

Team Leader:M.DILLI RANI

Email ID:m.dillirani2003@gmail.com

Team member:L.ASWINI

Email ID:aswinilaswini9@gmail.com

Team member:M.HARISRI

Email ID:harisrim13@gmail.com

Team member:B.GATCHIYAL

Email ID:gatchiyalg@gmail.com

RYTHMIC TUNES

Abstract for the Program "Rhythmic Tunes":

"Rhythmic Tunes" is an innovative and engaging musical program designed to explore the art of rhythm and melody through diverse musical styles and genres. The program aims to foster creativity, improve musical literacy, and provide a platform for individuals to connect with the transformative power of music. Through interactive sessions, participants will dive deep into rhythmic patterns, understanding their role in creating harmony and musical flow. "Rhythmic Tunes" features hands-on exercises, expert-led discussions, and collaborative performances, allowing individuals of all skill levels to develop their musical talents and gain a deeper appreciation for the rhythmic foundation of music. Whether a beginner or a seasoned musician, the program invites everyone to discover the beauty of rhythm in an inclusive, inspiring .

Introduction:-

Welcome to the future of musical indulgence – an unparalleled audio experience awaits you with our cutting-edge Music Streaming Application, meticulously crafted using the power of React.js. 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 React-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.

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

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 Intro:-

Imagine stepping onto a bustling city street, the sounds of cars honking, people chatting, and street performers playing in the background. You're on your way to work, and you need a little something to elevate your mood. You pull out your phone and open your favorite music streaming app, "RythimicTunes."

With just a few taps, you're transported to a world of music tailored to your tastes. As you walk, the app's smart playlist kicks in, starting with an upbeat pop song that gets your feet tapping. As you board the train, the music shifts to a relaxing indie track, perfectly matching your need to unwind during the commute.

Target Audience:-

Music Streaming is designed for a diverse audience, including:

- **Music Enthusiasts:** People passionate about enjoying and listening Music Through out there free time to relax themselves.

Project Goals and Objectives:-

The primary goal of Music Streaming is to provide a seamless platform for music enthusiasts, enjoying, and sharing diverse musical experiences. Our objectives include:

User-Friendly Interface: Develop an intuitive interface that allows users to effortlessly explore, save, and share their favorite music tracks and playlists.

Comprehensive Music Streaming: Provide robust features for organizing and managing music content, including advanced search options for easy discovery.

Modern Tech Stack: Harness cutting-edge web development technologies, such as React.js, to ensure an efficient and enjoyable user experience while navigating and interacting with the music streaming application.

Key Features:-

- ? **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.
- ? **Search Functionality:** Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

PRE-REQUISITES:-

Here are the key prerequisites for developing a frontend application using React.js:

? **Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions:
<https://nodejs.org/en/download/package-manager/>

? **React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

- Create a new React app:

```
npm create vite@latest
```

Enter and then type project-name and select preferred frameworks and then enter

- Navigate to the project directory:

```
cd project-name npm install
```

- Running the React App:

With the React app created, you can now start the development server and see your React application in action.

- Start the development server:

```
npm run dev
```

This command launches the development server, and you can access your React app at <http://localhost:5173> in your web browser.

? **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.

? **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://git-scm.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.

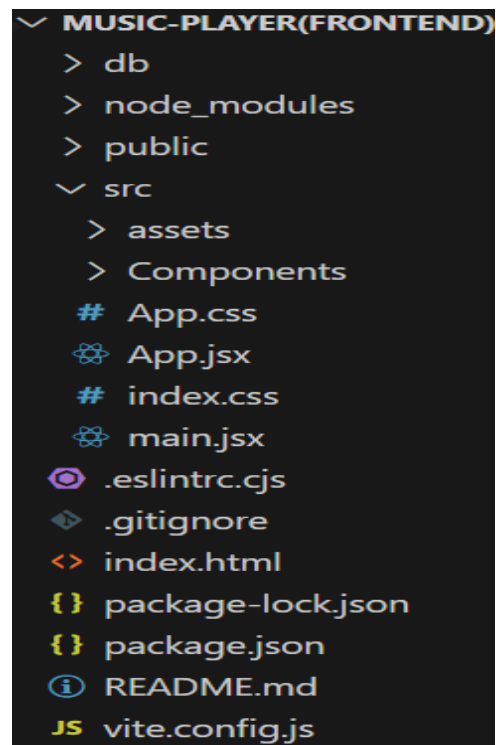
- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>

•WebStorm:Download

from

<https://www.jetbrains.com/webstorm/download>

Project structure:



```

MUSIC-PLAYER(FRONTEND)
├── db
├── node_modules
├── public
├── src
│   ├── assets
│   ├── Components
│   ├── App.css
│   ├── App.jsx
│   ├── index.css
│   ├── main.jsx
│   ├── .eslintrc.cjs
│   ├── .gitignore
│   ├── index.html
│   ├── package-lock.json
│   ├── package.json
│   ├── README.md
│   └── vite.config.js

```

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:-

Project demo:

Before starting to work on this project, let's see the demo.

Demolink:

https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=drive_link

Use the code in:

https://drive.google.com/drive/folders/1BkYWfW_K3ek_UgtXNTAsDqlhdCuqz6nT?usp=drive_link

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- **Installation of required tools:**

1. Open the project folder to install necessary tools In this project, we use:
 - React Js
 - React Router Dom
 - React Icons
 - Bootstrap/tailwind css
 - Axios

- For further reference, use the following resources
 - <https://react.dev/learn/installation>
 - <https://react-bootstrap-v4.netlify.app/getting-started/introduction/>
 - <https://axios-http.com/docs/intro>
 - <https://reactrouter.com/en/main/start/tutorial>

Milestone 2: Project Development:

1. Setup React Application:

- Create React application.
- Configure Routing.

- Install required libraries. Setting Up Routes:-

```
import 'bootstrap/dist/css/bootstrap.min.css';
import './App.css'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import Songs from './Components/Songs'
import Sidebar from './Components/Sidebar'
import Favorites from './Components/Favorites'
import Playlist from './Components/Playlist';

function App() {

  return (
    <div>
      <BrowserRouter>
      <div>
        <Sidebar/>
      </div>
      <div>
        <Routes>
          <Route path="/" element={<Songs/>} />
          <Route path="/favorites" element={<Favorites/>} />
          <Route path="/playlist" element={<Playlist/>} />
        </Routes>
      </div>
    </BrowserRouter>
  </div>
  )
}

export default App
```

Code Description:-

- Imports Bootstrap CSS (bootstrap/dist/css/bootstrap.min.css) for styling components.
- Imports custom CSS (./App.css) for additional styling.

- Imports `BrowserRouter`, `Routes`, and `Route` from `react-router-dom` for setting up client-side routing in the application.
- Defines the `App` functional component that serves as the root component of the application.
- Uses `BrowserRouter` as the router container to enable routing functionality.
- Includes a `div` as the root container for the application.
- Within `BrowserRouter`, wraps components inside two `div` containers:
 - The first `div` contains the `Sidebar` component, likely serving navigation or additional content.
 - The second `div` contains the `Routes` component from `React Router`, which handles rendering components based on the current route.
 - Inside `Routes`, defines several `Route` components:
 - `Route` with `path='/'` renders the `Songs` component when the root path is accessed (`/`).
 - `Route` with `path='/favorites'` renders the `Favorites` component when the `/favorites` path is accessed.

- Route with `path='/playlist'` renders the Playlist component when the

`/playlist` path is accessed.

- Exports the App component as the default export, making it available for use in other parts of the application.

```

import React, { useState, useEffect } from 'react';
import { Button, Form, InputGroup } from 'react-bootstrap';
import { FaHeart, FaRegHeart, FaSearch } from 'react-icons/fa';
import axios from 'axios';
import './sidebar.css'

function Songs() {
  const [items, setItems] = useState([]);
  const [wishlist, setWishlist] = useState([]);
  const [playlist, setPlaylist] = useState([]);
  const [currentlyPlaying, setCurrentlyPlaying] = useState(null);
  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {
    // Fetch all items
    axios.get('http://localhost:3000/items')
      .then(response => setItems(response.data))
      .catch(error => console.error('Error fetching items: ', error));

    // Fetch favorites items
    axios.get('http://localhost:3000/favorites')
      .then(response => setWishlist(response.data))
      .catch(error => {
        console.error('Error fetching Favvorities:', error);
        // Initialize wishlist as an empty array in case of an error
        setWishlist([]);
      });

    // Fetch playlist items
    axios.get('http://localhost:3000/playlist')
      .then(response => setPlaylist(response.data))
      .catch(error => {
        console.error('Error fetching playlist: ', error);
        // Initialize playlist as an empty array in case of an error
        setPlaylist([]);
      });

    // Function to handle audio play
    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', () => {
        handleAudioPlay(itemId, audioElement);
      });
    };

    // Add event listeners for each audio element
    items.forEach((item) => {
      const audioElement = document.getElementById(`audio-${item.id}`);
      if (audioElement) {
        handlePlay(item.id, audioElement);
      }
    });

    // Cleanup event listeners
    return () => {
      items.forEach((item) => {
        const audioElement = document.getElementById(`audio-${item.id}`);
        if (audioElement) {
          audioElement.removeEventListener('play', () => handleAudioPlay(item.id, audioElement));
        }
      });
    };
  }, [items, currentlyPlaying, searchTerm]);

  const addToWishlist = async (itemId) => {
    try {
      const selectedItem = items.find((item) => item.id === itemId);
      if (!selectedItem) {
        throw new Error('Selected item not found');
      }
      const { title, imgUrl, genre, songUrl, singer, id: itemId2 } = selectedItem;
      const response = await axios.post('http://localhost:3000/favorites', { itemId: itemId2, title, imgUrl, genre, songUrl, singer });
      setWishlist(response.data);
    } catch (error) {
      console.error('Error adding item to wishlist: ', error);
    }
  };

```

Code Description:-

- **useState:**

- items: Holds an array of all items fetched from

[http://localhost:3000/items.](http://localhost:3000/items)

wishlist: Stores items marked as favorites fetched from

[http://localhost:3000/favorites.](http://localhost:3000/favorites)

- playlist: Stores items added to the playlist fetched from
[http://localhost:3000/playlist.](http://localhost:3000/playlist)

- currentlyPlaying: Keeps track of the currently playing audio element.

- searchTerm: Stores the current search term entered by the user.

- **Data Fetching:**

- Uses useEffect to fetch data:

- Fetches all items (items) from
[http://localhost:3000/items.](http://localhost:3000/items)

- Fetches favorite items (wishlist) from
[http://localhost:3000/favorites.](http://localhost:3000/favorites)

- Fetches playlist items (playlist) from
[http://localhost:3000/playlist.](http://localhost:3000/playlist)

- Sets state variables (items, wishlist, playlist) based on the fetched data.

- **Audio Playback Management:**

- Sets up audio play event listeners and cleanup for each item:
 - `handleAudioPlay`: Manages audio playback by pausing the currently playing audio when a new one starts.
 - `handlePlay`: Adds event listeners to each audio element to trigger

`handleAudioPlay`.

- Ensures that only one audio element plays at a time by pausing others when a new one starts playing.

- **`addToWishlist(itemId)`:**

- Adds an item to the wishlist (favorites) by making a POST request to

<http://localhost:3000/favorites>.

- Updates the wishlist state after adding an item.

- **`removeFromWishlist(itemId)`:**

- Removes an item from the wishlist (favorites) by making a DELETE request to

<http://localhost:3000/favorites/{itemId}>.

- Updates the wishlist state after removing an item.

Adds an item to the playlist (playlist) by making a POST request to <http://localhost:3000/playlist>.

- Updates the playlist state after adding an item.

- **removeFromPlaylist(itemId):**

- Removes an item from the playlist (playlist) by making a DELETE request to

<http://localhost:3000/playlist/{itemId}>.

- Updates the playlist state after removing an item.

- **isItemInPlaylist(itemId):**

- Checks if an item exists in the playlist (playlist) based on its itemId.

- **filteredItems:**

- Filters items based on the searchTerm.
- Matches title, singer, or genre with the lowercase version of

searchTerm.

- **JSX:**

- Renders a form with an input field (Form, InputGroup, Button, FaSearch) for searching items.

- Maps over filteredItems to render each item in the UI.
- Includes buttons (FaHeart, FaRegHeart) to add/remove items from

wishlist and playlist.

- Renders audio elements for each item with play/pause functionality.

- **Error Handling:**

- Catches and logs errors during data fetching (axios.get).
- Handles errors when adding/removing items from wishlist and playlist.

Frontend Code For Displaying Songs:-

```

return (
  <div style={{display:"flex", justifyContent:"flex-end"}}>
    <div className="songs-container" style={{width:"1300px"}}>
      <div className="container mx-auto p-3">
        <h2 className="text-3xl font-semibold mb-4 text-center">Songs List</h2>
        <InputGroup className="mb-3">
          <InputGroup.Text id="search-icon">
            <FaSearch />
          </InputGroup.Text>
          <Form.Control
            type="search"
            placeholder="Search by singer, genre, or song name"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            className="search-input"
          />
        </InputGroup>
        <br />
        <div className="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-4">
          {filteredItems.map((item, index) => (
            <div key={item.id} className="col">
              <div className="card h-100">
                <img
                  src={item.imgUrl}
                  alt="Item Image"
                  className="card-img-top rounded-top"
                  style={{ height: '200px', width: '100%' }}
                />
                <div className="card-body">
                  <div className="d-flex justify-content-between align-items-center mb-2">
                    <h5 className="card-title">{item.title}</h5>
                    {isItemInWishlist(item.id) ? (
                      <Button
                        variant="light"
                        onClick={() => removeFromWishlist(item.id)}
                      >
                        <FaHeart color="red" />
                      </Button>
                    ) : (
                      <Button
                        variant="light"
                        onClick={() => addToWishlist(item.id)}
                      >

```

```

                        <FaRegHeart color="black" />
                      </Button>
                    )}
                  </div>
                  <p className="card-text">Genre: {item.genre}</p>
                  <p className="card-text">Singer: {item.singer}</p>
                  <audio controls className="w-100" id={`audio-${item.id}`} >
                    <source src={item.songUrl} />
                  </audio>
                </div>
                <div className="card-footer d-flex justify-content-center">
                  {isItemInPlaylist(item.id) ? (
                    <Button
                      variant="outline-secondary"
                      onClick={() => removeFromPlaylist(item.id)}
                    >
                      Remove From Playlist
                    </Button>
                  ) : (
                    <Button
                      variant="outline-primary"
                      onClick={() => addToPlaylist(item.id)}
                    >
                      Add to Playlist
                    </Button>
                  )}
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
);
}

export default Songs;

```

- **Container Setup:**

- Uses a div with inline styles (`style={{display:"flex", justify-content:"flex-end"}}`) to align the content to the right.
- The main container (`songs-container`) has a fixed width (`width:"1300px"`) and contains all the UI elements related to songs.

- **Header:**

- Displays a heading (`<h2>`) with text "Songs List" centered (`className="text-3xl font-semibold mb-4 text-center"`).

- **Search Input:**

- Utilizes `InputGroup` from `React Bootstrap` for the search functionality.
- Includes an input field (`Form.Control`) that allows users to search by singer, genre, or song name.
- Binds the input field value to `searchTerm` state (`value={searchTerm}`) and updates it on change (`onChange={(e) => setSearchTerm(e.target.value)}`).
- Styled with `className="search-input"`.

- **Card Layout:**

- Uses Bootstrap grid classes (row, col) to create a responsive card layout (className="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-4").
- Maps over filteredItems array and renders each item as a Bootstrap card (<div className="card h-100">).

● Card Content:

- Displays the item's image (), title (<h5 className="card-title">), genre (<p className="card-text">), and singer (<p className="card-text">).
- Includes an audio player (<audio controls className="w-100" id={audio-\${item.id}}>) for playing the song with a source (<source src={item.songUrl} />).

● Wishlist and Playlist Buttons:

- Adds a heart icon button (<Button>) to add or remove items from the wishlist (isItemInWishlist(item.id) determines which button to show).
- Includes an "Add to Playlist" or "Remove From Playlist" button (<Button>) based on whether the item is already in the playlist (isItemInPlaylist(item.id)).

● Button Click Handlers:

- Handles adding/removing items from the wishlist (addToWishlist(item.id), removeFromWishlist(item.id)).
- Manages adding/removing items from the playlist (addToPlaylist(item.id), removeFromPlaylist(item.id)).

- **Card Styling:**

- Applies Bootstrap classes (card, card-body, card-footer) for styling the card components.
- Uses custom styles (rounded-top, w-100) for specific elements like images and audio players.

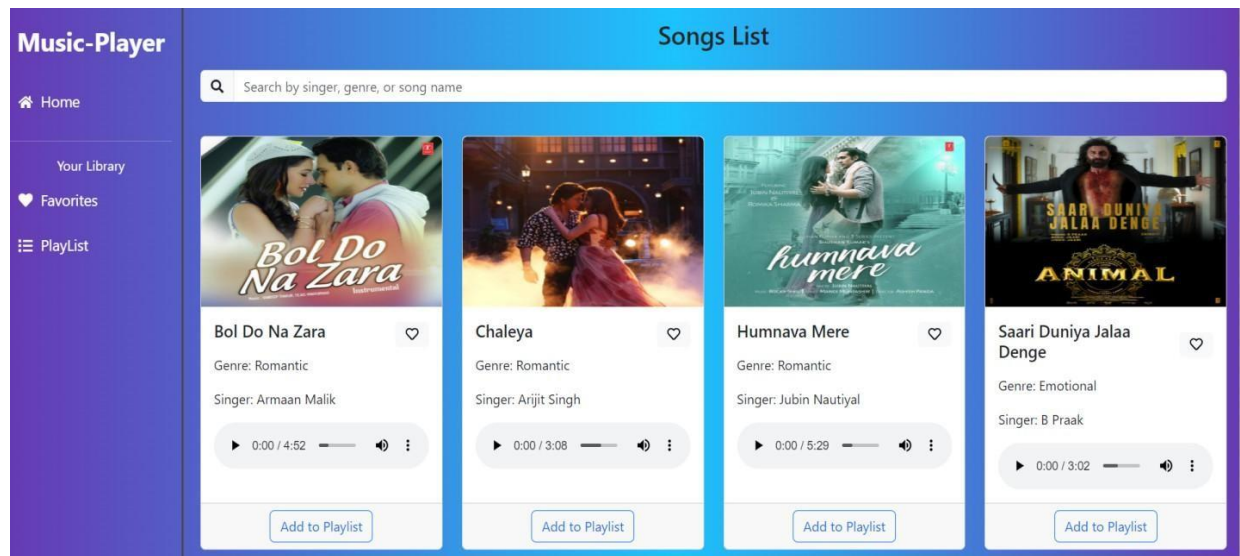
Project Execution:

After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

And the Open new Terminal type this command “json-server --watch ./db/db.json” to start the json server too.

After that launch the Rythimic Tunes.

Here are some of the screenshots of the application.



? **Hero components**

Music-Player

- Home
- Your Library
- Favorites
- PlayList

Singer: Armaan Malik

0:00 / 4:52

Add to Playlist

Singer: Arijit Singh

0:00 / 3:08

Add to Playlist

Singer: Jubin Nautiyal

0:00 / 5:29

Add to Playlist

Singer: B Praak

0:00 / 3:02

Add to Playlist

Sanam Teri Kasam

Genre: Emotional

Singer: Ankit Tiwari

0:00 / 5:14

Add to Playlist

Tum Hi Ho

Genre: Emotional

Singer: Arijit Singh

0:00 / 4:22

Add to Playlist

zihal-e-misk

Genre: Emotional

Singer: Shreya Ghoshal

0:00 / 4:03

Add to Playlist

Music-Player

- Home
- Your Library
- Favorites
- PlayList

Playlist

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

? **Playlist**

? **Favorites**

Music-Player



🏠 Home

📖 Your Library

♥ Favorites

☰ PlayList

Favorites

#	Title	Genre		Actions
1	 Chaleya Arijit Singh	Romantic	♥	▶ 0:00 / 3:08 ⏮ ⏭ 🔊 ⋮
2	 Bol Do Na Zara Armaan Malik	Romantic	♥	▶ 0:00 / 4:52 ⏮ ⏭ 🔊 ⋮

Project Demo link:

https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=drive_link

FUTURE SCOPE

Future Scope of "Rhythmic Tunes":

The future of "Rhythmic Tunes" holds immense potential for growth and expansion, particularly as interest in music education and creative expression continues to rise globally. Some key areas for future development include:

- 1. Expansion into Digital Platforms:** With the rise of online learning and virtual experiences, "Rhythmic Tunes" could develop a digital platform or app, offering interactive lessons, virtual workshops, and global collaborations. This would make the program accessible to a larger audience and break down geographical barriers.
- 2. Incorporating Technology in Music Creation:** As technology continues to evolve, the program can integrate

digital tools such as music production software, drum machines, and virtual instruments to give participants a deeper understanding of rhythm in modern music production.

3. **Collaborations with Schools and Universities:** "Rhythmic Tunes" can partner with educational institutions to create specialized curriculums or extracurricular programs. This would provide students with a comprehensive understanding of rhythm across various genres, from classical to contemporary styles.
4. **Global Cultural Exchange:** Expanding the program to include rhythm traditions from diverse cultures worldwide would broaden participants' musical horizons, fostering global musical understanding and collaboration.
5. **Workshops and Masterclasses with Renowned Artists:** By inviting renowned musicians and percussionists for workshops and masterclasses, the program can continue to offer unique learning opportunities, inspiring future generations of musicians.

6. Integration with Other Art Forms: Collaborating with dance, theater, and visual arts to explore the intersection of rhythm and movement could offer a holistic approach to creative expression, enhancing the learning experience for participants.

7. Certification and Professional Development: Offering certification programs or professional development tracks for music educators and performers could elevate the program's credibility and offer tangible career benefits to participants.

In summary, "Rhythmic Tunes" has the opportunity to evolve into a versatile and globally recognized program that adapts to the evolving needs of the music community while embracing new technologies and artistic disciplines.

CONCLUSION

"Rhythmic Tunes" serves as a dynamic platform for individuals to explore and celebrate the essential role of rhythm in music. By blending traditional and modern approaches to music education, the program fosters creativity, collaboration, and a deeper understanding of musical expression. Its ability to adapt to technological advancements and cultural diversity ensures that it can cater to a wide audience, from beginners to seasoned musicians. As it continues to grow, "Rhythmic Tunes" will remain a source of inspiration, encouraging lifelong learning and fostering a global community united through the universal language of rhythm. Ultimately, the program not only enhances musical skills but also nurtures a profound appreciation for the power of rhythm in shaping both music and culture.