

RhythmicTunes: Your Melodic Companion

1. Introduction

Project Title : RhythmicTunes: Your Melodic Companion

TEAM ID: NM2025TMID45115

TEAM LEADER : NARMATHA S ROLE-(Project Coding and Demo Video)

EMAIL ID: narmathan362@gmail.com

NM ID: 57D21A9FBE8EE8881984E0764565C3B1

Team Members :

PRABA M-ROLE-(Project Coding and Demo Video)

EMAIL ID: prabamani000@gmail.com

NM ID: 5D531A6EEBE709E887D430224A2B406E

PREETHI A-ROLE-(Project Coding)

EMAIL ID: preethi232456@gmail.com

NM ID: F224BA0866A3E77A53C48181AC86361D

SARATHI S-ROLE-(Project Documentation)

EMAIL ID: sarathibebe@gmail.com

NM ID: EBD747AA9139E1C96E547E8E92CEDAB4

2. Project Overview

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 welcome a world of possibilities with our React-based 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 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 Objective

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.

3. Setup Instructions

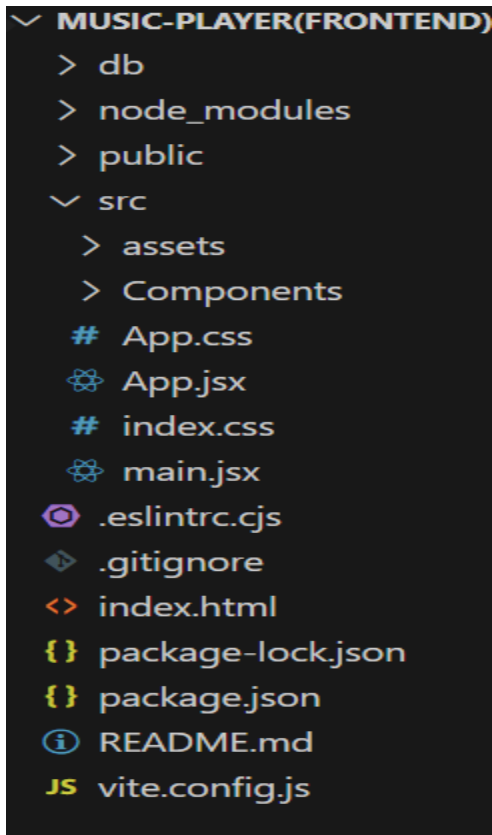
PRE-REQUISITES

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.

4. Folder 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.

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.

5. 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.

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
    };
  });
}
```

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`.
 - 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`.
 - 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.
- **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.imageUrl}
                  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)}
                      >

```

Code Description:-

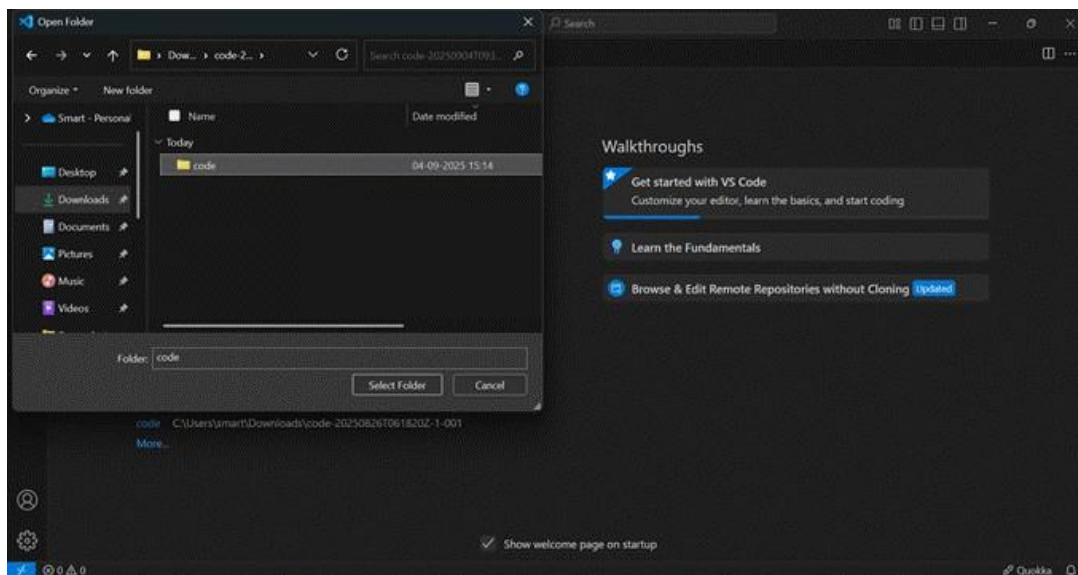
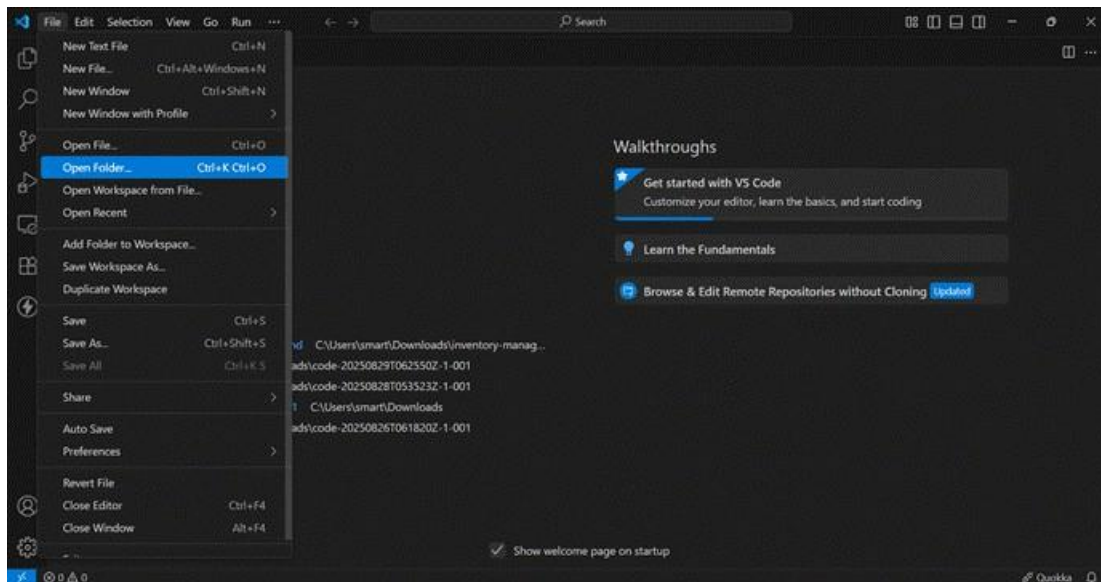
- **Container Setup:**
 - Uses a div with inline styles (style={{display:"flex", justifyContent:"flex-end"}}) to align the content to the right.
 - The main container (songs-container) has a fixed width (width:"1300px") and contains all the UI elements related to songs.
- **Header:**
 - Displays a heading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

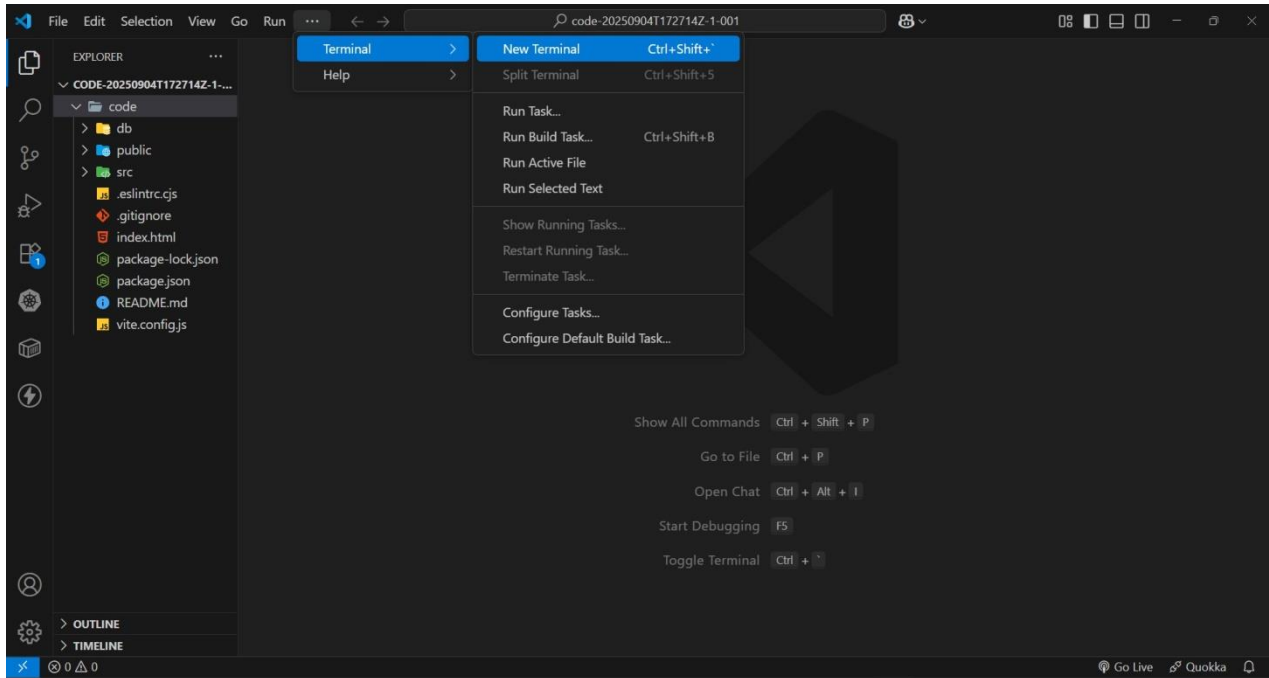
Uses custom styles (rounded-top, w-100) for specific elements like images and audio players.

6. Project Implementation & Execution

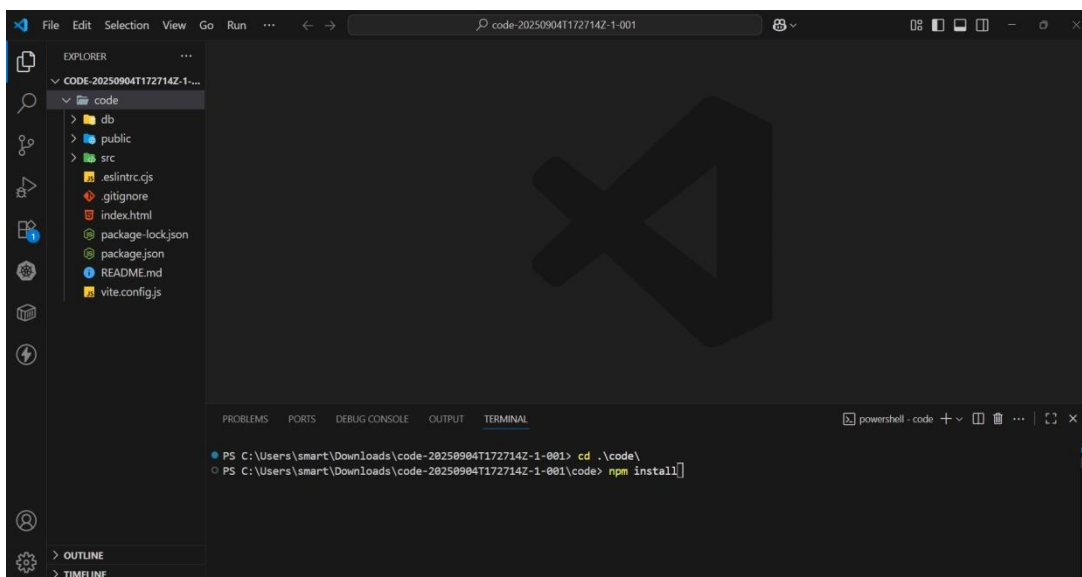
- Now open VS code and open the **code** folder from the **extracted folder**.



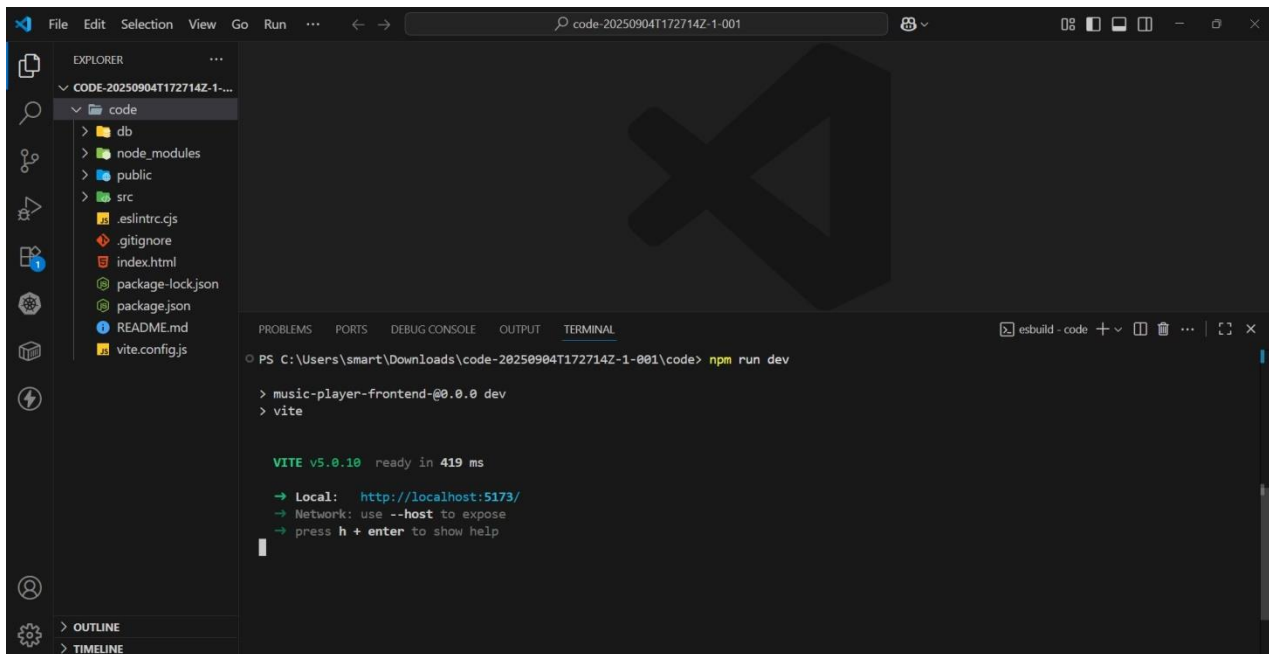
- Now open a new terminal.



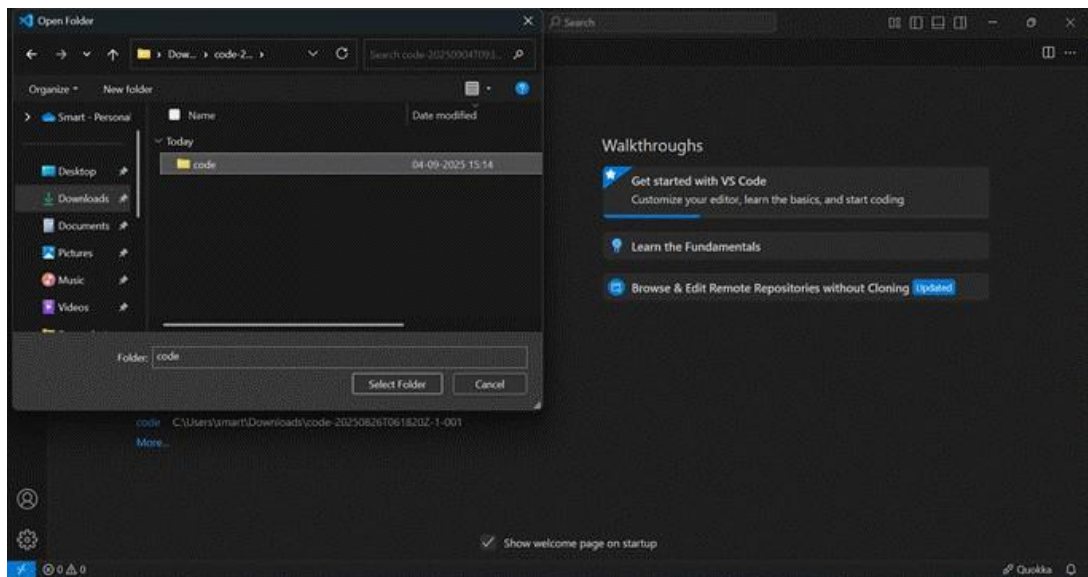
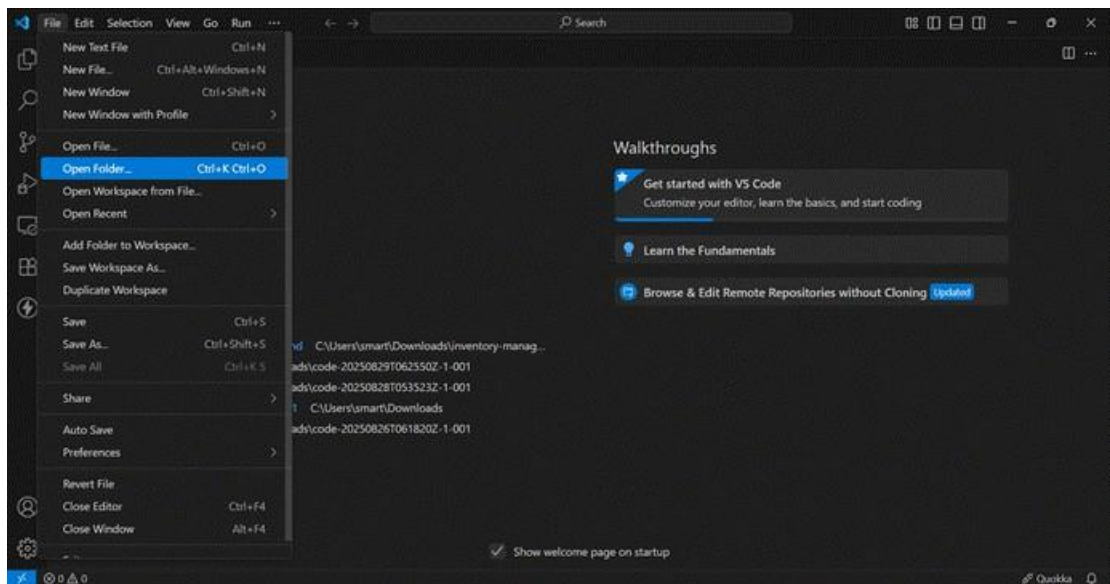
- Type **npm install** in your terminal and press **Enter** and wait till all the dependencies gets download



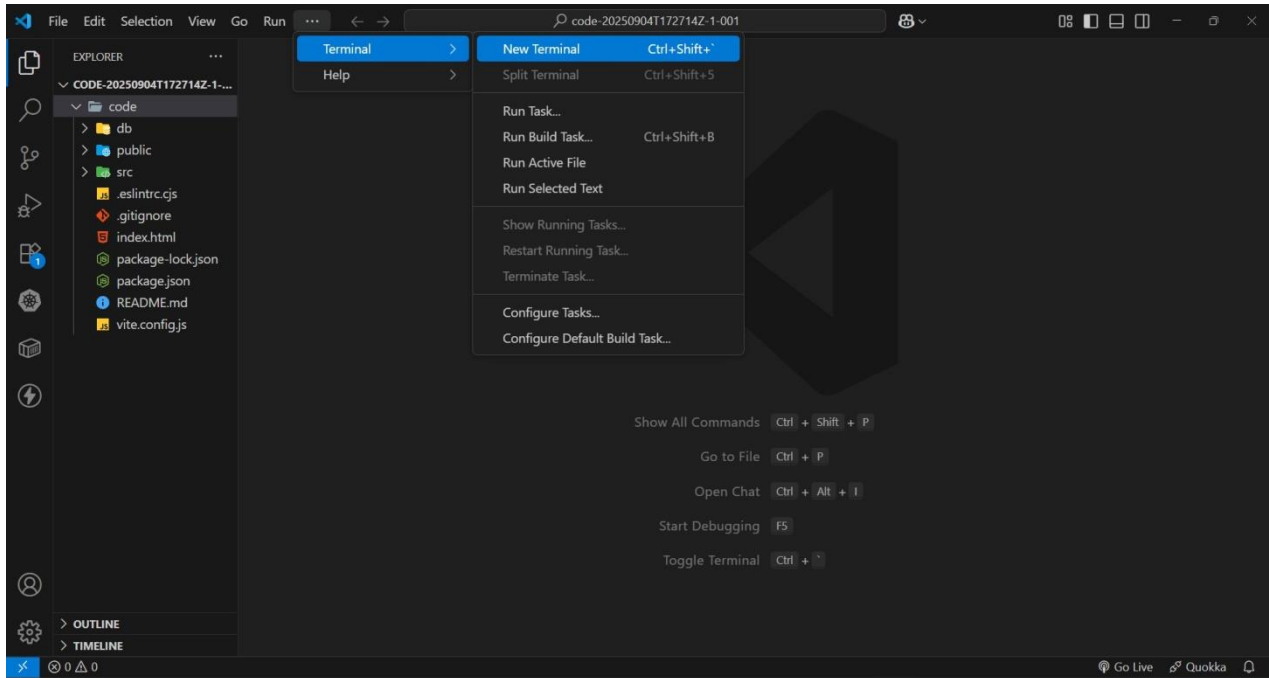
- After all the downloads finishes, now type **npm run dev** and press **Enter**



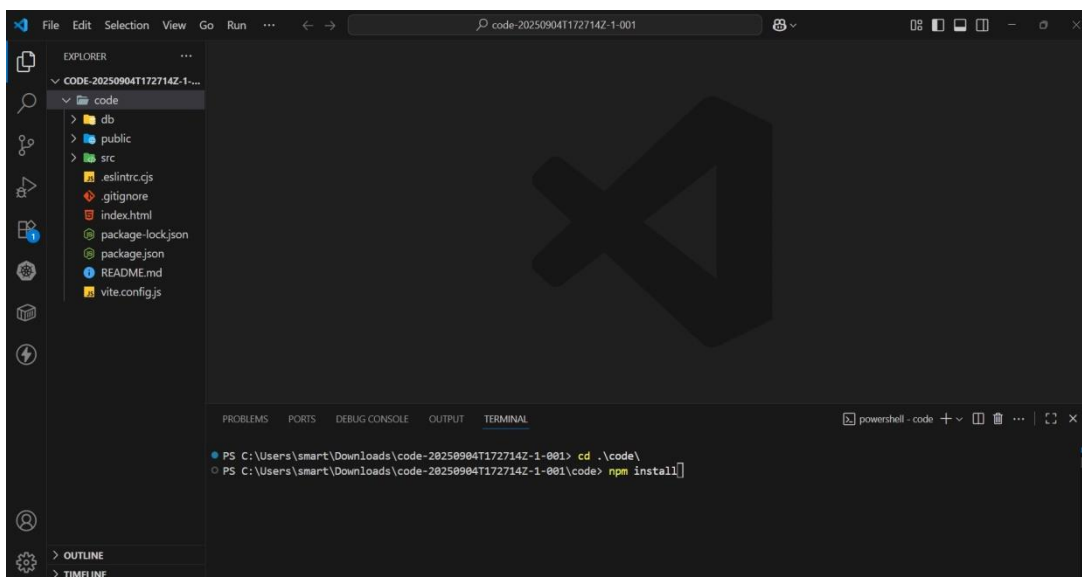
- Now your application will be opened in your browser with url <http://localhost:5173>
- Now open VS code and open the **code** folder from the **extracted folder**.



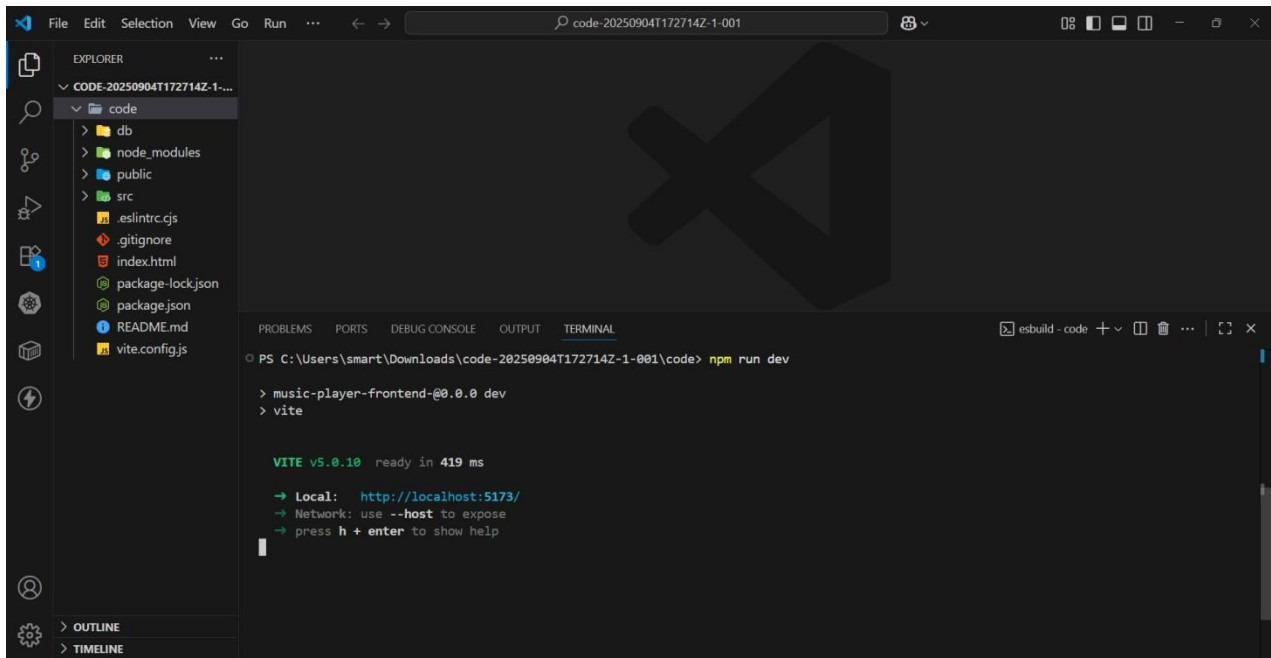
- Now open a new terminal.



- Type **npm install** in your terminal and press **Enter** and wait till all the dependencies gets download



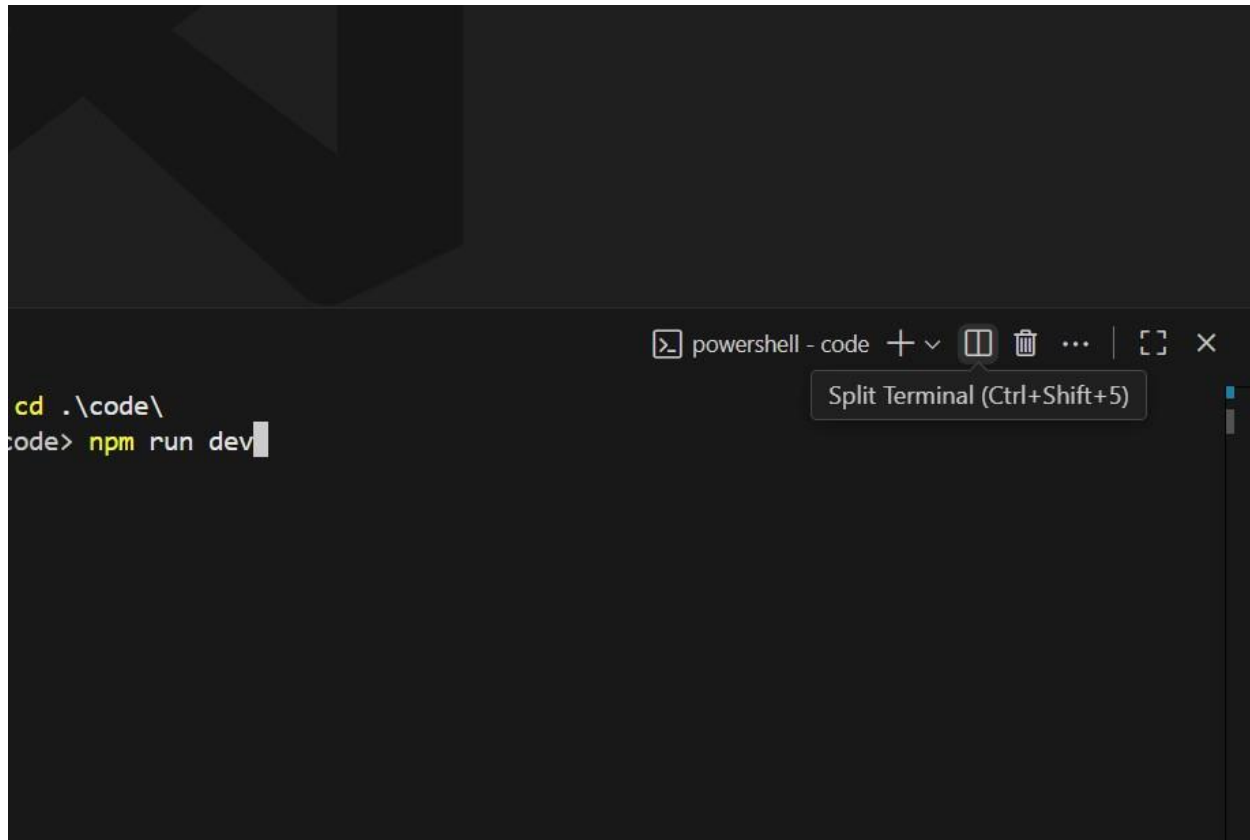
- After all the downloads finishes, now type **npm run dev** and press **Enter**



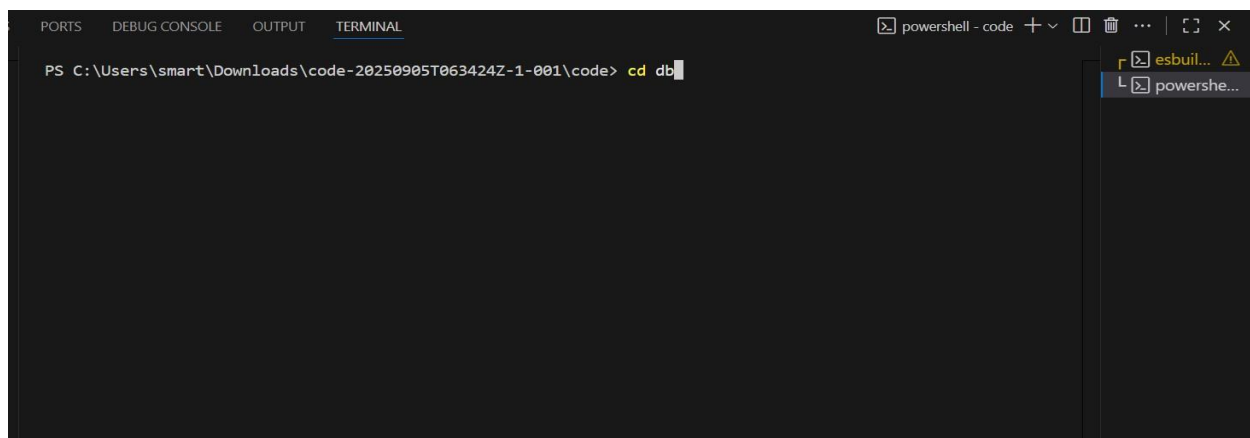
- Now your application will be opened in your browser with url <http://localhost:5173>

Create The JSON Server :

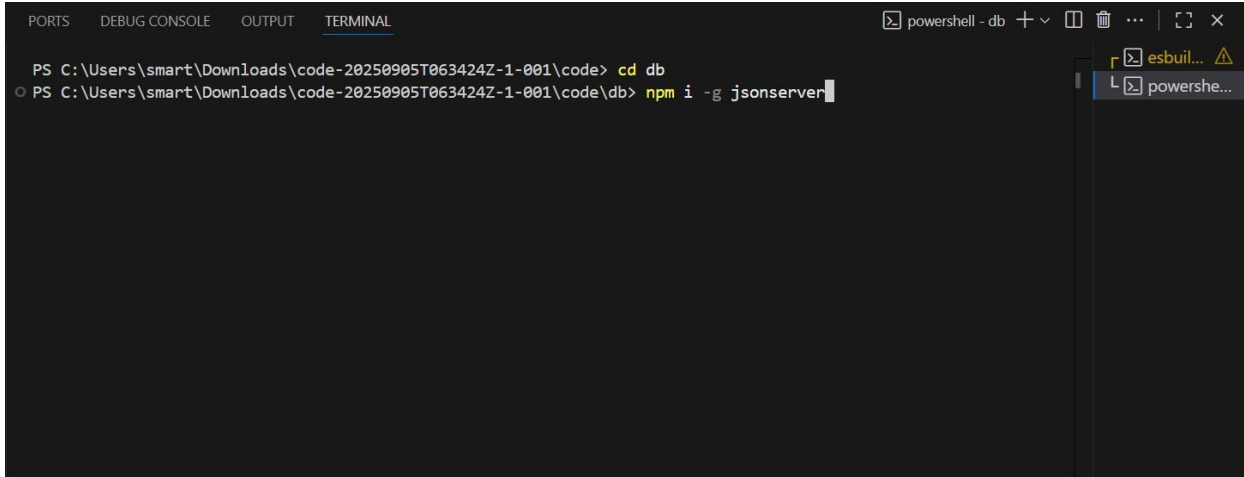
- Split the terminal



- Change the directory from code to db using the command **cd db**

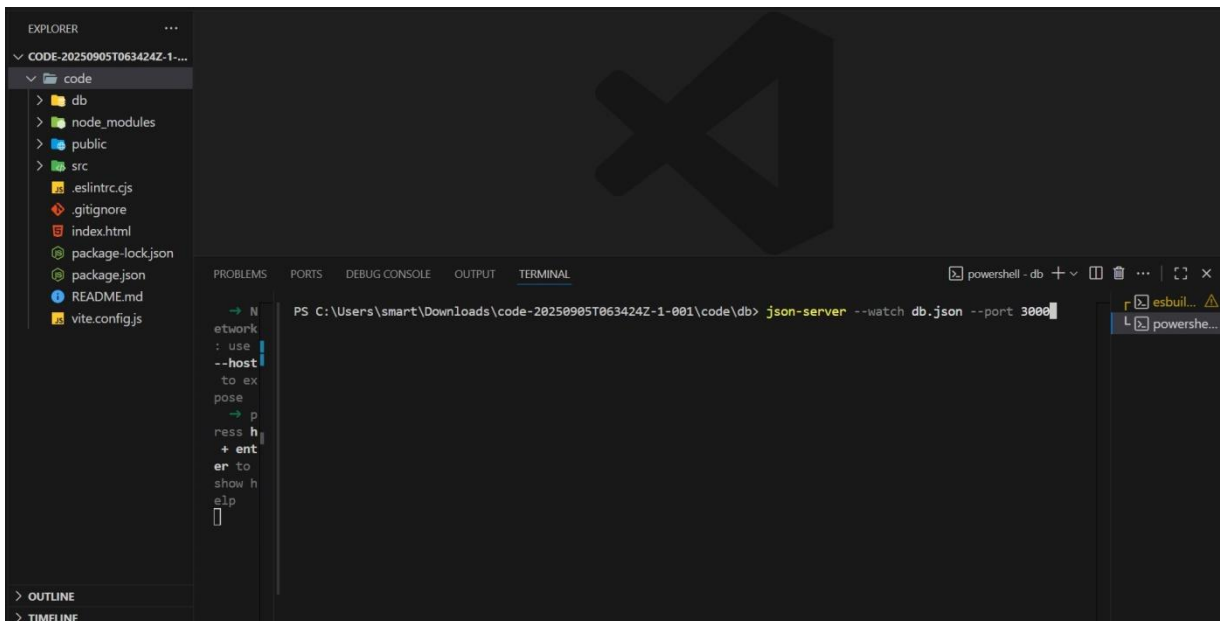


- Run the command **npm i -g jsonserver** , it will globally install the json server



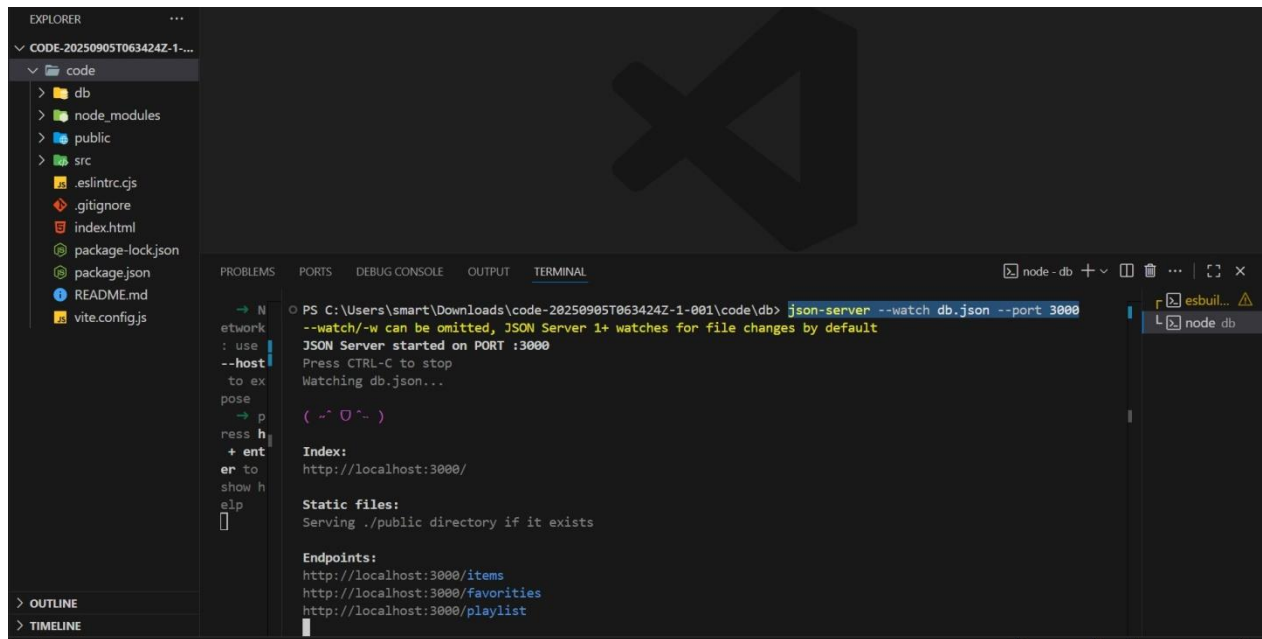
```
PS C:\Users\smart\Downloads\code-20250905T063424Z-1-001\code> cd db
PS C:\Users\smart\Downloads\code-20250905T063424Z-1-001\code\db> npm i -g jsonserver
```

- After install the server then run the following command **json-server --watch db.json --port 3000**



```
EXPLORER
CODE-20250905T063424Z-1-...
code
  db
  node_modules
  public
  src
  .eslintrc.cjs
  .gitignore
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js

TERMINAL
PS C:\Users\smart\Downloads\code-20250905T063424Z-1-001\code\db> json-server --watch db.json --port 3000
```



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a file tree for a project named 'CODE-20250905T063424Z-1-...'. The tree includes a 'code' directory with subfolders 'db', 'node_modules', 'public', and 'src', and files '.eslintrc.js', '.gitignore', 'index.html', 'package-lock.json', 'package.json', 'README.md', and 'vite.config.js'. The main editor area is split into two panes. The left pane shows a file named 'network' with some text, and the right pane shows a file named 'node db'. The bottom pane is a terminal window titled 'node - db'. It shows the command 'json-server --watch db.json --port 3000' being executed in a PowerShell prompt. The output indicates that the JSON Server has started on port 3000 and is watching for file changes in db.json. It also lists the index URL (http://localhost:3000/), static files (serving ./public directory), and endpoints (http://localhost:3000/items, http://localhost:3000/favorites, and http://localhost:3000/playlist).

```
PS C:\Users\smart\Downloads\code-20250905T063424Z-1-001\code\db> json-server --watch db.json --port 3000
--watch/-w can be omitted, JSON Server 1+ watches for file changes by default
JSON Server started on PORT :3000
Press CTRL-C to stop
Watching db.json...

( ^ _ ^ )

Index:
http://localhost:3000/

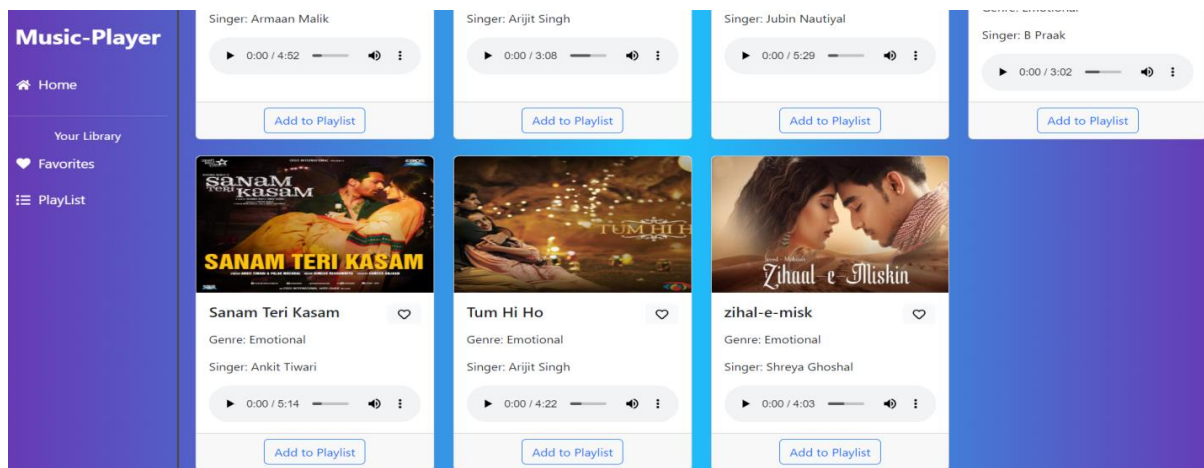
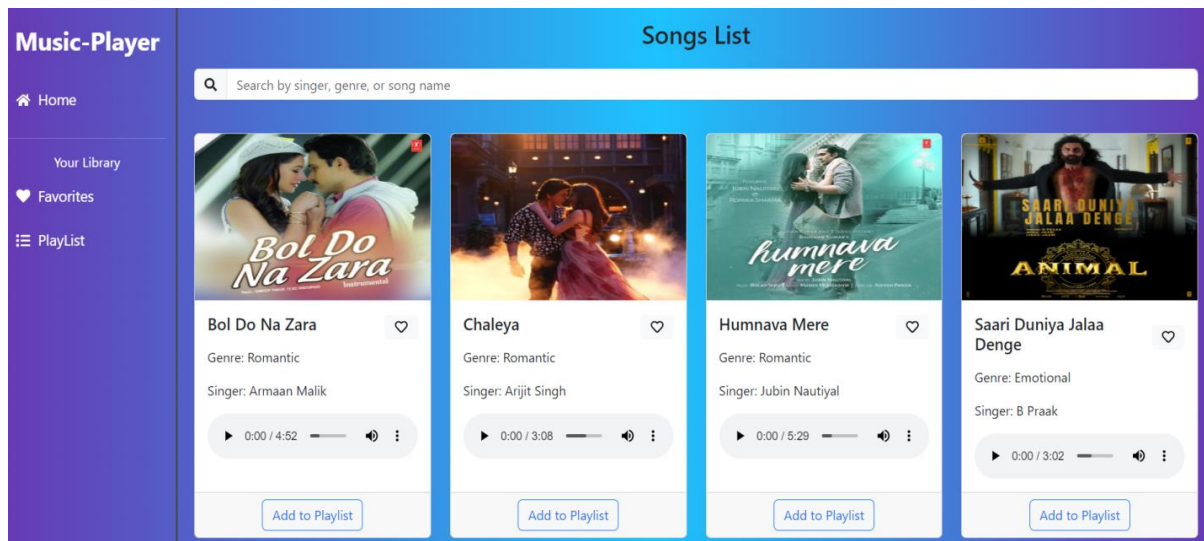
Static files:
Serving ./public directory if it exists

Endpoints:
http://localhost:3000/items
http://localhost:3000/favorites
http://localhost:3000/playlist
```

- After successfully running the command the json server will start and you can see the similar kind of output in the terminal which is there in the below image.

7.Screenshots or Demo

- Hero components



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>Remove</div>
2	 Humnava Mere Jubin Nautiyal	Romantic	<div>▶ 0:00 / 5:29 <div></div> 🔊 ⋮</div>	<div>Remove</div>
3	 Saari Duniya Jalaa Denge B Praak	Emotional	<div>▶ 0:00 / 3:02 <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>
2	 Bol Do Na Zara Armaan Malik	Romantic	♥	<div>▶ 0:00 / 4:52 <div></div> 🔊 ⋮</div>