Documentation Music-Player Frontend

by Elena Tret,

Cristian Stefanel-Petrea

Ichim Daniel-Nicolae

Table of contents:

1)Introduction

2)Core Components

* 2.1)DashboardPageComponent
* 2.2)PlaybarComponent
* 2.3)SearchBarComponent
* 2.4)SongListComponent
* 2.5)AlbumPageComponent

3)Services

* 3.1)MusicService
* 3.2)PlaybackService
* 3.3)UserService

4)Architecture and Design Patterns

5)Additional Features

6)Conclusion

1) Introduction:

This project is designed to provide users with an enjoyable and intuitive way to discover, listen to, and manage their favorite music.

At its core, this application serves as a digital music library and player. Users can search for songs, albums, and artists, accessing a wide range of musical content. The application includes a variety of user-friendly components, such as a search bar for finding new music, a song list to browse through available tracks, and a playbar for controlling music playback.

In addition to these basic features, our application also includes an album-focused view, where users can dive deep into specific albums, exploring all the tracks and details related to each one.

2)Core Components

2.1)DashboardPageComponent

* **HTML & SCSS**: The DashboardPageComponent serves as the main landing page, integrating the ‘SearchBarComponent’, ‘PlaybarComponent’, and ‘SongListComponent’. It provides a cohesive user experience by displaying search results, playback controls, and song lists in a unified layout.
* **TypeScript**: This component primarily aggregates and organizes other components, facilitating easy navigation and interaction without containing specific business logic.

2.2)PlaybarComponent

* **HTML & SCSS**: The PlaybarComponent includes controls for music playback, such as play, pause, skip, and volume adjustments. It also features a progress bar to indicate the current position in the track.
* **TypeScript**: The component handles playback control logic, interfacing with the ‘PlaybackService’ to manage audio states, including play, pause, and volume settings.

2.3)SearchBarComponent

* **HTML & CSS**: The SearchBarComponent allows users to search for music by typing in the search field. It provides real-time search suggestions as users type.
* **TypeScript**: The component uses Angular's reactive forms and event handling to process user input and communicate with the ‘MusicService’ to fetch and display search results.

2.4)SongListComponent

* **HTML & CSS**: The ‘SongListComponent’ displays a list of songs based on the user's search or selected playlist, including details such as song title, artist, album, and duration.
* **TypeScript**: This component manages the display and interaction with the list of songs, allowing users to select and play tracks. It interfaces with the ‘PlaybackService’ to update the current track being played.

2.5)AlbumPageComponent

* **HTML & CSS**: The ‘AlbumPageComponent’ is designed to display a list of songs from a specific album. It presents detailed information about the album, such as the album cover, release date, and tracklist.
* **TypeScript**: This component retrieves album details and song lists from the ‘MusicService’. It provides functionalities for users to play songs from the album, view lyrics, and add songs to playlists. The ‘AlbumPageComponent’ enhances the user's experience by providing contextual information and easy access to album tracks.

3)Services

3.1)MusicService

* This service is responsible for making HTTP requests to external music APIs to fetch data about songs, albums, and artists. It abstracts the data-fetching logic, providing a clean interface for the components to interact with.

3.2)PlaybackService

* The ‘PlaybackService’ manages the audio playback functionality. it includes methods for playing, pausing, skipping tracks, and adjusting volume. It also keeps track for the playback state, which is shared across the application to maintain synchronization.

3.3)UserService

* The ‘UserService’ handles user data and preferences, such as favorite songs, playlists, and recently played tracks. This service communicates with a backend system to persist user settings and preferences, enabling a personalized user experience.

4)Architecture and Design Patterns

* The application is designed with a modular architecture, ensuring each component and service is well-encapsulated and reusable. This modularity facilitates easy maintenance and scalability, allowing developers to add new features or update existing ones with minimal disruption.
* **Responsive Design**: The application's UI is designed to be responsive, ensuring a seamless experience across various devices, including desktops, tablets, and smartphones. The layout uses SCSS Flexbox and Grid to adapt to different screen sizes and orientations.

5)Additional Features

* **Real-time Lyrics Display**: The application can display synchronized lyrics for the currently playing song, enhancing the listening experience. This feature uses third-party APIs to fetch and display lyrics in real-time.
* **Offline Mode**: Users can download songs and playlists for offline listening. The application leverages browser storage APIs to cache the downloaded content, ensuring users can enjoy music even without an internet connection.

6)Conclusion

* In conclusion, this project has demonstrated the importance of creating an organized and intuitive user interface for showcasing albums and their associated songs. By implementing a well-structured Album-Page Component, we provide users with an engaging and accessible way to explore music collections. The design principles employed—such as clear categorization, consistent formatting, and interactive elements—enhance the overall user experience.