Minor Project – 01

Synopsis

On

EchoBeat - Music Player

Submitted by

Sneha Naraniwal	R2142221360
Uday Surothiya	R2142220683
Priyanshu Chauhan	R2142220605
Tarushi Chaudhary	R2142220369

Under the guidance of

Dr. Avita Katal

Assistant professor (SG)

Cloud Software Operations Cluster

Cloud Software Operations

School of Computer Science

Aug – Nov, 2024

Dr. Avita Katal
(Project guide)

Dr. Hitesh Kumar Sharma
(Cluster Head)

1. Abstract

Music streaming services increasingly rely on playlists, often manually curated, which becomes challenging with vast music libraries. This research explores techniques to streamline playlist creation using:

1. **Playing Coefficient**: This metric measures how frequently users play specific tracks, serving as implicit feedback to understand preferences without explicit ratings. It addresses data sparsity and improves recommendation accuracy by focusing on actual listening habits, thus solving issues like the cold-start and gray-sheep problems.
2. **Clustering**: To overcome inefficiencies in collaborative filtering with extensive music libraries, we propose using clustering algorithms. These algorithms group similar tracks and simplify recommendation processes by reducing dimensionality, enhancing speed while maintaining recommendation quality.
3. **Matrix Factorization**: Similar to clustering, matrix factorization techniques help in handling large datasets by decomposing user-item interactions into latent factors, thereby improving the efficiency and quality of recommendations.

These techniques collectively enhance playlist generation in music streaming services, making the process more manageable and personalized.

2. Introduction

Music player applications are a staple in modern software, providing users with the ability to manage and play their music collections. While many existing music players offer basic functionality, they often lack efficiency in handling operations such as searching for songs, managing playlists, or dynamically arranging tracks.

This project focuses on developing a simple yet effective music player that utilizes Data Structures and Algorithms (DSA) to enhance performance and usability. By employing techniques such as efficient searching, sorting, and playlist management, the music player will demonstrate how tailored DSA solutions can improve the user experience even in a small-scale application. The goal is to create a lightweight and responsive music player that provides smooth navigation and interaction with music files, offering a practical demonstration of DSA concepts in action. This project is ideal for educational purposes, showcasing how fundamental DSA principles can be applied to solve everyday problems in software development.

3. Problem Statement

The current music player landscape often lacks the advanced performance features needed for efficient search and playlist management. Moreover, emerging artists struggle to gain visibility due to inefficient highlighting on existing platforms. This project aims to develop a music player that leverages Data Structures and Algorithms (DSA) to enhance performance and provide better support for independent artists.

4. Objectives

- The objective of this project is to design and implement a cutting-edge music player that harnesses the power of Data Structures and Algorithms (DSA) to significantly improve the user experience.
- This includes optimizing search algorithms for faster music retrieval, developing data structures for efficient playlist management, and ensuring smooth, lag-free interactions.
- Beyond just performance, the project also focuses on creating a platform that empowers emerging artists by providing them with increased visibility and better opportunities to connect with their audience.

5. Methodology

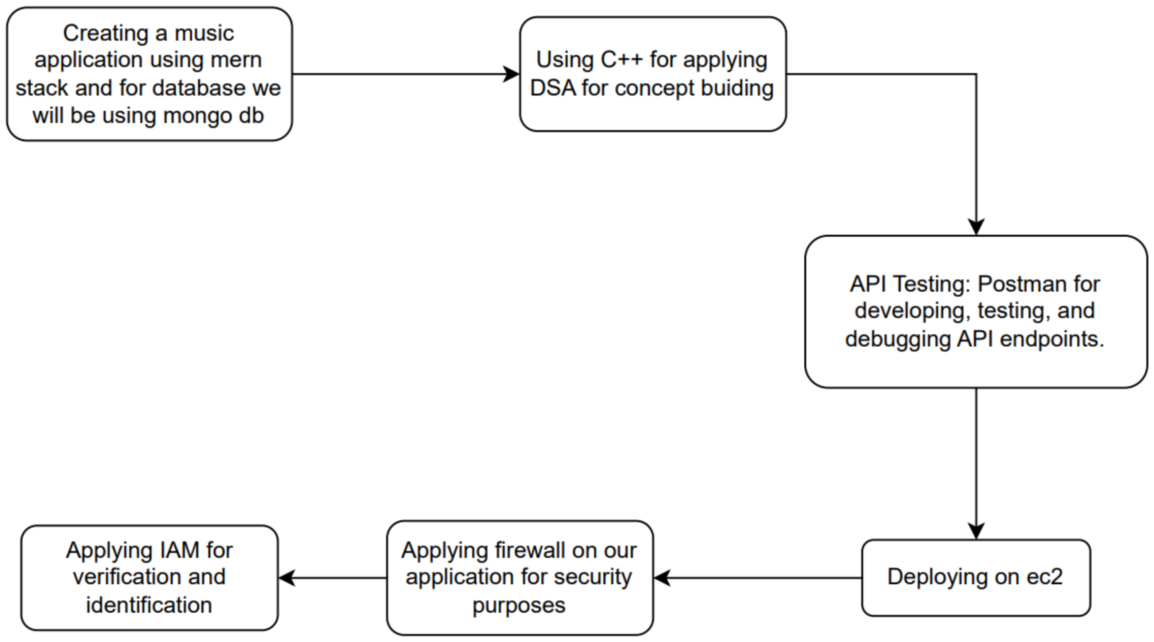

Creating a music application using mern stack and for database we will be using mongo db
Using C++ for applying DSA for concept buiding
API Testing: Postman for developing, testing, and debugging API endpoints.
Deploying on ec2
Applying firewall on our application for security purposes
Applying IAM for verification and identification

Figure 1

5.1 Data Structure

- Doubly Linked List
    - Playlist Management
        - We are inserting, removing songs with the help of it.
        - It allows users to navigate between songs (forward and backward) efficiently.
        - It allows users to access the last N played songs.
- Stack Data structure
    - Recently Played or History Management
        - A stack can be used to push and pop songs in a LIFO (Last-In-First-Out) manner, showing the most recently played song first.

- Trie and Hash Map:
    - Song Search:
        - A Trie is useful for fast search operations. For example, if a user wants to search for a song by title, the BST enables quick lookups by dividing the search space in half at each step.
        - A hash map can map song names or metadata (e.g., artist, genre) to song objects for faster lookup.

- Queue
    - Song Streaming
        - Use a queue to handle streaming data where chunks of songs are loaded and played in sequence
- Graphs
    - Songs Recommendation
        - A graph can represent users and songs as nodes, with edges between them indicating interactions such as song likes, plays, or ratings. Based on these relationships, the system can recommend new songs by finding similar users or songs with shared attributes.

5.2 Algorithmic Implementation

- Shuffling Algorithm
    - Fisher-Yates Shuffle: Implement the Fisher-Yates Shuffle algorithm to generate random permutations of music tracks. This well-known algorithm ensures a fair and unbiased shuffle, providing a random yet uniformly distributed arrangement of tracks for the user.
- Recommendation Algorithm
    - Collaborative Filtering: Employ collaborative filtering techniques to suggest songs based on the user's past listening experience. This approach will analyze user behavior and preferences to recommend music that aligns with their tastes.
- PageRank Algorithm
    - Adapted for Music Discovery: Utilize a variant of the PageRank algorithm to rank songs or artists based on their popularity and relevance within a network. This adaptation will help in discovering new music by highlighting popular or highly recommended tracks and artists.

5.3 Application Architecture

- Modular Architecture
    - User Interface (UI): Design a user-friendly interface that allows users to interact with the application seamlessly. The UI will include features for browsing the music library, managing playlists, and viewing recommendations.
    - Music Library Management: Develop a module for managing the music library, including adding, updating, and organizing music files. This module will integrate with the search and sorting functionalities.
    - Search and Retrieval: Implement functionalities for efficient searching and retrieval of music files, incorporating binary search and trie structures for optimized performance.

    - Playlist Management: Create a module for handling playlists, utilizing graph algorithms to dynamically generate and update playlists based on user preferences and listening history.

6. Technical Requirements

6.1 Hardware Requirements

- Processor:
    - Minimum: Intel Pentium or equivalent.
    - Recommended: Intel Core i3 or AMD Ryzen 3 for better performance.
- RAM:
    - Minimum: 4 GB.
    - Recommended: 8 GB to handle multiple tasks efficiently.
- Storage:
    - Minimum: 1 GB of free disk space for application files.
    - Recommended: 5 GB or more for storing music files and application cache.
- Input / Output Devices:
    - Keyboard and mouse.
    - Optional: Speakers or headphones for testing audio playback.

6.2 Software Requirements

- Operating System:
    - Windows: Windows 7 or later.

6.3 Development Environment:

- Compiler:
    - Windows: MinGW (GCC) or Microsoft Visual C++.
- IDE:
    - Windows: Code Blocks, Visual Studio Community Edition.
- Libraries and Frameworks:
    - Standard Template Library (STL): For data structures and algorithms.
- Version Control:
    - Git: For version control, especially if working on a team or planning to manage multiple versions.

6.4 Frontend

- Framework:
    - React: For building the user interface and managing the frontend logic.
    - Styling: Use CSS along with modern frameworks like Tailwind CSS or Bootstrap for advanced animations and dynamic styles.

  - User Engagement: Implement Toast notifications for real-time feedback and interactive features.
  - JavaScript: Core logic and API integration within the React framework.

6.5 Backend

- Node.js & JavaScript: Core logic and API development using Node.js.
- MongoDB: For efficient data storage and retrieval of user data, playlists, and song metadata.
- Input Validation: Use ZOD for schema declaration and robust input validation.

7. SWOT Analysis

Strengths:

- Our competitive advantage lies in the integration of Data Structures and Algorithms (DSA) to enhance music management functionality, providing efficient search, sorting, and playlist management capabilities that outperform standard music players in terms of speed and user experience.
- Our core music management functionalities, such as quick search and efficient playlist organization, are expected to perform well due to the use of optimized algorithms
- Additionally, features promoting new artists can attract a dedicated user base interested in discovering fresh music.

Weaknesses:

- Initially, features related to artist promotion may underperform if not marketed effectively to both users and artists. The music discovery algorithm might also need continuous refinement to effectively highlight emerging artists without overwhelming users

Opportunities:

- We can leverage machine learning to enhance the music recommendation engine, which could suggest songs based on user behavior and preferences.
- Integrating cloud storage could also allow users to access their music libraries from multiple devices

Threats:

- Changes in digital content licensing laws and regulations around music streaming could impact our ability to legally provide certain features.

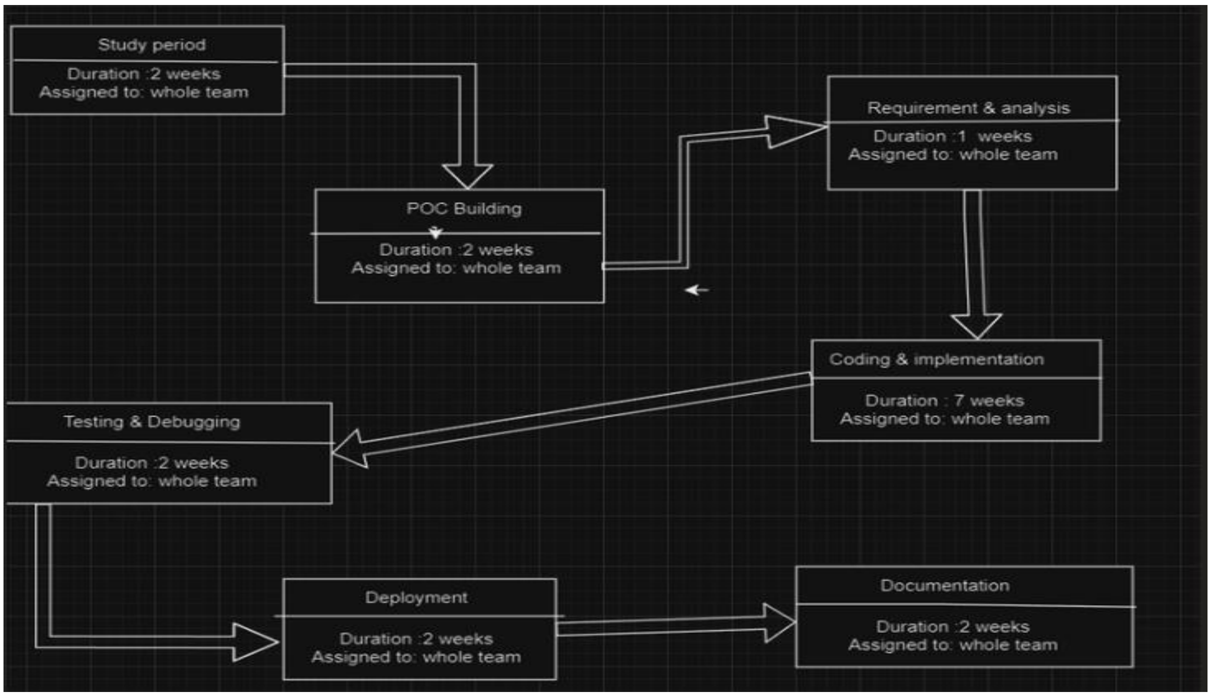
- Additionally, evolving data privacy laws may require us to update how we handle user data. Users also increasingly expect seamless integration with their social media and smart devices, which could pose a challenge if we do not adapt to these trends.

8. Scope of the project (area of application)

The project involves developing a music player application will use various data structures and algorithms to manage and optimize how music is stored, retrieved, and interacted with. The music player application focuses on advanced data management and user-centric features. Key aspects of the application include:

- **Music Playback**: Users can play, pause, skip, and shuffle music tracks with support for various audio formats.
- **Music Library Management**: The application will efficiently organize and manage a large collection of music files using algorithms for sorting and searching.
- **Playlist Management**: Users can create, modify, and manage playlists. The system will use algorithms to dynamically suggest playlists based on user preferences and listening history.
- **Personalized Recommendations**: The application will analyze user behavior to provide tailored song and artist recommendations.
- **User Interface**: A modern and responsive UI built with React, featuring interactive elements and real-time feedback.

9. Pert Chart



10. References

1. Aljunid, M. F., & Dh, M. (2020). An efficient deep learning approach for collaborative filtering recommender system. *Procedia Computer Science*, *171*, 829–836. https://doi.org/10.1016/j.procs.2020.04.090

2. Aaron, V. D. O., Dieleman, S., & Schrauwen, B. (2013). *Deep content-based music recommendation*. https://proceedings.neurips.cc/paper/2013/hash/b3ba8f1bee1238a2f37603d90b58898d-Abstract.html

3. Sánchez-Moreno, D., González, A. B. G., Vicente, M. D. M., Batista, V. F. L., & García, M. N. M. (2016). A collaborative filtering method for music recommendation using playing coefficients for artists and users. *Expert Systems With Applications*, *66*, 234–244. https://doi.org/10.1016/j.eswa.2016.09.019

4. Liao, C., & Lee, S. (2016). A clustering based approach to improving the efficiency of collaborative filtering recommendation. *Electronic Commerce Research and Applications*, *18*, 1–9. https://doi.org/10.1016/j.elerap.2016.05.001

5. *Automatic playlist generation using Convolutional Neural Networks and Recurrent Neural Networks*. (2019, September 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8903002?casa_token=57Ip3JWapxAAAAAA:OVtxa1fiUdIOsQx_t7gXHYa_x1sbWEL5fDTlwBrFRgPVR678YHTySbpCH5Gxhlgdmz0n8A_24etuVbs

6. *Music recommendation system using human activity recognition from accelerometer data*. (2019b, August 1). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/8743473

7. Park, S., Laplante, A., Lee, J. H., & Kaneshiro, B. (2019). *Tunes Together: Perception and experience of collaborative playlists*. https://www.semanticscholar.org/paper/Tunes-Together%3A-Perception-and-Experience-of-Park-Laplante/9761c7ce3db2ef0c421c37eec916c7c9904b5917

8. *Semantic Scholar topic*. (n.d.). https://topics-beta.apps.semanticscholar.org/topic/4577034884?corpusId=195720338