Table of Contents

Table of Figures

1 Introduction

1.1 Background

Music plays a pivotal role in entertainment, productivity, and daily life, with streaming services like Spotify and Apple Music revolutionizing how users interact with audio content. Central to these platforms is playlist management, which relies on efficient data structures and algorithms (DSA) to handle operations such as adding songs, shuffling, and maintaining playback queues (School, n.d.). Key DSA concepts include lists for organized storage, stacks for last-in-first-out (LIFO) operations like undo features, queues for first-in-first-out (FIFO) processing such as playback order, linked lists for dynamic insertion and deletion, and recursion for breaking down complex problems like shuffling into simpler subproblems. (Patel, 2023)

This project develops a command-line interface (CLI) simulation of a music playlist manager, emphasizing DSA implementation over actual audio playback. By focusing on these foundational concepts, the application serves as an educational tool for computer science students, illustrating how theoretical DSA principles translate into practical software solutions. Unlike commercial music players that abstract these complexities, this simulation exposes the underlying mechanics, fostering a deeper understanding of data management in real-world applications. (Geeksforgeeks, 2025)

1.2 Problem Statement

Modern music players often conceal the intricate DSA required for features like playlist curation, random shuffling, playback history tracking, and priority queues for user-voted songs. This abstraction can hinder DSA learners from connecting classroom theory, such as the efficiency of linked lists in navigation or the recursive nature of algorithms, to tangible examples. The challenge is to create an accessible tool that demystifies these concepts while providing functional playlist management.

This project addresses this by simulating a music playlist manager where each feature is deliberately built using specific data structures: lists for filtering, recursion for shuffling,

linked lists for playlist representation (TutorialPoint), stacks for history (GeeksforGeeks, 2025), and queues for playback prioritization (Programiz, 2023). By doing so, it bridges the gap between abstract DSA and practical implementation, enabling users to experiment with and observe these structures in action.

1.3 Scope of Work

The project focuses on:

- Implementing a CLI-based music playlist manager.
- Incorporating DSA concepts: lists for filtering, recursion for shuffling, linked lists for playlists, and stacks/queues for playback features.
- Supporting playlist creation, song management, filtering, shuffling, and prioritized playback.
- Persisting data using JSON for cross-session continuity.
- Excluding actual audio playback, focusing on simulation.

Exclusions include real audio playback (focusing solely on simulation), graphical user interfaces (GUIs), and advanced integrations like external databases or cloud services. This scope ensures a focused, educational prototype that highlights DSA efficiency in a controlled environment.

2 Aims and Objectives

The main aim of this project is to design and implement a CLI-based Music Playlist Manager that demonstrates the practical application of core Data Structures and Algorithms (DSA) such as lists, stacks, queues, recursion, and linked lists.

The objectives of this project are:

- To simulate playlist creation, management, and navigation using appropriate data structures.
- To apply queues for play-next functionality and stacks for listening history.
- To implement recursion for shuffle and playlist traversal.
- To use linked lists for representing and navigating playlists.
- To provide a user-friendly menu-driven CLI interface that allows multiple playlist management.
- To reinforce theoretical DSA concepts by integrating them into a real-world simulation problem.

3 Expected Outcomes and Deliverables

3.1 Expected Outcomes

The project will yield a robust CLI application with the following capabilities:

- Comprehensive playlist management: Users can add/delete songs, create/switch playlists, and filter by artist using list comprehensions for efficient querying.
- Advanced features: Recursive shuffling to randomize song order, a queue for immediate "play next" songs, a priority queue simulation for "party mode" upvoting, and a stack-based history to track recently played tracks in reverse order.
- Simulated playback: Displays "now playing" without audio, drawing from queues or playlists while logging to history.
- Persistence: Playlists saved in JSON format, loaded on startup to preserve state.
- Educational value: Demonstrates DSA in context, such as the advantages of doubly linked lists for bidirectional navigation and stacks for undo-like history retrieval.

Overall, it will provide an efficient, modular system that highlights DSA's role in software development.

3.2 Deliverables

The project provides following deliverables:

- Python source code of the CLI music playlist manager (music_playlist_manager.py).
- JSON file (playlistsCLI.json) for data persistence.
- Documentation report explaining design and DSA implementations.
- Screenshots of CLI outputs to act as the “dashboard”.

4 Project Risks and Contingency Plans

4.1 Project Risks

Different risks can be observed in the project.

- Time constraints lead to incomplete features, such as advanced queue prioritization.
- Challenges in implementing linked list traversal correctly in Python, potentially causing pointer errors.
- Logical flaws in recursion, like exceeding Python's default recursion limit (1000) for large playlists, result in stack overflows.
- CLI's inherent lack of intuitiveness compared to GUIs risks user confusion.
- JSON file issues, such as corruption during saves, can lead to data loss.
- Input validation gaps are causing crashes from invalid user entries.
- Scalability limits in the list-based party queue, inefficient for frequent shifts (O(n) time).

4.2 Contingency Plans

The contigency plan to overcome the project risk can be as following:

- Prioritize **core features** first (add/delete, playlist creation, linked list navigation).
- Test recursive functions thoroughly with small datasets.
- Provide clear **menus and instructions** to make CLI more user-friendly.
- Maintain backups of working versions of code.
- Prioritize essential features (e.g., playlist CRUD operations) and modularise code for phased development.
- Conduct thorough unit testing on recursive functions with small inputs, increasing the recursion limit if needed via sys. setrecursionlimit.
- Enhance CLI with clear prompts, error messages, and input validation using try-except blocks.
- Maintain code backups via version control (e.g., Git) and add JSON validation on load.
- Switch to an iterative shuffle algorithm for large datasets to avoid recursion depth issues.

- Create JSON backups before overwrites and implement data recovery mechanisms.
- Upgrade party queue to a heap-based structure (e.g., heapq) for O(log n) operations if scalability becomes critical.

5 Methodology

5.1 Approach/ Design

The system is designed modularly, aligning with a five-week DSA learning structure:

- **Lists**: Use lists for song filtering and reporting.
- **Recursion**: Implement recursive playlist shuffling.
- **Functions & Loops**: Build a menu-driven CLI with functions for user interactions.
- **Linked Lists**: Use doubly linked lists for playlists to support efficient navigation and modifications.
- **Stacks & Queues**: Implement a deque for “play next,” a list-based priority queue for “party mode,” and a stack for listening history.

The design prioritizes simplicity, using Python’s standard libraries (random, json, collections.deque, os) for portability.

Python's standard libraries ensure portability: random for shuffling, json for serialization, collections.deque for efficient queues, and os for file handling. This simplicity avoids dependencies, focusing on core DSA demonstrations.

5.2 Implementation Steps

The steps implemented for building this project are:

- Step 1: Define a data model for songs (title, artist).
- Step 2: Load songs into a list.
- Step 3: Implement filtering by artist.
- Step 4: Implement recursive shuffle.
- Step 5: Create a menu system for user interaction.
- Step 6: Build a playlist using linked list nodes.
- Step 7: Implement a stack for history.
- Step 8: Implement a queue and a priority queue for play-next.
- Step 9: Test and debug.
- Step 10: Prepare documentation and screenshots.

5.3 Data / Testing

- **Data**: Playlists stored in playlistsCLI.json as a dictionary of playlist names mapping to lists of song dictionaries ({"title": "...", "artist": "..."}).
- **Testing**:
    - Unit tests for linked list operations (add/remove songs).
    - Manual testing of CLI menu for usability.
    - JSON file read/write validation to ensure persistence.
    - Edge cases: Empty playlists, invalid inputs, large playlists (e.g., 1000 songs).

6 Resources and Requirements

6.1 Hardware Requirements

The basic hardware requirements are:

- Processor: Intel i3 or equivalent for basic execution.
- RAM: Minimum 4GB to handle recursion and data structures.
- Storage: At least 500MB free for code and JSON files.
- Compatible with standard systems running Python, no specialized hardware needed.

6.2 Software Requirements

The basic software requirements are:

- Python 3.x, leveraging built-in modules for DSA implementations.
- OS: Windows, macOS, or Linux for cross-platform compatibility.
- IDE/Text Editor: VS Code or PyCharm for development and debugging.

6.3 Other Resources

The other sersources are:

- Documentation tools like MS Word or Google Docs for report creation.
- Terminal for CLI testing and execution.

7 Work Breakdown Structure and Milestones

7.1 Work Breakdown Structure

The WBS for carrying out this project is:

- Data modelling and basic structures (lists, nodes).
- Core DSA implementations (recursion, linked lists, stacks, queues).
- User interface development (CLI menu, inputs).
- Persistence and integration (JSON handling).
- Testing and validation.
- Documentation preparation.

7.2 Milestones

- **August 8, 2025**: Lists & Filtering (Filter songs by artist implemented).
- **August 13, 2025**: Recursive Shuffle (Recursive shuffle function completed).
- **August 20, 2025**: CLI Menu (Functional menu with add/remove song options).
- **August 22, 2025**: Linked List Playlist (Doubly linked list for playlist management).
- **August 24, 2025**: Playback Features (Play next, party mode, and history implemented).
- **August 26, 2025**: Integration & Persistence (JSON saving/loading, full system testing).
- **August 28, 2025**: Documentation (Comprehensive project documentation completed).

8 Tools and Technologies Used

The tools and technologies used for this project are:

- **Programming Language**: Python 3, chosen for its simplicity and rich standard library.
- **Data Structures**: Lists (filtering), Linked Lists (playlists), Stacks (history), Queues (playback), Recursion (shuffling).
- **Environment**: CLI for text-based interaction.
- **Libraries**: random, json, collections.deque, os.
- **Tools**: VS Code for coding, word processors for docs.
- No external dependencies to maintain portability.

9 System Dashboard & Screenshot

9.1 Main Menu

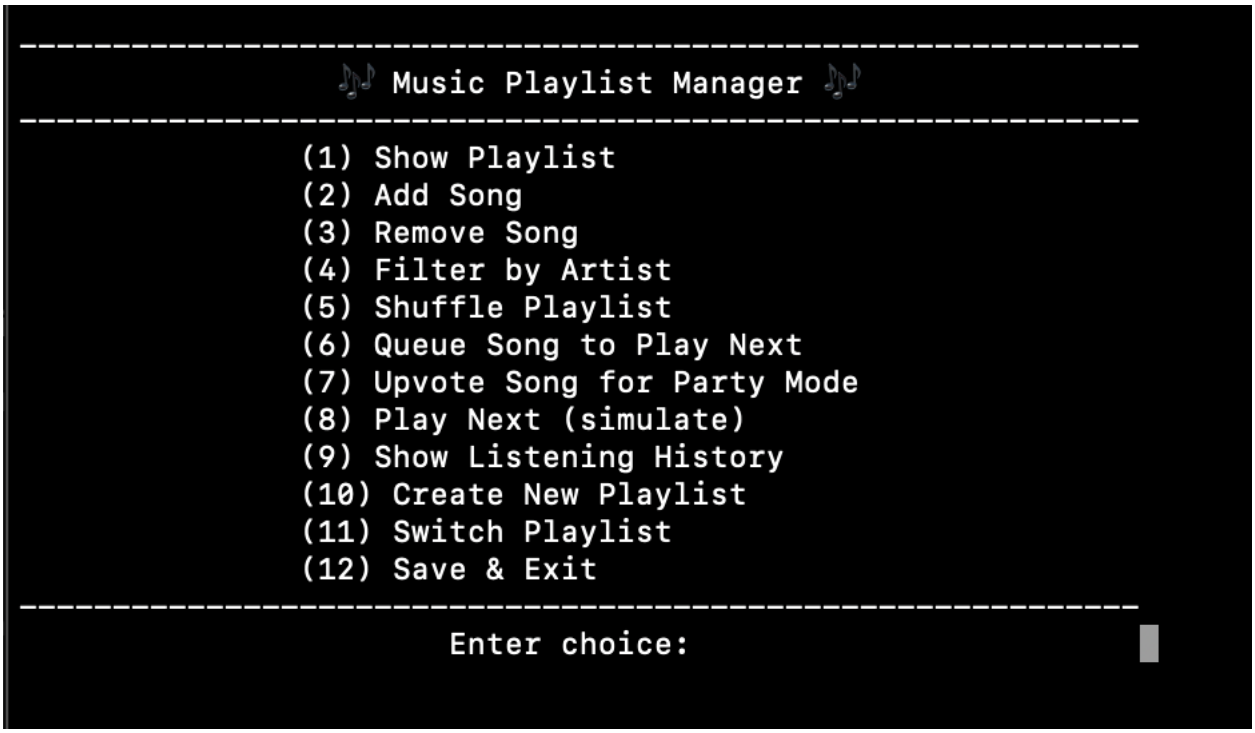

Figure 1 Main Menu

The main menu of the Music Playlist Manager system displays a bordered, enclosed design, where options are listed and centred, with the console and choice entered by the user.

9.2 Switch Playlist

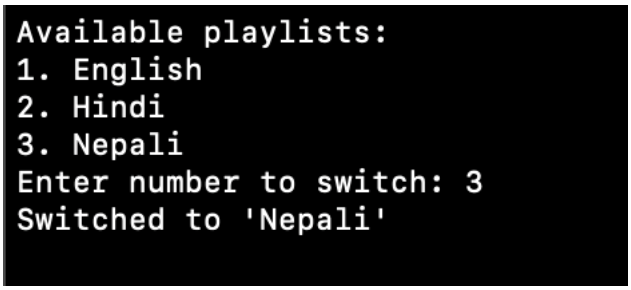

Figure 2 Switch Playlist

Option 11, Switch Playlist, shows the available playlists, and the desired playlist to be switched is entered. Here, the ‘Nepali’ playlist is switched.

9.3 Playlist Display

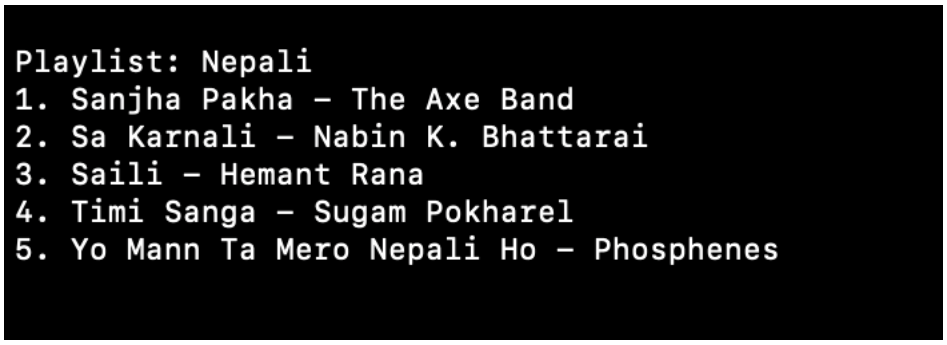

Figure 3 Showing Playlist

Option 1 shows the playlist and the song under it.

9.4 Adding song

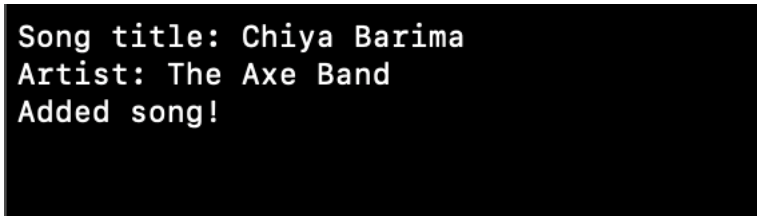

Figure 4 Adding Song

Option 2 allows the user to add the song, where the user must enter the song title and the artist's name.

Figure 5 Song added to the Playlist

Here, it shows the song has been added.

9.5 Remove Song

```
------------------------------------------------------------
                🎶 Music Playlist Manager 🎶
------------------------------------------------------------
               (1) Show Playlist
               (2) Add Song
               (3) Remove Song
               (4) Filter by Artist
               (5) Shuffle Playlist
               (6) Queue Song to Play Next
               (7) Upvote Song for Party Mode
               (8) Play Next (simulate)
               (9) Show Listening History
               (10) Create New Playlist
               (11) Switch Playlist
               (12) Save & Exit
------------------------------------------------------------
                       Enter choice:                       3
------------------------------------------------------------

Playlist: Nepali
1. Timi Sanga - Sugam Pokharel
2. Saili - Hemant Rana
3. Sa Karnali - Nabin K. Bhattarai
4. Yo Mann Ta Mero Nepali Ho - Phosphenes
5. Chiya Barima - The Axe Band
6. Sanjha Pakha - The Axe Band

Song number to remove: 4
Removed: Yo Mann Ta Mero Nepali Ho by Phosphenes
```

Figure 6 Remove Song

```
Playlist: Nepali
1. Timi Sanga - Sugam Pokharel
2. Saili - Hemant Rana
3. Sa Karnali - Nabin K. Bhattarai
4. Chiya Barima - The Axe Band
5. Sanjha Pakha - The Axe Band
```

Option 3 removes the song of the user's choice. The playlists' songs will be shown, and the desired corresponding number of the song is entered and removed.

9.6 Filter songs by artist

```
Artist name: The Axe Band
Songs by The Axe Band:
1. Sanjha Pakha
2. Chiya Barima
```

Figure 7 Filter songs by artist

Option 4 filters the song by the artist's name. As per the entered name of the artist, the available songs will be displayed.

9.7 Shuffle Playlist

```
                (7) Upvote Song for Party Mode
                (8) Play Next (simulate)
                (9) Show Listening History
                (10) Create New Playlist
                (11) Switch Playlist
                (12) Save & Exit
------------------------------------------------------------
                        Enter choice:                      5
------------------------------------------------------------
Playlist shuffled!
```

Figure 8 Shuffle Playlist

Option 5 shuffles the playlist songs.

```
Playlist: Nepali
1. Timi Sanga - Sugam Pokharel
2. Saili - Hemant Rana
3. Sa Karnali - Nabin K. Bhattarai
4. Yo Mann Ta Mero Nepali Ho - Phosphenes
5. Chiya Barima - The Axe Band
6. Sanjha Pakha - The Axe Band
```

9.8 Queue Song to Play Next

Figure 9 Queue song to play next

Option 6 queues the next song to play. The desired song is entered, and the song is queued to play next.

9.9 Play Next Song

Figure 10 Play next song

Option 8 plays the song that is next or queued.

9.10 Listening History

Figure 11 Listening History

Option 9 shows the history of the song that has been played.

9.11 Upvote song

Figure 12 Upvote Song

Option 7 upvoted the song for party mode. This enables the song to play next directly.

9.12 Create New Playlist

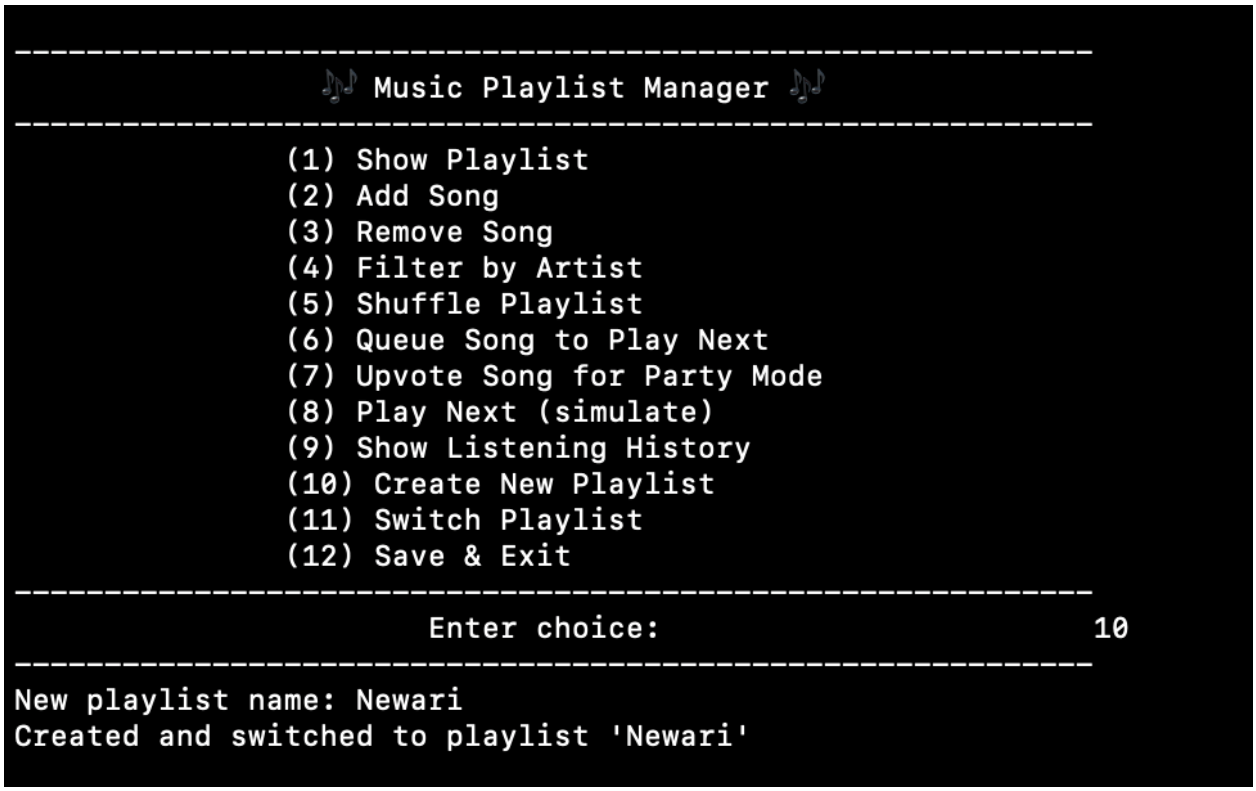

Figure 13 Create a new playlist

Option 10 created a new playlist with user‘s desired name.

9.13 Saving and Exiting

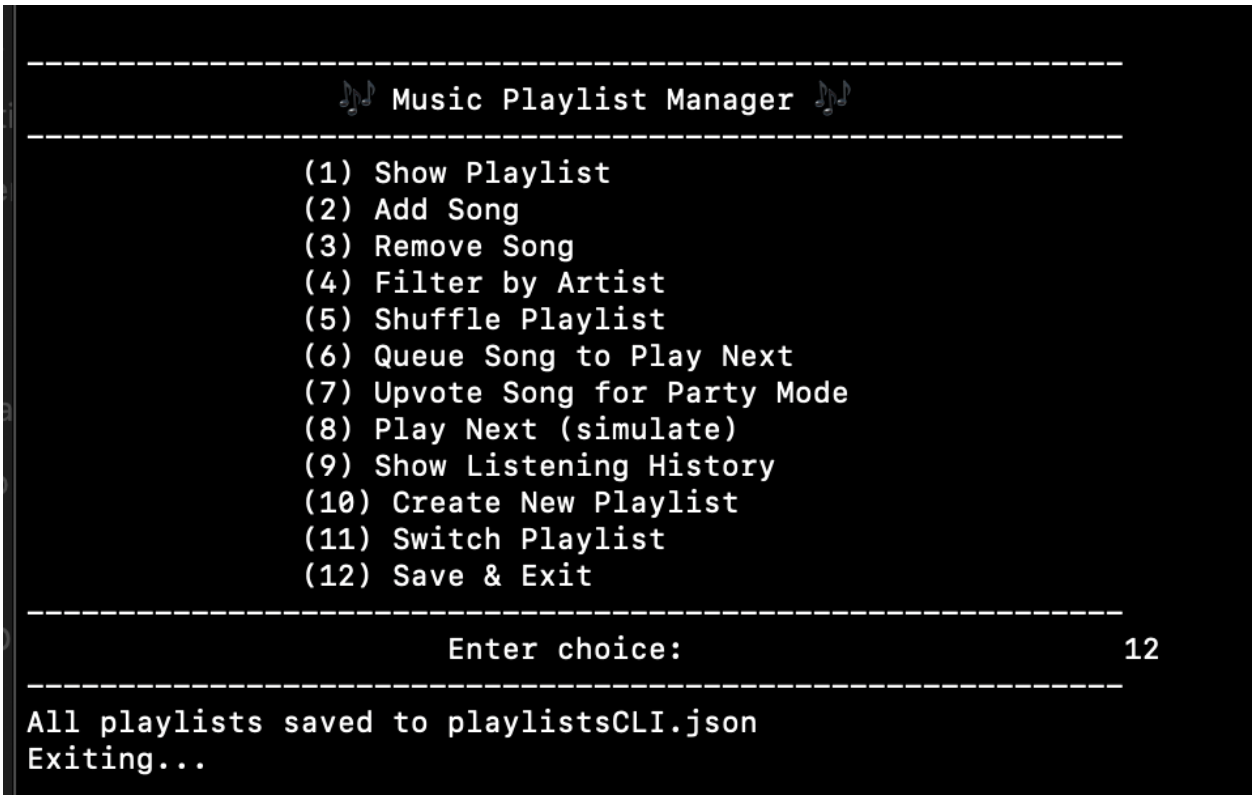

Figure 14 Saving and Exiting

Option 12 saves the changes made (add, remove, playlist creation) to the json file and exits the system.

9.14 PlaylistCLI.json

Figure 15 JSON Sample

This is a sample of the JSON file, which shows how the playlist with songs under them are listed.

10 Conclusion and Future Scope

This project successfully demonstrates how core Data Structures and Algorithms (DSA) can be applied to solve real-world inspired problems through a Music Playlist Manager. By simulating playlist management in a command-line interface, the project highlights the role of lists, stacks, queues, recursion, and linked lists in implementing essential features such as playlist creation, shuffle functionality, play-next queue, history tracking, and priority-based playback.

The implementation reinforces theoretical knowledge gained in the classroom by mapping abstract DSA concepts to a practical, relatable problem domain. This project also serves as an effective learning tool for understanding how multiple data structures interact to build a functional system.

Future Scope

While the current project is limited to a CLI-based simulation, there is significant potential for further enhancement:

- GUI/Web Interface: Extend the project to a Tkinter GUI or a Flask/Django web application for a better user experience.
- Integration with Actual Music Files: Link with pygame or a media library to play real audio files.
- Database Support: Store songs, playlists, and history persistently using SQLite or PostgreSQL.
- User Profiles: Allow multiple users with personalized playlists and histories.
- Recommendation System: Add AI/ML-based suggestions (e.g., most played songs, similar artists).
- Cloud Deployment: Deploy as a web app so multiple users can interact with playlists in real time.

11 Bibliography

Patel, A., 2023. *Mastering Stacks and Queues: Understanding LIFO and FIFO Data Structures.* [Online] Available at: https://blog.bitsrc.io/mastering-stacks-and-queues-understanding-lifo-and-fifo-data-structures-531c8d17194c
[Accessed 26 Aug 2025].

School, W., n.d. *Introduction to Data Structures and Algorithms.* [Online] Available at: https://www.w3schools.com/dsa/dsa_intro.php

Geeksforgeeks, 2025. *Recursive Algorithms.* [Online] Available at: https://www.geeksforgeeks.org/dsa/recursion-algorithms/ [Accessed 26 Aug 2025].

TutorialPoint, n.d. *Doubly Linked List Data Structure.* [Online] Available at: https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm
[Accessed 26 Aug 2025].

GeeksforGeeks, 2025. *Stack Data Structure.* [Online] Available at: https://www.geeksforgeeks.org/dsa/stack-data-structure/ [Accessed 26 Aug 2025].

Programiz, 2023. *Queue Data Structure.* [Online] Available at: https://www.programiz.com/dsa/queue [Accessed 26 Aug 2025].