



MUSIC RECOMMENDATION SYSTEM

A Phase 1 Report on Building a Scalable Web Platform for Music Recommendation System Using MERN Stack

¹ Mr. Shravan, ²Dr. Shashidhar Kini K

¹Student, ²Professor & Head,

¹Department of Master of Computer Applications,

¹Srinivas Institute of Technology, Valachil Mangalore, Karnataka , India

Abstract : In the digital era, music consumption has evolved from physical formats to on-demand digital streaming platforms. Personalization has become a central component in enhancing user engagement and satisfaction. This paper presents a Phase 1 report on the development of a scalable, full-stack Music Recommendation System using the MERN stack (MongoDB, Express.js, React.js, Node.js). The platform aims to recommend personalized music to users based on listening behavior, mood, and preferences. Key features include user registration, playlist management, genre filtering, mood-based discovery, and basic collaborative filtering. The paper discusses the architecture, design methodology, backend logic, and testing outcomes of this intelligent music recommendation platform.

IndexTerms - Music recommendation, MERN stack, collaborative filtering, web application, user experience, personalization.

I.INTRODUCTION

As streaming services become the primary medium for music consumption, the need for intelligent music recommendation systems has grown significantly. Traditional methods of music discovery like radio or static playlists are being replaced by algorithm-driven engines that suggest content tailored to individual tastes. This shift demands highly responsive and user-centric platforms capable of learning user behavior and predicting future preferences.

Music Recommendation Systems (MRS) use user profiles, listening history, ratings, and collaborative filtering techniques to suggest tracks that align with user interests. By integrating advanced filtering techniques and user behavior analysis, platforms can improve listener retention and satisfaction. However, the development of such systems poses challenges in data handling, scalability, and responsiveness.

This paper outlines Phase 1 of the Music Recommendation System developed using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The goal is to provide a seamless music discovery experience while maintaining flexibility for future AI integration. The system supports functionalities like real-time playback, playlist creation, track rating, and genre/mood-based recommendations. It is optimized for both performance and usability, ensuring accessibility across devices.

The architecture supports enhancements like machine learning-based models, audio analysis integration, social features, and third-party music APIs. This research focuses on the initial planning, design, and implementation phases, ensuring the core recommendation logic and user experience foundation are robust and scalable.

The system architecture is designed with modularity in mind, allowing for easy expansion into areas such as machine learning-based recommendation models, sentiment and audio feature analysis (e.g., tempo, key, energy), and natural language processing for lyric analysis. Additionally, the system is built to support the integration of third-party music APIs (e.g., Spotify, Last.fm) for enriched content and broader track libraries.

Security, user privacy, and data integrity are also considered in the early stages of development, with secure user authentication and protected API routes implemented using JWT and role-based access control. The system also supports responsive design principles to ensure a consistent experience across web and mobile platforms.

II. EASE OF USE

The Music Recommendation System is built to be user-friendly and accessible to a wide demographic, including casual listeners, audiophiles, and tech-savvy users. The interface is designed using clean layouts, dynamic components, and responsive elements that function seamlessly across desktops, tablets, and smartphones.

Users can sign up or log in using standard or social authentication methods. Onboarding allows users to select genres, moods, or artists they like, which informs initial recommendation data. The dashboard provides access to trending music, personalized playlists, favorite tracks, and listening history. A minimalistic player interface allows users to play, pause, skip, or repeat tracks in real-time

Key usability features include an advanced search system with filters for genre, artist, mood, and popularity; customizable playlist management with drag-and-drop support; a rating system to fine-tune recommendations; mood-based music discovery through tags like "chill," "romantic," or "energetic"; and real-time notifications for new song or playlist updates. The frontend, built with React.js, connects to backend services through secure RESTful APIs using Axios and JWT-based token authentication, ensuring robust session management. Backend optimizations, including indexed MongoDB queries, data caching, and asynchronous task handling, support rapid response times and a smooth user experience even under heavy load. Future enhancements are planned, such as Progressive Web App (PWA) capabilities for offline use, background playback support, voice command integration, and multilingual interface options, all aimed at making music discovery more seamless, intelligent, and engaging..

The system dashboard serves as a centralized hub where users can explore trending music, receive AI-driven playlist suggestions, view their favorite and recently played tracks, and manage their personal content. A built-in media player remains fixed at the bottom of the interface, offering real-time controls for play, pause, skip, repeat, shuffle, and volume adjustments. The player supports visual progress bars, album art, and future extensibility for lyrics display or waveform visualizations. Key usability features further enhance the user experience, such as a powerful search bar with auto-complete and multi-criteria filtering (by genre, artist, mood, popularity), the ability to create and organize custom playlists via drag-and-drop functionality, and a rating system that refines the quality of future recommendations based on user feedback. Music discovery is enriched through mood-tagged content like "chill," "upbeat," "romantic," or "focus," allowing users to explore based on emotional context.

1. PREPARE YOUR PAPER BEFORE STYLING

Initial project planning emphasized a structured yet flexible approach to ensure clarity in both design and development. Beyond user surveys, stakeholder analysis was conducted to identify key roles, such as end-users, developers, system administrators, and potential third-party API providers, helping to align technical objectives with user-centric goals. The requirements-gathering process extended to online music communities, forums, and feedback from early testers, offering insight into both functional needs and non-functional expectations like performance, aesthetics, and privacy.

To translate insights into actionable designs, detailed wireframes and interactive UI prototypes were created using Figma, enabling early visualization of the user journey and interface flow. This collaborative prototyping stage involved multiple design iterations and usability walkthroughs to identify friction points before a single line of code was written. Special attention was paid to mapping out high-impact user scenarios, such as new user onboarding, first-time playlist creation, and mood-based browsing.

From a system architecture perspective, the application was divided into clearly defined microfunctional services. Express.js was chosen for its lightweight and scalable nature, enabling the creation of modular API endpoints that handled specific tasks like authentication, track metadata retrieval, and recommendation logic execution. MongoDB, managed through Mongoose ODM, supported the flexible storage of heterogeneous user data, including behavioral logs, audio preferences, and session interactions. Early schema validation was enforced to minimize data inconsistencies and reduce debugging overhead.

Security was embedded from the beginning as a design priority rather than a post-development patch. Passwords were securely hashed using bcrypt, and all authentication flows used JWTs with short-lived tokens and refresh strategies to mitigate risks. Additionally, HTTPS was enforced for all communications, and CORS policies were fine-tuned to prevent cross-site vulnerabilities during both development and deployment.

2. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even if they have been defined in the abstract. Do not use abbreviations in section titles unless unavoidable.

In this paper, the following abbreviations and acronyms are used:

- **API:** Application Programming Interface
- **MERN:** MongoDB, Express.js, React.js, Node.js
- **JWT:** JSON Web Token
- **UI:** User Interface
- **UX:** User Experience
- **DB:** Database

- **UAT:** User Acceptance Testing
- **CRUD:** Create, Read, Update, Delete
- **HTTPS:** HyperText Transfer Protocol Secure
- **CSS:** Cascading Style Sheets
- **HTML:** HyperText Markup Language
- **IDE:** Integrated Development Environment

III. RESEARCH METHODOLOGY

This section outlines the methodology adopted to develop a full-stack web platform for fostering engagement among alumni, students, and academic institutions. It encompasses the user population, data sources, theoretical framework, and the development and testing tools used throughout the initial phase of the alumni portal system implementation.

3.1 Study Population and Sample

The Music Recommendation System is purposefully designed to accommodate users across a broad demographic, accounting for diverse age groups, cultural backgrounds, and musical preferences. To ensure the platform meets real-world expectations, a mixed-method approach was employed in the user research phase. This involved conducting detailed surveys and structured interviews with over 100 participants, including casual listeners, audiophiles, amateur musicians, and playlist enthusiasts. Data was collected on preferred genres, discovery behaviors (e.g., mood-based, artist-based, or social sharing), and engagement patterns with existing streaming platforms. Participants also provided feedback on frustrations with current solutions—such as repetitive recommendations, poor personalization, and lack of cross-device syncing—which helped guide design improvements.

Based on analysis of the gathered data, three core user roles were defined: Listener, the general user focused on discovering and consuming music; Curator, a more active user who creates and shares playlists for themselves or others; and Admin, responsible for overseeing platform operations, managing content, and maintaining system integrity. These roles helped structure both the frontend experience and backend access levels, supporting tailored workflows and permission-based functionalities.

3.2 Data and Sources of Data

Survey Responses: Collected via Google Forms to gather genre preferences, artist likes, and feature expectations.

Public Datasets: Metadata and sample audio files from platforms like Last.fm and Million Song Dataset.

Mock Data: Synthetic data created to simulate listening sessions, user ratings, and song metadata.

UI Templates: Open-source React UI kits and audio player components from GitHub.

To support development and evaluation of the Music Recommendation System, a variety of data sources and development resources were utilized, both real and synthetic. Survey responses were collected using Google Forms, allowing the team to compile valuable insights from a broad audience on genre preferences, favorite artists, listening frequency, and expectations for personalized music features. The data helped validate the initial feature set and informed the design of the recommendation logic by highlighting commonly requested attributes like mood filters, smart playlists, and discovery feeds.

3.3 Theoretical framework

The system design is based on a combination of collaborative filtering principles and the Technology Acceptance Model (TAM). These frameworks ensured that both algorithmic relevance and user interaction factors were integrated into design decisions.

- **Collaborative Filtering:** Suggests songs based on the listening patterns and preferences of users with similar tastes. This method enables the system to surface new or niche music by identifying hidden patterns in group behavior, even without extensive metadata.
- **Content-Based Filtering (Planned):** Matches songs to users based on song metadata such as genre, mood, and artist. This future enhancement will allow recommendations even for new users (cold-start problem) and enable finer control over the personalization logic.
- **Technology Acceptance Model (TAM):** Evaluates perceived ease of use and perceived usefulness to predict user acceptance.

TAM principles were used to inform interface simplicity and emphasize core features that users found most beneficial in survey feedback.

These theoretical models shaped both the recommendation engine logic and the UI layout to drive engagement and long-term user retention.

3.4 Development Tools and Web Technologies

This phase integrates both conventional web technologies and modern development tools to implement and evaluate a scalable and responsive music recommendation system. The core objective is to develop a functional full-stack web application that enables users to discover new music, interact with personalized recommendations

3.4.1 Descriptive Statistics

Initial survey responses were analyzed using frequency charts and histograms to identify genre popularity and desired features. This analysis helped prioritize functionality such as mood filters and playlist customization in the system's MVP

3.4.2 Exploratory Feature Mapping

Wireframes were developed in Figma to visualize the flow of user interactions across devices. Miro boards illustrated key modules, user stories, and potential integration points for future AI-based enhancements.

3.4.3 Backend Architecture

Node.js and Express.js were used to build scalable API routes for authentication, music feeds, playlist actions, and feedback logging. Middleware functions handled request validation, JWT verification, and centralized error management for robust backend stability.

3.4.4 Database Design

MongoDB was chosen for its flexible document structure. Collections were created for users, events, jobs, and messages, with Mongoose schemas enforcing data consistency. MongoDB collections were organized into modular documents for users, songs, playlists, and ratings, ensuring flexible and scalable storage.

3.4.5 Frontend Framework

React.js powered the dynamic frontend, with reusable components for song display, playlist cards, filters, and media controls.

3.4.6 API Integration

Axios facilitated asynchronous data exchange between the frontend and backend services. JWT tokens were used to protect sensitive endpoints and enforce role-based access control (e.g., admin vs. listener).

3.4.7 Testing Strategies

Jest was employed for unit tests of backend routes, recommendation logic, and user session flows. Manual testing on Chrome, Firefox, iOS, and Android confirmed UI responsiveness, navigation accuracy, and audio playback stability.

3.4.8 Hybrid Deployment Plans

Initial deployment was configured using MongoDB Atlas for quick access and scaling. The structure supports containerization (Docker) and integration with institutional servers in future phases. The application was initially deployed on Heroku for rapid prototyping and demo access.

3.4.9 Evaluation Criteria

Performance and engagement metrics included API response time, session length, and the ratio of skipped to liked songs.

3.4.10 Feedback and Iteration

Beta testers provided structured input via in-app forms, covering everything from usability to music relevance.

IV. RESULTS AND DISCUSSION

4.1 Results of Descriptive Statics of Study Variables

Table 4.1: Descriptive Statics

Variable	Minimum	Maximum	Mean	Std. Deviation
Daily Listens	1	20	7.2	3.1
Playlist Created/User	0	10	3.6	2.5
Track Skip Rate (%)	5.00	75.00	33.40	15.80
Ratings Submitted/User	0	40	17.90	9.10
Recommendation Accuracy (%)	50.00	92.00	72.30	11.50

Table 4.1 These results indicate a promising level of user engagement across multiple dimensions. On average, users listened to 7.2 songs per day, suggesting consistent platform usage. The creation of playlists (mean of 3.6 per user) shows that users are actively organizing their music preferences, a behavior that supports the inclusion of customized playlist features. The track skip rate, averaging 33.4%, implies that while many recommendations are relevant, there is still room for improvement in matching music to user taste. A moderate standard deviation here indicates that skipping behavior varies, possibly depending on user familiarity or interest in suggested tracks. Users submitted an average of 17.9 ratings, indicating participation in feedback loops that are crucial for refining recommendation accuracy. Lastly, the recommendation accuracy—calculated based on user responses to suggested tracks—averaged 72.3%, reflecting a reasonably effective recommendation engine at this early stage of deployment. These metrics collectively validate the core functionality of the system and provide a strong foundation for implementing more advanced recommendation algorithms in future phases.

V. ACKNOWLEDGMENT

The author wishes to express sincere gratitude to the Project Guide and Head of the Department of MCA, Dr. Shashidhar Kini K, for his invaluable guidance, insightful suggestions, and continuous encouragement throughout the development of the Alumni Portal project. Special thanks are extended to the Principal, Dr. Shrinivasa Mayya D, for providing a supportive academic environment that made this work possible within the institution. The author also acknowledges the management of Srinivas Institute of Technology for their direct and indirect support during the course of this research.

Appreciation is due to all faculty members and non-teaching staff of the MCA department for their timely assistance, technical advice, and encouragement throughout the project phases. Lastly, the author is deeply thankful to parents, friends, and peers whose unwavering motivation and moral support were instrumental in completing this phase of the work successfully.

REFERENCES

- [1] Y. Deldjoo, M. Schedl, and P. Knees, "Content-driven Music Recommendation: Evolution, State of the Art, and Challenges," arXiv preprint arXiv:2107.11803, 2021.
- [2] K. Benzi, V. Kalofolias, X. Bresson, and P. Vandergheynst, "Song Recommendation with Non-Negative Matrix Factorization and Graph Total Variation," arXiv preprint arXiv:1601.01892, 2016.
- [3] K. Damak and O. Nasraoui, "SeER: An Explainable Deep Learning MIDI-based Hybrid Song Recommender System," arXiv preprint arXiv:1907.01640, 2019.
- [4] J. Lee, K. Lee, J. Park, J. Park, and J. Nam, "Deep Content-User Embedding Model for Music Recommendation," arXiv preprint arXiv:1807.06786, 2018.
- [5] V. Saini, "MusicBolt: A Simple Music Recommender System Built Using Collaborative Filtering and Spotify API," GitHub Repository, 2020.
- [6] K. Nair, "MusicAppMERN: A Music Recommender Web Application Using MERN Stack," GitHub Repository, 2021.
- [7] "Music Discovery App with MERN Stack," GeeksforGeeks, 2023.
- [8] P. Darshna, "Music Recommendation Based on Content and Collaborative Approach: Reducing Cold Start Problem," International Journal of Engineering Research & Technology (IJERT), vol. 7, no. 6, pp. 1–5, 2018.
- [9] J. H. Watson and L. A. Green, "User-Centric Design in Alumni Portals: Best Practices for User Engagement," Journal of Web Development and Design, vol. 29, no. 3, pp. 75–88, 2022.
- [10] A. Sharma and V. Thomas, "Data Privacy and Security Considerations in Educational Web Applications," Journal of Information Systems in Education, vol. 18, no. 4, pp. 221–228, 2021.