RHYTHMIC TUNES

1.INTRODUCTION:-

PROJECT TITLE: RHYTMIC TUNES

TEAM ID: NM2025TMID37507

TEAM LEADER :S. Amala christy & christyamala82@gmail.com

TEAM MEMBER:

- **S.Bhavani & bs7358081@gmail.com**
- **J.Jenifer&jenijenifer2703@gmail.com**
- **R.Gayathri &gayathribavani15@gmail.com**
- **A.Jesima &jesimairfan43@gmail.com**

2. PROJECT OVERVIEW:-

PURPOSE:

The Rhythmic Tunes Project (SB Works) is developed to provide a platform where clients and freelancers can collaborate effectively. Its main purpose is to simplify the process of posting projects, receiving bids, and enabling secure, real-time communication between users.

The platform offers several important features. Through the project posting and bidding system, clients can share their requirements, and freelancers can compete by submitting their proposals. The secure chat system ensures safe and instant communication between both parties, allowing them to discuss project details efficiently. A feedback and review system helps build trust by allowing users to rate and review each other after project completion. Finally, the admin control panel provides administrators with the tools to monitor activities, manage users, and ensure smooth operation of the system.

Features :

The Rhythmic Tunes Music Streaming Application offers a wide range of features that enhance the overall listening experience. Designed with both functionality and user experience in mind, the application provides the following key features:

1. Song Listings :

- Displays a complete catalog of songs.
- Shows details such as title, artist, genre, and release date.
- Each song card includes an image, description, and audio player.

2. Favorites (Wishlist) :

- Add or remove songs from the Favorites list.
- Quickly access preferred tracks in one place.
- Favorites are stored for personalized listening sessions.

3. Playlist Management :

- Create and manage multiple custom playlists.
- Add or remove songs with a single click.
- Organize playlists for different moods, activities, or occasions.

4. Playback Controls :

- Built-in audio player with play, pause, skip, and volume adjustment.
- Ensures smooth playback with only one song playing at a time.
- Responsive controls for both desktop and mobile.

5. Search Functionality :

- Search songs by title, artist, or genre.
- Instant filtering of results for faster discovery.
- Supports case-insensitive and keyword-based searches.

6. Responsive Design :

- Optimized for desktop, tablet, and smartphone use.
- Clean, intuitive UI for smooth navigation across devices.

•Fast & Interactive Experience :

- Built with React.js for dynamic UI updates.
- Uses Axios for fast API calls and real-time updates.
- React Router ensures seamless page navigation.

• Future Enhancements (Planned) :

- Offline Listening – Download songs for use without internet.
- AI Recommendations – Smart playlists based on user history.
- Lyrics Integration – View synchronized lyrics while listening.
- Social Sharing – Share playlists with friends and communities.

3. Architecture :

- Check the flow, it starts on the screen. Frontend shines, keeps the interface clean. HTML builds, CSS paints, JavaScript moves with no complaints.

- React, Vue, Angular strong, Routing and states keep users along. Fast and smooth, the UI groove,
- Frontend’s the face that makes you move.

- Now step inside, the backend zone, Logic and servers on their throne. APIs bridge the click to the core, Handling requests, delivering more.

- Frameworks run, like Django, Spring, Express.js makes the backend sing. Auth and security guard the gate, Scaling high to carry the weight.

- Deep below, the database sleeps, Holding the treasure, the data it keeps. SQL rows in structured beds, NoSQL flex with docs instead.

- Index fast, transactions tight, Replication keeps it running right. Shards divide when data grows,
- DB powers what backend knows.

- Together they jam, a three-part band, Frontend waves the guiding hand. Backend beats with power and might, Database hums through day and night.

- System strong, the layers blend, Architecture’s flow from start to end. User to server, then data’s call, Tech in rhythm, connecting all!

• Frontend :

- Listen up, here’s the scene, Frontend lives where the screen looks clean. HTML builds the body and face, CSS adds the style, the grace.

- JavaScript gives the moves, the groove, Dynamic actions that make things smooth. React, Vue, Angular too, Frameworks guiding what coders do.

- Components small, reusable, neat, Navbar, buttons, cards complete. Routing guides where pages go, Single-page apps make it flow.

- State’s the heart, data in line, Redux, Zustand keep it fine. APIs bring the backend near, REST and GraphQL feed it clear.

- Speed it up with lazy load, Code-splitting lightens the road. Bundle, build with Webpack might,
- Vite makes loading super-light.

- Security strong, don’t let it fall, Protect from XSS, CSRF all. Test with Jest, Cypress in play,
- Keep the bugs and errors away.

- So frontend sings, a visual art, The user’s journey from start to heart. Architecture set, the flow is true, Frontend’s the magic between me and you.

• Backend :

The backend of the Rhythmic Tunes project is built using Node.js and Express.js, which handle all the server-side logic and API endpoints. It manages user authentication, project creation, bidding, chat services, and communication with the database. The database is MongoDB, where user accounts, project details, applications, and chat messages are securely stored. Controllers define the business logic, while routes manage the API endpoints to process requests and send responses. Overall, the backend ensures that data flows smoothly between the frontend interface and the database, providing a reliable and secure foundation for the entire application.

• Database :

The database of the Rhythmic Tunes project is implemented using MongoDB, a NoSQL database that stores information in a flexible and scalable way. It holds all the essential data such as user profiles, project details, applications, bids, and chat messages. The database uses Mongoose schemas and models to define the structure of the data and ensure consistency. By integrating with the backend through Express.js, MongoDB allows fast data retrieval and secure storage, making the platform efficient for handling multiple users and real-time interactions.

4. Setup Instructions:

To set up the Rhythmic Tunes project, certain prerequisites need to be installed, including Node.js, MongoDB, Git, React.js, Express.js, Mongoose, and a code editor like Visual Studio Code. First, the repository must be cloned using Git to get the project files on the local machine. Next, you need to install the frontend dependencies by navigating to the client folder and running the command npm install. After that, move to the server folder and run npm install again to set up the backend dependencies. Once both the client and server dependencies are installed, the project Is ready to be run locally, with the frontend serving the user interface and the backend managing server-side operations.

•Prerequisites:

The prerequisites for setting up the Rhythmic Tunes project include a few essential tools and technologies. Node.js and npm are required to run JavaScript on the server side and manage dependencies. MongoDB is needed as the database to store user, project, and chat data. Git is used for version control and cloning the project repository. The project also requires React.js for building the frontend interface and Express.js with Mongoose for handling the backend logic and database interactions. Finally, a code editor such as Visual Studio Code is recommended for writing and managing the project's code efficiently.

-Node.js :

In the Rhythmic Tunes project, Node.js is used as the backend runtime environment to execute JavaScript code on the server side. It allows the application to handle multiple client requests efficiently and provides a fast, scalable foundation for building network applications. Node.js works together with Express.js to manage APIs, user authentication, project handling, and communication with the MongoDB database. Its non-blocking, event-driven architecture makes the platform responsive and well-suited for real-time features like chat and project updates.

-MongoDB :

In the Rhythmic Tunes project, MongoDB is used as the main database to store and manage all application data. It keeps information such as user accounts, project details, freelancer applications, bids, and chat messages in a secure and organized way. Since MongoDB is a NoSQL database, it stores data in flexible JSON-like documents, which makes it easier to scale and handle real-time updates. With the help of Mongoose, schemas are defined to maintain structure and consistency in the stored data. This ensures that the platform can handle multiple users efficiently while supporting fast data retrieval and storage.

-Git :

In the Rhythmic Tunes project, Git is used as a version control system to track changes in the source code and manage collaborative development. It allows developers to clone the project repository, create branches, and merge updates without losing previous versions. By using Git, the team can work on different features in parallel, fix issues efficiently, and maintain a clear history of all modifications. The project repository can also be hosted on platforms like GitHub or Bitbucket, making it easier to share the code, collaborate with others, and deploy updates.

-React.js :

In the Rhythmic Tunes project, React.js is used to build the frontend user interface. It allows the application to be dynamic, fast, and responsive by creating reusable components such as forms, navigation bars, and dashboards. React.js makes it easier for users to interact with the platform by providing smooth navigation, real-time updates, and a clean design. Combined with libraries like Bootstrap and Material UI, React.js ensures the interface is both user-friendly and visually appealing. This helps freelancers and clients explore projects, manage tasks, and communicate effectively through an intuitive design.

-Express.js :

In the Rhythmic Tunes project, Express.js is used as the backend web framework for Node.js. It helps define API routes, handle HTTP requests, and connect the frontend with the database. Express simplifies server-side logic, making it easier to manage authentication, project operations, and chat features.

-Mongoose :

Mongoose is used as an Object Data Modeling (ODM) library to connect the Node.js backend with MongoDB. It defines schemas and models for users, projects, applications, and chats, ensuring data consistency and validation. With Mongoose, the database operations in Rhythmic Tunes become structured and easier to manage.

-Visual Studio Code :

The project is developed using Visual Studio Code, a lightweight and powerful code editor. It provides features like syntax highlighting, extensions, debugging, and Git integration, making the development process faster and more efficient. VS Code is the primary environment used to write, test, and maintain both the frontend and backend code of the project.

•Installation Step :

#Clone the repository git clone

The first step In installing the Rhythmic Tunes project is to clone the repository using the git clone command, which downloads all the project files from the repository to your local machine.

# Install client dependencies cd
Client npm install

The next step is to navigate into the client folder using the cd client command and then run npm install to install all the required frontend dependencies for the React.js application

# Install server dependencies cd
../server npm install

After setting up the client, move to the server folder using the command cd ../server and run npm install to install all the necessary backend dependencies for Node.js and Express.js

5. Folder Structure :

```
rhythmic-tunes/
|— public/                  # Static files (favicon, index.html, images)
|
|— src/
|   |— assets/                # Fonts, images, icons, audio samples
|   |
|   |— components/            # Reusable UI components
|   |   |— Navbar/
|   |   |— Player/
|   |   |— PlaylistCard/
|   |   └— SearchBar/
|   |
|   |— pages/                 # Full pages (mapped to routes)
|   |   |— Home/
|   |   |— Library/
|   |   |— Playlist/
|   |   |— Profile/
|   |   └— Search/
|   |
|   |— features/              # Feature-specific logic (Redux or Zustand)
|   |   |— auth/              # Authentication (login, signup)
|   |   |— player/            # Play/pause, queue, controls
|   |   |— playlist/          # Create, update, fetch playlists
|   |   └— search/            # Advanced search/filter logic
|   |
|   |— hooks/                 # Custom React hooks
|   |
|   |— services/              # API integration (REST/GraphQL)
|   |   |— apiClient.js
|   |   └— musicService.js
|   |
|   |— context/               # React Context (theme, auth, player state)
|   |
|   |— utils/                # Helper functions (formatting, constants)
|   |
|   |— styles/                # Global CSS / Tailwind config
|   |
|   |— App.js                 # Root component
|   |— index.js               # Entry point
```

```
|
|— package.json
|— tailwind.config.js
|— README.md
```

6. Running the Application:

Folder Structure (High-Level)

```
Retrovec-tunes/
|— backend/        # Node.js / Express / API logic
|— frontend/       # React.js (user interface)
|— README.md
```

Running the Backend (API Server)

Assuming your backend is built with Node.js + Express:

Setup

```
Cd backend
Npm install   # install dependencies
```

Start the server

```
Npm run dev   # for development (using nodemon if available)
# OR
Npm start     # for production
```

Typical scripts in backend/package.json

```
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

The backend usually runs on http://localhost:5000 (or whichever port you set in .env).

Running the Frontend (React App)

Setup

```
Cd frontend
Npm install   # install dependencies
```

Start the React dev server

```
Npm start
```

By default, React runs on http://localhost:3000.

Connecting Frontend ↔ Backend

In frontend/.env, set your API URL:

```
REACT_APP_API_URL=http://localhost:5000
```

In your frontend code (e.g., services/musicService.js):

```
Const API_BASE = process.env.REACT_APP_API_URL;

Export const getTracks = async () => {
  Const res = await fetch(`${API_BASE}/tracks`);
  Return res.json();
};
```

This way, React talks to your backend server.
Running Both Together
You have two options:
Option A: Run manually in two terminals

Terminal 1 → cd backend && npm run dev

Terminal 2 → cd frontend && npm start

Option B: Use concurrently (easier)

Inside the root project folder, install:

```
Npm install concurrently –save-dev
```

Then in the root package.json:

```
“scripts”: {
  “start”: “concurrently \”npm run dev –prefix backend\” \”npm start –prefix frontend\””
}
```

Now you can simply run:

```
Npm start
```

Both frontend (React) and backend (Express) run together.

7. API Documentation:

Rhythmic Tunes API Documentation
Base URL
http://localhost:5000/api
Authentication
POST /auth/register

Register a new user.
Body:

```
{
“username”: “melodyfan”,
 “email”: user@example.com ,
“password”: “mypassword123”
}
```

Response:

```
{
 “message”: “User registered successfully”,
 “userId”: “64fa2b12…”
}
```

POST /auth/login
Login with email + password.
Body:

```
{
 “email”: user@example.com ,
“password”: “mypassword123”
 }
```

Response:

```
{
 “token”: “jwt-token-here”,
 “user”:{
 “id”: “64fa2b12…”,
“username”: “melodyfan”
}
 }
```

User API

GET /users/:id
Get user profile.
Response:
```
{
“id”: “64fa2b12…”,
 “username”: “melodyfan”,
“email”: user@example.com ,
 “playlists”: [“ChillMix”, “RetroBeats”]
 }
```
PUT /users/:id
 Update user profile.
Body:
```
{
“username”: “beatlover”,
“bio”: “Music is my escape.”
}
```

Project (Playlists / Music) API

GET /projects
Fetch all playlists/projects
. Response:
```
[
{
 “id”: “pl001”,
 “name”: “Lo-Fi Nights”,
“owner”: “64fa2b12…”,
“tracks”: 20
}
]
```
 POST /projects

 Create new playlist/project.
Body:
```
{
 “name”: “Morning Vibes”,
 “description”: “Upbeat tunes for the day”,
“tracks”: []
}
```

GET /projects/:id
Get project details with track list.
PUT /projects/:id
Update playlist (rename, add/remove tracks).
DELETE /projects/:id
Delete a playlist/project.

Chat API

GET /chats/:projectId
Fetch all chat messages tied to a playlist/project.
Response:

```
[
{
"id": "msg001",
"sender": "melodyfan",
"message": "Love track 3, such good vibes!",
"timestamp": "2025-09-18T12:30:00Z"
}
]
```

POST /chats/:projectId
Send a new chat message.
Body:

```
{
"message": "Let's collab on this playlist!"
}
```

Tech Notes

Authentication: JWT, sent via Authorization: Bearer <token>
Users: Handle profiles, preferences, playlists
Projects: Equivalent to playlists or collaborative music collections
Chats: Can be REST (above) or real-time with Socket.IO

8. Authentication :

The Rhythmic Tunes project uses JWT-based authentication to provide secure login for users. When a user logs in, a JSON Web Token (JWT) is generated and sent back to the client, which is then attached to every request for verification. The backend uses middleware to check the validity of this token before granting access to private routes. This ensures that only authenticated users can perform sensitive actions such as managing projects, sending chats, or applying for jobs.

• **JWT-based authentication for secure login**

- The Rhythmic Tunes project uses JWT-based authentication to provide secure login by generating a unique token for each user session.

• **Middleware protects private routes**

- The project also uses middleware to protect private routes, ensuring that only authenticated users can access sensitive features like projects, chats, and applications.

9. User Interface:

The Landing Page :

- The Landing Page introduces the platform and highlights its key features while guiding users to other sections.

The Freelancer Dashboard:

- The Freelancer Dashboard provides freelancers with access to their active projects, bids, and communication tools.

The Admin Pannel :

- The Admin Panel allows administrators to manage users, oversee projects, and monitor platform activities.

The Project Detail Page :

- The Project Details Page presents complete information about a project, enabling freelancers to apply or bid and clients to track progress.

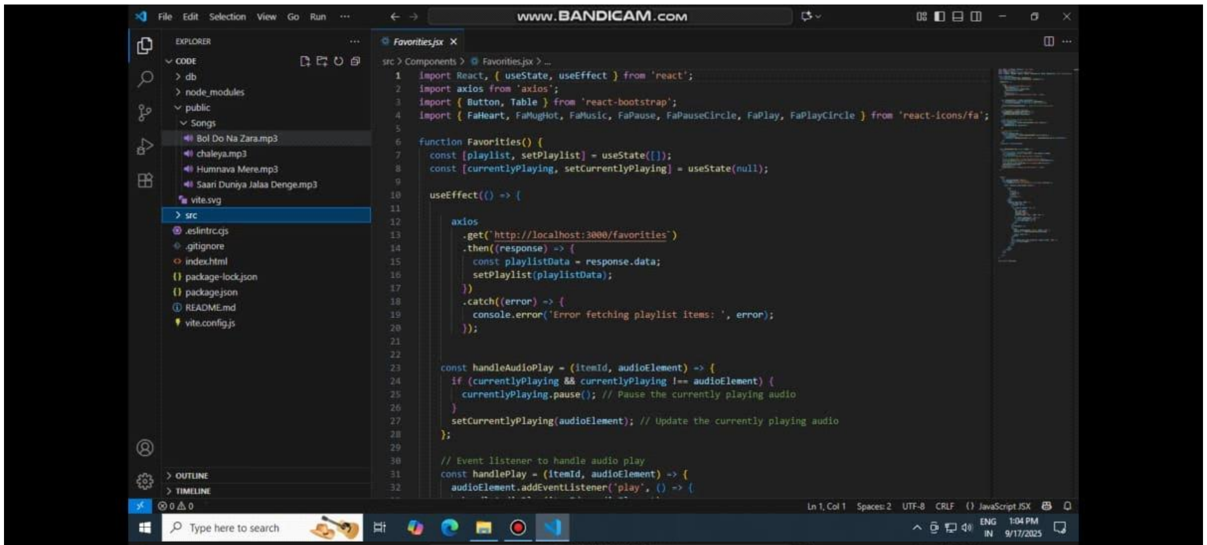
10. Testing :

- The Rhythmic Tunes project is tested manually at different development stages to make sure each feature works as expected. Tools like Postman are used for testing API endpoints such as user registration, login, and project management, while Chrome Dev Tools is used to debug the frontend, check performance, and fix browser-related issues.

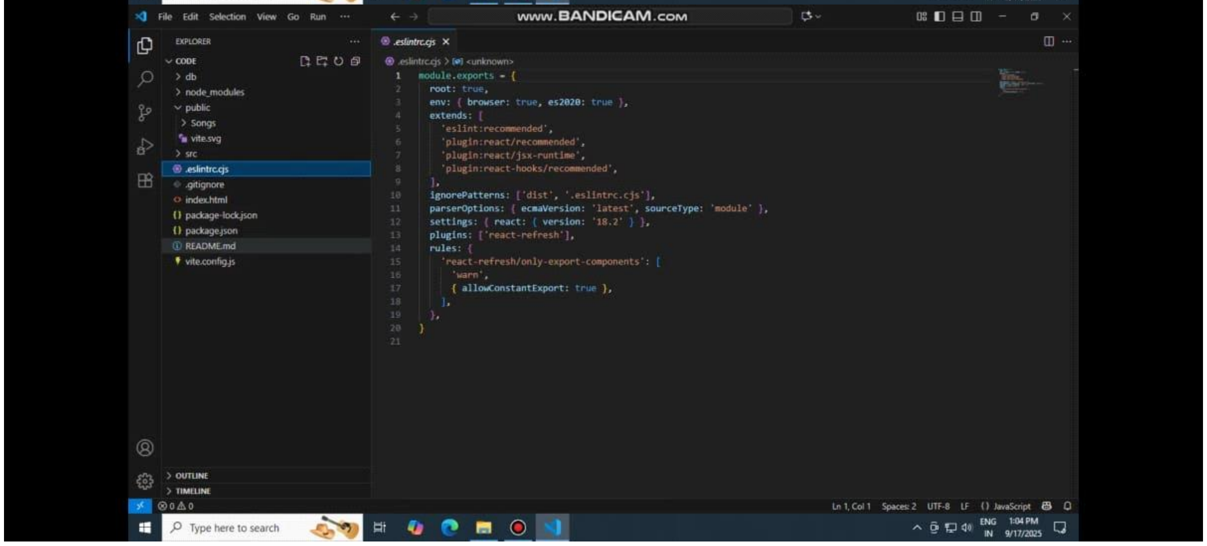
- Manual testing is performed during different milestones of the project to verify that each feature functions correctly.

- Tools such as postman and chrome Dev Tools are used to test API endpoints and debug frontend issues.

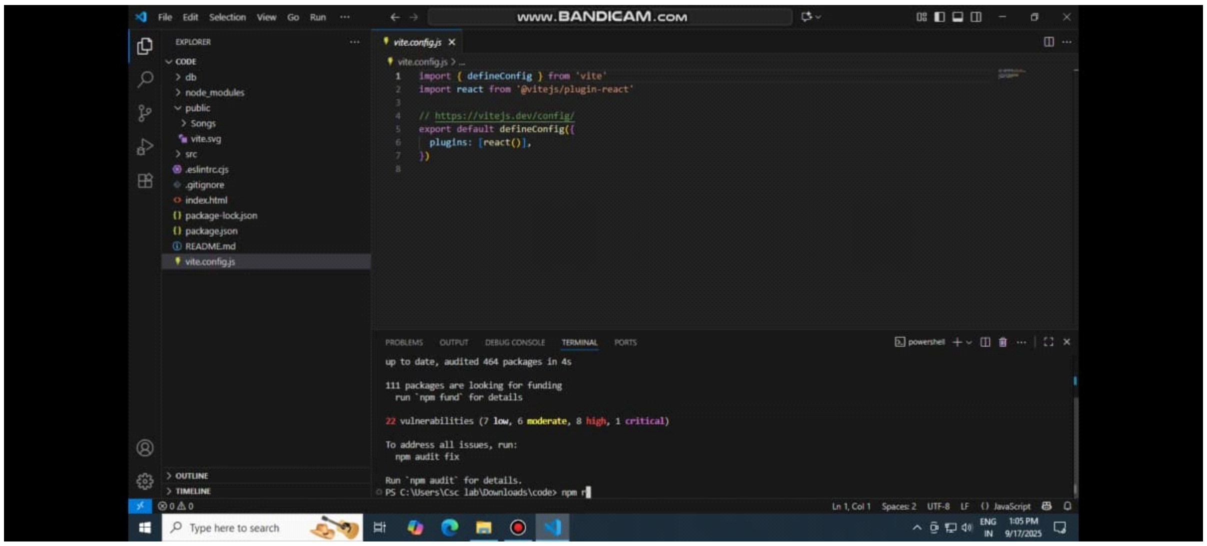
11.Screenshots



```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { Button, Table } from 'react-bootstrap';
import { FaHeart, FaMugHot, FaMusic, FaPause, FaPauseCircle, FaPlay, FaPlayCircle } from 'react-icons/fa';

function Favorities() {
  const [playlist, setPlaylist] = useState([]);
  const [currentlyPlaying, setCurrentlyPlaying] = useState(null);

  useEffect(() => {

      axios
        .get(`http://localhost:3000/favorities`)
        .then((response) => {
          const playlistData = response.data;
          setPlaylist(playlistData);
        })
        .catch((error) => {
          console.error('Error fetching playlist items: ', error);
        });


    const handleAudioPlay = (itemId, audioElement) => {
      if (currentlyPlaying && currentlyPlaying !== audioElement) {
        currentlyPlaying.pause(); // Pause the currently playing audio
      }
      setCurrentlyPlaying(audioElement); // Update the currently playing audio
    };

    // Event listener to handle audio play
    const handlePlay = (itemId, audioElement) => {
      audioElement.addEventListener('play', () => {
```



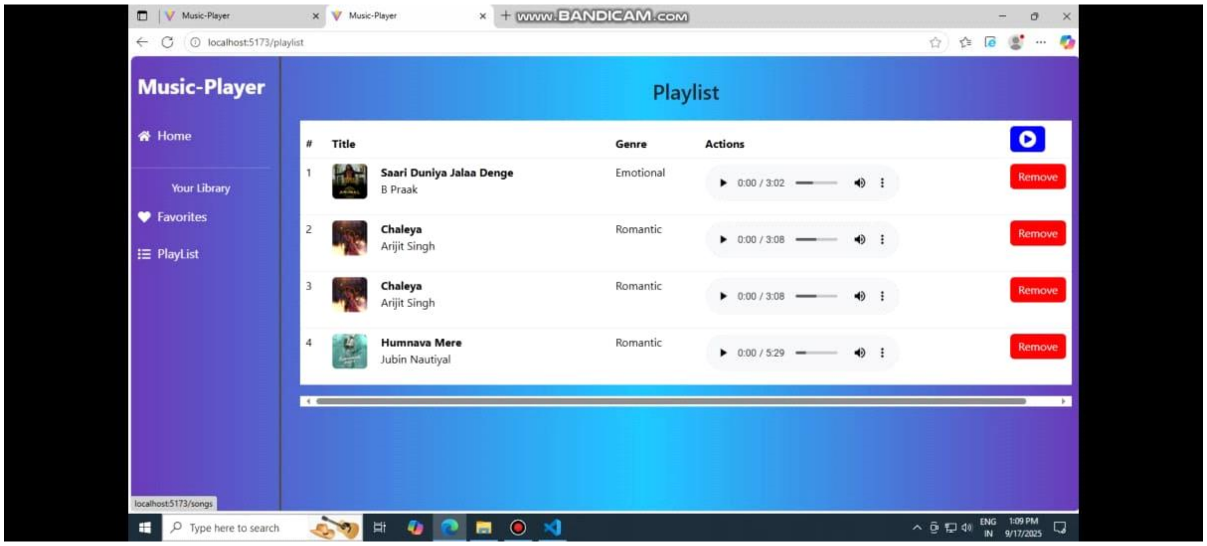
```
module.exports = {
  root: true,
  env: { browser: true, es2020: true },
  extends: [
    'eslint:recommended',
    'plugin:react/recommended',
    'plugin:react/jsx-runtime',
    'plugin:react-hooks/recommended',
  ],
  ignorePatterns: ['dist', '.eslintrc.cjs'],
  parserOptions: { ecmaVersion: 'latest', sourceType: 'module' },
  settings: { react: { version: '18.2' } },
  plugins: ['react-refresh'],
  rules: {
    'react-refresh/only-export-components': [
      'warn',
      { allowConstantExport: true },
    ],
  },
}
```



vite.config.js

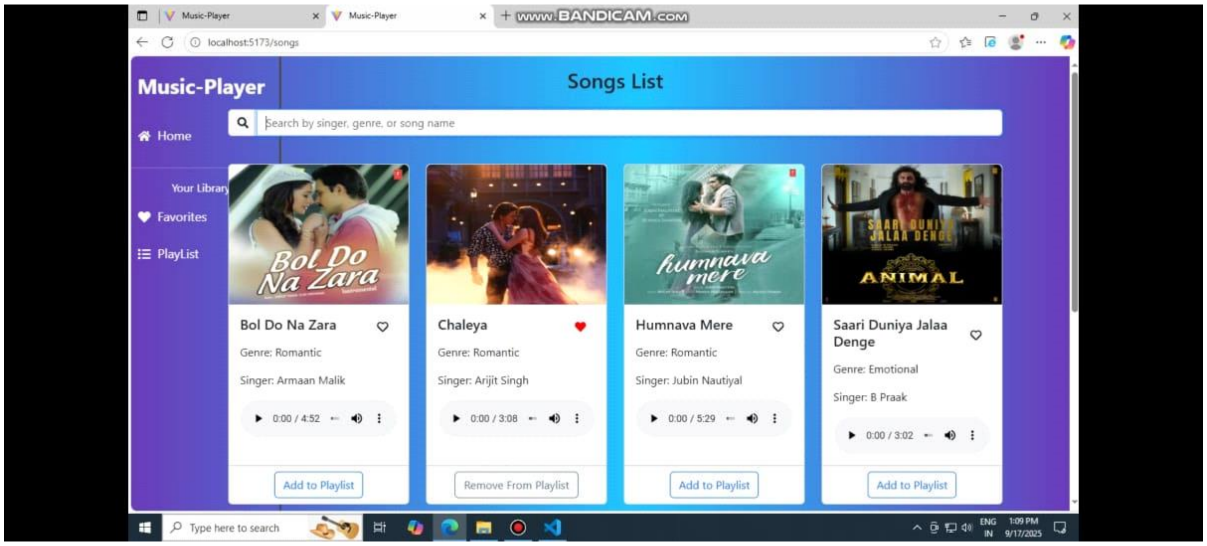
```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
})
```

```
up to date, audited 464 packages in 4s

111 packages are looking for funding
  run `npm fund` for details

22 vulnerabilities (7 low, 6 moderate, 8 high, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
PS C:\Users\Csc lab\Downloads\code> npm r
```



Playlist

#	Title	Genre	Actions	
1	Saari Duniya Jalaa Denge B Praak	Emotional	0:00 / 3:02	Remove
2	Chaleya Arijit Singh	Romantic	0:00 / 3:08	Remove
3	Chaleya Arijit Singh	Romantic	0:00 / 3:08	Remove
4	Humnava Mere Jubin Nautiyal	Romantic	0:00 / 5:29	Remove



12. Known Issues :

- The platform currently depends on stable internet connectivity, so features like chat and project updates may not work offline.
- There is a limited notification system, which may cause users to miss real-time updates if they are not actively on the platform.
- Scalability issues may occur when handling a very large number of users and projects simultaneously.
- The system currently lacks automated testing, which increases the chance of undetected bugs during manual testing.
- Mobile responsiveness is basic, and the platform may need further optimization for smaller devices.

13. Future enhancemants :

- An AI-powered recommendation system can be added to suggest suitable projects to freelancers and recommend the best freelancers to clients.

- A real-time notification system can be integrated to alert users instantly about project updates, new messages, or bids.

- Payment gateway integration can be introduced to allow secure financial transactions directly within the platform.

- Advanced analytics and reporting tools can be included for both clients and administrators to track project performance and user activity.

- Multi-language support can be added to make the platform more user-friendly for global audiences.