

ANNA ADARSH COLLEGE FOR WOMEN

DEPARTMENT OF BCA - SHIFT - II

COLLEGE CODE : 1353

TEAM ID :SWTID1741175743150646

TEAM MEMBERS NAME

Team Leader :

Deepikasri S (Documentation, Video, Coding)

Team member :

Ramya G (Coding)

Team member :

Dharshini V (Documentation)

Team member :

NehaShree K (Documentation)

Team member :

Pavithra Rani A S (Coding)

Rhythmic Tunes

1.Introduction:

- Rhythmic Tunes is a music-based project that aims to create an interactive platform for users to explore and engage with rhythmic patterns and melodies. The project combines music theory, technology, and user experience design to provide an immersive and educational experience.
- Design and develop a web-based application that allows users to interact with rhythmic patterns and melodies.
- Provide an engaging and informative experience for users to learn about music theory, rhythm, and melody.
- Encourage users to explore and create their own rhythmic patterns and melodies.

2.Project Overview:

PURPOSE:

- 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 in a visually stunning and interactive interface, where every click, scroll, and playlist creation feels like a musical revelation.
- Whether on a desktop, tablet, or smartphone, our responsive design ensures a consistent and enjoyable experience across all devices.

Scenario-Based

- Imagine stepping onto a bustling city street, the sounds of cars honking, people chatting, and street performers playing in the background.
- Way to work, and you need a little something to elevate your mood. You pull out your phone and open favorite music streaming app, "RythimicTunes."
- With just a few taps, transported to a world of music tailored to tastes.
- The app's smart playlist kicks in, starting with an upbeat pop song that gets feet tapping. As you board the train, the music shifts to a relaxing indie track, perfectly matching need to unwind during the commute.

Features:-

1. **Song Listings:** Display a comprehensive list of available songs with details such as title, artist, genre, and release date.
2. **Playlist Creation:** Empower users to create personalized playlists, adding and organizing songs based on their preferences.
3. **Playback Control:** Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback.
4. **Offline Listening:** Allow users to download songs for offline listening, enhancing the app's accessibility and convenience.
5. **Search Functionality:** Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

3.Architecture:

Component Structure:

1. **Login/Registration Screen:** Describe the layout, design, and functionality of the login and registration screens.

2. **Music Library:** Outline the organization and structure of the music library, including how users can browse, search, and select music.
3. **Player Controls:** Describe the player controls, such as play, pause, stop, and seek, and how they interact with the music library.
4. **Rhythm Analysis Visualizer:** If your application includes a visualizer for rhythm analysis, describe its functionality and design.
5. **Music Classification Model:** Describe the music classification model used, including any machine learning algorithms or feature extraction techniques.
6. **Genre Recognition Module:** Outline the functionality of the genre recognition module, including how it identifies genres from audio signals.
7. **Mood Analysis Module:** Describe the mood analysis module, including how it extracts mood features from audio signals.

State Management:

1. ***User Authentication State:*** Describe how your application manages user authentication, including login, registration, and session management.
2. ***Music Library State:*** Outline how your application manages the music library, including music metadata, playback state, and user preferences.
3. ***Rhythm Analysis State:*** Describe how your application manages the rhythm analysis state, including the current rhythm pattern, tempo, and time signature.
4. ***User Interactions:*** Describe how user interactions, such as button clicks or screen taps, update the application state.
5. ***API Calls:*** Outline how API calls, such as fetching music metadata or rhythm analysis results, update the application state.
6. ***Timer Events:*** Describe how timer events, such as playback progress or rhythm analysis updates, update the application state.

4.Setup Instructions:

PREREQUISITES:-

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 us 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 development machine, as they are required to run JavaScript on the server-side.

- Download: Node.js Version V22.14.0 [LTS]
- Installation **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, we can now start the development server and see React application in action.

- Start the development server:

```
npm run dev
```

This command launches the development server, and we can access 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

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:** <https://code.visualstudio.com/download>

To get the Application project from drive:

Follow below steps:

✓Get the code:

- Download the code from the drive link given below:

Install Dependencies:

- Navigate into the cloned repository directory and install libraries:

cd fitness-app-react

npm install

✓Start the Development Server:

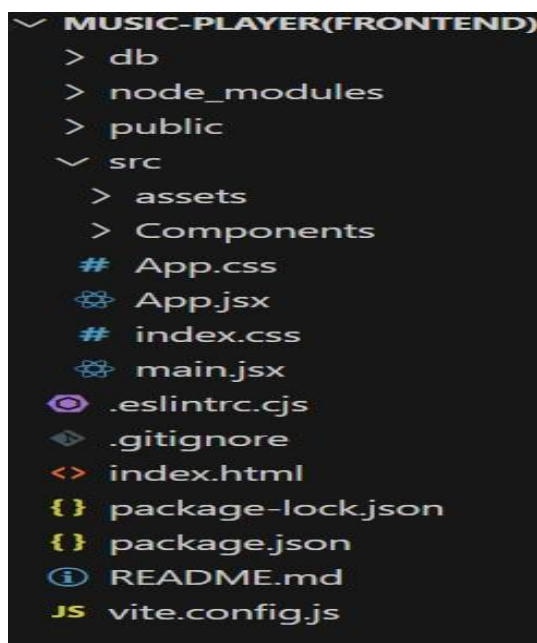
- To start the development server, execute the following command:

npm start

Access the App:

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the application's homepage, indicating that the installation and setup were successful.

Project structure:



- 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.

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 oReact Router
Dom oReact Icons
oBootstrap/tailwind css
- Axios

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 'module "c:/Users/arsha/OneDrive/Desktop/MY PROJECTS/Music-Player(Frontend)/src/Components/Playlist"'
import Favorities from './Components/Favorities'
import Playlist from './Components/Playlist';

function App() {

  return (
    <div>
      <BrowserRouter>
      <div>
        <Sidebar/>
      </div>
      <div>
        <Routes>
          <Route path="/" element={<Songs/>} />
          <Route path="/favorities" element={<Favorities/>} />
          <Route path="/playlist" element={<Playlist/>} />
        </Routes>
      </div>
    </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.

Fetching Songs:-

```

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;
      await axios.post('http://localhost:3000/favorites', { itemId: itemId2, title, imgUrl, genre, songUrl, singer });
      const response = await axios.get('http://localhost:3000/favorites');
      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`.
- wishlist: Stores items marked as favorites fetched from `http://localhost:3000/favorites`.
- playlist: Stores items added to the playlist fetched from `http://localhost:3000/playlist`. `ocurrentlyPlaying`: Keeps track of the currently playing audio element. `osearchTerm`: 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`.
- Fetches favorite items (wishlist) from `http://localhost:3000/favorites`.
- Fetches playlist items (playlist) from `http://localhost:3000/playlist`. `oSets` 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.
- **isItemInWishlist(itemId):**
 - Checks if an item exists in the wishlist (favorites) based on its itemId.
- **addToPlaylist(itemId):**
 - 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>
);
export default Songs;

```

Code Description:-

- **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.

Testing:

Black Box Testing:

- Black Box Testing, also known as Functional Testing, is a software testing technique where the tester has no knowledge of the internal workings or structure of the application.
- The focus is on the input and output of the application, without considering how it is implemented.

White Box Testing:

- White Box Testing, also known as Glass Box Testing or Clear Box Testing, is a software testing technique where the tester has knowledge of the internal workings or structure of the application.
- The focus is on the internal logic, code, and architecture of the application.

Integration Testing:

- Integration Testing is a software testing technique where individual modules or components are combined and tested as a group to ensure they work together seamlessly.
- It focuses on the interactions and interfaces between these components.

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

Songs List

Q

Search by singer, genre, or song name



Bol Do Na Zara

Genre: Romantic


Singer: Armaan Malik

▶ 0:00 / 4:52

🔊

⋮

Add to Playlist



Chaleya

Genre: Romantic


Singer: Arijit Singh

▶ 0:00 / 3:08

🔊

⋮

Add to Playlist



Humnava Mere

Genre: Romantic


Singer: Jubin Nautiyal

▶ 0:00 / 5:29

🔊

⋮

Add to Playlist



Saari Duniya Jalaa Denge

Genre: Emotional

Singer: B Praak

▶ 0:00 / 3:02

🔊

⋮

Add to Playlist

Music-Player

Home

Your Library

Favorites

PlayList


Singer: Armaan Malik

▶ 0:00 / 4:52

🔊

⋮

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

Singer: B Praak

▶ 0:00 / 3:02

🔊

⋮

Add to Playlist

Playlist:

Music-Player











Home

Your Library

Favorites

PlayList

Playlist

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

Favorites:

Music-Player









Home

Your Library

Favorites

PlayList

Favorites

| # | Title | Genre | Actions |
|---|---|----------|---|
| 1 |  Chaleya Arijit Singh | Romantic |  <div>▶ 0:00 / 3:08</div> <div><div></div></div> <div></div> <div></div> |
| 2 |  Bol Do Na Zara Armaan Malik | Romantic |  <div>▶ 0:00 / 4:52</div> <div><div></div></div> <div></div> <div></div> |