

SMARTINTERNZ

FULLSTACK DEVELOPER WITH MERN

PROJECT TITTLE:MUSIC STREAMING APP

SUBMITTED BY:

TEAM ID : LTVIP2024TMID05420

Team leader : Kakollu Divyadeepika

Team member : Nerella Srivyshnavidevi

Team member : Tellakula Sivanagajyothi

Team member : Arumalla Krishnaprsanna

Team member : Gottipati Meghana

Introduction:-

Music Player Application is a modern and intuitive platform designed to elevate your music listening experience. Our app offers a seamless way to enjoy your favorite tunes, whether you're on the go or relaxing at home. With a user-friendly interface, personalized playlists, and a vast library of songs, our Music Player App is your ultimate companion for all things music. Explore new tracks, create custom playlists, and immerse yourself in the world of music with our app. Download now and discover a new way to enjoy music!

Description:

Welcome to the future of musical indulgence – an unparalleled audio experience awaits you with our cutting-edge Music Streaming Application, meticulously crafted using the powerful MERN (MongoDB, Express.js, React, Node.js) Stack. Seamlessly blending innovation with user-centric design, our application is set to redefine how you interact with and immerse yourself in the world of music.

Designed for the modern music enthusiast, our MERN-based Music Streaming Application offers a harmonious fusion of robust functionality and an intuitive user interface. From discovering the latest chart-toppers to rediscovering timeless classics, our platform ensures an all-encompassing musical journey tailored to your unique taste.

Powered by MongoDB, our application ensures a scalable and efficient database, facilitating lightning-fast access to an extensive library of tracks. Express.js, with its minimalist web application framework, lays the foundation for a responsive and streamlined server, while Node.js ensures high-performance, non-blocking I/O operations, providing a fluid and seamless user experience.

The heart of our Music Streaming Application lies in React, a dynamic and feature-rich JavaScript library. Immerse yourself in a visually stunning and interactive interface, where every click, scroll, and playlist creation feels like a musical revelation. Whether you're on a desktop, tablet, or smartphone, our responsive design ensures a consistent and enjoyable experience across all devices.

Say goodbye to the limitations of traditional music listening and welcome a world of possibilities with our MERN Stack Music Streaming Application. Join us on this journey as we transform the way you connect with and savor the universal language of music. Get ready to elevate your auditory experience – it's time to press play on a new era of music streaming.

Scenario Based Case Study:

Allen is a passionate music enthusiast who loves discovering new artists and genres. He uses a popular music streaming app to listen to music throughout the day, whether he's working, exercising, or relaxing at home. However, recently, Allen has noticed that he's not as excited about using the app as he used to be. He feels like he's listening to the same songs and artists repeatedly, and the app's recommendations don't seem as personalized or engaging as they once were. Allen is considering trying out a different music streaming service unless the app can offer him a more personalized and exciting music experience.

- **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.

SYSTEM OVER VIEW :

For a system overview documentation of a music streaming app, you'll want to cover various aspects like architecture, user flow, functionalities, and technologies used. Here's a basic outline you can follow:

1. *Introduction:*

- Brief overview of the music streaming app.
- Purpose of the document.

2. *System Architecture:*

- High-level architecture diagram.
- Components and their interactions (front end, back end, database, etc.).
- Scalability and resilience considerations.

3. *User Roles and Permissions:*

- Types of users (listeners, artists, admins, etc.).
- Access levels and permissions for each role.

4. *User Flow:*

- Registration and authentication process.
- Searching for music.
- Creating and managing playlists.
- Listening to music (streaming, offline mode).
- Interacting with social features (following artists, liking songs, sharing playlists, etc.).

5. *Functionalities:*

- Music catalog management (uploading, metadata management).
- Payment processing (subscription, ad-supported, premium features).
- Recommendations and personalized content.
- Social features (following, sharing, commenting).
- Offline mode and downloads.

6. *Technologies Used:*

- Front end (frameworks, libraries).
- Back end (programming languages, frameworks, databases).
- Cloud services (hosting, storage, CDN).
- Third-party APIs (payment gateways, music metadata providers, social media integration).

7. *Security Considerations:*

- Data encryption (user data, payment information).
- Authentication and authorization mechanisms.
- Preventing common security threats (SQL injection, XSS attacks, etc.).

8. *Performance and Scalability:*

- Load balancing.
- Caching strategies.
- Database optimization.
- Monitoring and analytics.

9. *Testing:*

- Types of testing (unit, integration, performance, security).
- Tools and frameworks used for testing.

10. *Deployment:*

- Continuous integration/continuous deployment (CI/CD) pipelines.
- Deployment environment (cloud, on-premise).
- Rollback strategies.

11. *Maintenance and Support:*

- Bug tracking and issue resolution.
- Software updates and versioning.
- User support channels (helpdesk, FAQs, community forums).

12. *Future Enhancements:*

- Potential features for future releases.
- Technologies to explore.
- Feedback mechanisms for users.

13. *Conclusion:*

- Summary of the system overview.
- Importance of the music streaming app and its impact.

INSTALLATION AND SETUP :

1. Install required tools and software:

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

2. Create project folders and files:

- Client folders.
- Server folders.

3. Install Packages:

Frontend npm Packages

- Axios.
- React-Router –dom.
- Bootstrap.
- React-Bootstrap.
- React-icons.

Backend npm Packages

- Express.
- Mongoose.
- Cors.

Referance Link:-

https://drive.google.com/file/d/1Acv3Lx3PtJcOYkUjREWAzIoC-i6w96Tl/view?usp=drive_link

BACK END DEVELOPMENT :

- **Setup express server**
 1. Create index.js file in the server (backend folder).
 2. Create a .env file and define port number to access it globally.
 3. Configure the server by adding cors, body-parser.
- **User Authentication:**
 - Create routes and middleware for user registration, login, and logout.
 - Set up authentication middleware to protect routes that require user authentication.
- **Define API Routes:**
 - Create separate route files for different API functionalities such as users orders, and authentication.
 - Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
 - Implement route handlers using Express.js to handle requests and interact with the database.
- **Implement Data Models:**
 - Define Mongoose schemas for the different data entities like products, users, and orders.
 - Create corresponding Mongoose models to interact with the MongoDB database.
 - Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
- **User Authentication:**
 - Create routes and middleware for user registration, login, and logout.
 - Set up authentication middleware to protect routes that require user authentication.

- **Error Handling:**

- Implement error handling middleware to catch and handle any errors that occur during the API requests.
- Return appropriate error responses with relevant error messages and HTTP status codes.

Referance Link:-

https://drive.google.com/file/d/1ff8bKL5XFYRQhggqn_Jn6QEcyjS8CCGc/view?usp=sharing

Database

1. Configure MongoDB:

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

2. Connect database to backend:

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.

```
//  
const PORT = process.env.PORT || 6001;  
mongoose.connect(process.env.MONGO_URL, {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
}).then(()=>{  
  
  server.listen(PORT, ()=>{  
    console.log(`Running @ ${PORT}`);  
  });  
  
}).catch((err)=>{  
  console.log("Error: ", err);  
})
```

3. Configure Schema:

Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.

```
JS User.js  X
server > models > JS User.js > ...
1  import mongoose from 'mongoose';
2
3  const UserSchema = new mongoose.Schema({
4    username:{
5      type: String,
6      require: true
7    },
8    email:{
9      type: String,
10     require: true,
11     unique: true
12   },
13   password:{
14     type: String,
15     require: true
16   },
17 });
18
19 const User = mongoose.model("users", UserSchema);
20 export default User;
```

Frontend 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.

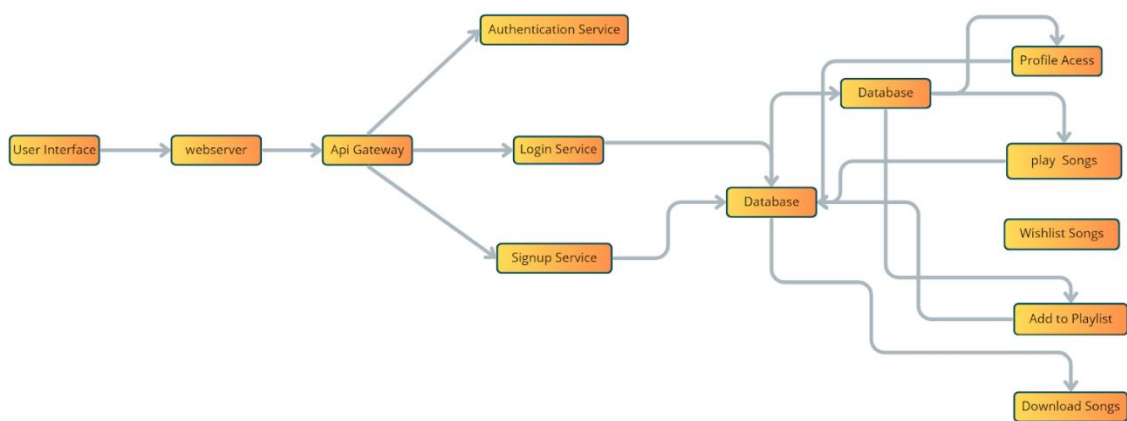
3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Referance:-

https://drive.google.com/file/d/1Bqxdk51ZTeHF2z1qyFw5m1bWHot89LdS/view?usp=drive_link

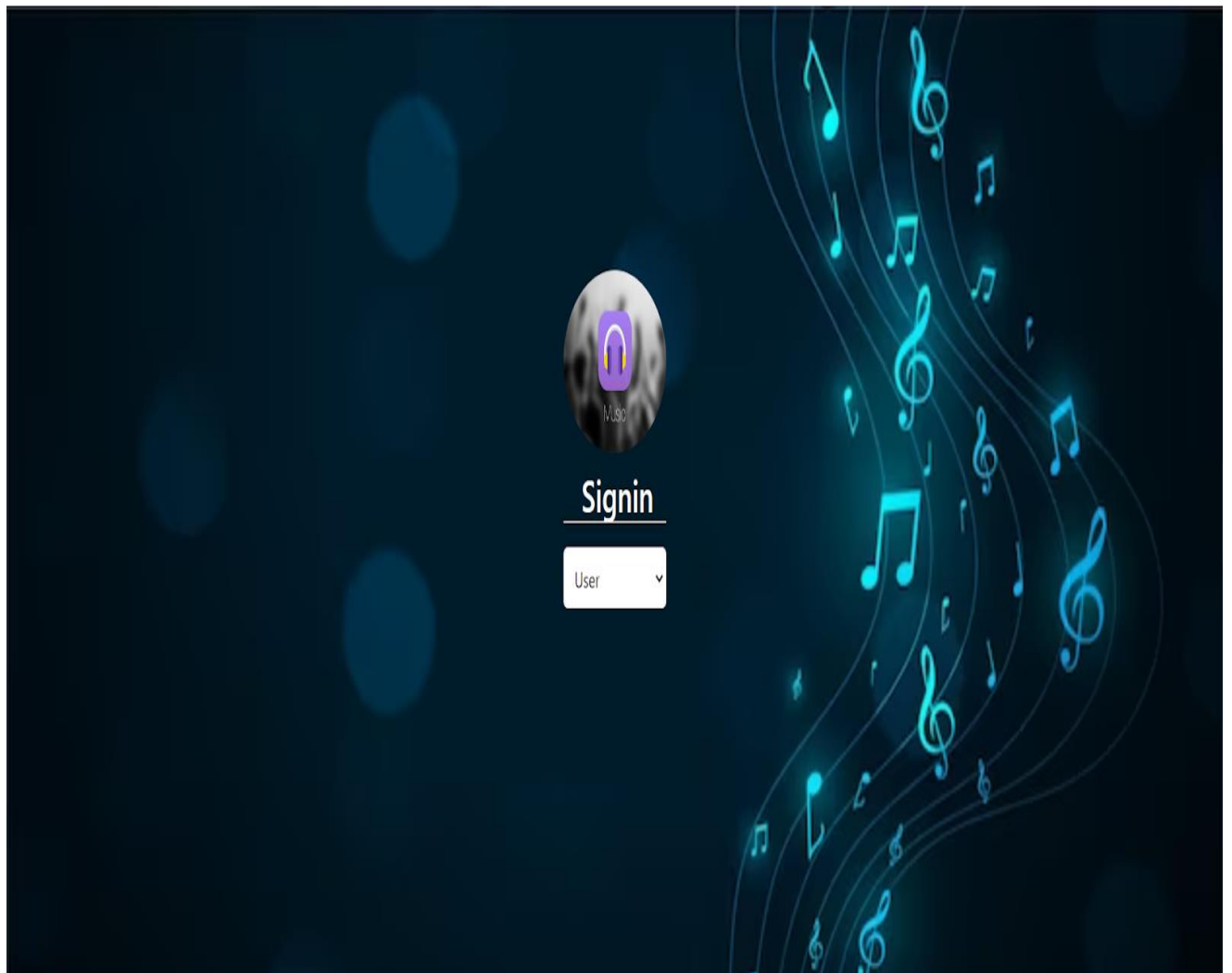
TECHNICAL ARCHITECTURE:



Project Implementation:


Finally, after finishing coding the projects we run the whole project to test it's working process and look for bugs. Now, let's have a final look at the working of our Cab Booking application.

Landing page:-



Login Page:-





Login to user account

Email address

Password

[Log in](#)

Don't have an account? [Create](#) [Signup](#)

Signup Page:-





Signup

Name

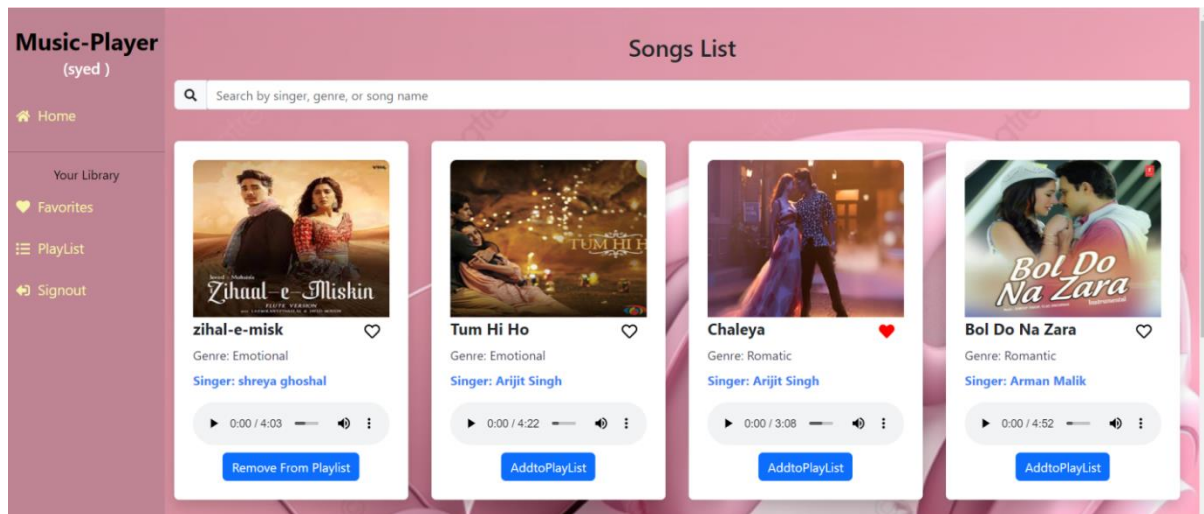
Email address

Password

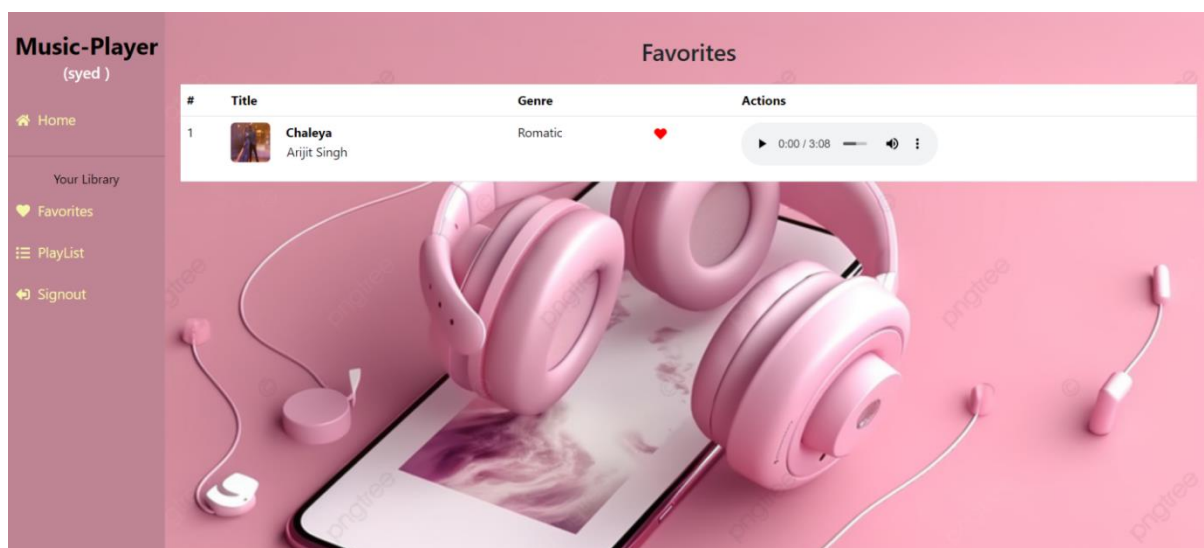
[Signup](#)

Already have an account? [Login](#)

Songs Page:-



Favorites Page :-



Playlist Page:-

Music-Player
(syed)

Home









Your Library

Favorites

PlayList

Signout

Playlist

#	Title	Genre	Actions	
1	 Ishq Tera Guru Randhawa	Romantic	▶ 0:00 / 3:31  ⋮	Remove
2	 zihal-e-misk shreya ghoshal	Emotional	▶ 0:00 / 4:03  ⋮	Remove
3	 Tum Hi Ho Arijit Singh	Emotional	▶ 0:00 / 4:22  ⋮	Remove
4	 Bol Do Na Zara Arman Malik	Romantic	▶ 0:00 / 4:52  ⋮	Remove

Users Page:-

Music-Player
(Admin)

Users

Songs

AddSong

Signout

Users

sl/no	UserId	User name	Email	Operation
1	6576eaac459cb2b913bd3581	Arshad	arshad@gmail.com	✕ ✕
2	65799c3ede4c9f4501cf3e8d	syed	syed@gmail.com	✕ ✕

Admin songs List:-

Music-Player
(Admin)

Users


Songs

AddSong


Signout


Songs List

🔍 Search by singer, genre, or song name





Genre: Emotional
Singer: shreya ghoshal

▶ 0:00 / 4:03  ⋮





Genre: Emotional
Singer: Arijit Singh

▶ 0:00 / 4:22  ⋮




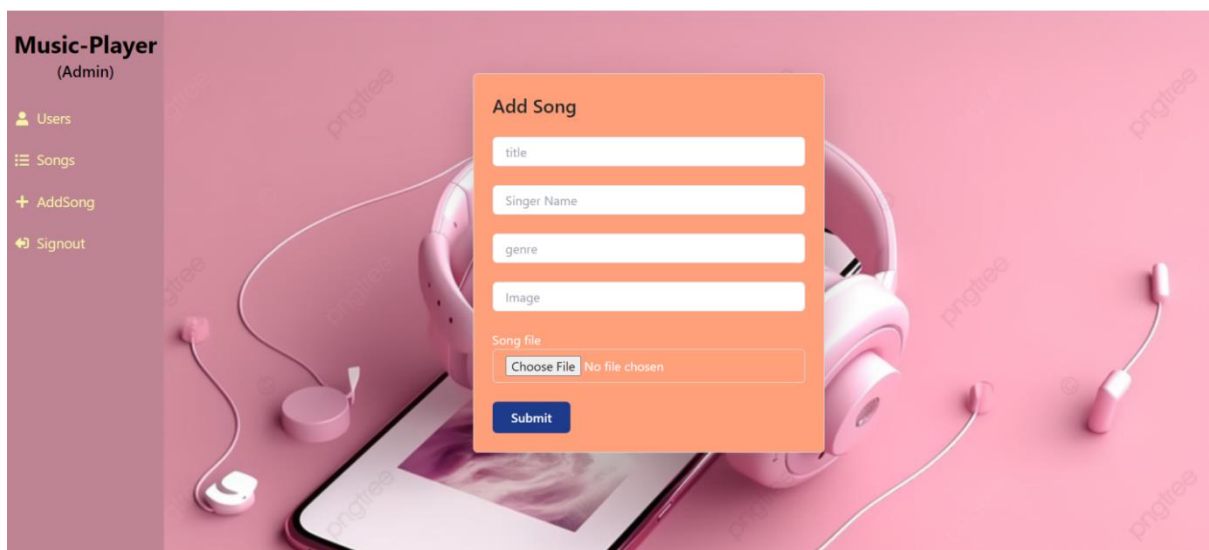
Genre: Romatic
Singer: Arijit Singh

▶ 0:00 / 3:08  ⋮



Genre: Romantic
Singer: Arman Malik

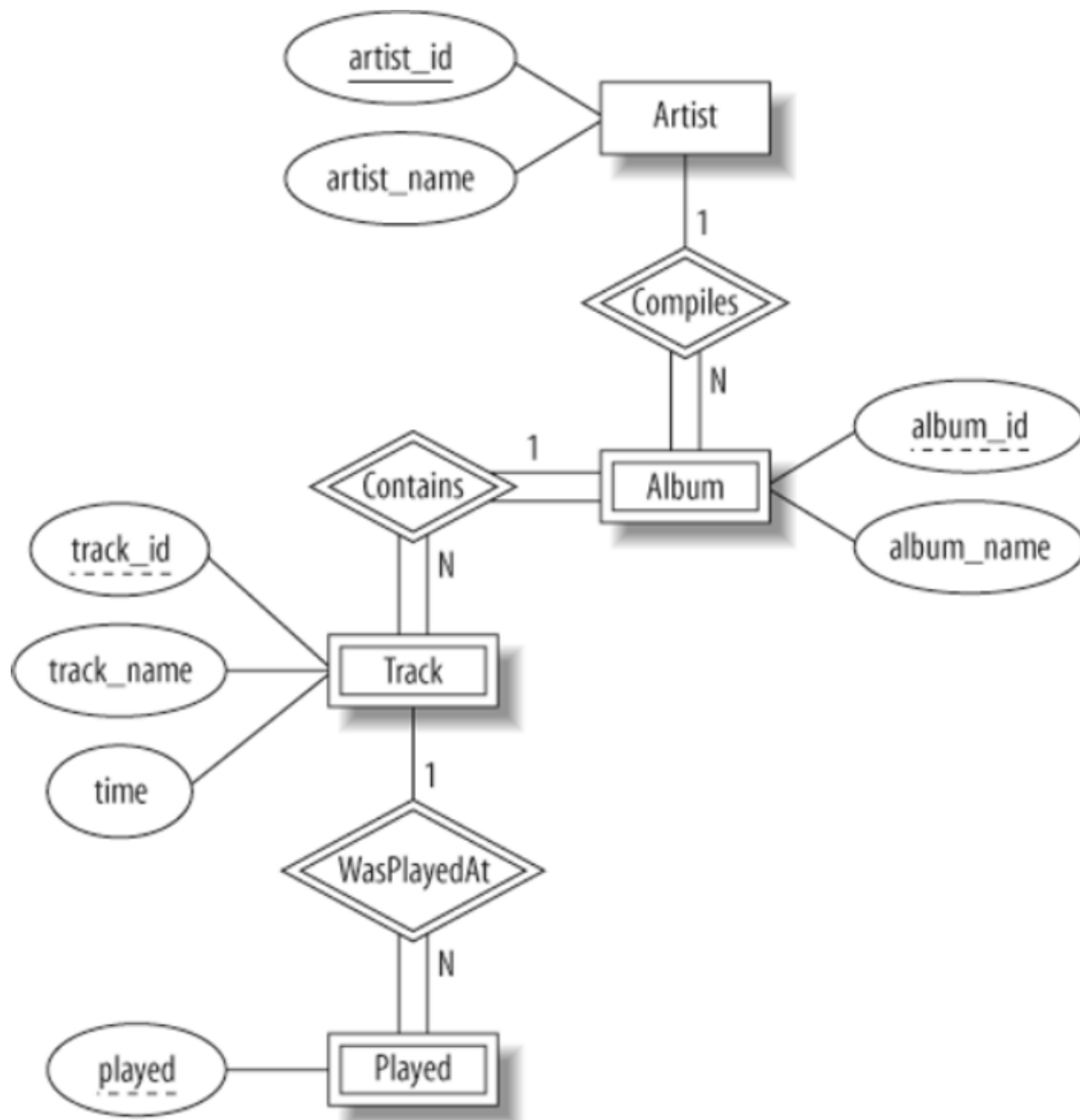
▶ 0:00 / 4:52  ⋮



The demo of the app is available at:-

https://drive.google.com/file/d/1WwxlyubB-C0mnVmHlevEPJKkTB6ymjH3/view?usp=drive_link

ER DIAGRAM:



CONCLUSION: A conclusion for a music streaming app could highlight its impact on the music industry, the convenience it offers to users, the variety of music available, and its potential for future growth and innovation. It could also mention the importance of user experience, personalized recommendations, and the role of technology in shaping the future of music consumption. Overall, the conclusion should emphasize the significance of music streaming apps in shaping how people discover, enjoy, and share music in the digital age.