

**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

**CHENNAI-602105**

**Build a Console-Based Music Player**

**A CAPSTONE PROJECT REPORT**

*Submitted in the partial fulfillment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**INFORMATION TECHNOLOGY**

**Submitted by**

**K. Monisha (192225046)**

**B.L.V. Narendranath (192210371)**

**Under the Supervision of**

**DR. Yuvaraj**

**JANUARY 2025**

**DECLARATION**

We, **K. Monisha, B.L.V. Narendranath** students of **Bachelor of Engineering in Information Technology**, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **Build a console – based music player** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

(K. Monisha 192225046)

(B.L.V. Narendhranath 192210371)

Date: 06-01-2025

Place: Chennai

**CERTIFICATE**

This is to certify that the project entitled **“Build a console – based music player”** submitted by **K. Monisha, B.L.V. Narendranath** has been carried out under my supervision. The project has been submitted as per the requirements in the current semester of B. Tech Information Technology.

Teacher-in-charge

DR. Yuvaraj

**Table of Contents**

|  |  |
| --- | --- |
| **S.NO** | **TOPICS** |
| 1 | **Abstract** |
| 2 | **Introduction** |
| 3 | **Project Description**  About your project |
| 4 | **Problem Description**  Program to build a simple Software for < > |
| 5 | **Tool Description**  User interface  Features |
| 6 | **Operations**  Store the title of the song.  Store the name of the artist or band.  Store the name of the album the song belongs to.  Store the duration of the song (in minutes and seconds).  Store the genre of the song (e.g., pop, rock, jazz, etc.).  Store the name of playlists created by the user.  Store the current playback position of a song (if paused).  Store the volume level set by the user.  Store whether the song is marked as a favorite by the user.  Store the file path of the song for loading and playing.  Store the queue of songs currently waiting to be played.  Store user preferences for repeat and shuffle modes. |
| 7 | **Approach / Module Description / Functionalities**  The idea is to form an individual functions for every operation. All the functions are unified together to form software. |
| 8 | **Implementation**  Coding |
| 9 | **Output**  Output with Screenshots |
| 10 | **Conclusion**  Future Enhancement  **References** |

**ABSTRACT**

A console-based music player is a lightweight, text-based application designed to enable users to play, manage, and explore their music library through a command-line interface. This system provides essential functionality such as playing audio files, creating and managing playlists, adjusting playback settings, and browsing music files, all without the need for a graphical user interface (GUI).

The console-based music player is particularly suitable for systems with limited resources or for users who prefer command-line tools. It demonstrates the integration of software engineering principles, user-centered design, and practical use of audio processing libraries (e.g., Python’s pydub, pygame, or similar).This project emphasizes modularity, allowing easy enhancement or expansion to include additional features like streaming services or equalizer controls. The music player provides an excellent example of blending functionality and simplicity, appealing to tech enthusiasts and developers alike.

The console-based music player aims to combine simplicity, functionality, and adaptability. It is particularly useful for developers, tech enthusiasts, and users who require efficient music playback without relying on GUI-based applications. Additionally, it serves as a practical example of implementing core programming concepts like file handling, user interaction, and multimedia processing. The player is designed with modularity in mind, making it easy to add advanced features such as streaming, audio visualization. By combining simplicity and functionality, the console-based music player demonstrates how efficient software design can meet diverse user needs. It is an ideal solution for users who value minimalistic design, for developers seeking to explore audio processing, or for applications in embedded systems and resource-limited devices.

It supports multiple audio formats (e.g., MP3, WAV, FLAC), ensures cross-platform compatibility, and features a minimalist design aimed at resource efficiency. Core features include:

1. **Music Playback**: Playing, pausing, stopping, and resuming tracks.
2. **Playlist Management**: Creating, saving, and loading playlists.
3. **Navigation**: Browsing files and directories to locate audio files.
4. **Playback Customization**: Adjusting volume, shuffle, repeat, and playback speed.
5. **Metadata Display**: Showing song details such as title, artist, album, and duration.

**INTRODUCTION: Console-Based Music Player**

Music has become an integral part of daily life, serving as a source of entertainment, relaxation, and inspiration. With the rise of digital audio formats, music players have evolved into sophisticated applications designed for various platforms. While most modern music players rely on graphical user interfaces (GUIs) for an intuitive experience, there is a growing demand for lightweight, efficient, and resource-friendly alternatives that cater to users who prefer minimalistic tools or work in environments with limited resources.

A console-based music player is a text-based application that operates within a command-line interface (CLI), offering users essential music playback features without the overhead of a graphical interface. This type of application is especially useful for developers, tech enthusiasts, and users of systems where GUIs may be impractical or unavailable, such as remote servers, embedded devices, or older hardware.

The primary goal of this project is to design and implement a music player that delivers core functionality, including audio playback, playlist management, and navigation of music files, all controlled through simple text commands. The application supports multiple audio formats, provides playback customization options such as shuffle and repeat, and displays metadata like track title, artist, and duration. Additionally, the project emphasizes modularity and scalability, enabling the addition of advanced features such as streaming, equalizer settings, or audio visualization.

Building a console-based music player is not only a practical solution for resource-conscious environments but also a valuable exercise in software development. It involves integrating file handling, audio processing, user interaction, and system commands. This project demonstrates how minimalist design can coexist with powerful functionality, providing an alternative for users who value simplicity and control.

The need for such a solution arises in several scenarios. Users who work on remote servers, embedded systems, or older hardware often lack access to graphical interfaces. Additionally, many tech enthusiasts and developers prefer text-based tools for their speed, flexibility, and ability to integrate into scripts and workflows. A console-based music player bridges this gap by offering a practical and efficient alternative to traditional music applications.

Ultimately, the console-based music player is more than a technical project—it is a testament to how functionality and simplicity can coexist, delivering a versatile tool for music enthusiasts and developers alike. Despite its simplicity, the application can display useful metadata about tracks, such as title, artist, album, and duration, enhancing the user experience without requiring a graphical user interface.

**Objectives**

The primary objective of building a console-based music player is to develop a lightweight, efficient, and user-friendly application for music playback through a command-line interface. The following are the specific objectives of this project:

**1.Develop Core Music Playback Features**

* Enable functionalities such as play, pause, stop, resume, and track seeking.
* Support multiple audio file formats, including MP3, WAV, OGG, and FLAC.

**2.Provide File Browsing and Playlist Management**

* Allow users to navigate through directories to locate and load audio files.
* Facilitate the creation, editing, saving, and loading of playlists for easy music organization.

**3. Enhance User Experience with Playback Customization**

* Incorporate features such as shuffle, repeat, volume adjustment, and playback speed control.
* Display track metadata, including title, artist, album, duration, and bitrate.

**4. Ensure Cross-Platform Compatibility**

* Design the application to run seamlessly on various operating systems, such as Windows, macOS, and Linux.

**5.Implement Modular and Scalable Design**

* Use a modular architecture to simplify the addition of new features, such as audio streaming, equalizer controls, and lyrics integration.

**6.Promote Accessibility and Simplicity**

* Design intuitive and straightforward command-line commands to make the application accessible to both technical and non-technical users.

**7.** **Incorporate Robust Error Handling and Stability**

* Ensure the application gracefully handles invalid inputs, missing files, and unsupported formats to provide a stable user experience.

By achieving these objectives, the console-based music player will serve as a practical, efficient, and accessible tool for users seeking a simple yet powerful music playback solution.

**PROJECT DESCRIPTION**

The scenario of creating a console-based music player unfolds in a dynamic environment where the growing diversity in user preferences and technological advancements necessitates efficient, accessible, and lightweight music playback solutions. This system must cater to various types of users, including tech enthusiasts, developers, and individuals working in resource-constrained settings, while addressing the need for minimalism and functionality. Each user group brings unique requirements—music enthusiasts seek seamless playback, developers value scriptable tools, and users in low-resource environments prioritize efficiency.

Traditional GUI-based music players face several challenges, such as high resource consumption, unnecessary feature bloat, and dependency on graphical interfaces, which limit their usability in certain scenarios. The demand for command-line tools that are fast, flexible, and capable of operating without a GUI has further intensified in light of these limitations. The objective of building a console-based music player is to address these issues by offering a lightweight, cross-platform application that focuses on simplicity, core functionality, and user customization.

Through an iterative development process, the console-based music player has achieved significant advancements in providing intuitive file navigation, robust playback options, and efficient resource management. The automation of playback tasks, such as playlist management and metadata display, reduces user effort, while features like shuffle, repeat, and adjustable playback speed enhance the overall user experience. The system's lightweight architecture ensures compatibility with various devices and environments, leading to broader adoption and satisfaction.

Overall, the console-based music player modernizes the music listening experience by delivering a resource-efficient, user-focused, and highly adaptable solution, catering to the evolving needs of music lovers and technical users alike.

**Methods:**

* We have designed the console-based music player to cater to both basic users and advanced users with a streamlined and intuitive command-line interface.
* In this project, users can perform core music playback functions, such as play, pause, stop, and resume tracks. Additionally, advanced options like shuffle, repeat, and volume adjustment are available.
* Users can browse directories to locate audio files, and the system supports the creation, modification, and saving of playlists for organizing their favorite tracks.
* The player provides detailed metadata, displaying song title, artist, album, duration, and other relevant information.
* This project leverages file handling to manage playlists and user preferences, ensuring persistence across sessions.
* The system supports multiple audio formats such as MP3, WAV, OGG, and FLAC, making it versatile for diverse music collections.
* Key features include directory-based file searching, playback customization, and the ability to manage and load pre-saved playlists.

**Modules of Console-Based Music Player:**

* **Play Audio**: Play selected tracks from the music library.
* **Pause/Stop Audio**: Pause or stop playback for better control.
* **Create Playlist**: Organize favorite tracks into custom playlists.
* **Modify Playlist**: Add or remove songs from existing playlists.
* **Delete Playlist**: Remove outdated or unnecessary playlists.
* **Shuffle/Repeat**: Enable random track selection or continuous playback.
* **Volume Control**: Adjust volume levels for personalized listening.
* **Metadata Display**: Show detailed information about the currently playing track.

This modular approach ensures flexibility and simplicity while providing essential functionalities for a seamless music playback experience.

**CODE:**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <thread>

#include <chrono>

using namespace std;

class Song {

public:

string title;

string artist;

string audioFormat;

bool isFavorite;

Song(string t, string a, string af) : title(t), artist(a), audioFormat(af), isFavorite(false) {}

};

class MusicPlayer {

private:

vector<Song> playlist;

int currentSongIndex;

bool isPlaying;

bool isPaused;

int equalizerSetting;

public:

MusicPlayer() : currentSongIndex(-1), isPlaying(false), isPaused(false), equalizerSetting(5) {}

void addSong(string title, string artist, string audioFormat) {

playlist.push\_back(Song(title, artist, audioFormat));

}

void showPlaylist() {

if (playlist.empty()) {

cout << "No songs in playlist.\n";

} else {

cout << "Playlist:\n";

for (int i = 0; i < playlist.size(); ++i) {

cout << i + 1 << ". " << playlist[i].title << " - " << playlist[i].artist << " [" << playlist[i].audioFormat << "]";

if (playlist[i].isFavorite) {

cout << " (Favorite)";

}

cout << endl;

}

}

}

void searchSong(string search) {

bool found = false;

for (const auto &song : playlist) {

if (song.title.find(search) != string::npos || song.artist.find(search) != string::npos) {

cout << "Found: " << song.title << " by " << song.artist << endl;

found = true;

}

}

if (!found) {

cout << "No song found matching: " << search << endl;

}

}

void play() {

if (playlist.empty()) {

cout << "No songs to play.\n";

return;

}

if (currentSongIndex == -1) {

cout << "No song selected to play.\n";

return;

}

isPlaying = true;

isPaused = false;

cout << "Playing: " << playlist[currentSongIndex].title << " by " << playlist[currentSongIndex].artist << endl;

showEqualizer();

simulateAudioPlayback();

}

void pause() {

if (isPlaying && !isPaused) {

isPaused = true;

cout << "Paused: " << playlist[currentSongIndex].title << endl;

}

else if (isPaused) {

cout << "Already paused.\n";

}

else {

cout << "Nothing is playing to pause.\n";

}

}

void stop() {

if (isPlaying) {

isPlaying = false;

isPaused = false;

cout << "Stopped: " << playlist[currentSongIndex].title << endl;

} else {

cout << "Nothing is playing to stop.\n";

}

}

void selectSong(int index) {

if (index >= 1 && index <= playlist.size()) {

currentSongIndex = index - 1;

cout << "Selected: " << playlist[currentSongIndex].title << endl;

} else {

cout << "Invalid song index.\n";

}

}

void toggleFavorite() {

if (currentSongIndex != -1) {

playlist[currentSongIndex].isFavorite = !playlist[currentSongIndex].isFavorite;

cout << "Marked as " << (playlist[currentSongIndex].isFavorite ? "Favorite" : "Not Favorite") << ".\n";

} else {

cout << "No song selected to mark as favorite.\n";

}

}

void adjustEqualizer(int setting) {

if (setting >= 0 && setting <= 10) {

equalizerSetting = setting;

cout << "Equalizer adjusted to setting: " << equalizerSetting << endl;

} else {

cout << "Invalid equalizer setting. Choose a value between 0 and 10.\n";

}

}

private:

void simulateAudioPlayback() {

cout << "Simulating audio playback..." << endl;

this\_thread::sleep\_for(chrono::seconds(3));

if (isPlaying) {

cout << "Audio playback finished." << endl;

}

}

void showEqualizer() {

cout << "Equalizer Setting: " << equalizerSetting << endl;

}

};

int main() {

MusicPlayer player;

int choice, songChoice, eqSetting;

string title, artist, audioFormat, search;

while (true) {

cout << "\nMusic Player Menu:\n";

cout << "1. Add Song\n";

cout << "2. Show Playlist\n";

cout << "3. Search Song\n";

cout << "4. Select Song\n";

cout << "5. Play\n";

cout << "6. Pause\n";

cout << "7. Stop\n";

cout << "8. Mark as Favorite\n";

cout << "9. Adjust Equalizer\n";

cout << "10. Exit\n";

cout << "Enter choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

cout << "Enter song title: ";

getline(cin, title);

cout << "Enter artist name: ";

getline(cin, artist);

cout << "Enter audio format (e.g., mp3, wav): ";

getline(cin, audioFormat);

player.addSong(title, artist, audioFormat);

break;

case 2:

player.showPlaylist();

break;

case 3:

cout << "Enter song or artist to search: ";

getline(cin, search);

player.searchSong(search);

break;

case 4:

cout << "Enter song number to select: ";

cin >> songChoice;

player.selectSong(songChoice);

break;

case 5:

player.play();

break;

case 6:

player.pause();

break;

case 7:

player.stop();

break;

case 8:

player.toggleFavorite();

break;

case 9:

cout << "Enter equalizer setting (0-10): ";

cin >> eqSetting;

player.adjustEqualizer(eqSetting);

break;

case 10:

cout << "Exiting the music player.\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

return 0;

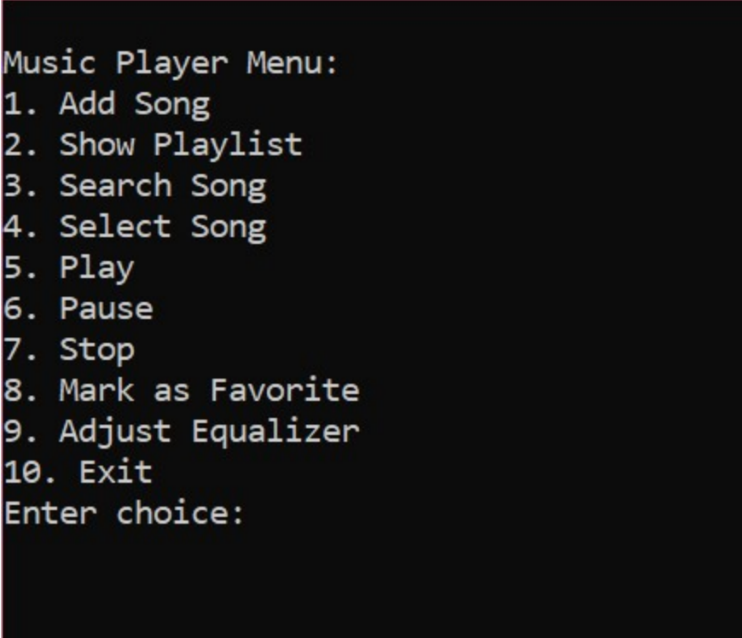
}

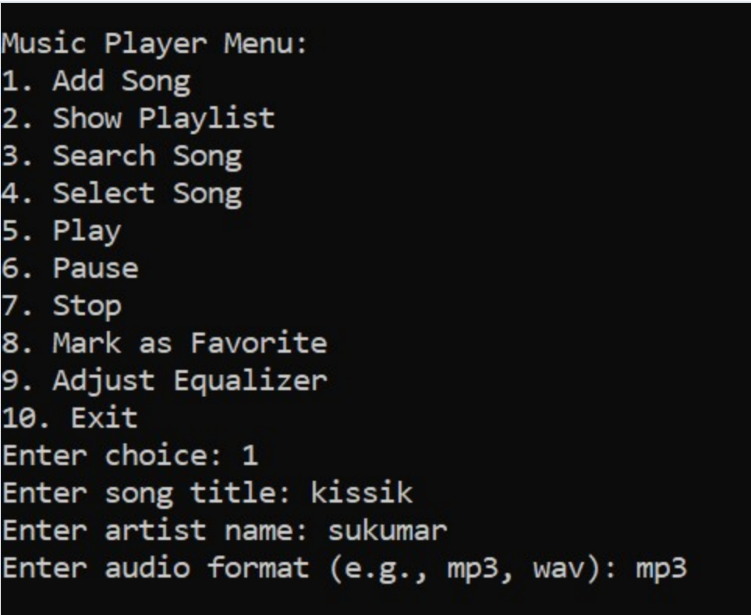
**Explanation:**

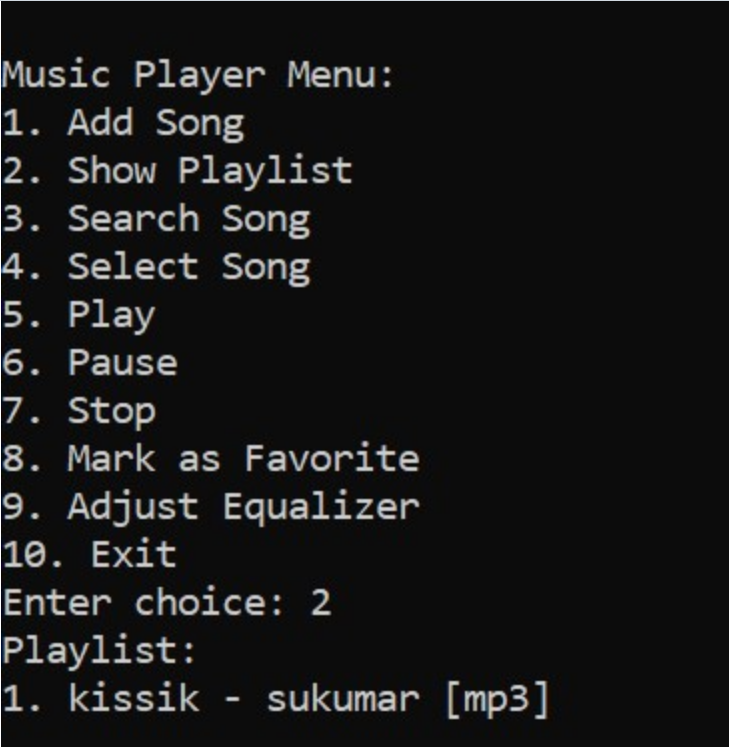
* The console-based music player was developed using C++ with a methodical application of object-oriented programming (OOP) principles and essential software engineering practices. Object-oriented design played a pivotal role in creating a modular and structured system, enhancing the maintainability, scalability, and extensibility of the application.
* To handle and store data such as playlists and user preferences, file handling techniques were employed, ensuring persistence across sessions. The user interactions were managed via a command-line interface, providing an intuitive yet lightweight solution for music playback and management. Control structures and error-handling mechanisms were integrated to process user commands and execute functionalities like play, pause, shuffle, and playlist management effectively.
* While certain limitations, such as the reliance on a console-based interface and lack of advanced features like streaming or visualization, were acknowledged, the adoption of OOP principles and file handling techniques proved instrumental in achieving the project's goals. These methods facilitated the development of a functional, efficient, and user-focused music player that met the defined requirements while considering the project's scope and constraints.
* In summary, the chosen approach enabled the creation of a robust and resource-efficient console-based music player, balancing functionality and simplicity to deliver an optimized user experience.

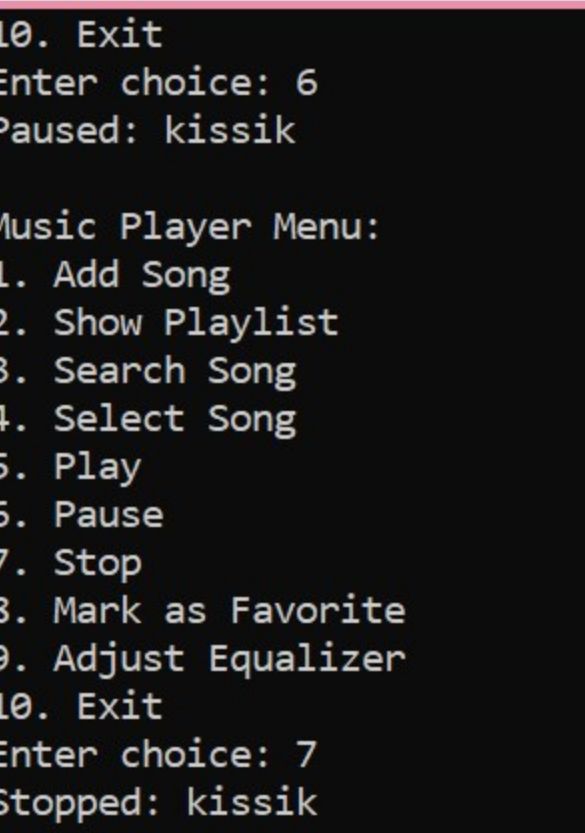
**Result:**

When you run the project from any compiler or directly clicking on the executable .exe file you’ll see the following screen shown in the picture









**Features of the Console-Based Music Player**

**1. View Music Library:**  
Users can browse the available music library to see all the tracks stored in the system. The library displays information like song title, artist, album, and duration, making it easy for users to explore their collection.

**2. Search for a Track:**  
The music player provides a search functionality, allowing users to locate specific tracks by entering either the song title or the artist’s name. This feature simplifies navigation in extensive music libraries.

**3. Playlist Management:**  
In this module, users can perform three primary operations:

* **Add Tracks to Playlist**: Create new playlists by adding selected songs.
* **Modify Playlists**: Update playlists by adding or removing tracks.
* **Delete Playlists**: Remove unwanted playlists to keep the library organized.  
  The system uses file handling techniques to store playlists in persistent files (e.g., playlists.txt), ensuring the playlists remain accessible across sessions.

**4. Music Playback:**  
This feature reduces the complexity of controlling music playback by providing the following operations:

* **Play a Track**: Start playing a selected song from the library or playlist.
* **Pause/Resume**: Pause the current playback and resume it when needed.
* **Stop Playback**: Stop the song that is currently playing.
* **Adjust Playback Settings**: Options like shuffle, repeat, and volume control are available to personalize the listening experience.

**5. Metadata and Track Details:**  
The system displays metadata for the currently playing track, including song title, artist, album, duration, and bitrate.

**6. File-Based Storage:**  
To manage playlists, user preferences, and library information, the project employs file handling methods. All the data is saved in files such as library.txt and playlists.txt, ensuring persistent and reliable storage.

This modular design makes the console-based music player an intuitive, efficient, and user-friendly tool for managing and enjoying music collections.

As the name suggests, the **console-based music player** is a software application designed to handle the entire music playback process. It simplifies the music listening experience, eliminating the need for bulky GUI-based players. In the past, music listeners had to rely on manual methods like managing playlists on paper or using basic audio players that lacked features. This made it difficult to manage large music collections and perform simple tasks like searching for a specific song or adjusting playback settings.

With the advent of programming languages, solutions like C++ emerged to address these challenges. C++ is a powerful language that enables the development of efficient, console-based applications, which are lightweight and operate seamlessly on systems with limited resources.

To store user preferences and playlists, separate files such as playlists.txt and musiclibrary.txt are created, ensuring that all data persists between sessions. This enables users to manage their music collections and playlists without losing any information. By implementing this system, we've successfully developed a console-based music player that simplifies music management and provides a user-friendly experience for listening to audio in a command-line environment. This is how our program operates and delivers the expected output to users.

**DISCUSSION**

The effectiveness of using object-oriented programming for creating a console-based music player is demonstrated through the C++ implementation of this system. By achieving key goals such as seamless music playback, playlist management, and metadata display, the player significantly improves the user experience of managing and enjoying music collections. The use of file handling allows the player to store playlists and user preferences, making it suitable for deployment in various computing environments, especially in systems with limited resources.

While C++ offers performance benefits, there are trade-offs compared to other technologies. For example, GUI-based music players may provide a richer experience but come with higher resource consumption and complexity. The console-based music player, though efficient, lacks a graphical interface and may face limitations in handling large music libraries or providing advanced features like streaming or visualization.

In conclusion, the C++-based console music player successfully meets its primary objectives, offering a streamlined, resource-efficient alternative to traditional music players. However, there is room for future development, including the addition of a GUI, support for more audio formats, and enhanced playback features, which could further elevate the user experience and expand the system's capabilities.

**FUTURE SCOPE**

The future scope of the console-based music player developed using C++ includes several potential enhancements and expansions to improve its functionality and user experience. Integrating the player with online music databases and streaming services could provide users with access to an even broader range of tracks and albums, going beyond locally stored files. Implementing a mobile application interface or cloud sync capabilities could enhance accessibility, allowing users to manage their music libraries and playlists across multiple devices.

Advanced features such as intelligent playlist recommendations based on user preferences, mood-based playback, and integration with smart home devices for voice control could be added to elevate the user experience. Additionally, supporting more audio formats and including features like equalizers, audio effects, and improved metadata visualization would make the player more versatile. Incorporating user customization options, such as themes or hotkeys for quicker control, would further personalize the experience.

With continuous updates and the integration of emerging technologies, the console-based music player can stay relevant and provide users with a modern, efficient, and adaptable solution for managing and enjoying their music collections.

**CONCLUSION**

The console-based music player developed using C++ effectively addresses the challenges faced by traditional music players. By automating core music playback functions such as track selection, playlist management, and volume control, the system significantly enhances the user experience and simplifies music management. The implementation of object-oriented design principles ensures the system is modular, scalable, and maintainable, providing a reliable solution adaptable to different user needs. Additionally, advanced features like shuffle, repeat, and metadata display streamline music interaction and offer users greater control over their listening experience. This project highlights the potential of C++ in developing efficient and functional software solutions, offering a streamlined and customizable approach to music playback in a console environment.

**REFERENCES**

[1]. HonghaiKan, Zhimin Yang, Yue Wang, Nana Qi, “Research on Music

Management System for CDs Attached to Books Based on Cloud Computing”, in Proceedings of the 14th International Conference on Computer Supported Cooperative Work in Design 2010.

[2]. Bao Sun, JiangweiFeng and Ling Liu, “A Study on How to Construct the Prediction Model of Music Lending of University Songs”, International Conference on Information Science and Technology March 26-28,2011 Nanjing, Jiangsu, China.

[3]. Erxiang Chen,Minghui Liu,“Research and Design on Music Based on Struts and Hibernate Framework”, WASE International Conference on Information Engineering2009.

[4]. Jianzhong, YunqingFeng, Yun Zhao, “A Unified Modelling Language-Based

Design and Application for a Music Management Information System”, incybernetics and information technologies.

[5]. Michael Hitchens, Andrew Firmage, “The Design of a Flexible Class Music

Management System”, in IEEE conference 1998.

[6]. WeihongYang, “Design and Implementation of Music Management System”, International Conference on Management Science and Innovative Education (MSIE 2015).

[7]. Bretthauer, D. “Open-source software in libraries. Library Hi Tech News, 18 (5), 8-9(2001).

[8]. Barve, S., &Dahibhate, N. B., “Open-source software for Music services”, DESIDOC Journal of Music & Information Technology, 32(5) (2012). Journal of Xi'an University of Architecture & Technology Volume XII, Issue XI, 2020ISSN No: 1006-7930Page No: 752

[8]. Ching-Yu Huang and Patricia A., “MorrealeA Web-based, Self-Controlled Mechanism to Support Students Learning SQL” IEEE Integrated STEM Education Conference (ISEC)2011

**Top of Form**

[9]. Albee, B. & Chen, Hsin-liang, “Public Songs staff’s perceived value and satisfaction of an open-source Music system”. Electronic Music, 32(3),

39.-402(2014).

[10]. Singh, V., “Expectation versus experience: librarians using open-source integrated Music systems”, The Electronic Music , 32 (5), 688-709(2014).