist name: d

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 4

Enter the song title to remove: d

d removed from the playlist.

Menu:1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 5

Playing: b - b

Playing: c - c

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 6

Playing: c - c

Playing: b - b

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 7

Playing: b - b

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 8

Enter the song title: e

Enter the artist name: e

Song added to the queue.

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 9

Queue:

e - e

Menu:

1. Create a new playlist

2. Insert a song into the current playlist

3. Shuffle the current playlist

4. Remove a song from the current playlist

5. Play the current playlist (Added First)

6. Play the current playlist (Latest Added)

7. Play the current playlist (Shuffle)

8. Add a song to the queue to play next

9. View the queue

10. Exit

Enter your choice: 10

**CONCLUSION**

In conclusion, this Java-based “Transaction Backend API” is a comprehensive and practical application that effectively simulates real-world banking operations. It supports various types of accounts - Student, Current, and Savings, and allows for essential banking transactions like depositing, withdrawing, and transferring money.

The use of Java ensures robustness and cross-platform compatibility, while the GUI design enhances user experience and interaction. This project not only serves as a valuable learning resource for understanding the intricacies of banking systems but also showcases the potential of Java in developing complex, real-world applications. It's not just a project; it's a step towards understanding and innovating within the banking sector. This project is a testament to the power of technology in transforming and simplifying financial systems. It holds immense potential for future enhancements and scalability.

# REFERENCE

[**https://www.bizzbuzz.news/h-upload/2021/07/28/1204161-dig.webp**](https://www.bizzbuzz.news/h-upload/2021/07/28/1204161-dig.webp)