Final Project Documentation

Darat’s Music Library

Professor: Ramsey Muvva

INF651 – Front End Web Development

Name: Darat Heng

Dec 10, 2025

Repository URL: https://github.com/liori-01/Heng_INF651

Table of Contents

1. Project Overview

This project is an interactive web application called **Darat's Music Library**, designed to showcase my understanding of core JavaScript concepts such as events, DOM manipulation, arrays/objects, functions, and conditional logic. The site functions as a personal music hub where users can:

- Browse a curated **3×3 grid of favorite albums** that link out to Spotify.
- Listen to my original tracks with a sortable track list.
- Submit **music recommendations** through a dynamic form that validates required fields and displays submissions in real time.
- See a **daily album recommendation pill** on the home page that updates once per day using JavaScript, arrays, and localStorage.

The application focuses on a cohesive user experience through consistent layout, a vertical navigation rail, full-screen background imagery, and JavaScript-driven interactions that make the page feel alive and personalized. Styling and layout are handled through a single main stylesheet, styles.css, which defines the layout, background images, cards, and responsive behavior.

2. Application Structure & Pages

The site is organized into four main pages:

1. **Home (index.html)** – A landing page with a hero section, navigation rail, and a "Today's Recommendation" area featuring a daily album pill and a small recommendation card.
2. **Favorites / 3×3 Albums (favorites.html)** – A dedicated page that showcases a 3×3 grid of favorite albums. Each album card displays album information and links out to Spotify. JavaScript dynamically fetches album art from the iTunes API.
3. **My Tracks (my-music.html)** – A page for my own original tracks, including a sortable list (by title and year) and embedded audio players for each track.
4. **Recommend (recommend.html)** – A page with a recommendation form where users can suggest songs or albums. JavaScript validates basic fields, stores recommendations in an array, and renders them dynamically on the page.

All pages share:

- A fixed vertical **side navigation bar** (.landing-side-nav) with links to each page and a "DARAT" logo.
- Full-screen background images and overlay sections that give each page a unique mood while keeping the layout consistent.

3. Main Features & Functionality

3.1 Home Page – Daily Recommendation & CTA

The home page introduces **Darat's Music Library** and immediately shows a **Daily Album Pill** and a "Today's Recommendation" panel.

Key features:

- **Daily Album Pill (#dailyAlbumMini):**
  - Displays one album from my favoriteAlbums array each day.
  - Uses localStorage and today's date as a key so the recommendation stays consistent for that day and rotates automatically the next day.
  - Shows album art (fetched via the iTunes API), title, artist, and year, and links to the album's Spotify page.
- **Call-to-Action Buttons:**
  - Buttons navigate to **Favorites** and **My Tracks**, encouraging users to explore the rest of the site.

This page demonstrates user interaction (click events), DOM updates, external API usage, and data handling using arrays and objects.

3.2 Favorites / Album 3×3 Page

The favorites page displays a **3×3 grid of my all-time favorite albums**. Each album is shown as a card with:

- Album art (loaded dynamically with JavaScript),
- Title, artist, and metadata (year + genre),
- A clickable area that opens the album on Spotify in a new tab.

Each card is represented in HTML with a .album-art <div> that uses data-album and data-artist attributes. JavaScript reads these attributes and uses the iTunes Search API to fetch album cover art and apply it as a background image.

This page highlights:

- **DOM manipulation** (selecting .album-art elements and updating their inline style),
- **Loops and functions** (loadArtworkForAlbumGrid() iterates over each album element),
- **Data attributes** as a clean way to connect HTML to JavaScript.

3.3 My Tracks Page – Original Music & Sorting

The **My Tracks** page is dedicated to my own music. The tracks are stored as an array of objects in myMusicTracks, each containing:

- title,
- description,
- mood,
- year,
- and a local audio file path (file).

JavaScript features on this page include:

- **Dynamic track rendering** with renderMyMusic(list):
  - Loops over the myMusicTracks array.

    - Builds a .track-card for each item with a title, description, mood/year metadata, and an <audio> player if a file is provided.
- **Sorting functionality** with applyMusicSort(criteria):
    - A <select> element allows users to sort by:
        - Title (A → Z),
        - Year (Newest → Oldest),
        - Year (Oldest → Newest).
    - On change, JavaScript sorts a copy of myMusicTracks and re-renders the list.

This page demonstrates:

- Arrays and objects for structured data,
- Event listeners on form controls,
- Conditional logic and array sorting,
- Reusable functions for rendering and updating the DOM.

3.4 Recommend Page – Interactive Form & Live List

The **Recommend** page contains a responsive form that allows users to submit song or album recommendations. The form includes fields for:

- Name (required),
- Song/album title (required),
- Artist (optional),
- Link (optional, URL type),
- Reason (optional).

JavaScript functionality:

- **Form submission handler** handleRecommendationSubmit(event):
    - Prevents the default page reload.
    - Uses FormData to read input values.
    - Validates that at least name and title are filled in.
    - Shows an error message if fields are missing.

    - Pushes a new recommendation object into the recommendations array and re-renders the recommendation list.
- **DOM rendering** via renderRecommendations():
    - If no recommendations exist, shows a “No recommendations yet” message.
    - Otherwise, it builds .recommendation-card elements showing:
        - Title + artist,
        - Recommender’s name,
        - Optional reason (in quotes),
        - Optional external link.

This page showcases:

- Event handling (submit),
- Real-time DOM updates,
- Basic form validation and error messaging,
- Data storage in arrays and dynamic list rendering.

4. JavaScript Concepts Demonstrated

All JavaScript logic is contained in script.js, which is shared across the site and initialized on DOMContentLoaded.

4.1 Events & User Interactions

- window.addEventListener("DOMContentLoaded", ...) to initialize each page only after the DOM is ready.
- click listeners on buttons (e.g., adding/removing favorites, navigation via buttons).
- change event on the sort dropdown for My Tracks.
- input event on a search bar (if used/added) to dynamically filter content.
- submit event on the recommendation form for validation and live updates.

4.2 DOM Manipulation

Throughout script.js, the app uses document.getElementById, document.querySelectorAll, innerHTML, and style changes to:

- Insert track and album cards into containers.
- Update artwork images after API responses.
- Display error messages and empty-state messages.
- Change attributes and classes for dynamic behavior (e.g., turning the daily album pill into a clickable link).

4.3 Logic, Loops, and Conditionals

Examples include:

- For-loops to iterate over arrays and NodeLists when rendering cards or loading album art.
- Conditional checks to:
  - Avoid errors on pages that don't have certain elements.
  - Only show audio players if a file path exists.
  - Show different content when there are no items (e.g., "No recommendations yet.").
- Sorting logic in applyMusicSort that uses conditionals to choose the correct sorting strategy.

4.4 Data Handling with Arrays & Objects

The app uses several arrays of objects:

- myMusicTracks for original tracks and audio files.
- favoriteAlbums for the 3×3 grid and daily album pill, including title, artist, year, genre, and Spotify URL.
- userFavorites (optional feature) to capture a user-built playlist based on core favorites.
- recommendations to store user-submitted music suggestions.

These arrays demonstrate:

- Data modeling in JavaScript,
- Passing arrays into rendering functions,
- Filtering and sorting over arrays.

4.5 Reusable Functions

The code is organized into reusable functions, for example:

- renderMyMusic(list) – render a list of tracks.
- applyMusicSort(criteria) – sort tracks based on a given option.
- loadArtworkForAlbumGrid() and fetchArtworkForAlbumElement(el) – fetch and apply album art.
- showDailyAlbumMini() – build the daily album pill from the favoriteAlbums array.
- handleRecommendationSubmit() and renderRecommendations() – manage recommendation form logic and display.

By reusing these functions, the code stays more modular and easier to maintain.

5. Design & User Experience

The user experience is built around a cohesive aesthetic and intuitive layout:

- **Vertical navigation rail** on the left that is consistent across all pages.
- **Full-screen background images** for different pages (e.g., bg1.jpg, bg3.jpg, bg4.jpg) to create distinct moods while keeping typography and layout consistent.
- **Cards and pills** are used for albums, tracks, and recommendations, making information scannable and visually appealing.
- Responsive design tweaks via media queries hide the side nav on smaller screens and adjust padding/layout so the site remains usable on different screen sizes.

JavaScript enhances UX by:

- Avoiding page reloads when submitting recommendations.
- Providing live feedback (error messages, empty-state text).
- Making the site feel personalized via the album-of-the-day feature.
- Giving users control over how they view data (sorting on My Tracks).

6. Sketches & Planning (Conceptual)

During planning, I designed simple wireframes for each page:

- **Home:**
  - **Left:** vertical nav rail.
  - **Center:** hero text.
  - **Bottom-left:** daily album pill + Today’s Recommendation card.
  - **Bottom-right:** CTA buttons.
- **Favorites:**
  - Title at the top.
  - 3×3 album grid in the center, each card clickable.
- **My Tracks:**
  - Title block at top.
  - Sort dropdown in the top-right of the content panel.
  - Vertical list of track cards with descriptions and audio players.
- **Recommend:**
  - Title block at top.
  - Recommendation form centered in a blurred-glass card.
  - List of submitted recommendation cards below.

In the written documentation, these can be described as wireframes or supplemented with screenshots of the implemented pages.

7. Challenges & How I Overcame Them

1. **Getting the layout and navigation consistent across pages**
   - **Challenge:** Making sure the side nav, background images, and overlays lined up correctly so every page felt like part of the same app.
   - **Solution:** I centralized most layout rules in styles.css and reused shared classes like .landing-side-nav, .music-overlay, .recommend-overlay, and .page-title-block, then added small page-specific overrides.
2. **Dynamic album art with external APIs**
   - **Challenge:** Loading album art from the iTunes API based on album and artist names, and ensuring the UI didn't break if the API returned no results.
   - **Solution:** I wrote helper functions like fetchArtworkForAlbumElement that safely check for results before applying background images. I also used data attributes in the HTML so the JS could stay generic.
3. **Keeping code working across multiple pages**
   - **Challenge:** script.js is loaded on every page, but not all pages have the same elements. That can lead to null errors when querying the DOM.
   - **Solution:** I added checks like if (document.getElementById("myMusicList")) { ... } before attaching event listeners or rendering content, so each block only runs on the relevant page.
4. **Form validation and user feedback**
   - **Challenge:** Ensuring the recommendation form gives clear messages when required fields are missing.
   - **Solution:** I used a dedicated error element (#formError) and simple conditional checks in handleRecommendationSubmit to show messages without reloading the page.

8. Potential Future Enhancements

If I had more time or backend support, there are several ways I would extend this project to feel even more like a real, living music platform:

1. **Persistent Data and User Accounts**
   I would add persistent storage for recommendations and user favorites using localStorage or a backend database so data remains across visits. On top of that, I would add authentication so users can create accounts, log in, and save their own playlists and favorite tracks directly in the app.

2. **Deeper Spotify Integration**
   A major next step would be to integrate the Spotify Web API. Instead of only linking out to albums, the site could:
   - Pull in my actual Spotify playlists and display them in a dedicated section
   - Generate recommendations based on my listening history and liked songs
   - Show live data such as recently played tracks, top artists, or mood based playlists

   This would turn the site into a more personalized extension of my Spotify profile instead of a static snapshot of my taste.

3. **Music Sharing Between Friends and Peers**
   I would like to turn the site into a small community space for music production and sharing. Possible features include:
   - A section where friends can upload demos and work in progress tracks (IDs)
   - Comment or feedback threads on each demo so peers can leave reactions and suggestions
   - Optional privacy settings so some demos are only visible to close friends or collaborators

   This would make the platform useful not only for sharing finished songs, but also for supporting each other creatively.

4. **Upcoming Releases and Teasers**
   Another feature I would add is an "Upcoming Releases" area where I can:
   - Announce upcoming tracks or EPs
   - Share short audio teasers or preview clips
   - Show release dates and link to pre save or release pages

   This would turn the site into a central place for people to follow what I am planning to release next.

5. **Advanced Browsing and Discovery Tools**
   I would improve the Favorites and My Tracks pages with more advanced filtering and discovery features, for example:
   - Filters by mood, genre, BPM, or year
   - Tag based browsing such as "study", "night drive", or "gym"
   - Sorting by popularity or play count

6. **Improved Mobile Experience**
   Finally, I would refine the mobile experience by:
   - Adding a collapsible navigation menu
   - Increasing tap target sizes and spacing for buttons and cards
   - Tuning layout and typography so the site feels as smooth on a phone as it does on a desktop

Together, these enhancements would turn Darat’s Music Library from a single user project demonstration into a more complete music hub that connects my Spotify activity, my own productions, and the music shared within my friend group.

9. Conclusion

Darat’s Music Library successfully meets the project’s objective of building an interactive, JavaScript-driven web application that goes beyond a static website. It combines:

- Multiple pages with distinct but cohesive layouts,
- Rich interaction via events and DOM manipulation,

- Structured data with arrays and objects,
- External API integration for album art,
- Clear UX patterns like sorting, form validation, and daily recommendations.

Overall, this project allowed me to practice not just technical JavaScript skills, but also planning, UI design, and thinking about how each interactive feature improves the user experience.