TABLE OF CONTENTS

Introduction

This project, *Rhythmic Tunes – A Music Companion*, is a frontend-based music player developed using React.js as the core framework, with Vite as the build tool and JSON Server serving as a lightweight backend. The primary motivation for building this project was to explore how modern web technologies can be combined to create an interactive, responsive, and user-friendly application that mimics the basic functionality of popular music platforms.

Music has become an inseparable part of our daily lives, and digital platforms have completely changed the way people access and experience it. Applications such as Spotify, YouTube Music, and Apple Music are widely used around the world because they provide simple interfaces and easy accessibility. Developing a music companion, even at a smaller scale, is therefore an excellent way to understand how these applications are structured internally and how core web development concepts come together to build something both functional and enjoyable.

The project focuses on providing a smooth user experience by implementing essential music player features. Users can view a playlist of available songs, check details such as the title, artist, and genre, and control playback through buttons such as play, pause, next, and previous. The data for all songs is stored in a JSON file (db.json), which contains metadata including the song name, artist, genre, album cover image, and audio file URL. This file is served using JSON Server, which acts as a simulated backend by exposing REST API endpoints. This setup allows the React frontend to fetch data dynamically, just as it would from a real database or external API, while keeping the overall project lightweight and easy to run.

From a learning perspective, this project highlights the use of component-based architecture in React. Instead of writing one large block of code, the application is divided into smaller, reusable components such as MusicPlayer, Controls, and Playlist. Each component is responsible for a specific part of the interface, making the codebase more modular, maintainable, and scalable. Additionally, React hooks such as useState and useEffect are used for managing the state of the application, such as keeping track of which song is currently playing and updating the UI whenever the state changes.

Another key highlight is the integration of Vite as the build tool. Unlike older tools like Create React App, Vite offers much faster project setup and development server speeds, improving developer productivity. It also ensures that the project remains efficient and optimized for future scalability. The inclusion of JSON Server further adds value by teaching how frontend applications can consume APIs and interact with structured data in a realistic way without needing to build a complex backend.

The overall aim of *Rhythmic Tunes – A Music Companion* is not only to demonstrate technical knowledge but also to deliver a working application that can serve as a foundation for future development. By extending this project, features like volume control, progress tracking, playlists, and even integration with real-world APIs (such as Spotify or YouTube Music) can be added. In its current form, however, the project already achieves its goal of showcasing the implementation of a modern frontend system that connects design, interactivity, and functionality.

In conclusion, this project combines creativity with technical learning. It bridges the gap between classroom concepts and real-world applications by allowing students to engage with tools and practices that are directly relevant to the industry. Through the development of *Rhythmic Tunes – A Music Companion*, a strong foundation has been built in React.js development, state management, API handling, and project organization, all of which are critical skills for any aspiring web developer.

RHYTHMIC TUNES –YOUR MELODIC COMPANION

Submitted by

[S.Rachitha]

[R.Vidhya Sree]

[P.Rajeshwari]

[A.Yuvasri]

Department of Computer Applications

[Dr. M. G. R. Janaki College of Arts and Science for Women]

Academic Year: 2025 – 2026

Project Overview

Purpose

The purpose of this project was to design and implement a functional music companion app using modern web technologies. The project provided a platform to explore React.js for frontend development and JSON Server for backend simulation. It allowed the practical application of programming concepts in a way that directly contributes to building a real-world style project.

Features

- **Dynamic Playlist**: Songs stored in db.json are fetched and displayed in the app.
- **Playback Controls**: Play, pause, next, and previous functions.
- **Metadata Display**: Displays song details such as title, artist, and genre.
- **Album Art Integration**: Each track shows its cover image for a better user experience.
- **Responsive Design**: Works smoothly across devices of different screen sizes.

Architecture

Component Structure

The application is divided into multiple reusable components:

- App Component: Root component that initializes the app.
- MusicPlayer Component: Handles the playback and displays details of the current song.
- Controls Component: Provides play, pause, next, and previous buttons.
- Playlist Component: Fetches and displays songs from JSON Server.

This modular design makes the project easier to maintain and extend in the future.

State Management

React hooks are used for managing state across the application:

- useState: For managing playback status and track index.
- useEffect: For fetching data from the JSON Server when the app loads.

Since the project is small, global state management tools like Redux or Context API were not required.

Routing

The application does not use routing, as the project scope was limited to a single functional page.

Setup Instructions

Prerequisites

- Node.js (v14 or higher)
- npm or yarn package manager
- Vite (for React setup)
- JSON Server (for backend simulation)

Installation

1. Clone the project repository.
2. Install dependencies using npm install.
3. Start JSON Server using:
4. npx json-server --watch db/db.json --port 3000
5. Run the frontend using:
6. npm run dev
7. Open the application in your browser using the Vite local server link.

Folder Structure

- **db/** → Contains db.json with all song details.
- **public/** → Stores static assets.
- **src/** → Main source code.
    - **components/** → Contains MusicPlayer, Controls, and Playlist.
    - **App.jsx** → Root component.
    - **index.css** → Styles.
    - **main.jsx** → Entry point.

Running the Application

1. Start JSON Server with the database.
2. Run the frontend development server using npm run dev.
3. Access the app in the browser at http://localhost:5173/.

Component Documentation

Key Components

- **MusicPlayer**: Core component that plays audio and shows details.
- **Controls**: Provides functional buttons linked to player logic.
- **Playlist**: Fetches songs from JSON Server and allows selection.

Reusable Components

- Buttons and styling elements reused across controls.

State Management

Global State

Not implemented due to small project size.

Local State

Local state was sufficient to handle:

- Track index
- Playback status (playing, paused)
- Fetched song list

User Interface

The user interface (UI) of *Rhythmic Tunes – A Music Companion* has been designed with a focus on simplicity, clarity, and ease of use. A music player is an application that users interact with frequently, and therefore, it must have an intuitive design that minimizes complexity while maximizing functionality. For this reason, the interface of the application has been kept minimalistic, ensuring that users can easily navigate through the playlist and control playback without facing unnecessary distractions.

The layout is divided into two major sections: the **playlist area** and the **music player area**. The playlist area displays all the available songs fetched from the JSON Server. Each entry in the playlist includes details such as the title of the song, the singer/artist name, the genre, and the album cover image. This not only provides important context about the song but also enhances the visual appeal of the application by incorporating graphics instead of relying only on text.

The music player area is where the currently selected track is displayed in detail. Here, the album cover image is shown prominently alongside the song title and artist. Below this section, the playback controls are placed in an easy-to-reach position. These include the **play/pause button**, **next track button**, and **previous track button**, all of which have been styled consistently to maintain uniformity. The buttons are designed with clear icons so that even new users can understand their purpose without needing any additional instructions.

Responsiveness has also been given importance while designing the interface. The use of Flexbox and Grid in CSS ensures that the layout adjusts automatically depending on the device size. On larger screens such as desktops or laptops, the playlist and player appear side by side for better visibility, while on smaller screens like smartphones, the layout stacks vertically to make navigation easier. This adaptability ensures that the application delivers a smooth experience across a wide range of devices.

Color choices in the UI were made to strike a balance between readability and attractiveness. A light background was selected to provide clarity, while important elements like the control buttons were highlighted using distinct colors so that they immediately catch the user's attention. Text was kept simple and legible with proper spacing to avoid visual clutter. Album cover images act as additional design elements, adding vibrancy to the interface without overcomplicating the overall theme.

Usability testing was carried out informally by interacting with the application multiple times. During this process, attention was given to whether users could quickly identify how to start playback, switch songs, and return to the playlist. Based on these tests, the final layout was refined to reduce unnecessary steps and make the flow as smooth as possible.

In summary, the user interface of *Rhythmic Tunes – A Music Companion* successfully combines **aesthetic appeal with functionality**. By keeping the design minimal yet engaging, the application ensures that users can enjoy their music without being overwhelmed by complex controls or unnecessary elements. The inclusion of responsive design principles also makes the app future-proof and adaptable, ensuring usability on multiple devices. With further enhancements, such as adding a progress bar, volume control, and themes (light/dark mode), the UI can be taken to an even higher level of interactivity and user satisfaction.

Styling

- **Plain CSS** was used for styling.
- Layout handled using Flexbox and Grid.
- A light theme with highlighted buttons was chosen for simplicity and clarity.

Testing

Testing was done manually by:

- Checking playback controls.
- Verifying playlist functionality.
- Ensuring songs fetch correctly from JSON Server.
- Testing across multiple screen sizes.

In addition to manual testing, the project was also verified for **error handling and stability**. For example, cases where the JSON Server was not running or when an invalid song file path was encountered were checked to ensure the application did not crash unexpectedly. The console logs were monitored during testing to identify potential warnings or errors, and necessary fixes were applied. These small but important tests helped improve the reliability of the application and ensured that the music player functioned smoothly under normal usage conditions.

Screenshots or Demo

Screenshots

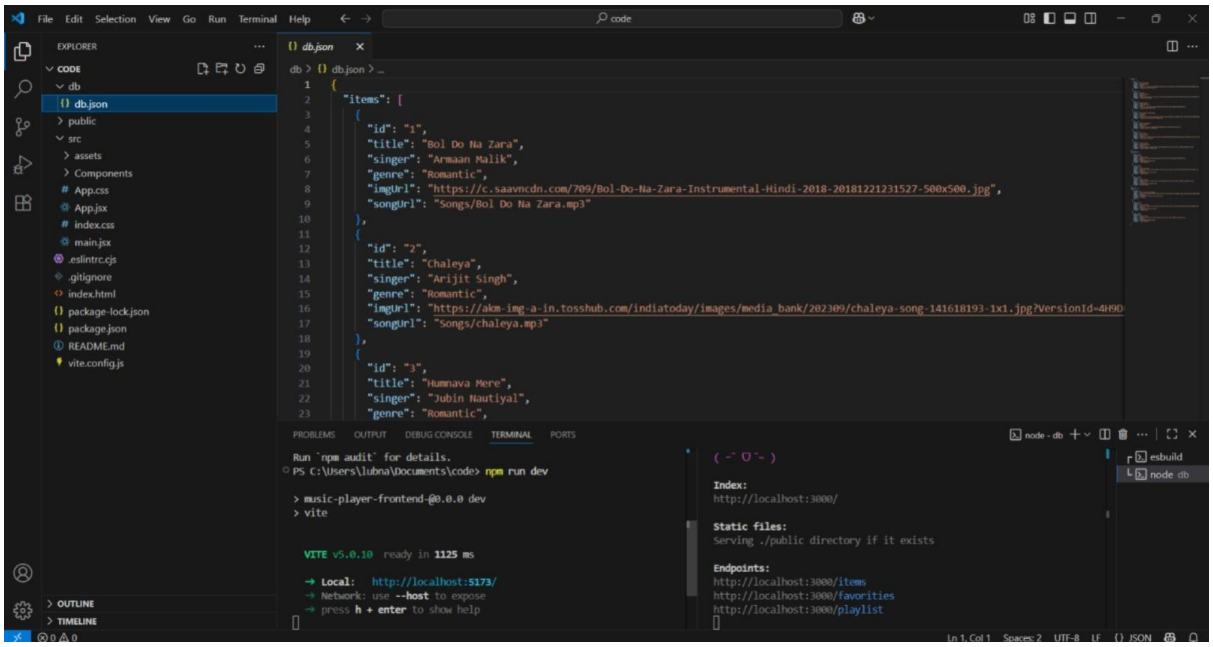


Figure 1VS Code Terminal

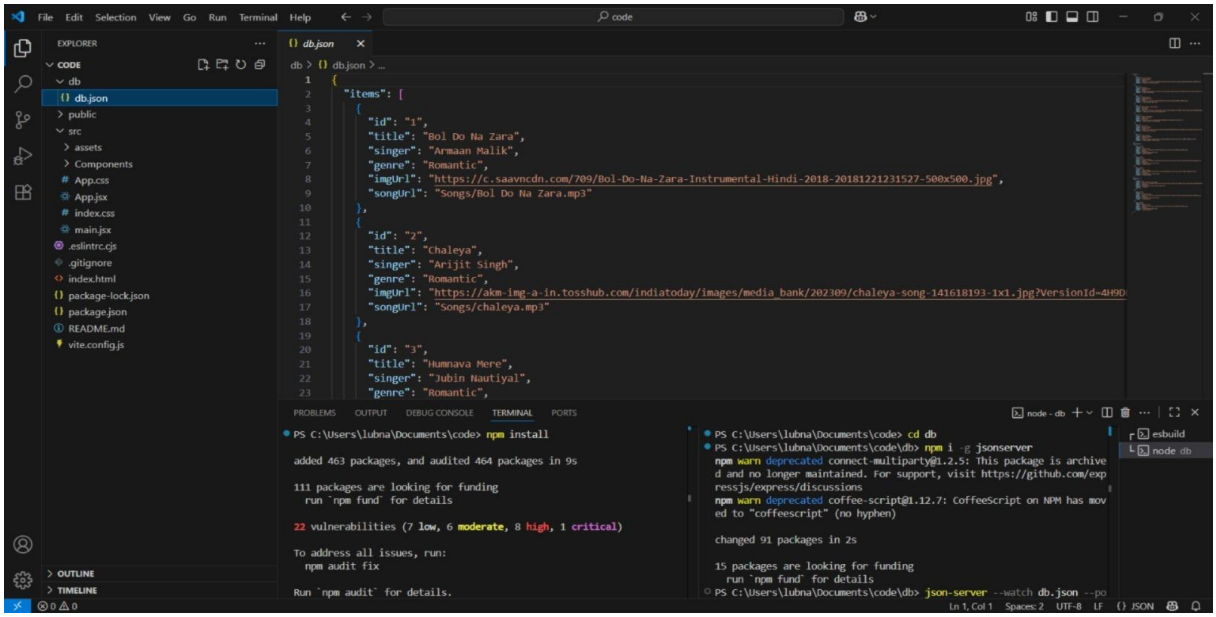


Figure 2 VS Code Terminal

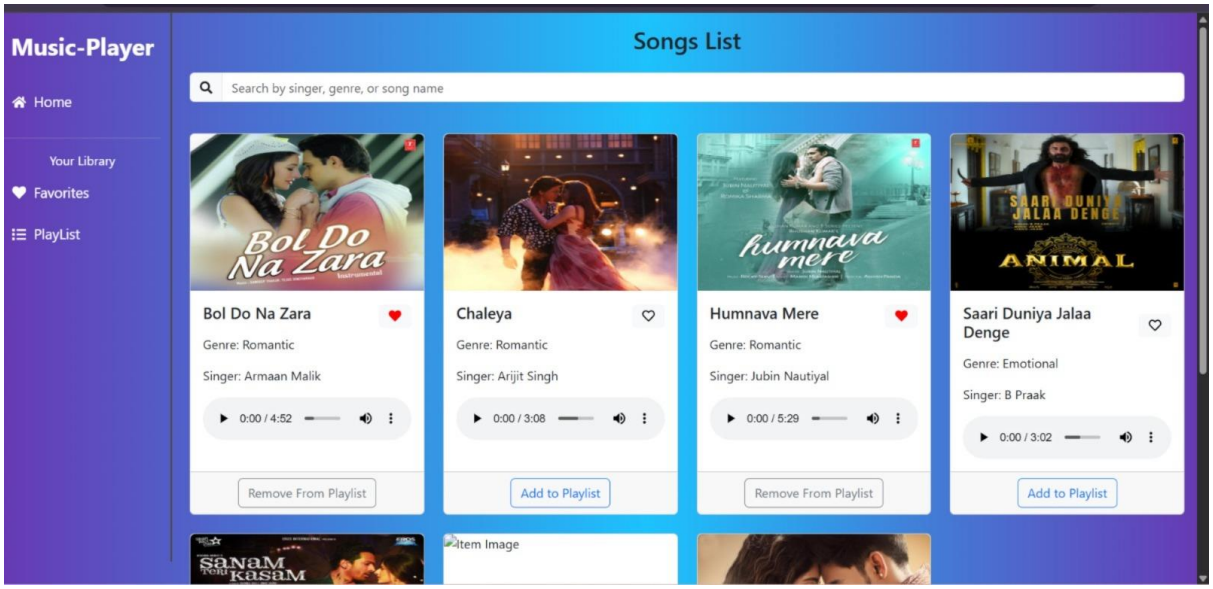
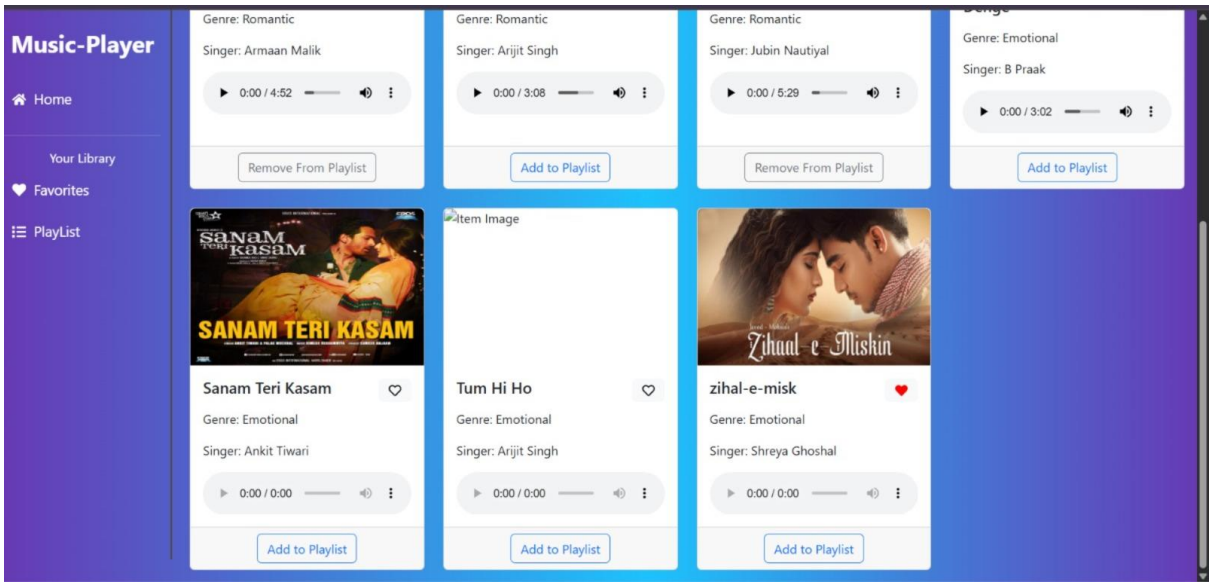
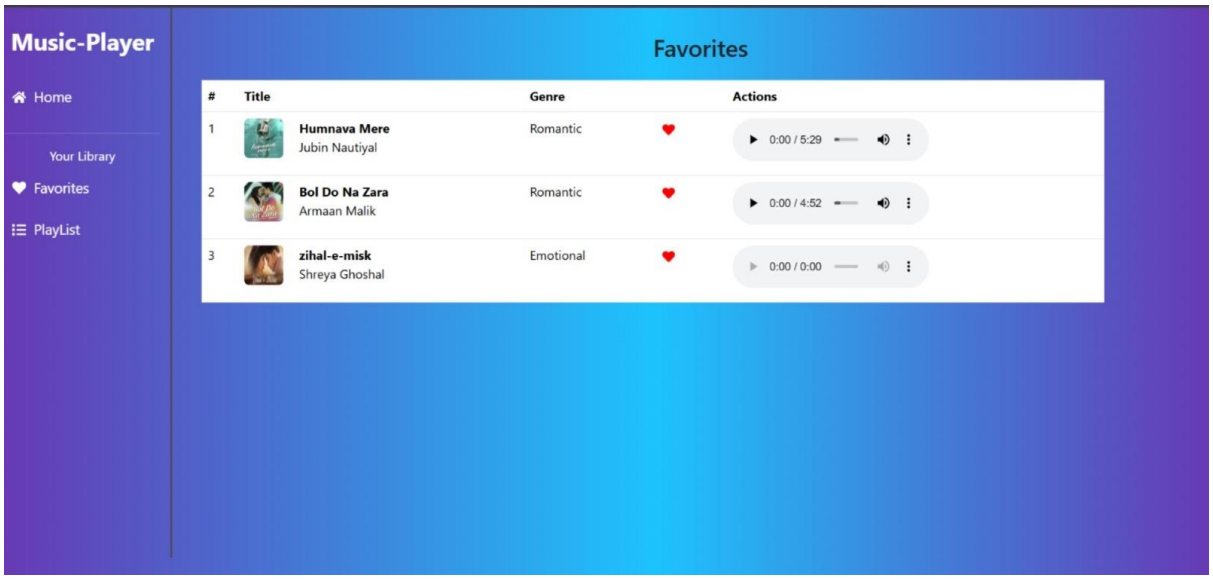


Figure 3 Music player





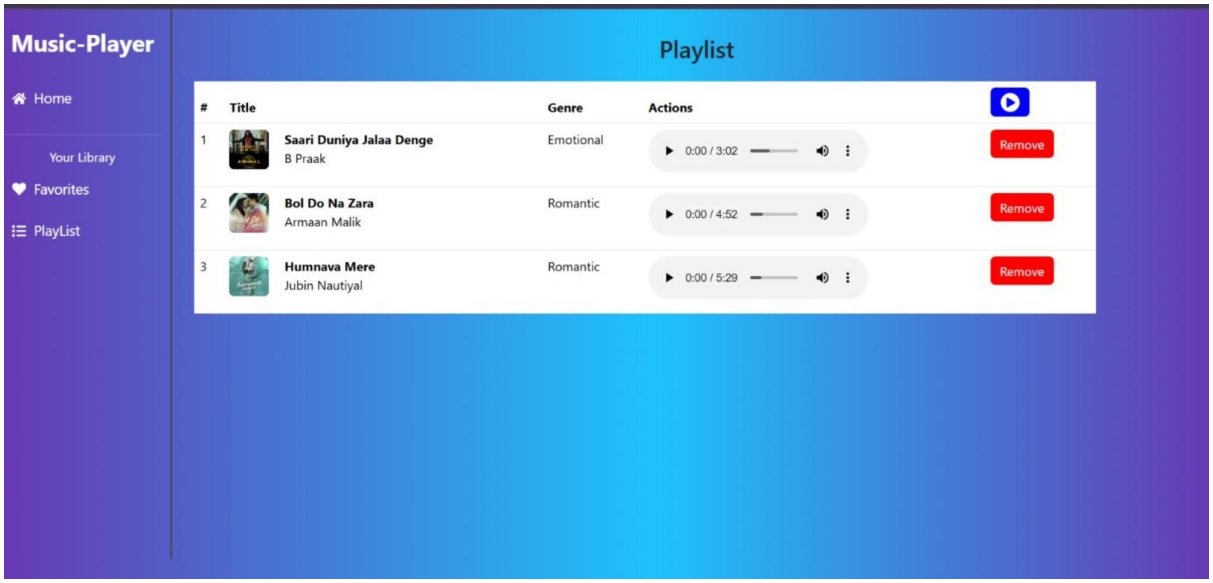


Figure 4 Music player showing favorites, playlists and controls

Demo Video

To demonstrate the working of *Rhythmic Tunes – A Music Companion*, a short demo video has been recorded. The video showcases the main features of the application, including the playlist display, song selection, and playback controls such as play, pause, next, and previous. It also highlights the responsive design of the user interface and how the application interacts with the JSON Server to fetch and display song data in real time. This video serves as a quick walkthrough of the project and provides a clear understanding of its functionality.

Link to the Demo Video

https://drive.google.com/file/d/18ERtJlvRRQ4HRN39g_RQ6NlS5yw08rtU/view?usp=sharing

Known Issues

Although *Rhythmic Tunes – A Music Companion* functions smoothly in most scenarios, there are a few limitations and issues that were identified during testing. At present, the application works only with songs stored locally in the db.json file and requires JSON Server to be running in the background. This means that without the server, the playlist cannot be loaded. In addition, advanced features such as volume control, a progress bar for tracking song duration, shuffle, and repeat options are not yet implemented. The application has also not been tested extensively across multiple browsers, so there may be minor compatibility issues on certain platforms. Finally, the current design focuses on core functionality and therefore lacks additional styling features such as dark mode or animation, which could improve user experience. These issues do not affect the basic functionality of the project but highlight areas for improvement in future versions.

Future Enhancements

While the current version of *Rhythmic Tunes – A Music Companion* achieves its primary goal of functioning as a simple music player, there are several features and improvements that can be added to enhance both functionality and user experience. One of the immediate improvements would be the addition of **volume control and a progress bar**. These would allow users to adjust the playback volume directly from the interface and also track or seek specific portions of a song. Similarly, implementing options such as **shuffle and repeat** would make the player more versatile and enjoyable for users who prefer customized listening experiences.

Another important enhancement would be the inclusion of **theming options**, such as light mode and dark mode. This would give users greater flexibility and improve usability under different lighting conditions. The user interface could also be improved further with the addition of **smooth animations, hover effects, and modern design elements** to make the application more visually appealing. In terms of accessibility, features such as keyboard shortcuts or screen reader compatibility could also be introduced to ensure that the application is inclusive for all users.

In the long term, the project could be expanded beyond local data storage by integrating with **external music APIs** such as Spotify, YouTube Music, or SoundCloud. This would allow users to stream real-world music content and manage larger playlists. Deploying the project on platforms like **Netlify or Vercel** would also make it accessible to anyone with an internet connection, eliminating the need for manual setup with JSON Server. Finally, advanced features such as **user accounts, personalized playlists, search functionality, and recommendations** could transform *Rhythmic Tunes – A Music Companion* into a more complete and scalable music application.