**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

**A Project Report**
**on**
# “Playlist Sorter”

**[Code No: COMP 202]**

**(For partial fulfillment of II Year/ Semester I in Computer Science/Engineering)**

**Submitted by**

**Sangya Pandey (Roll No. 32)**

**Submitted to**

**Sagar Acharya**

**Department of Computer Science and Engineering**

**Submission Date: 02/25/2026**

## Abstract

Playlist Sorter is a simple desktop music management system that is made using C++ an Qt framework. The main object of this project is implement fundamental data structures and algorithms concept to make a practical application. This project allows users to add, delete, sort, search and play the songs users add to the application. When adding the songs to the application, users can assign a mood to each songs and the application organizes the songs according to the mood filter as well.

This project uses multiple data structures. It uses doubly linked list to store the songs dynamically and allow bidirectional traversal. It uses binary search tree for searching and stack for song history. It uses merge sort for sorting the songs according to artist or title. The list of songs are stored in a text file.

The application uses Qt framework for the user interface and the multimedia functions are handled using QMediaPlayer. This project is a simple and lightweight multimedia application that is made for organizing scattered audio files using relevant concepts of data structures and algorithms.

**Keywords:** *Playlist Sorter, Data Structures and Algorithms, Doubly Linked List, Binary Search Tree, Merge Sort, Stack, Qt Framework, C++*

# Acknowledgement

I would like to express my sincere gratitude to my course instructor, Sagar Acharya, for his continuous guidance and support throughout the completion of this project. It would not have been possible without his instructions and lectures that helped me understand the concepts of data structures and algorithms that were required to complete this project. I would also like to thank the entire Department of Computer Science and Engineering for giving me the opportunity to work on this project.

My sincere thanks goes to my peers and faculty members that were supportive throughout the course of this project. I am also grateful to the Qt company for providing documentation that helped me understand the application of multimedia extension.

# Table of Contents

# List of Figures

# Abbreviations

| | |
|---|---|
| **GUI** | Graphical User Interface |
| **BST** | Binary Search Tree |
| **DSA** | Data Structures and Algorithms |
| **DMA** | Dynamic Memory Allocation |
| **XMMS** | X Multimedia System |

# Chapter 1

## Introduction

Life has become easier with fast digitalization of many things. One of them is music systems that have gone from being simple playback tools to systems that allow the users to personalize their listening experience by organizing and searching songs.

This project implements C++ and Qt framework to create a ‘Playlist Sorter’ that helps users add, organize, search, sort and play songs based on the moods they assign to each song like happy, chill, workout, and sad. Different data structures such as linked list, binary search trees (BST), stack and merge sort are used to make this project more efficient and user friendly.

### 1.1 Background

Multimedia applications need frameworks that can manage audio processing, GUI, and event-driven programming. Qt framework offers cross-platform development tools and multimedia components like QMediaPlayer and QAudioOutput, which facilitate efficient meda playback (Qt Company, 2023).

Data Structures are commonly employed to manage digital libraries. Linked lists enable DMA and efficient insertion and deletion. BSTs improve search efficiency. Merge sort is used due to their stable and predictable O(n log n) time complexity (Sedgewick & Wayne, 2011).

### 1.2 Objectives

This project had the following objectives:

- To develop a music player that helps users personalize their music
- To implement dynamic song storage using doubly linked list
- To implement song sorting using merge sort
- To implement search using BST
- To manage history using stack
- To allow users to assign each song a mood

## 1.3 Motivation and Significance

The motivation for this project came from the fact that music players are either too vast or don't have the features that some users want. Many users still have audio files in their personal computers that are scattered across different directories and need an application to manage and look through these songs. Having to download a new application that consumes significant space on our desktop can be a hassle. So, the idea for 'Playlist Sorter' comes not only from the need to integrate different DSA concepts in a practical application but also to solve the problem of not having a simple, lightweight application to manage different audio files in my desktop.

The significance of this project lies in the fact that it is lightweight and has all the necessary features for personalized management of songs that are in the local drive. It has multi-structure integration where different DSA concepts are used to handle different features like efficient searching using BST, data sorting using merge sort, temporary history tracking using stack and doubly linked list for storing songs and navigating to 'previous' and 'next' songs in the list.

# Chapter 2

## Related Works

There are many music systems catering to user needs from simple applications that allow music playback offline to vast libraries that allow users to listen to all songs through their phones and laptops when online. Some widely used music players include apple music, spotify and youtube music but these applications are too vast to be compared to a project that was created with the purpose of practical implementation of data structures and algorithm concepts. While it is difficult to list all of the applications, some applications similar to the project are:

- **Strawberry Music Player:** It is a prominent open-source collection and player forked from Clementine. Strawberry is built using C++ and Qt. It is known for its local library management. It has vast features including stream integration and advanced audio transcoding which can be resource-heavy for a user seeking simple and fast organizational tool
- **Foobar2000**: It is known for its modular design. It utilizes tree-based structures and SQL components to manage massive libraries.
- **Quod Libet:** It is built around the idea of "regex" searching and massive tag-based organization. It prioritizes the search function as the primary way to interact with music.
- **Sayonara Player:** It is a lightweight music player written in C++ and Qt. It is designed specifically for performance on low-resource systems. It uses a highly optimized library management system that allows for mast sorting and searching.
- **Audacious:** It is a descendant of XMMS, Audacious is a high-performance, open-source audio player that prioritizes "playing music how you want it, without stealing away your computer's resources" (Audacious Team, 2025). It is significant because it demonstrates the professional application of C++ for "drag and drop" playlist management and real-time library searching. While Audacious offers extensive plugin support and equalizer features, this project focuses specifically on the data structure logic behind searching (BST) and history (Stacks).

## 2.1 Problem Statement

Despite the abundance of music streaming platforms and media players, there is no lightweight, localized solution that prioritizes algorithmic efficiency. Most existing local players rely on basic linear search methods with time complexity of O(n) that becomes slow as the music library expands.

Furthermore, standard applications don't have temporary history navigation that allows users to trace back to their listening session.

Additionally, while most applications let users add songs to 'favourites', they don't have a way for users to categorize songs according to moods like 'happy', 'sad', etc.

This project addresses these inefficiencies by using BST for fast searching, doubly linked list for bidirectional traversal and stack for history tracking. It also lets users choose to navigate the list according to different moods.

# Chapter 3

## Design and Implementation

The system is developed using C++ and Qt framework and the architecture integrates GUI components with backend data structures

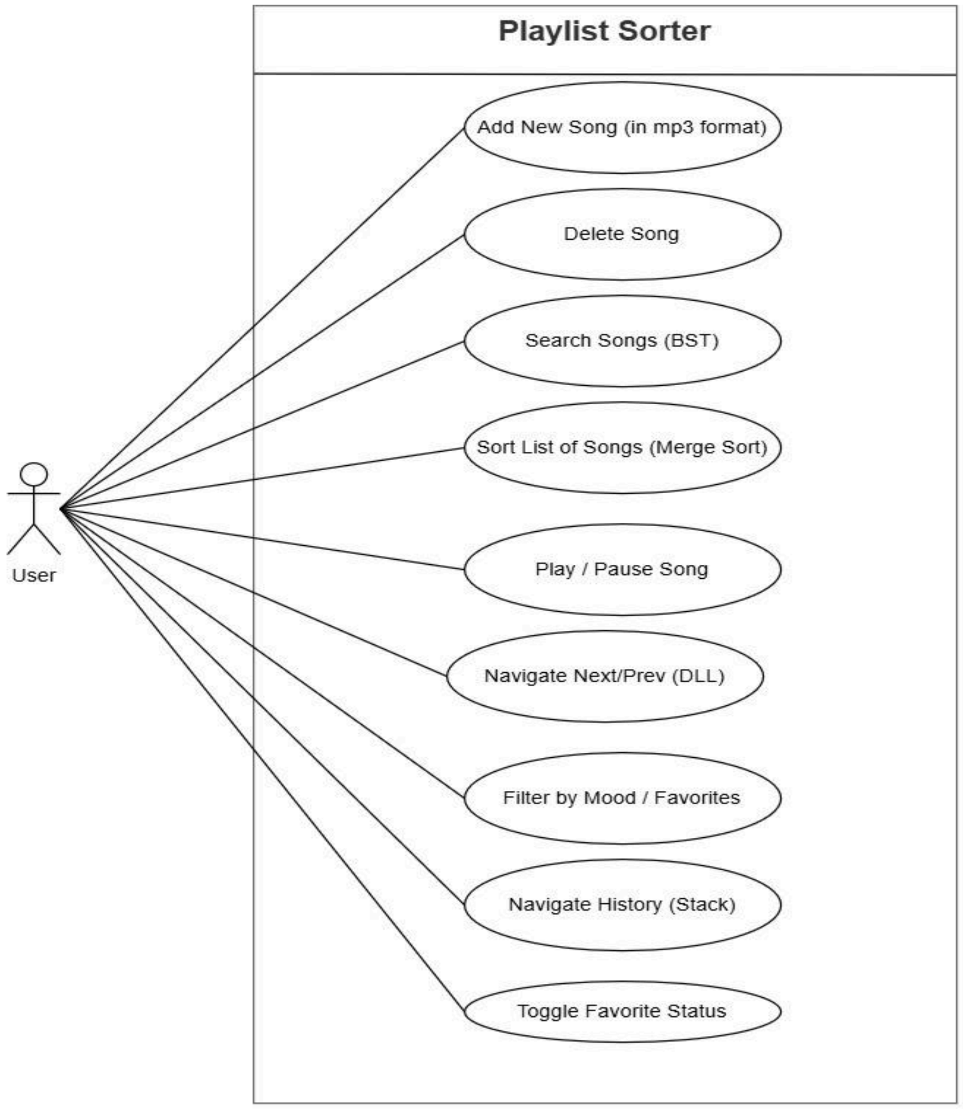


*Figure 3.1: Use Case Diagram*

The use case diagram shows how a user interacts with the project. A user can delete, search, sort, play/pause, navigate, filter, navigate history, toggle favourite status for any song that the user adds to the system.

## 3.1 System Requirement Specification

### 3.1.1 Software Specifications

- **Operating system:** Windows/Linux/Mac
- **Programming Language:** C++
- **Framework:** Qt
- **Multimedia Module:** QMediaPlayer
- **File Handling:** QTextStream

### 3.1.1 Hardware Specifications

- **Processor:** Intel Core i3 or equivalent
- **RAM:** 4GB
- **Audio:** Standard Sound Card with high-bitrate MP3 decoding
- **Display:** 1366 × 768 resolution
- **Storage:** 50 MB for application (plus music storage)

# Chapter 4

# Discussion on Achievements

## 4.1 Features

This project integrates the fundamental data structures and algorithms concepts into a simple and functional desktop application. The feature are:

1. **Add Songs**
   Users can add new songs dynamically to the music library. It is achieved using a doubly linked list. The system also checks if the song already exists to stop duplication. The BST is rebuilt after insertion of the song.

2. **Delete Songs**
   Users can delete selected song node from the linked list. It then clears the history stack and rebuilds the BST.

3. **Song Navigation**
   The users can navigate the songs using the previous and next buttons. This bidirectional traversal is achieved by the use of a doubly linked list.

4. **Search Song**
   Users can search a specific song using the search bar. It is achieved by constructing a BST with song titles as the nodes. Each song is inserted alphabetically into the BST and searching is done using recursive traversal.

5. **Sorting**
   Users can sort songs by title or by artist. This feature is achieved using merge sort.

6. **Playback History**
   Users can see the list of songs they've listened to. They can also navigate through the history list too. This is achieved using stack where previously played songs are pushed to stack and the 'back' button pops the song from the stack.

7. **Mood-based Filtering**
   Users can assign the mood to each song that they add to the library. Then they can filter songs based on moods. It is achieved by traversing the linked list and displaying the songs based on mood or favourites.

8. **Add Song to Favourites**
   Users can add the songs they like to favourites. When the list is created each song node has a boolean that is initially set to false and changes when the users click on the 'favourite' button for the selected song.

9. **Persistent Storage**
   This feature allows the songs that users insert to the library to be there even when the application is restarted. This is achieved using file handling.

# Chapter 5

# Conclusion and Recommendations

The ‘Playlist Sorter’ is a desktop application designed to help users organize their audio files and personalize their music experience with interactive organization. It helps in organizing the music library with the use of fundamental data structures and algorithm concepts.
The application is designed to be simple, lightweight and efficient in managing the music files in desktop with features like mood-based filtering, favourites, playback history, sorting and sorting to make the listening experience better for each user.
Additionally, it has the potential to me a more advanced and scalable application that is capable of giving users more personalized experience and handling larger libraries with further enhancements.

## 5.1 Limitations

Although the application is functional, there are many limitations to it. Some of them are:

1. **Non Self Balancing Search Tree**
   Currently, the application uses BST for searching songs but it is not self balancing and in the worst case scenario it could be completely skewed and result it in reducing the efficiency to that of linear search. This could affect the performance when the music library becomes larger.
2. **Linear Filtering**
   Some operations like duplicate checking and mood filtering uses linear traversal of the linked list. This also becomes a problem as the music library becomes larger.
3. **File Storage**
   This application currently stores the music data in a text file which is not as efficient as using a database.
4. **Lack of Audio Format Support**
   The application only supports MP3 files.
5. **Lack of Advanced User Interface Customization**
   The GUI currently lacks customization options like changing themes, or other visual effects that attracts users.

## 5.2 Future Enhancements

Below are some of the future enhancements that would overcome the current limitations.

1. **Database Integration**
   Integrating database like MySQL, SQLite would help in indexing and query search. It would also support larger library.
2. **Advanced Search Option**
   Searching would be faster if prefix trees or hash indexing were used. It would also support filtering options.
3. **Support Different Audio Formats**
   The application would be more flexible if different audio formats like WAV were also allowed.
4. **Advanced User Interface Customization**
   Different features like themes, dark mode, drag-and-drop function would help improve user experience.
5. **Additional Playback Features**
   To make the application better, features like shuffle, looping songs, creation of playlists could be added.

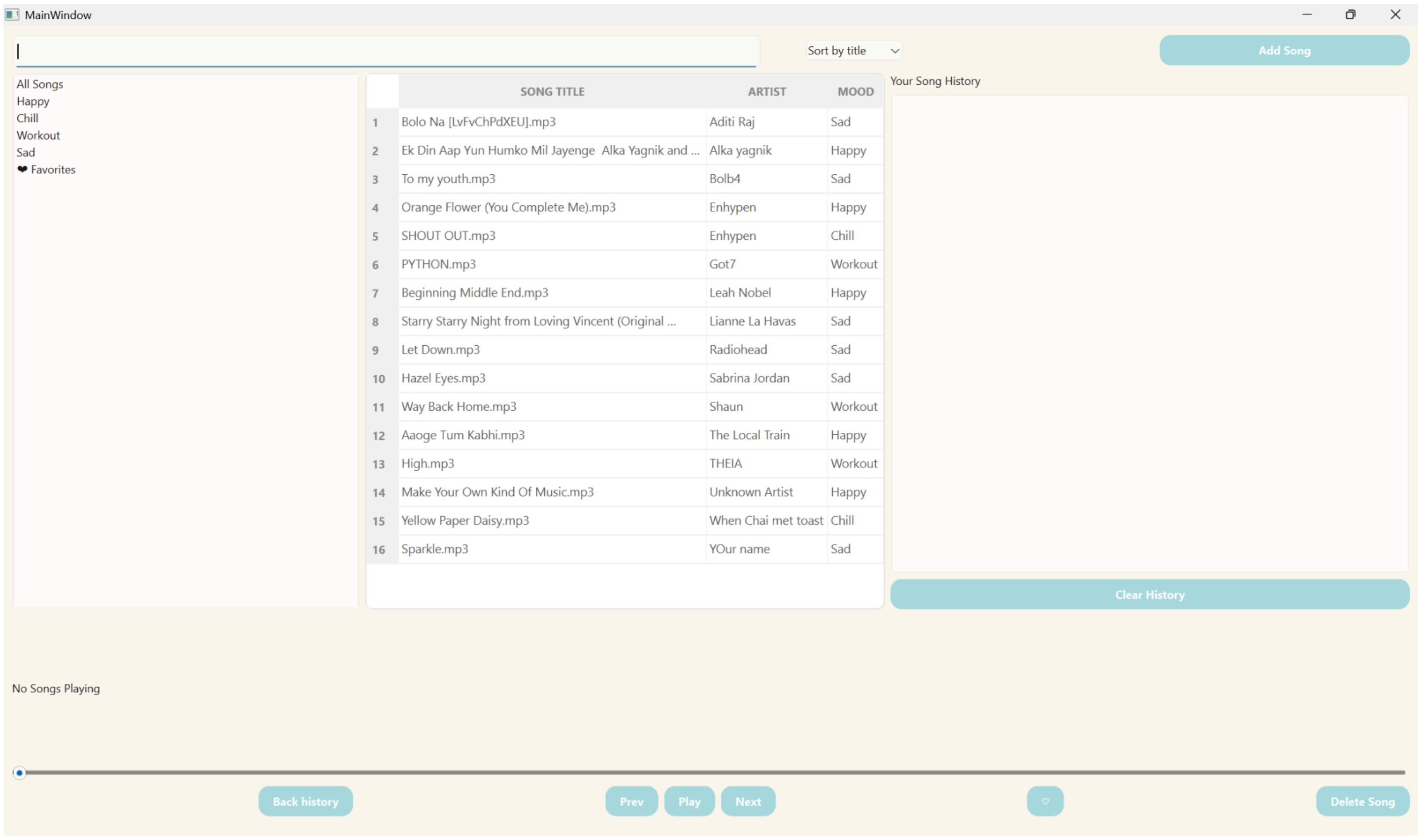
## References

*Qt Company. (2023). Qt documentation.* [*https://doc.qt.io*](https://doc.qt.io)

*Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.*

*Audacious Team. (2025, September 7). Audacious - An advanced audio player.* [*https://audacious-media-player.org/*](https://audacious-media-player.org/)

# Appendix


MainWindow

Sort by title

Add Song

- All Songs
- Happy
- Chill
- Workout
- Sad
- ❤ Favorites

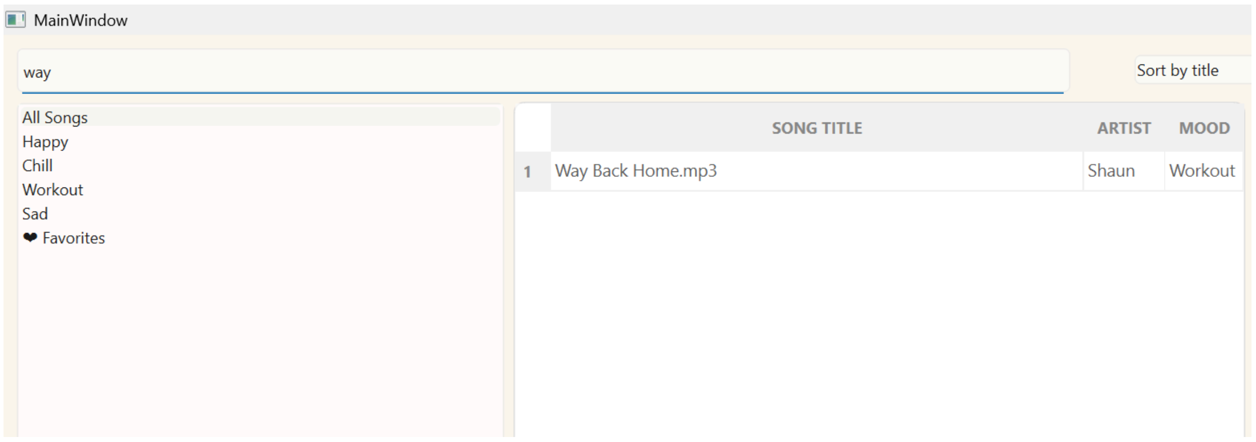
| | SONG TITLE | ARTIST | MOOD |
|---|---|---|---|
| 1 | Bolo Na [LvFvChPdXEU].mp3 | Aditi Raj | Sad |
| 2 | Ek Din Aap Yun Humko Mil Jayenge  Alka Yagnik and ... | Alka yagnik | Happy |
| 3 | To my youth.mp3 | Bolb4 | Sad |
| 4 | Orange Flower (You Complete Me).mp3 | Enhypen | Happy |
| 5 | SHOUT OUT.mp3 | Enhypen | Chill |
| 6 | PYTHON.mp3 | Got7 | Workout |
| 7 | Beginning Middle End.mp3 | Leah Nobel | Happy |
| 8 | Starry Starry Night from Loving Vincent (Original ... | Lianne La Havas | Sad |
| 9 | Let Down.mp3 | Radiohead | Sad |
| 10 | Hazel Eyes.mp3 | Sabrina Jordan | Sad |
| 11 | Way Back Home.mp3 | Shaun | Workout |
| 12 | Aaoge Tum Kabhi.mp3 | The Local Train | Happy |
| 13 | High.mp3 | THEIA | Workout |
| 14 | Make Your Own Kind Of Music.mp3 | Unknown Artist | Happy |
| 15 | Yellow Paper Daisy.mp3 | When Chai met toast | Chill |
| 16 | Sparkle.mp3 | YOur name | Sad |

Your Song History

Clear History

No Songs Playing

*Figure A.1: MainWindow with Sort by title*


MainWindow

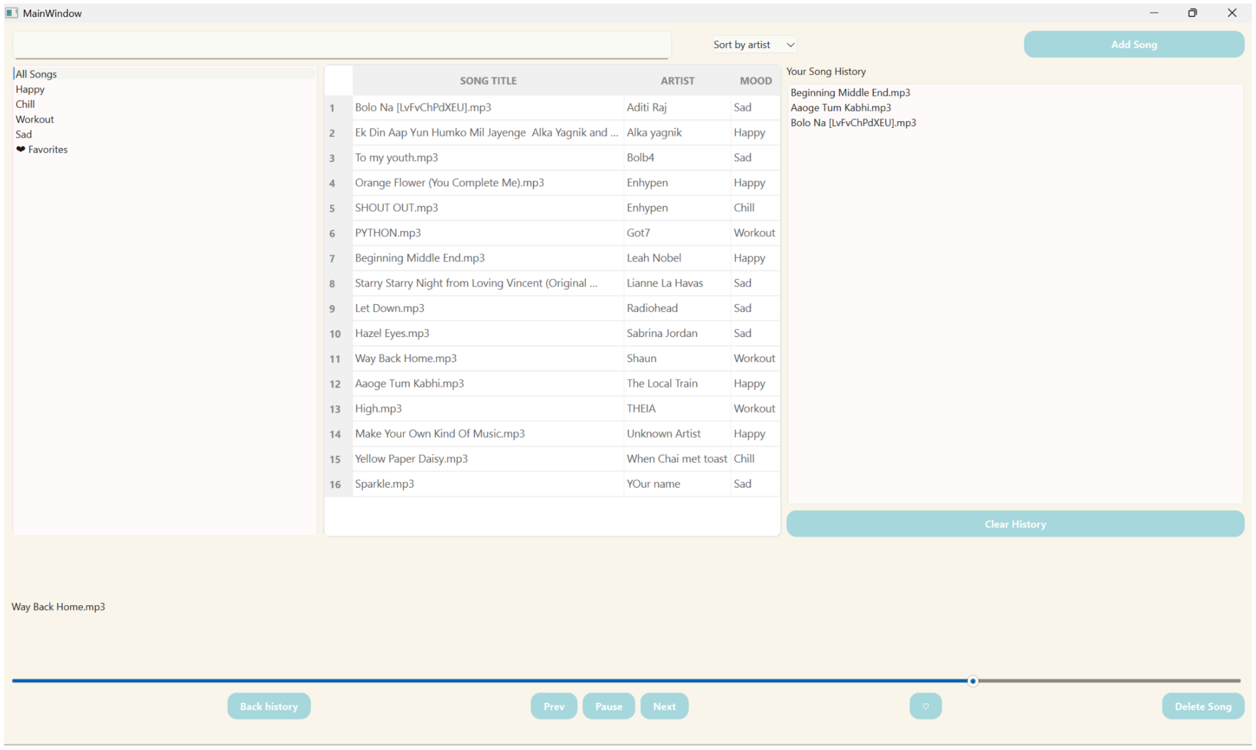
way

Sort by title

- All Songs
- Happy
- Chill
- Workout
- Sad
- ❤ Favorites

| | SONG TITLE | ARTIST | MOOD |
|---|---|---|---|
| 1 | Way Back Home.mp3 | Shaun | Workout |

*Figure A.2: Searching*



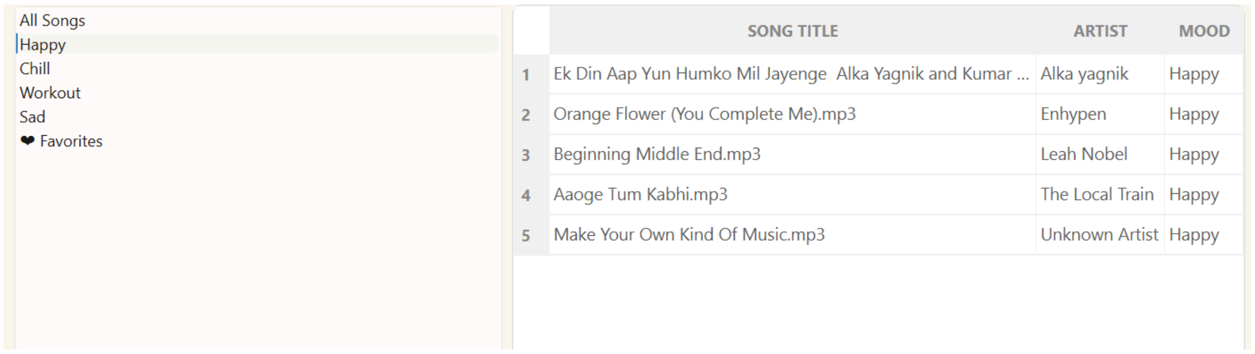
| | SONG TITLE | ARTIST | MOOD |
|---|---|---|---|
| 1 | Bolo Na [LvFvChPdXEU].mp3 | Aditi Raj | Sad |
| 2 | Ek Din Aap Yun Humko Mil Jayenge  Alka Yagnik and ... | Alka yagnik | Happy |
| 3 | To my youth.mp3 | Bolb4 | Sad |
| 4 | Orange Flower (You Complete Me).mp3 | Enhypen | Happy |
| 5 | SHOUT OUT.mp3 | Enhypen | Chill |
| 6 | PYTHON.mp3 | Got7 | Workout |
| 7 | Beginning Middle End.mp3 | Leah Nobel | Happy |
| 8 | Starry Starry Night from Loving Vincent (Original ... | Lianne La Havas | Sad |
| 9 | Let Down.mp3 | Radiohead | Sad |
| 10 | Hazel Eyes.mp3 | Sabrina Jordan | Sad |
| 11 | Way Back Home.mp3 | Shaun | Workout |
| 12 | Aaoge Tum Kabhi.mp3 | The Local Train | Happy |
| 13 | High.mp3 | THEIA | Workout |
| 14 | Make Your Own Kind Of Music.mp3 | Unknown Artist | Happy |
| 15 | Yellow Paper Daisy.mp3 | When Chai met toast | Chill |
| 16 | Sparkle.mp3 | YOur name | Sad |

Your Song History

Beginning Middle End.mp3
Aaoge Tum Kabhi.mp3
Bolo Na [LvFvChPdXEU].mp3

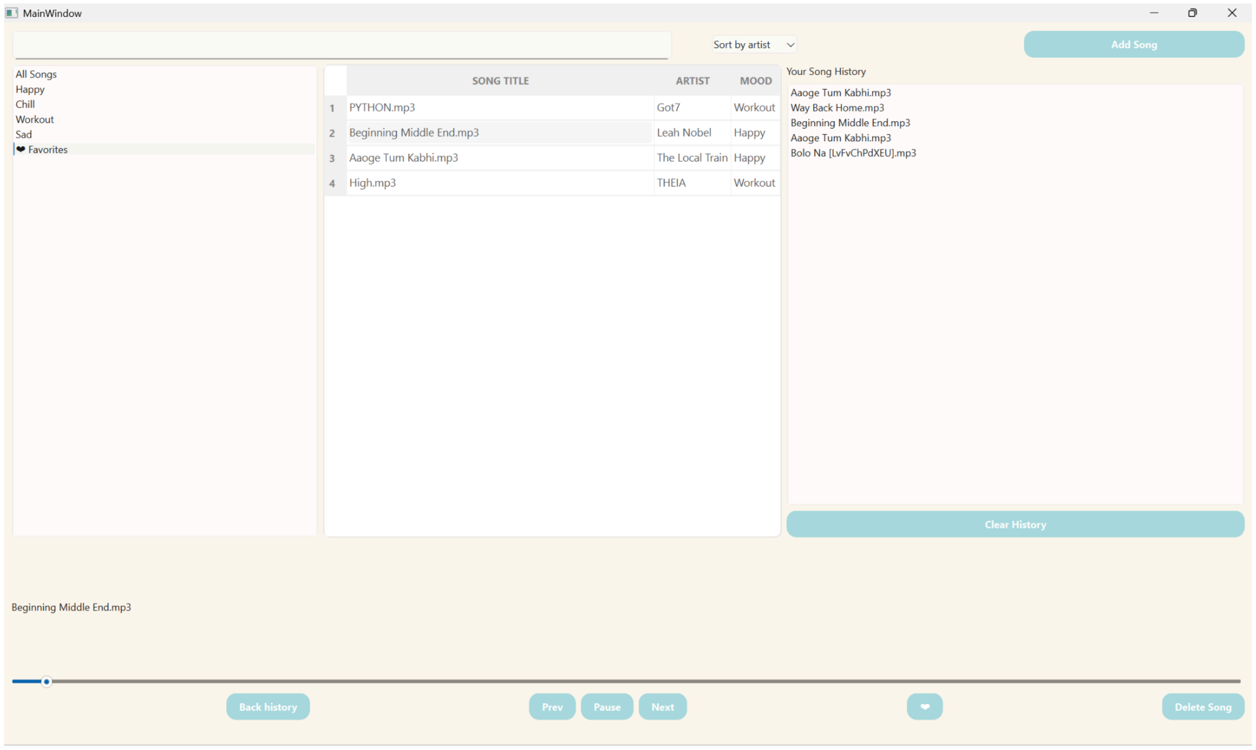
Way Back Home.mp3

*Figure A.4: MainWindow with Artist Sort and Song history*



| | SONG TITLE | ARTIST | MOOD |
|---|---|---|---|
| 1 | Ek Din Aap Yun Humko Mil Jayenge  Alka Yagnik and Kumar ... | Alka yagnik | Happy |
| 2 | Orange Flower (You Complete Me).mp3 | Enhypen | Happy |
| 3 | Beginning Middle End.mp3 | Leah Nobel | Happy |
| 4 | Aaoge Tum Kabhi.mp3 | The Local Train | Happy |
| 5 | Make Your Own Kind Of Music.mp3 | Unknown Artist | Happy |

*Figure A.4: Happy Mood Filtering*



| | SONG TITLE | ARTIST | MOOD |
|---|---|---|---|
| 1 | PYTHON.mp3 | Got7 | Workout |
| 2 | Beginning Middle End.mp3 | Leah Nobel | Happy |
| 3 | Aaoge Tum Kabhi.mp3 | The Local Train | Happy |
| 4 | High.mp3 | THEIA | Workout |

Aaoge Tum Kabhi.mp3
Way Back Home.mp3
Beginning Middle End.mp3
Aaoge Tum Kabhi.mp3
Bolo Na [LvFvChPdXEU].mp3

Beginning Middle End.mp3

*Figure A.5: MainWindow with favourites list playing*