Music Playlist Manager Project Summary

Introduction:

- The **Music Playlist Manager** is an interactive web application designed to help users organize their favourite music. It allows users to add, view, edit, and delete songs from a playlist. The user interface includes a drag-and-drop song list for reordering, a modal form for adding and editing songs, and a dynamic playback progress bar to simulate real-time song play. The system supports smooth user experience through animations and responsive layouts.

Team Structure:

- This project was developed as a collaborative effort by the entire class working together as a single team. Previously, all projects were completed individually and hosted on GitHub. Shifting to a team-based approach introduced real-world challenges in project coordination, communication, and integration. This experience helped everyone gain insight into the importance of shared understanding, collaborative development, customer requirement analysis, and overall project workflow.

Modular Approach

- To ensure clarity of responsibilities and enable efficient parallel development, the project was divided into key functional modules. Each module was led by a **Module Lead** and supported by two **Engineers**, ensuring accountability, specialization, and streamlined integration across the system.

Modules & Responsibilities:

Module No	Module Name	Module Responsibility
01	Search module	To implement a responsive and dynamic search bar that allows users to search for songs by **title** or **genre**. The module must support **live suggestions** as the user types, with a **keyboard-navigable dropdown list** showing relevant results
02	Add Songs module	To design and develop a responsive **modal form** that allows users to **add new songs** to the playlist. The form should include fields for **title**, **artist**, **genre**, and **duration**, with both **client-side and server-side validation**. On successful submission, the song data must be stored in the **MySQL database**, and appropriate **toast messages** should be shown for success or failure
03	Edit/Delete module	To create and edit form with client side and server-side validation and delete button that deletes the movie and messages has to be in toast message

04	Song Details	To display detailed information about a selected song, including **title**, **artist**, **genre**, and **duration**, in a clean and accessible **single-column layout**. The module should feature a **client-side animated playback progress bar** that fills smoothly from 0% to 100% over the song's duration
05	Filter module	A module where the movies will be listed based on genre and genre should be listed in navigable clickable list
06	Drag and drop	To implement a **drag-and-drop interface** that enables users to **reorder songs** within the playlist dynamically. The module must ensure that the new order is **reflected immediately in the UI** and **persisted in the database** by updating each song's position field. On page reload, the songs should load in the saved order from the database.
07	Song Lists module	To fetch and display all songs from the **MySQL database** on the homepage in the form of **responsive, clickable song cards**. The module must support **drag-and-drop reordering**, **real-time search filtering**, and **genre-based filtering**. It should handle **loading states**, **API errors**, and display appropriate messages such as "No songs found."

Features Implemented

Created a responsive Add Song Form with:

Client-side and server-side validation.

Modal implementation using animations (Framer Motion).

Real-time validation feedback for required fields.

Toast messages for success and error feedback.

Data storage in a MySQL database.

Displayed all songs on the Song List Page:

Responsive single-column layout across devices.

Songs displayed as animated list items.

Integrated drag-and-drop reordering using react-beautiful-dnd.

Genre filter and real-time search with debounced input.

Developed Edit/Delete form:

Modal-based edit form with live validation.

Song update with persistent changes in the database.

Delete with confirmation and toast feedback.

Added a responsive Search Bar:

Live suggestions based on any part of the movie title.

Keyboard-navigable dropdown list.

Implemented Genre-based navigation module:

Genre-based filtering using a dropdown menu.

Enabled full drag-and-drop reordering:

Songs can be reordered with smooth transitions.

Updated order is stored in the database.

Order persists after page reloads

Created a Song Details Page:

Single-column responsive layout.

Simulated playback progress bar animated from 0% to 100%.

Smooth content fade-in on load.

ARIA labels for playback progress and accessibility

Issue Identified During Testing:

- **Tailwind CSS styles were not working:**

  Fixed by properly setting content paths in the Tailwind config file

- **Toast notifications were not showing or caused backend crashes**

  Fixed by separating toast logic from backend code.

- **Modal didn't open or close properly.**

  Fixed by using correct state toggles and class handlers.

- **Form inputs were not updating values**

  Fixed by connecting inputs to name, value, and on Change.

- **Submit button didn't work**

  Fixed with a proper onSubmit function and event. prevent Default ().

- **Validation messages didn't appear**

  Fixed by showing messages based on input state.

- **Modal background was still clickable**

  Fixed by adding a Fullscreen backdrop and disabling scroll.

- **Title accepted symbols; fields allowed blank values**
  Fixed with validation for clean input.
- **Duration accepted negative numbers**
  Fixed by allowing only positive values.
- **API error messages didn’t show**
  Fixed with toast notifications for API responses.
- **Update and cancel buttons sometimes failed**
  Fixed by improving button event handling.
- **Details page layout and buttons weren’t responsive**
  Fixed with single-column layout and stacked mobile buttons.
- **Playback progress animation wasn’t smooth or accessible**
  Fixed with smooth transitions, ARIA labels, and keyboard-friendly buttons.
- **Loading spinner or 404 message was missing on details page**
  Fixed by adding both loader and error message.
- **Song order didn’t persist after drag-and-drop**
  Fixed by storing song positions in the database using a position field.
- **Navigation buttons didn’t work as expected**
  Fixed to navigate correctly to edit, delete, and back to list views.

Integration Challenges & Fixes

Duplicate function names caused errors

Fixed by renaming and organizing code properly.

- **Modules didn’t work well together**
  Module leads collaborated and resolved overlaps.
- **Port conflicts in development environments**
  Fixed by assigning and using port 5000 consistently.
- **Base URLs and paths caused API routing issues**
  Fixed by standardizing routes across frontend and backend.
- **Accidental deletes during editing**
  Fixed by adding a separate confirmation step before deleting.
- **Default HTML validation was confusing**
  Replaced with clean, custom JavaScript validation.
- **Missing user feedback after API calls**
  Fixed by adding toast messages for all actions.
- **No confirmation before deleting a song**
  Fixed by adding a delete confirmation dialog.

CONCLUSION:

The **Music Playlist Manager** project was successfully completed with full functionality, including adding, editing, deleting, searching, filtering, viewing, and reordering songs within a playlist. The application features a modern, responsive UI with smooth animations, real-time feedback, and accessible design. Key enhancements such as drag-and-drop ordering with persistent storage, live search suggestions, and an animated playback progress bar contributed to a highly interactive user experience.

During development and testing, several issues were identified, including layout inconsistencies, validation gaps, broken interactions, and integration conflicts. These were resolved through collaborative debugging, proper use of state management, modular refactoring, and the implementation of standardized feedback mechanisms using toast notifications.

Integration across modules was improved through consistent design, routing, and port configuration. The addition of a position column in the database ensured consistent drag-and-drop behaviour, while accessible and responsive design patterns ensured usability across devices and user needs.

Overall, the project provided valuable hands-on experience in team-based development using modern web technologies such as **React**, **Node.js**, **MySQL**, **Tailwind CSS**, and libraries like **react-beautiful-dnd** and **framer-motion**. It served as a strong learning platform for understanding frontend-backend integration, user interface best practices, and real-world problem-solving in software development.

The final product is **stable, user-friendly, and meets all intended requirements**, offering a polished solution for managing a personal music playlist.

Following is the Folder structure, Stylings and Port Number:

standardised for the entire team:

Folder Structures:

Music playlist Manager

├── client/frontend

│ └── src

│ ├── pages

│ │ ├── HomePage.js

│ │ └── SongDetails.js

│ ├── components

│ │ ├── AddSongForm.js

│ │ ├── Search.js

| | ├── SongList.js

| | ├── Filter.js

| | ├── SongList.js

| | ├── EditDeleteForm.js

|

| | └── Reordering.js

| └── CSS

| ├── ... .CSS

| └── ... .CSS

├── server/backend

| └── server.js

| └──. env

| └── uploads

Stylings And Port Number:

Font-size: Heading 32 px, Paragraph 18 px

Font-Style: Seoge UI, sans-serif

DB Name - MUSICDB

Bg-color and colour: bg-purple-5000

PORT NO: 5000

The final integration was far easier with the above standardisation.

Test case was built and executed, bugs were found, reported and fixed and update to the final version was created.

The test cases built were rendered on Bugzilla, after bugs were found they were logged into Bugzilla.