Front-End UI/UX Mini Project Report

MelodyMuse - An Interactive Music Playlist Organizer

- **Submitted By:**
  - **Team Members:**
    - Aaron V Shibu (2462002)
    - Alan (2462021)
    - Albert B Varghese (2462024)
  - **College-E-mail ID:**
    - aaron.v@btech.christuniversity.in
    - alan.j@btech.christuniversity.in
    - albert.b@btech.christuniversity.in

- **Course:** Front-End UI/UX Development
- **Instructor Name:** Mrs. Nagaveena
- **Institution:** Christ University
- **Date of Submission:** 25/09/2025

2. Abstract

This project presents "MelodyMuse," an interactive single-page web application designed for efficient music playlist management. The key goal was to develop a visually engaging and user-friendly platform where users can browse, organize, create, edit, and delete their music playlists. The application is built using core front-end technologies including HTML5, CSS3, and JavaScript, and is enhanced with Bootstrap for responsiveness and jQuery for dynamic DOM manipulation. For data persistence, the browser's Local Storage API is utilized as a client-side backend. The final outcome is a fully functional, dark-themed music organizer with dynamic, genre-based color theming and a seamless user experience.

3. Objectives

The primary objectives set for the MelodyMuse project were:

- To design and develop a user-friendly, dark-themed interface with modern UI/UX principles.
- To build a fully responsive layout that functions seamlessly on desktop, tablet, and mobile devices using the Bootstrap framework.
- To implement dynamic content rendering for playlists and songs using jQuery to manipulate the DOM.
- To enable full CRUD (Create, Read, Update, Delete) functionality for both playlists and individual songs.
- To implement a persistent client-side backend using the browser's Local Storage API, allowing user data to be saved across sessions.
- To integrate key features such as real-time searching, multi-criteria sorting, and filtering by genre and artist.
- To create a visually dynamic UI where playlist art is color-coded based on its primary music genre.

4. Scope of the Project

The project is a front-end application focused on user interface and client-side functionality. The scope is defined by the following boundaries:

- **Client-Side Focus:** The application operates entirely within the user's browser. There is no server-side integration or communication with an external database.
- **Local Backend:** The "backend" is powered by the browser's Local Storage, meaning data is persistent on a per-device, per-browser basis.

- **Device Compatibility:** The application is designed to be fully responsive and functional across modern desktop, tablet, and mobile viewports.
- **Technology Stack:** The project utilizes standard open-source tools and libraries (Bootstrap, jQuery, Font Awesome) and does not rely on any proprietary software.

5. Tools & Technologies Used

Tool/Technology	Purpose
HTML5	Markup and semantic structure of the application content.
CSS3	Custom styling, animations, and the dynamic dark theme.
JavaScript (ES6)	Core application logic, interactivity, and state management.
jQuery	Simplified DOM manipulation, event handling, and AJAX-like operations.
Bootstrap 5	Responsive grid system, modals, and base component styling.
Local Storage API	Client-side "database" for data persistence across sessions.
Font Awesome	Scalable vector icons used throughout the user interface.
Google Fonts	Provided the "Inter" typeface for clean and modern typography.
VS Code	Code editor for development.
Chrome DevTools	Indispensable tool for testing, debugging, and performance profiling.

6. HTML Structure Overview

The application is built as a Single-Page Application (SPA) within a single index.html file.

- **Semantic Tags:** The structure uses semantic HTML5 tags like <nav> and <main> for better accessibility and organization.

- **View-Based Layout:** The UI is divided into two main containers that act as "views":
    - #playlist-list-view: The main dashboard displaying all playlist cards.
    - #playlist-details-view: A detailed view showing the tracklist for a selected playlist. JavaScript is used to toggle the visibility of these views, creating a seamless single-page experience.
- **Modals:** Bootstrap modals are used for forms, such as creating new playlists, adding songs, and confirming deletions, preventing page reloads and keeping the user in context.

7. CSS Styling Strategy

All custom styling is managed in an external style.css file to separate presentation from structure.

- **CSS Variables:** The dark theme is built using CSS variables (:root), making it easy to manage the color palette and ensure consistency. This also allows for dynamic theming, such as changing hover colors based on genre.
- **Animations and Transitions:** Subtle CSS keyframe animations (@keyframes) are used for fade-in effects on page load. Smooth transitions are applied to hover effects on cards and buttons to create a fluid user experience.
- **Typography-First Design:** After encountering issues with external images, the design pivoted to a typography-focused aesthetic, using CSS gradients to create visually appealing "cover art" for playlists. This approach is both stylish and 100% reliable.
- **Responsive Design:** While Bootstrap provides the grid, media queries are used for fine-tuning and ensuring that the layout and typography are optimized for all screen sizes.

8. Key Features

Feature	Description
Dynamic Rendering	Playlists and tracklists are dynamically generated from a JavaScript array and rendered to the DOM, eliminating static content.
Full CRUD Operations	Users can create new playlists, add songs to them, remove individual songs, and delete entire playlists.

Persistent Storage	All user modifications are automatically saved to the browser's Local Storage, ensuring data is not lost when the tab is closed.
Search, Sort & Filter	A multi-functional toolbar allows users to perform a real-time text search, filter by genre or artist, and sort playlists.
Dynamic Genre Theming	Each playlist card and details page features a unique CSS gradient based on its primary music genre, creating a vibrant and organized UI.
Responsive Design	The interface seamlessly adapts from large desktop monitors to small mobile screens using Bootstrap's responsive grid system.

9. Challenges Faced & Solutions

Challenge	Solution
Unreliable External Image Links	Initially, the app used external URLs for playlist covers, which were prone to breaking. The solution was to pivot to a more robust, typography-centric design using CSS gradients generated dynamically based on genre. This is 100% reliable and visually appealing.
Keeping UI in Sync with Data	Managing the state of the playlists array and ensuring the UI always reflected the latest changes was complex. This was solved by creating dedicated rendering functions (renderPlaylists, renderPlaylistDetails) that are consistently called after any data modification (add, delete, etc.), ensuring the UI is always a perfect representation of the data.
Ensuring Data Persistence	To prevent users from losing their work on page reload, the Web Storage API was integrated. savePlaylists() and loadPlaylists() functions were created to serialize the playlists array to a JSON string and save/retrieve it from Local Storage, effectively creating a client-side backend.

10. Outcome

The project successfully achieved its goal of creating a clean, consistent, and visually engaging front-end application for music playlist management. All key features, including full CRUD operations and a persistent local backend, were implemented and

function as intended. The process provided deep, practical insights into building data-driven, single-page applications using core front-end technologies.

11. Future Enhancements

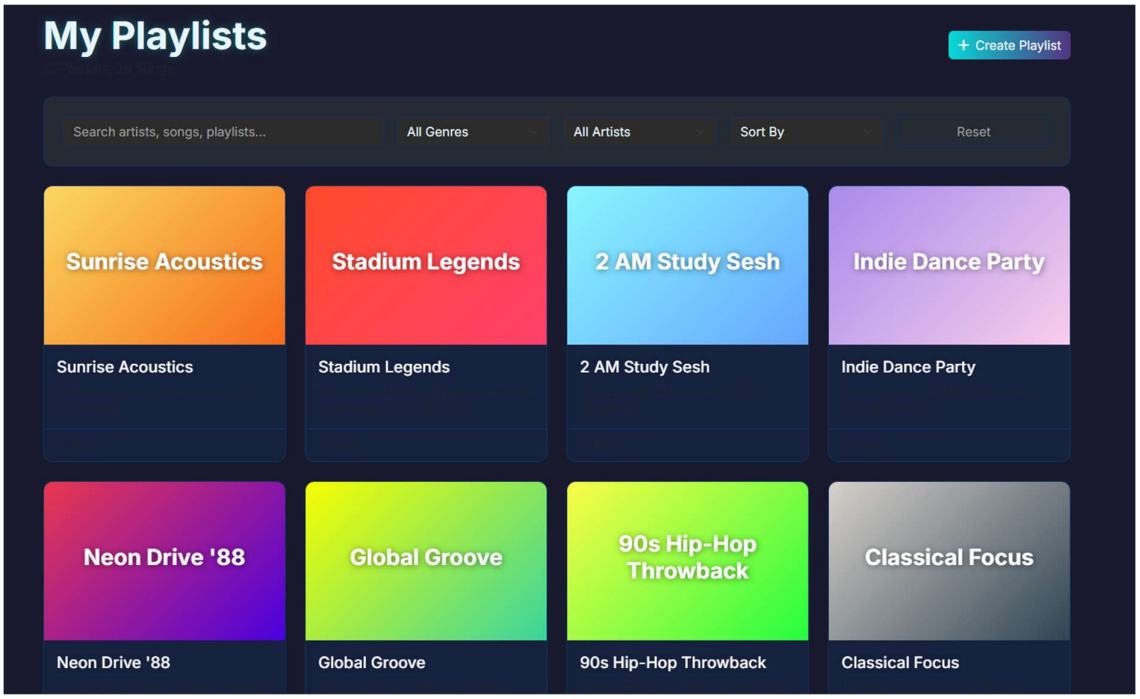
- **API Integration:** Integrate a third-party music API (e.g., Spotify API) to fetch real song data, album art, and artist information.
- **User Authentication:** Implement a user login system to allow different users to save their own private playlist collections.
- **Full Backend Integration:** Replace Local Storage with a server-side backend (e.g., Node.js, Express, MongoDB) to allow data to be stored persistently and accessed from any device.
- **Drag-and-Drop:** Add functionality to allow users to reorder songs within a playlist using a drag-and-drop interface.

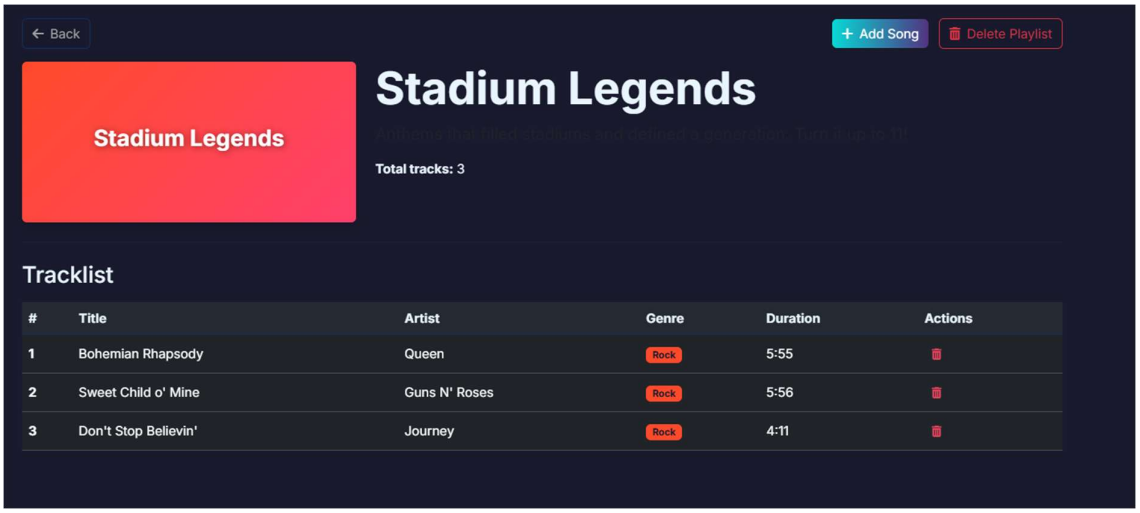
12. Sample Code

```
        <a class="navbar-brand" href="#"><i class="fas fa-compact-disc me-2"></i>MelodyMuse</a>
    </div>
</nav>

<div class="container mt-4">
    <!-- Playlist Listing View -->
    <div id="playlist-list-view">
        <div class="d-flex justify-content-between align-items-center mb-4 flex-wrap">
            <div class="me-3">
                <h1 id="main-title" class="display-5 fw-bold mb-0">My Playlists</h1>
                <p class="text-muted mb-0">
                    <span id="playlist-count">0</span> Playlists, <span id="song-count">0</span> Songs
                </p>
            </div>
            <button class="btn btn-gradient" data-bs-toggle="modal" data-bs-target="#createPlaylistModal"><i class="fas fa-
        </div>


        <!-- Filter and Search Section -->
        <div class="filter-section mb-4">
            <div class="row g-3">
                <div class="col-md-4">
                    <input type="text" id="search-input" class="form-control" placeholder="Search artists, songs, playlists
                </div>
                <div class="col-md-2">
                    <select id="genre-filter" class="form-select">
                        <option value="">All Genres</option>
                    </select>
                </div>
                <div class="col-md-2">
                    <select id="artist-filter" class="form-select">
                        <option value="">All Artists</option>
```

```
// --- RENDER FUNCTIONS ---
function renderPlaylists() {
    const grid = $('#playlist-grid');
    grid.empty();

    let processedPlaylists = [...playlists];

    // Sorting
    const sortBy = $('#sort-by').val();
    if (sortBy === 'title-asc') {
        processedPlaylists.sort((a, b) => a.title.localeCompare(b.title));
    } else if (sortBy === 'tracks-desc') {
        processedPlaylists.sort((a, b) => b.songs.length - a.songs.length);
    } else if (sortBy === 'tracks-asc') {
        processedPlaylists.sort((a, b) => a.songs.length - b.songs.length);
    }

    // Filtering
    const searchTerm = $('#search-input').val().toLowerCase();
    const genreFilter = $('#genre-filter').val();
    const artistFilter = $('#artist-filter').val();

    let filteredPlaylists = processedPlaylists.filter(p => {
        const titleMatch = p.title.toLowerCase().includes(searchTerm);
        const songMatch = p.songs.some(s =>
            s.title.toLowerCase().includes(searchTerm) ||
            s.artist.toLowerCase().includes(searchTerm)
        );
        const genreMatch = genreFilter ? p.songs.some(s => s.genre === genreFilter) : true;
        const artistMatch = artistFilter ? p.songs.some(s => s.artist === artistFilter) : true;
```

13. Screenshots of Final Output





14. Conclusion

This mini-project was a comprehensive exercise in modern front-end development. Building the MelodyMuse application from the ground up provided practical experience in structuring a single-page application, managing client-side data, and creating a dynamic, responsive user interface. The project reinforced my skills in HTML5, CSS3, and JavaScript (specifically with jQuery and the Local Storage API). I gained valuable insights into the importance of a robust design strategy, particularly when facing challenges like unreliable external assets. The hands-on implementation of these principles has significantly enhanced my understanding of building user-centric web applications.

15. References

- Bootstrap 5 Documentation: https://getbootstrap.com/docs/5.3/getting-started/introduction/
- jQuery API Documentation: https://api.jquery.com/
- MDN Web Docs - Local Storage API: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- Font Awesome Icons: https://fontawesome.com/
- Google Fonts: https://fonts.google.com/