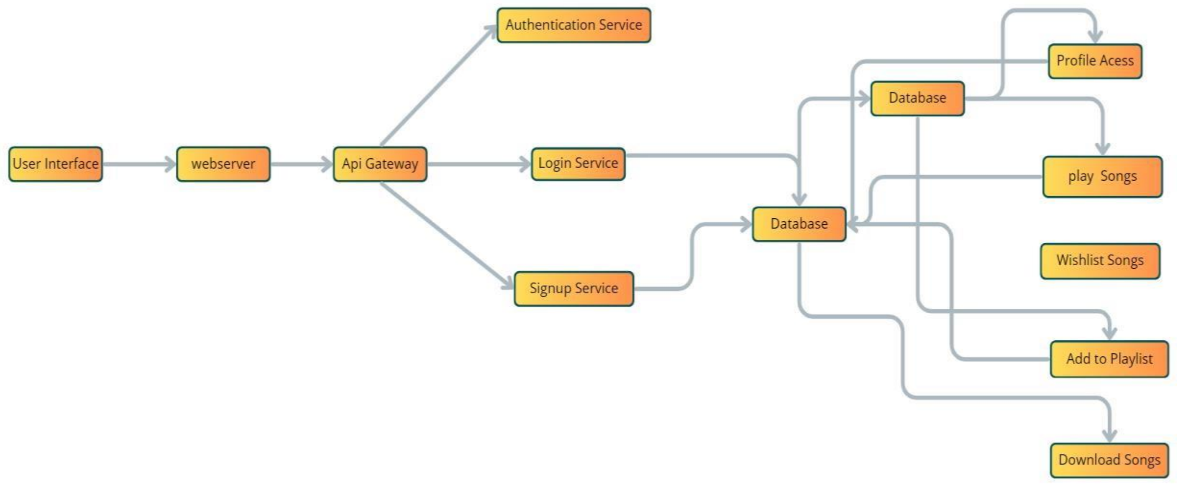
Music Streaming App

Introduction:-

Music Streaming Application, crafted using the powerful MERN (MongoDB, Express.js, React, Node.js) Stack. Designed for the modern music enthusiast, our application combines robust functionality with an intuitive user interface. Discover new hits, revisit classics, and enjoy a seamless musical journey tailored to your taste. Powered by MongoDB for fast access, Express.js for a responsive server, Node.js for high performance, and React for a visually stunning interface, our application delivers a consistent and enjoyable experience across all devices. Say goodbye to traditional music listening and embrace a new era of music streaming. Press play and elevate your auditory experience today!

Technical Architecture:

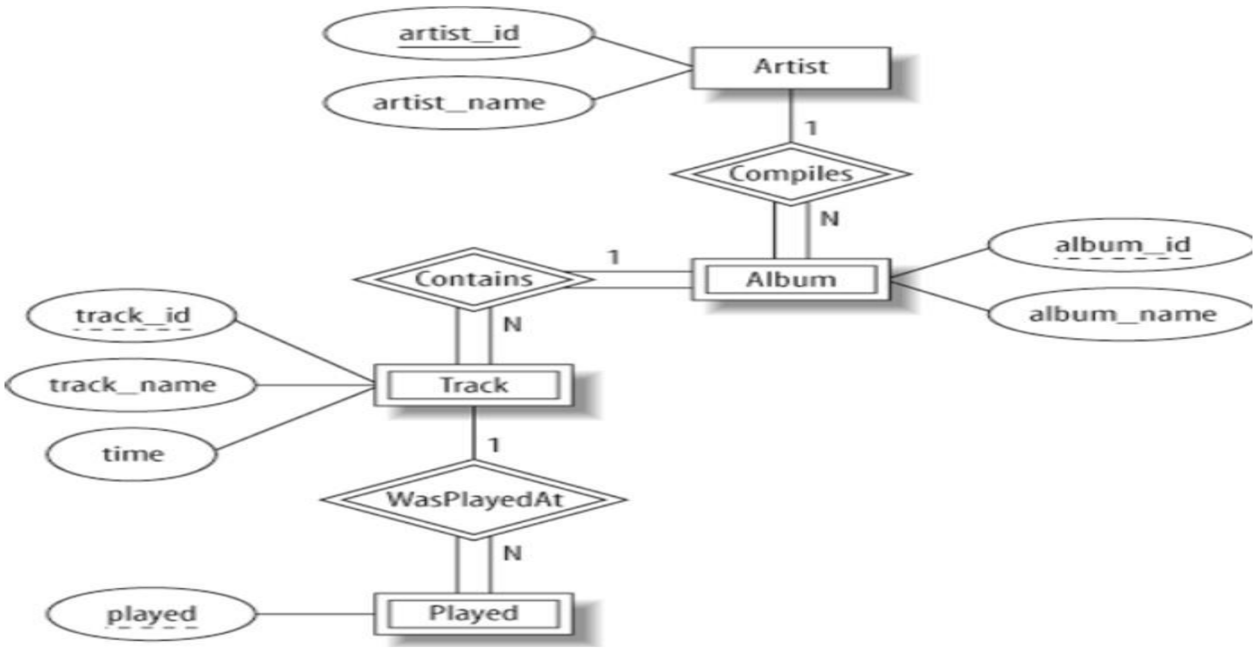


In this technical architecture, The Music Streaming App consists of several components:

- User Interface: Represents the web interface through which users interact with the system to search for events, select seats, and make bookings.
- Web Server: Hosts the user interface and serves the web pages to the users.

- API Gateway: Acts as a single entry point for client requests and routes them to the respective services.
- Authentication Service: Handles user authentication and authorization for secure access to the system.
- Service: Manages -related information such as Song details, Atrist details, and user information.
- Database: Stores persistent data related to events, bookings, payments, and other system entities.
- Song details: Manages information about Songs, Genre, Type and Duration.
- Artist details: User can access Artist details and songs that are sung by that artist
- Playlist: User can add songs to playlist and remove songs from playlist and play All the Songs in Loop.

ER-Diagram:



User:

- UserID (Primary Key)
- Name

- Email
- Phone

Song:

- ID (Primary Key)
- Title
- Genre
- Artist

Play Song:

- SongID (Primary Key)
- Add to Favorite
- Add to Playlist
- Duration
- Download Song

The relationships between the entities are as follows:

- User has a one-to-many relationship with Songs (one user can play multiple Songs).
- Song has a one-to-many relationship with Playlist (one Song can be played multiple Times).

The foreign key relationships are established between the entities using the primary keys from the respective tables.
Please note that this is just an example, and you can modify or expand it based on your specific requirements for the Music Streaming App.

Key Features:-

- User Registration and Authentication: Enable users to create accounts, log in securely, and authenticate their identity to access the music streaming app.
- Song Listings: Display a comprehensive list of available songs with details such as title, artist, genre, and release date.
- Playlist Creation: Empower users to create personalized playlists, adding and organizing songs based on their preferences.
- Playback Control: Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback.

- Offline Listening: Allow users to download songs for offline listening, enhancing the app's accessibility and convenience.
- Library Management: Provide users with the ability to manage their music library, including adding, removing, and organizing saved songs and playlists.
- Search Functionality: Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.
- User Profile: Allow users to customize their profiles, including profile pictures, bio, and other personalization options.
- Admin Dashboard: Provide an administrative interface for managing songs, user accounts, playlists, and app content.
- Data Analytics: Incorporate analytics tools to gather insights into user behavior, popular songs, and other relevant metrics for continuous improvement.

Keep in mind that the features mentioned are adaptable and can be tailored based on the specific goals and target audience of the music streaming application.
These are just a few key features, and you can customize and expand them based on your specific requirements and the scale of the Music Streaming Application you are building.

PRE REQUISITES:

To develop a full-stack Ecommerce App for Furniture Tool using React js, Node.js,Express js and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.
• Download: https://nodejs.org/en/download/
•Installation instructions:
https://nodejs.org/en/download/package-manager/

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service. • Download: https://www.mongodb.com/try/download/community
• Installation instructions:https://docs.mongodb.com/manual/installation/

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

React js: React is a JavaScript library for building client-side applications. And Creating Single Page Web-Appliaction

Getting Started

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

Quik Start

npx create-react-app my-app cd
my-app
npm start

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app or yarn global remove create-react-app to ensure that npx always uses the latest version.

Create a new React project:

- Choose or create a directory where you want to set up your React project.
- Open your terminal or command prompt.
- Navigate to the selected directory using the cd command.

- Create a new React project by running the following command: npx createreact-app your-app-name.Wait for the project to be created:
- This command will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

- After the project creation is complete, navigate into the project directory by running the following command**: cd your-app-name**

Start the development server:

- To launch the development server and see your React app in the browser, run the following command: **npm start**
- The npm start will compile your app and start the development server.
- Open your web browser and navigate to http://localhost:3000 to see your React app.

You have successfully set up React on your machine and created a new React project. You can now start building your app by modifying the generated project files in the src directory.

Please note that these instructions provide a basic setup for React. You can explore more ad- vanced configurations and features by referring to the official React documentation: https://react.dev/

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Library: Utilize React to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: https://gitscm.com/downloads

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm. • Visual Studio Code: Download

from https://code.visualstudio.com/download • Sublime Text: Download from
https://www.sublimetext.com/download
• WebStorm: Download from
https://www.jetbrains.com/webstorm/download

Roles and Responsibility

User:-

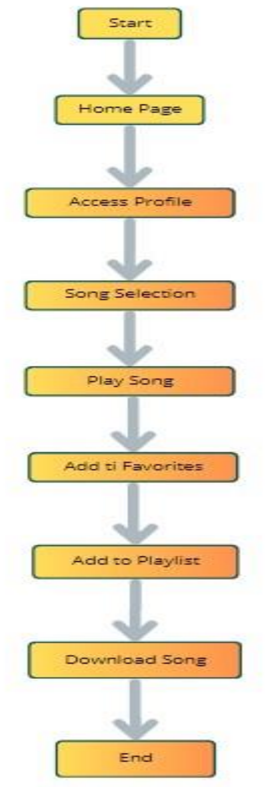
- Registration: Users are responsible for creating an account on the booking system by providing necessary details like name, email, and phone number.
- Search : Users can search for songs and view details such as title, genre, and release date, and Duration.
- Add to Favorites: Users can Add Songs to Favorites and can Remove from Favorities.
- Add to Playlist: Users can Add Songs to Playlist and can Remove from Playlist and finally he can play all the songs in loop.
- Download: user can download the song and he can listen in offline.

Admin:-

- System Management: The admin is responsible for managing the overall functioning and configuration of the Music Streaming Application.
- Management: The admin can add, update, or remove s from the system, including details like title, genre, and release date,song.

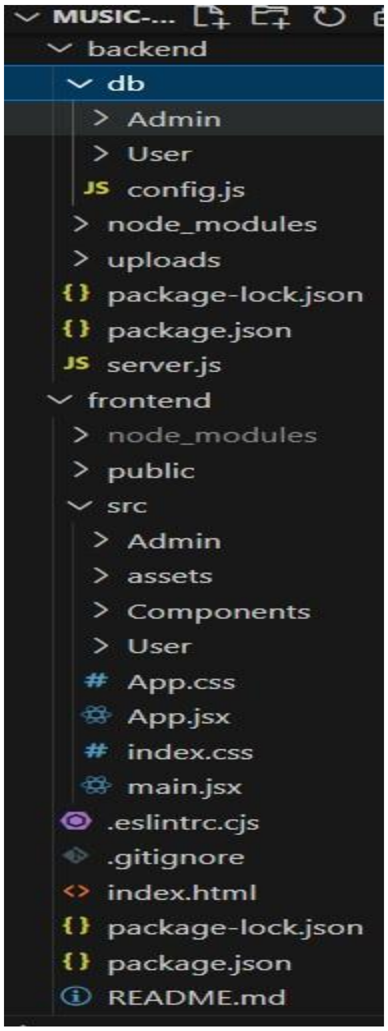
- Song Management: The admin is responsible for configuring and maintaining the Songs like create, update,delte Song.
- User Management: The admin can manage user accounts, including user registration, authentication, and handling user-related issues or inquiries.
- These roles and responsibilities can vary based on the specific requirements and scope of the Music Streaming Application.

User Flow:



In this user flow, the process starts at the "Start" node, then moves to the "Home Page" where the user can browse songs. From there, the user selects a song at the "Song Selection" node.
Next, the user proceeds to the "Play Song" node to play their preferred song. After that, the user can add songs to "Favorites" and to "Playlist".
From there, the user moves to the "Download" node to download songs. Finally, the process ends at the "End" node.
The style statements at the bottom can customize the appearance of specific nodes. In this example, the "Start" and "End" nodes have a light gray background.
Feel free to modify the code or add additional nodes as needed.

Project Structure:



```
MUSIC-...
  backend
    db
      Admin
      User
      config.js
    node_modules
    uploads
    package-lock.json
    package.json
    server.js
  frontend
    node_modules
    public
    src
      Admin
      assets
      Components
      User
      App.css
      App.jsx
      index.css
      main.jsx
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
```

The project structure may vary depending on the specific library, framework, programming language, or development approach used. It's essential to organize the files and directories in a logical and consistent manner to improve code maintainability and collaboration among developers.

app/app.component.css, src/app/app.component: These files are part of the main AppComponent, which serves as the root component for the React app. The component handles the overall layout and includes the router outlet for loading different components based on the current route.

PROJECT FLOW:-

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Node.js.
- MongoDB.
- Create-react-app.

2. Create project folders and files:

- Client folders.
- Server folders.

Milestone 2: Backend Development: Setup express server:

- Install express.
- Create app.js file. Define API's

Configure MongoDB:

- Install Mongoose.
- Create database connection.
- Create Models.

Implement API end points:

- Implement CRUD operations.
- Test API endpoints.

Milestone 3: Web Development:

1. Setup React Application:

• Create React application.

• Configure Routing.

• Install required libraries.

2. Design UI components:

• Create Components.

• Implement layout and styling.

• Add navigation.

3. Implement frontend logic:

• Integration with API endpoints.

• Implement data binding.

Backend:

- **Server**: Handles incoming requests, processes user actions (music playback, playlist creation, search queries, account management).
- **API Endpoints**: Facilitate communication between client and server (song retrieval, playlist management, user authentication).
- **Database**: Stores and retrieves music data, user profiles, playlists.
- **Song Selection**: Retrieves song details and sends to client-side for display.
- **Playlist Management**: Manages playlists, associates them with users, handles song additions/removals.
- **Search Functionality**: Processes search queries, retrieves and sends matching results.
- **Playback Control**: Manages streaming, handles buffering, sends audio stream to client-side, controls playback actions.
- **User Authentication**: Secures user accounts, verifies credentials, issues access tokens.

The server ensures data consistency, security, and a seamless user experience through validations, error handling, and business logic. Implementation may vary based on chosen technologies and frameworks.