Frontend Development with React.js
Project Documentation format

- **Introduction**
  - **Project Title**:RhythmicTunes:Your Melodic Dashboard
  - **Team Members**:
  - C Auxilia Sharin
  - Prenitha K
  - V Kavyashree
  - G Divya Bharathi
- **Project Overview**
  - **Purpose**: RhythmicTunes is a React.js-based music streaming application designed to offer a seamless and enjoyable user experience.
  - It allows users to discover, play, and manage their favorite songs with an intuitive interface.
  - The goal is to enhance engagement through features like playlist creation, offline listening, and personalized recommendations..
  - **Features**: Song Listings – Browse a collection of songs with details.
  - Playlist Creation – Users can create and manage playlists.
  - Playback Control – Play, pause, skip, and adjust volume.
  - Offline Listening – Download songs for offline use.
- **Architecture**
  - **Component Structure**: The application follows a modular component-based architecture for scalability and maintainability. The major components include:
  -
  - **State Management**: Context API is used for global state management to handle user playlists, favorites, and current song playback.✅ Local State is handled within components using useState for UI interactions like search filtering
  - **Routing**: Uses React Router to manage page navigation efficiently:
  - / → Home Page (Song Listings)
  - /playlist → User Playlists

    /favorites → Favorite Songs Uses React Router to manage page
- **Setup Instructions**
  - **Prerequisites**: Ensure the following dependencies are installed:
  - Node.js & npm – Install from nodejs.org

- React.js – UI Framework
- React Router Dom – Navigation
- Axios – API requests
- **Installation**:
- Clone the repository:
- git clone https://github.com/your-repo/RhythmicTunes.git
- cd RhythmicTunes
- Install dependencies:
- npm install
- Start the application:
- npm start
- Start the backend (JSON Server for mock API):
- json-server --watch ./db/db.json

- **Folder Structure**
  - **Client**:
  - 🎁 RhythmicTunes/
  - ├📂 public/
  - │└📄 index.html
  - ├📂 src/
  - │├📂 components/
  - ││├📂 Navbar/
  - │││└📄 Navbar.jsx
  - ││├📂 Sidebar/
  - │││└📄 Sidebar.jsx
  - ││├📂 SongList/
  - │││├📄 SongList.jsx
  - │││├📄 SongItem.jsx
  - │││└📄 SongList.css
  - ││├📂 Playlist/
  - │││├📄 Playlist.jsx
  - │││└📄 PlaylistItem.jsx
  - ││├📂 Favorites/

- │ │ │ ├ 📄 Favorites.jsx
- │ │ │ └ 📄 FavoriteItem.jsx
- │ │ ├ 📂 Player/
- │ │ │ ├ 📄 Player.jsx
- │ │ │ └ 📄 PlayerControls.jsx
- │ ├ 📂 pages/
- │ │ ├ 📄 Home.jsx
- │ │ ├ 📄 PlaylistPage.jsx
- │ │ ├ 📄 FavoritesPage.jsx
- │ ├ 📂 context/
- │ │ └ 📄 MusicContext.jsx
- │ ├ 📂 hooks/
- │ │ └ 📄 useFetchSongs.js
- │ ├ 📂 utils/
- │ │ └ 📄 api.js
- │ ├ 📂 assets/
- │ ├ 📄 App.jsx
- │ ├ 📄 index.js
- ├ 📂 db/
- │ └ 📄 db.json
- ├ 📜 package.json
- ├ 📜 README.md
-
- **Utilities:1. Layout Components**
- 📌 Navbar.jsx – Displays the app logo, search bar, and user profile.
- 📌 Sidebar.jsx – Contains navigation links (Home, Favorites, Playlists).
- 📌 Footer.jsx – Displays copyright info and quick links.
-
- 🎵 2. Songs Components
- 📌 SongList.jsx – Fetches and displays a list of songs.
- 📌 SongItem.jsx – Represents a single song (title, artist, play button).

- 📌 SongDetails.jsx – Shows song details when clicked.
-
- ◆ Example: SongItem.jsx
-
- jsx
- Copy
- Edit
- const SongItem = ({ song, onPlay }) => {
- return (
-
- <h4>{song.title} - {song.artist}</h4>
- <button onClick={() => onPlay(song)}>▶ Play</button>
-
-);
- };
- export default SongItem;
- 🎧 3. Player Components
- 📌 Player.jsx – The main music player UI.
- 📌 PlayerControls.jsx – Controls (Play, Pause, Skip, Volume).
- 📌 ProgressBar.jsx – Displays song progress.
-
- ◆ Example: PlayerControls.jsx
-
- jsx
- Copy
- Edit
- const PlayerControls = ({ isPlaying, onPlayPause }) => {
- return (
-
- <button onClick={onPlayPause}>{isPlaying ? "⏸ Pause" : "▶ Play"}</button>

```
      </div>
    );
  };
  export default PlayerControls;
```

📜 4. Playlist Components

📌 Playlist.jsx – Shows user playlists.

📌 PlaylistItem.jsx – A single playlist entry.

📌 PlaylistForm.jsx – Allows users to create/edit playlists.

💜 5. Favorites Components

📌 Favorites.jsx – Displays favorited songs.

📌 FavoriteItem.jsx – Handles individual favorite actions.

🔍 6. Search Components

📌 SearchBar.jsx – Allows users to search for songs.

📌 SearchResults.jsx – Displays search results dynamically.

🔷 Example: SearchBar.jsx

jsx

Copy

Edit

```
const SearchBar = ({ onSearch }) => {
  return (
    <input
      type="text"
      placeholder="Search songs..."
      onChange={(e) => onSearch(e.target.value)}
    />
  );
};
```

- export default SearchBar;
- 📄 7. Page Components
- 📌 Home.jsx – Displays trending songs & recommendations.
- 📌 PlaylistPage.jsx – Shows a specific playlist.
- 📌 FavoritesPage.jsx – Displays the user's favorite songs.
- 📌 NotFound.jsx – Handles 404 errors for invalid routes.
-
- 🌐 8. Context API (Global State Management)
- 📌 MusicContext.jsx – Manages playlists, favorites, and playback state.
-
- 🔷 Example: MusicContext.jsx
-
- jsx
- Copy
- Edit
- import { createContext, useState } from "react";
-
- export const MusicContext = createContext();
-
- export const MusicProvider = ({ children }) => {
- const [favorites, setFavorites] = useState([]);
-
- return (
- <MusicContext.Provider value={{ favorites, setFavorites }}>
- {children}
- </MusicContext.Provider>
-);
- };
- ⚙ 9. Custom Hooks
- 📌 useFetchSongs.js – Fetches songs from API.
- 📌 useAudioControls.js – Manages audio play/pause logic.

-
- ◆ Example: useFetchSongs.js
-
- jsx
- Copy
- Edit
- import { useState, useEffect } from "react";
- import axios from "axios";
-
- const useFetchSongs = () => {
- const [songs, setSongs] = useState([]);
-
- useEffect(() => {
- axios.get("http://localhost:3000/songs")
- .then(res => setSongs(res.data))
- .catch(err => console.error(err));
- }, []);
-
- return songs;
- };
-
- export default useFetchSongs;
- 🔗 10. Utility Functions
- 📌 api.js – Handles API calls (GET, POST, DELETE).
- 📌 formatTime.js – Formats time in mm:ss format for song duration.
-
- ◆ Example: formatTime.js
-
- js
- Copy
- Edit

- export const formatTime = (seconds) => {
- const minutes = Math.floor(seconds / 60);
- const secs = Math.floor(seconds % 60);
- return `${minutes}:${secs < 10 ? "0" : ""}${secs}`;
- };.

- **Running the Application**
  - Provide commands to start the frontend server locally.
    - **Frontend**: npm start in the client directory.
- **Component Documentation**
  - **Key Components**:
  - App.jsx – Root component managing high-level state and routing.
  - Navbar.jsx – Navigation and search bar.
  - Sidebar.jsx – Contains links to playlists and favorites.
  - Player.jsx – Manages playback, progress, and controls.
  - SongList.jsx – Displays the list of available songs.
  - Playlist.jsx – Allows users to create and manage playlists.
  - Favorites.jsx – Displays and manages favorited songs.
  - SearchBar.jsx – Provides search functionality to filter songs.
  - Explains how these components interact to provide a smooth user experience.
  - Uses React's useState and useEffect for managing local state.Implements Context API for global state sharing across components.
  - Lists key state variables such as items, wishlist, playlist, currentlyPlaying, and search.Term.Routing utilizes React Router (react-router-dom) for client-side navigation.
  - **Reusable Components**:
  - SongItem.jsx – Reusable component for displaying a single song with play and favorite options.
  - PlaylistItem.jsx – Handles individual playlists within the playlist page. FavoriteItem.jsx – Represents a single song in the Favorites section.
  - PlayerControls.jsx – Controls for play, pause, skip, and volume adjustment.
  - ProgressBar.jsx – Displays the current progress of a song.
  - SearchResults.jsx – Displays filtered search results dynamically.
  -
- **State Management**

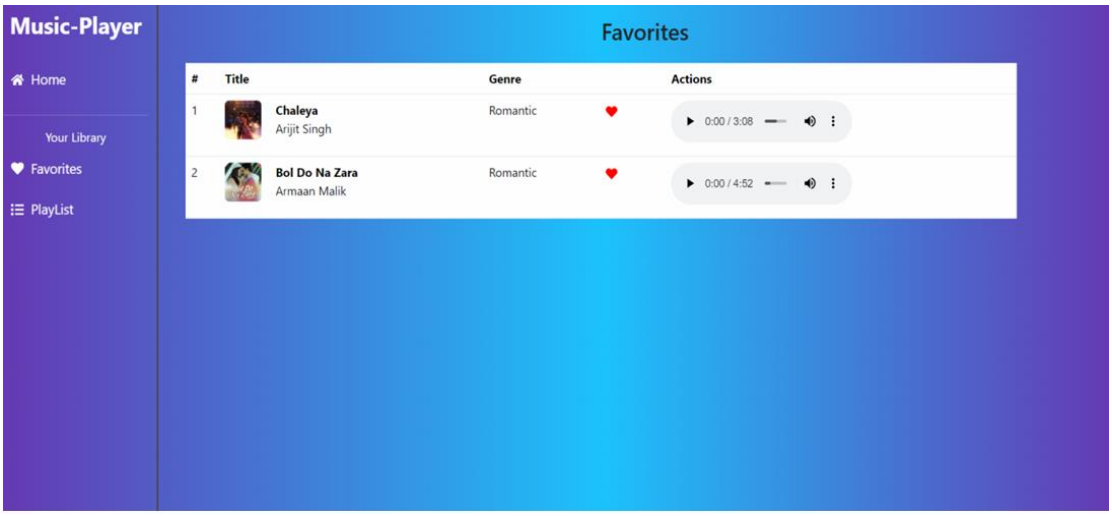
    - **Global State**: Context API is used for global state management to handle user playlists, favorites, and current song playback.
    - **Local State**:Local State is handled within components using useState and useEffect for managing search filtering and song playback.
- **User Interface**
    - Home Page – Displays all songs with search and filter options.
    - Playlist Page – Users can create, edit, and delete playlists.
    - Favorites Page – Shows songs marked as favorites.
    - Player Component – Interactive music player UI.
- **Styling**
- **CSS Frameworks/Libraries**:Tailwind CSS / Bootstrap – For styling and responsive UI design.

  Custom CSS – Minor customizations for a better UX.
- **Theming**:
- **Testing**
- **Testing Strategy**:
- Jest + React Testing Library – Unit testing for components.
- Cypress – End-to-end testing for UI interactions.
- Code Coverage:
- Istanbul (nyc) for tracking test coverage.
- Testing major features like playback, search, and playlist management.
- **Code Coverage**: Core Features
- ✅ Song Listings – Ensure the API fetches and displays songs correctly.
- ✅ Search Functionality – Test search queries, including case insensitivity and partial matches.
- ✅ Playlist Management – Adding, removing, and verifying duplicates in playlists.
- ✅ Favorites (Wishlist) Management – Ensure favoriting/unfavoriting songs updates correctly.
- ✅ Playback Controls – Test play, pause, skip, and volume control behavior.
-
- UI & Component Testing
- ✅ Navigation & Routing – Ensure routes (/, /favorites, /playlist) render the correct pages.
- ✅ Buttons & Click Events – Test add/remove playlist buttons, favorite buttons, and search input.
- ✅ Mobile Responsiveness – Ensure UI adapts correctly across different screen size

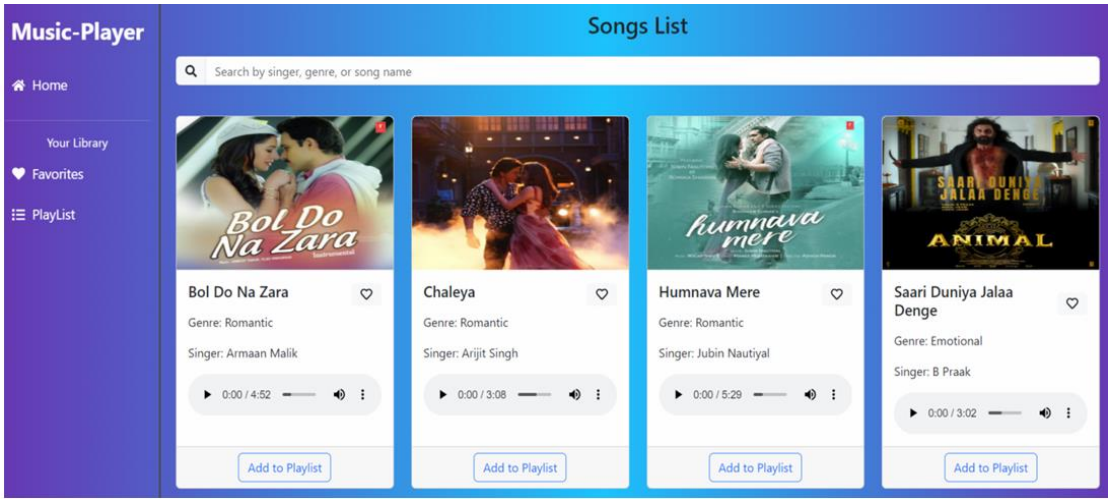
- API & Data Handling
- ✅ Mock API Calls – Verify that fetching data from json-server works as expected.
- ✅ Error Handling – Test scenarios where the API fails or returns no data.
- ✅ Data Persistence – Ensure added favorites/playlists are stored properly and retrieved correctly.

  Security & Performance
- ✅ Unauthorized Access – If authentication is added later, test protected routes.
- ✅ Load Testing – Check if the app performs well with a large song database.
- **Screenshots or Demo**
- The document includes Google Drive links for the demo and screenshots:
- Project Demo Link:

Favtorites

- 

- Home page

- 

-
-
- **Known Issues**
- **Multiple Audio Overlaps – Songs might overlap if currentlyPlaying state is not managed correctly.**
- **Lag in Playback Control – Delayed UI updates when skipping or pausing songs.**
- **Search Functionality Bugs – Case sensitivity and slow response on large datasets.**
- **Mobile Responsiveness – Some UI elements may need better optimization for small screens.**
- **Future Enhancements**
- .🚀 Crossfade & Gapless Playback – Smooth transitions between songs
- .🚀 Lyrics Display – Show synchronized lyrics while playing music.
- 🚀 User Authentication – Implement login/signup for a personalized experience
- .🚀 PWA Support – Enable offline access and a native-like experience.
-