**TRINITY MUSIC PLAYLIST**

MINOR PROJECT REPORT

By

**SOUMI MONDAL(RA2212702010004)**

**AKANKSHYA PANDA(RA2212702010019)**

**SHIVANSH SRIVASTAVA(RA2212702010007)**

Under the guidance of   
**Dr. MEENAKSHI N***In partial fulfilment for the Course*

of

**21CSC201J – DATA STRUCTURES AND ALGORITHMS**

in **COMPUTATIONAL INTELLIGENCE**(**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 ALGORITHMS** entitled in "**TRINITY MUSIC PLAYLIST**" is the bonafide work of **SOUMI MONDAL (RA2212702010004), SHIVANSH SRIVASTAVA (RA2212702010007)** and **AKANKSHYA PANDA (RA221272010019)** who carried out the work under my supervision.

# SIGNATURE SIGNATURE

Dr. Meenakshi N Dr Annie Uthra R

# DSA – Course Faculty Head of the Department

Assistant ProfessorProfessor

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 System is a dynamic and user-friendly application designed for our Data Structures and Algorithms (DSA) project. This system offers a seamless and personalized music listening experience, featuring efficient data structures and algorithms to manage and optimize playlist creation and song recommendations. Users can create, edit, and manage playlists with ease, utilizing data structures like linked lists and arrays for efficient storage and retrieval of songs. To enhance user engagement, the system employs algorithms for intelligent song recommendations, considering user preferences, genre, and song popularity, resulting in a customized and enjoyable music journey. Moreover, advanced sorting algorithms facilitate playlist organization and search operations, ensuring quick access to desired tracks. The Music Playlist System also focuses on minimizing time complexities and memory usage, providing a fast and responsive user interface. In this DSA project, we showcase the implementation of key data structures and algorithms to deliver a robust and efficient music playlist system that caters to the ever-evolving tastes of music enthusiasts.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S. No** | **CONTENTS** | **PAGE NO** |
| **1.** | **INTRODUCTION** | 5-8 |
|  | 1.1 Motivation | 5 |
|  | 1.2 Objective | 6 |
|  | 1.3 Problem Statement | 7 |
|  | 1.4 Challenges | 8 |
| **2.** | **REQUIREMENT ANALYSIS** | 9 |
| **3.** | **ARCHITECTURE AND DESIGN** | 10 |
| **4.** | **IMPLEMENTATION/ OUTPUT** | 11-16 |
| **5.** | **CONCLUSION** | 17 |
| **6.** | **REFERENCES** | 18 |

**CHAPTER 1**

**INTRODUCTION**

## MOTIVATION

The motivation for developing a music playlist system as a Data Structures and Algorithms (DSA) project lies in the ubiquitous role of music in people's lives. Creating an efficient system for managing playlists offers a practical application for DSA concepts. It enables users to organize, personalize, and enjoy their music collections, while simultaneously challenging developers to implement algorithms for efficient song selection, sorting, and user-friendly navigation, contributing to a deeper understanding and application of essential data structures and algorithms.

## OBJECTIVE

The objective of our music playlist system DSA (Data Structures and Algorithms) project is to design and implement an efficient and user-friendly platform that allows users to create, manage, and enjoy personalized music playlists. This project aims to employ various data structures and algorithms to optimize playlist creation, search, and recommendation functions, providing a seamless and responsive user experience. By prioritizing algorithmic efficiency and robust data structures, we intend to deliver a feature-rich music playlist system that enhances music discovery and organization for users while serving as a practical and educational demonstration of DSA principles.

## PROBLEM STATEMENT

The objective of this DSA (Data Structures and Algorithms) project is to design and implement a music playlist system that efficiently manages and organizes a collection of songs. This system will allow users to create, modify, and navigate playlists, search for songs based on various criteria, and provide a seamless and responsive user experience. The project aims to optimize data structures and algorithms for tasks like adding and removing songs, shuffling playlists, and managing user preferences, ensuring the system can handle large music libraries while maintaining fast and resource-efficient operations.

## CHALLENGES

Designing a music playlist system for a DSA (Data Structures and Algorithms) project presents several challenges, including efficient data storage and retrieval, playlist customization, and optimization for seamless user experience. Implementing data structures like linked lists or trees to store and manage songs, ensuring fast searching and sorting algorithms for playlist creation, and handling user preferences and dynamic updates are critical. Additionally, addressing potential issues with memory management, scalability, and playlist synchronization across multiple devices are key considerations in this complex project, requiring a balance between functionality and performance for an engaging music streaming experience.

# CHAPTER 2

# REQUIREMENTS ANALYSIS

## Requirement Analysis

From the given scenario, we draw the following requirements:

1. User Interface and Interaction:
   * Users should be able to create new playlists with unique names.
   * Users should be able to add songs to a playlist and remove them.
   * Users should be able to rename playlists, reorder songs, and delete playlists.
2. Data Structures and Algorithms:
   * Choose an appropriate data structure (e.g., linked list, array, etc.) to represent a

playlist.

* + Decide how to store information about each song. Consider using a data structure like a struct or class to encapsulate this information.

1. Error Handling and Validation:
   * Ensure that user inputs are properly validated to prevent invalid operations (e.g., attempting to remove a song that doesn't exist).
   * Provide clear and informative error messages to guide users when they make mistakes or encounter issues.
2. Performance and Scalability:
   * Ensure that the system performs efficiently for typical operations even with large playlists.

.

## Hardware Requirement

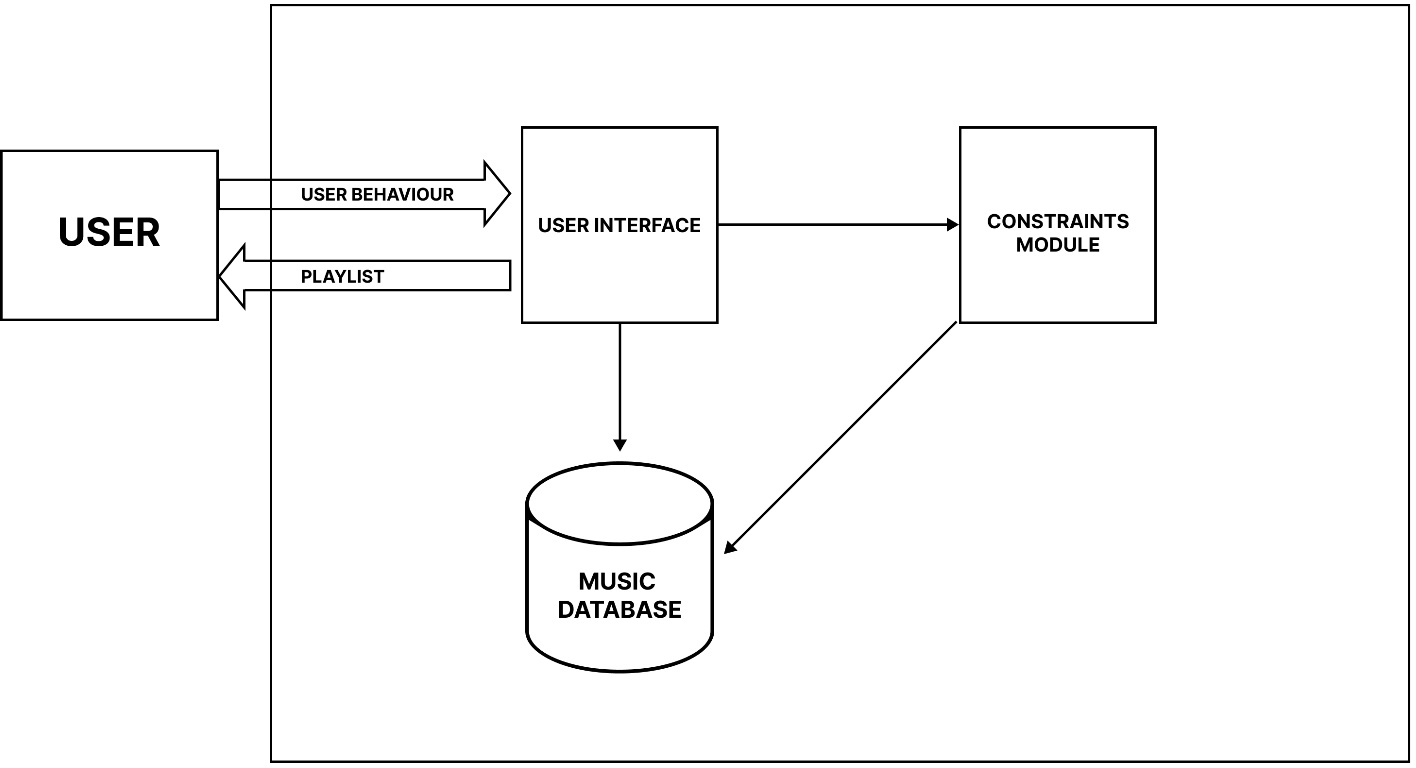
1. Processor: Dual-core or higher processor.
2. RAM: 2 GB of RAM or more.
3. Storage: At least 8 GB of internal storage.
4. Display: A screen with a resolution of 720x1280 pixels or higher.
5. Battery: A battery with sufficient capacity to run the device for a reasonable amount of time, typically around 2000 mAh or more.
6. Operating System: Android 6.0 (Marshmallow) or higher.

# CHAPTER 3

# ARCHITECTURE AND DESIGN

## Architecture

The architecture is as follows:



The architecture consists of the following management modules:

* User Interface Module
* Music Database Module
* Constraints Module

These modules are interconnected with each other.

# CHAPTER 4

# IMPLEMENTATION

## PROGRAM:

#include <stdio.h> #include <stdlib.h> #include <string.h>

// Define a structure to represent a song struct Song {

char title[100]; char artist[100];

int duration; // in seconds struct Song\* next;

};

// Function to create a new song

struct Song\* createSong(const char\* title, const char\* artist, int duration) { struct Song\* song = (struct Song\*)malloc(sizeof(struct Song)); strcpy(song->title, title);

strcpy(song->artist, artist); song->duration = duration; song->next = NULL; return song;

}

// Function to add a song to the playlist

void addSong(struct Song\*\* playlist, struct Song\* newSong) { if (\*playlist == NULL) {

\*playlist = newSong;

} else {

struct Song\* current = \*playlist;

while (current->next != NULL) {

current = current->next;

}

current->next = newSong;

}

}

// Function to display the playlist

void displayPlaylist(struct Song\* playlist) {

struct Song\* current = playlist; int songNumber = 1;

while (current != NULL) {

printf("%d. %s - %s (%d seconds)\n", songNumber, current->artist, current->title, current-

>duration);

current = current->next; songNumber++;

}

}

// Function to play the playlist

void playPlaylist(struct Song\* playlist) {

struct Song\* current = playlist;

while (current != NULL) {

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

sleep(current->duration); printf("Song finished.\n"); current = current->next;

}

}

int main() {

struct Song\* playlist = NULL; int choice;

char title[100]; char artist[100]; int duration;

while (1) {

printf("\nMusic Playlist Menu:\n"); printf("1. Add a song\n"); printf("2. Display playlist\n"); printf("3. Play playlist\n"); printf("4. Exit\n");

printf("Enter your choice: "); scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter song title: "); scanf("%s", title);

printf("Enter artist: "); scanf("%s", artist);

printf("Enter song duration (in seconds): "); scanf("%d", &duration);

struct Song\* newSong = createSong(title, artist, duration); addSong(&playlist, newSong);

break; case 2:

displayPlaylist(playlist);

break; case 3:

if (playlist == NULL) {

printf("The playlist is empty. Add songs to play.\n");

} else {

playPlaylist(playlist);

}

break; case 4:

exit(0);

default:

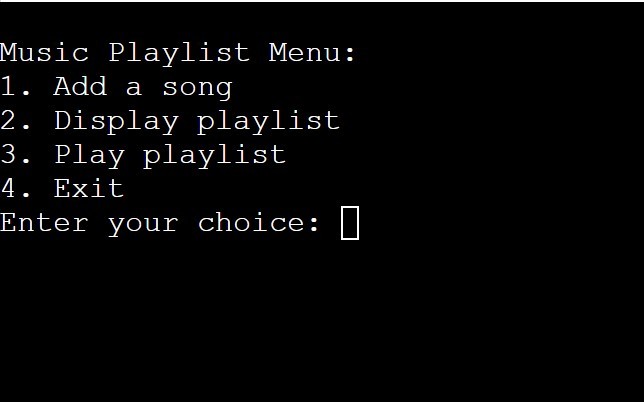
printf("Invalid choice. Please enter a valid option.\n");

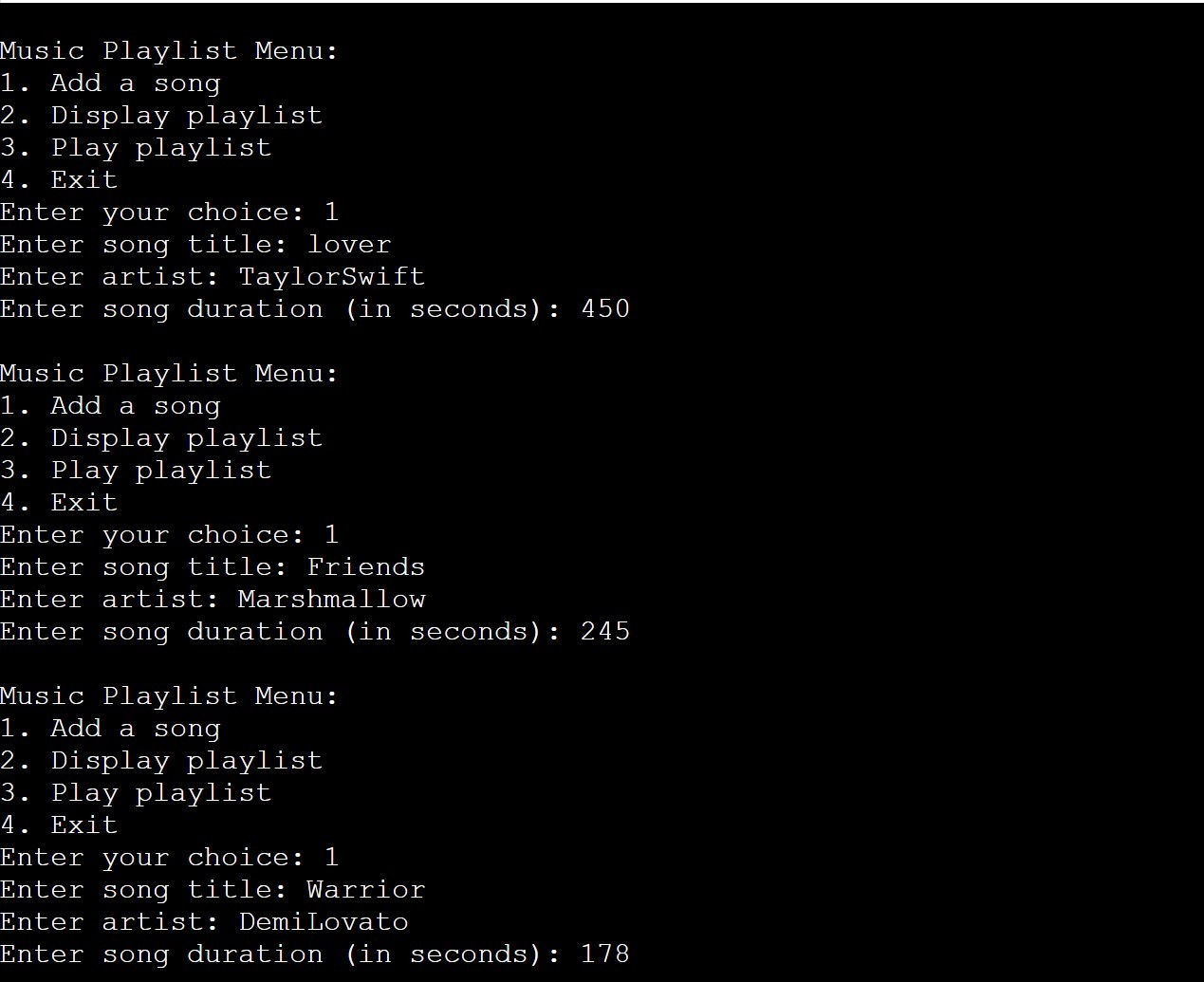
}

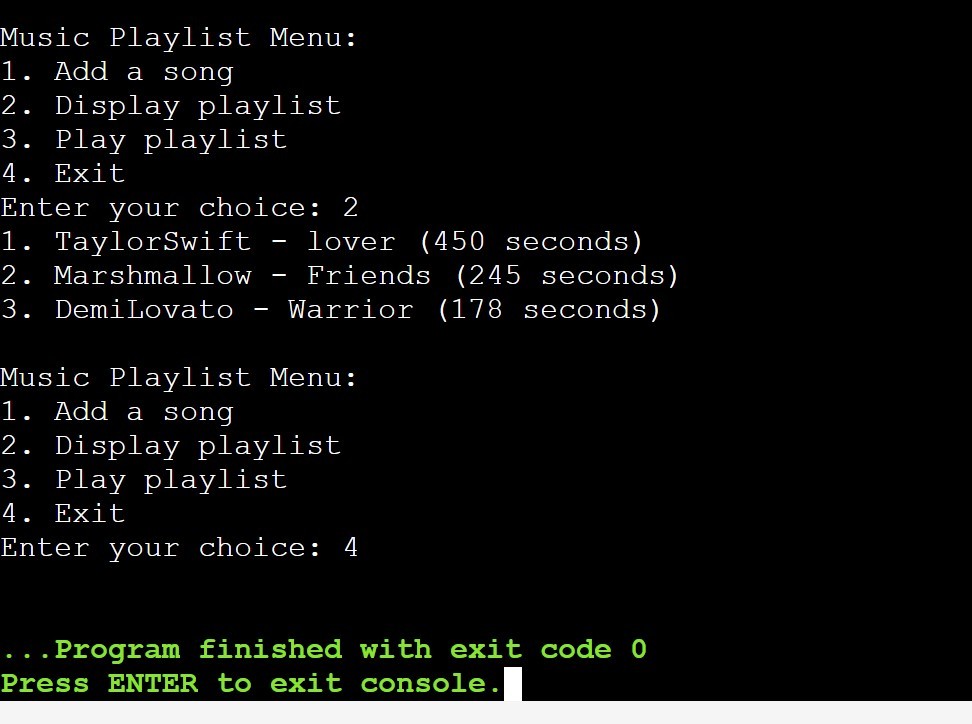
}

return 0;

}

**OUTPUT**





**CHAPTER 5**

**CONCLUSION**

In conclusion, the music playlist system developed for this DSA (Data Structures and Algorithms) project in C++ provides an efficient and user-friendly platform for managing and enjoying music. The system incorporates essential data structures such as linked lists and dynamic arrays to handle the storage and organization of songs and playlists.

Key features of the system include the ability to create, edit, and delete playlists, as well as adding and removing songs from playlists. Additionally, the system allows for seamless navigation through the library, enabling users to search for songs by title, artist, or genre.

The implementation of algorithms like sorting and searching ensures that the system operates efficiently, even with a large dataset. The system's user interface is intuitive, providing clear instructions and feedback to the user.

While the current version of the system meets the basic requirements, there is room for further enhancement. Future iterations could incorporate features such as a graphical user interface (GUI), support for additional audio formats, and integration with external music libraries or streaming services.

Overall, this music playlist system serves as a solid foundation for further development and expansion, demonstrating the successful application of data structures and algorithms in creating a practical and functional software solution.

# CHAPTER 6

# REFERENCES

* <https://www.geeksforgeeks.org/>
* <https://webflow.com/blog/website-ideas>
* [https://www.onlinegdb.com/online\_c++\_compiler](https://www.onlinegdb.com/online_c%2B%2B_compiler)
* <http://www.w3schools.com>
* http://www.webopedia.com