

RYTHEMIC TUNES

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.

Say goodbye to the limitations of traditional music listening and welcome a world of possibilities with our 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 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 Website, "MUMU TUNES"

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.

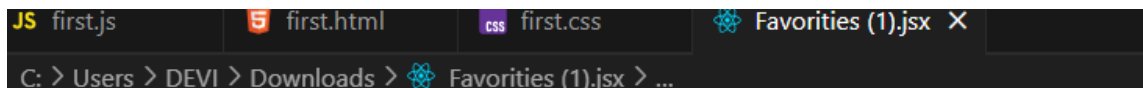
Key Features:-

- ? **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:-

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

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.

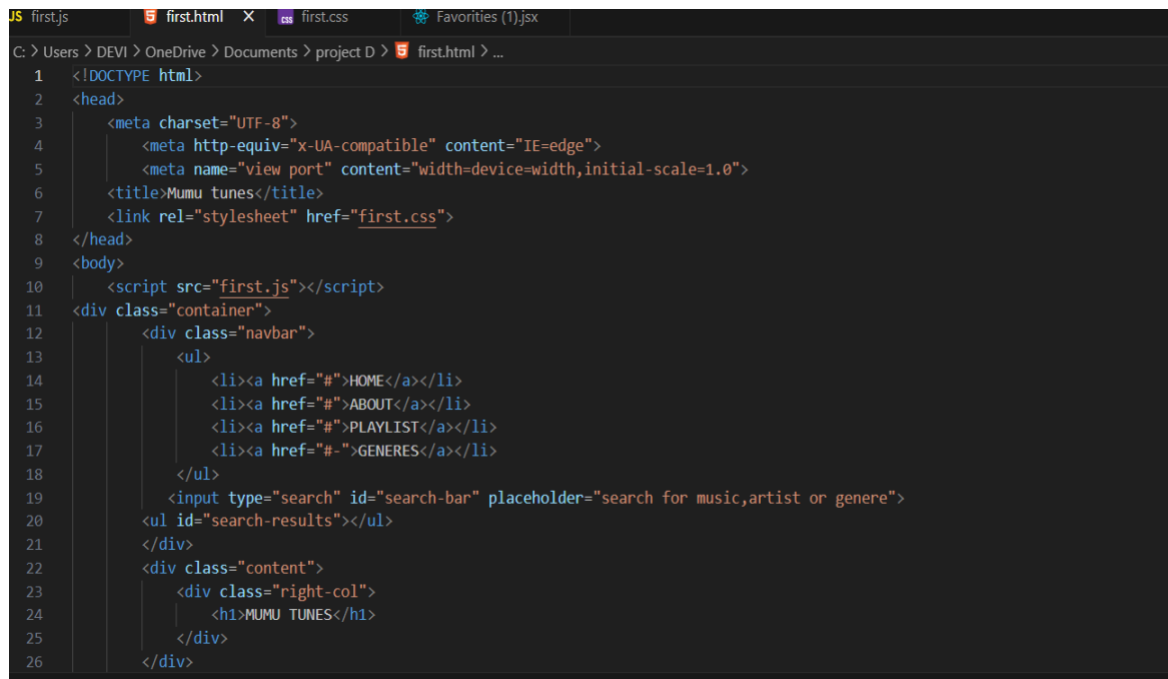
PROJECT FLOW:-

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Installation of required tools:

imports custom html (./first.html) for adding code



```
1 <!DOCTYPE html>
2 <head>
3     <meta charset="UTF-8">
4     <meta http-equiv="x-UA-compatible" content="IE=edge">
5     <meta name="view port" content="width=device=width,initial-scale=1.0">
6     <title>Mumu tunes</title>
7     <link rel="stylesheet" href="first.css">
8 </head>
9 <body>
10     <script src="first.js"></script>
11 <div class="container">
12     <div class="navbar">
13         <ul>
14             <li><a href="#">HOME</a></li>
15             <li><a href="#">ABOUT</a></li>
16             <li><a href="#">PLAYLIST</a></li>
17             <li><a href="#">GENERES</a></li>
18         </ul>
19         <input type="search" id="search-bar" placeholder="search for music,artist or genere">
20         <ul id="search-results"></ul>
21     </div>
22     <div class="content">
23         <div class="right-col">
24             <h1>MUMU TUNES</h1>
25         </div>
26     </div>
```

2. Code Description:-

Imports custom CSS (./first.css) for additional styling

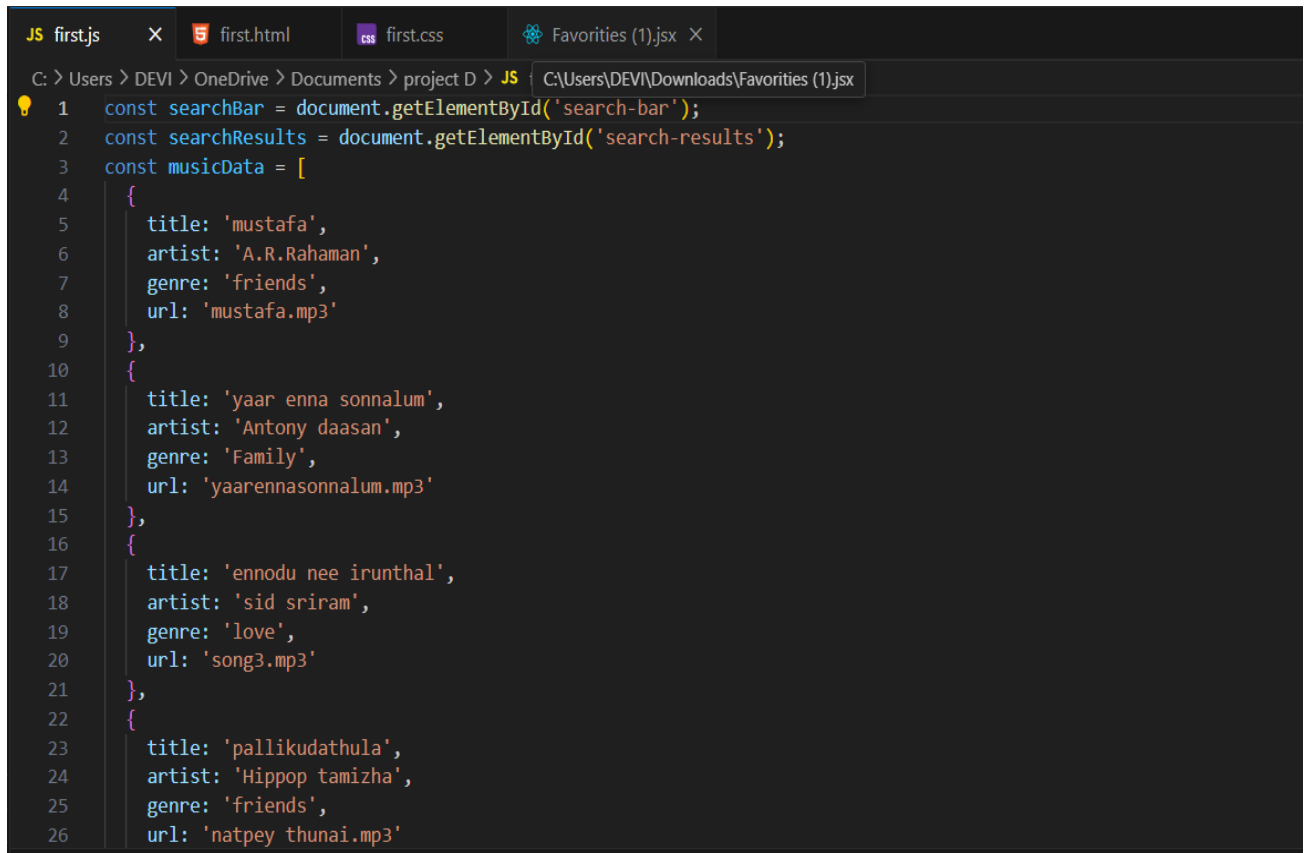
JS first.jsfirst.htmlfirst.cssX Favorites (1).jsx

C: > Users > DEVI > OneDrive > Documents > project D > first.css > .right-col h1

```
1  *{
2    margin: 0;
3    padding: 0;
4    font-family: sans-serif;
5  }
6  .container input{
7    height: 50px;
8    width: 70%;
9    margin-top: 10px;
10   margin-right: 5px;
11 }
12 .container{
13   height: 100%;
14   width:100%;
15   background-color: #f80a95;
16   background-size: cover;
17   background-position: center;
18   position:relative;
19 }
20 .navbar{
21   width:88%;
22   margin:auto;
23   margin-left: 75px;
24   padding:15px 0;
25   display:flex;
26   align-items: center;
```

Import java script file

(first.js)



```
1 const searchBar = document.getElementById('search-bar');
2 const searchResults = document.getElementById('search-results');
3 const musicData = [
4   {
5     title: 'mustafa',
6     artist: 'A.R.Rahaman',
7     genre: 'friends',
8     url: 'mustafa.mp3'
9   },
10  {
11    title: 'yaar enna sonnalum',
12    artist: 'Antony daasan',
13    genre: 'Family',
14    url: 'yaarennasonnalum.mp3'
15  },
16  {
17    title: 'ennodu nee irunthal',
18    artist: 'sid sriram',
19    genre: 'love',
20    url: 'song3.mp3'
21  },
22  {
23    title: 'pallikudathula',
24    artist: 'Hippop tamizha',
25    genre: 'friends',
26    url: 'natpey thunai.mp3'
```

- 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

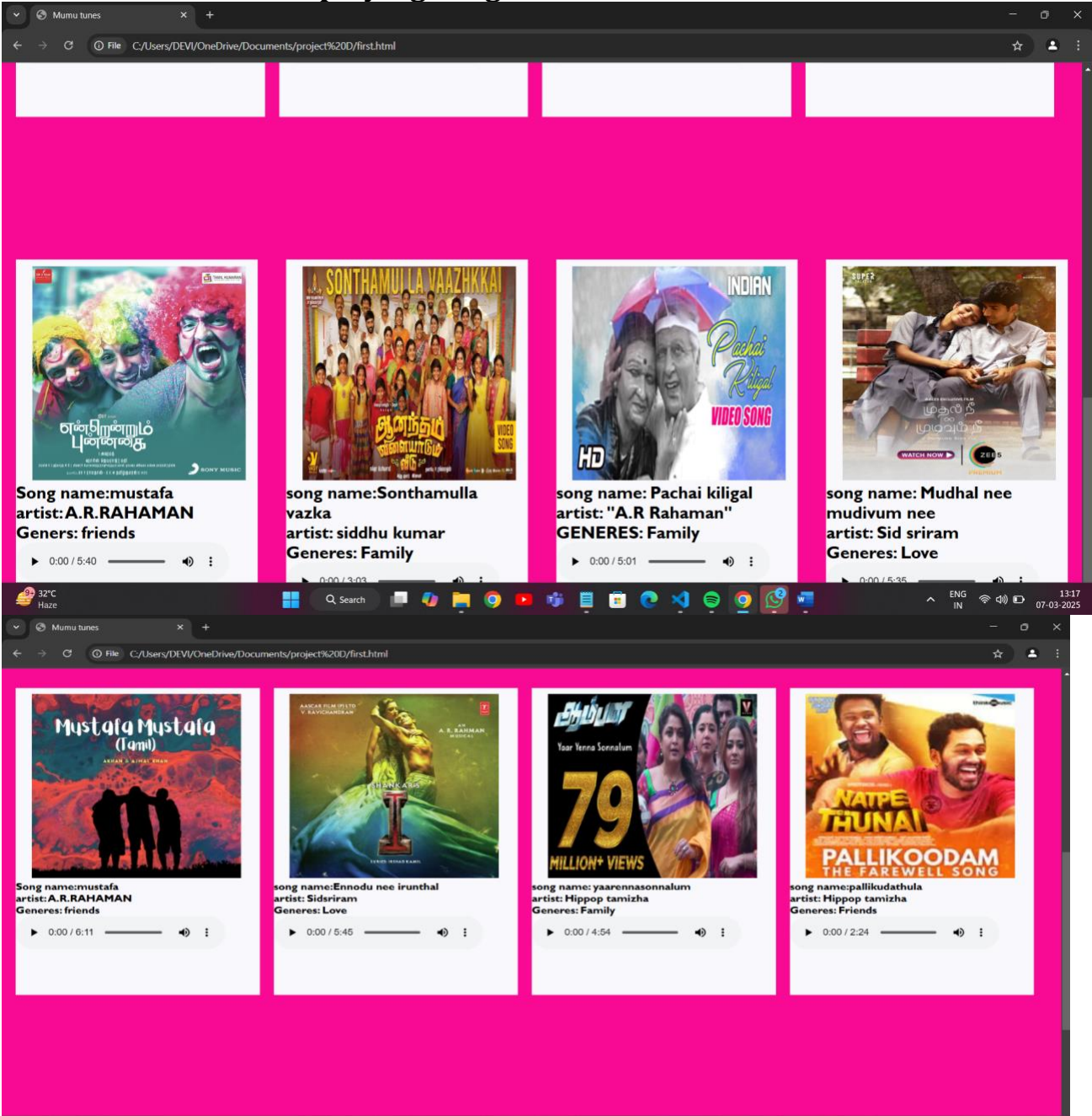
useState:

- `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`.
 - `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`.
 - Fetches favorite items (`wishlist`) from `http://localhost:3000/favorites`.
 - Fetches playlist items (`playlist`) from `http://localhost:3000/playlist`.
 - **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`.
 - **addToWishlist(itemId):**
 - Adds an item to the wishlist (`wishlist`) by making a POST request to `http://localhost:3000/favorites`.
 - Updates the `wishlist` state after adding an item.
 - **removeFromWishlist(itemId):**
 - Updates the `wishlist` state after removing an item.
 - **isItemInWishlist(itemId):**
 - Checks if an item exists in the wishlist (`wishlist`) 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`.
- **Error Handling:**
 - Catches and logs errors during data fetching (`axios.get`).
 - Handles errors when adding/removing items from `wishlist` and `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`.
- **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

Frontend Code For Displaying Songs



Frontend Code For Displaying Songs

