

The Dead Media Catalogue

Overview

I developed a single-page web application to catalogue a media collection using HTML, CSS, and JavaScript. The app provides users with the ability to view, add, remove, edit, sort, and search media entries, ensuring all data meets defined constraints.

All functional and technical requirements for the project were successfully met.

Design

UI/UX

The UI is built for maximum efficiency, focusing on the core user tasks: displaying, adding, and editing media entries. Drawing from Swiss design principles, it's clean and functional, allowing users to manage media seamlessly.

index.html

I have used semantic HTML using elements like `<header>`, `<form>`, and `<dialog>` to enhance both accessibility and SEO. Dialogs keep the add and edit media forms hidden when not needed. Each input has a `<label>`, and appropriate tags like `<input>`, `<select>`, and `<textarea>`. Basic client-side validation ensures required fields follow certain rules, but since it can be easily bypassed, additional JavaScript checks are included. Although those checks should be on server side, it's outside the scope of this assignment. Finally, JavaScript is loaded with the `defer` attribute to improve performance by downloading and running it after the HTML is fully loaded.

mediaManipulations.js

Provides basic key functions for manipulating a media list:

- Add, Delete, Update
- Validation
- Transformation
- Render

The `transformMediaList` and `validateMediaList` functions ensure the media data is ready to use from the start. Transformation is not just about reformatting; it adds proper ids directly linked to each media entry and aligns the data with JavaScript naming conventions, making it easier to work with. Validation ensures that only correct, well-formed data moves forward, preventing potential issues down the line.

To work with the media list, there are functions to add, update, and delete entries by their unique id.

To render the item, there is a function to generate HTML markup from entry data, designed to prevent XSS attacks by safely injecting data into the DOM. It also embeds the entry's id into attribute, making it easier to target and when performing actions on specific entries.

All those functions are written in a pure functional style, meaning no internal state is managed within these functions. This keeps the code predictable and easier to test and maintain, as each function only works with the data it's given.

app.js

Uses functions from mediaManipulations.js to manage media entries, handle user interactions, and render the list dynamically. It adds event listeners for buttons and forms, manages the state of search queries and sorting, and uses rerenderList() to update the displayed list.

In rerenderList(), sorting is applied, based on attributes like name or type, and then the list is filtered by the search query. Although it's more efficient to filter first and then sort, keeping it simple by sorting first has minimal performance impact, given JavaScript's performance. Ideally, the filtering and sorting should be handled server-side for better scalability and efficiency. However, for this small project, handling it in JavaScript keeps the implementation straightforward without significant performance issues. Finally, if no entries match the search, a "No records found" message is displayed.

For form submissions, event listeners are attached to the form elements themselves rather than submit buttons, which simplifies extracting data and form structure directly matches the media entry schema.

styles.css

Global reset - ensures a uniform layout across browsers.

styles.css

```
box-sizing: border-box;  
margin: 0;  
padding: 0;
```

Responsive Layout - used minmax() and grid to automatically resize

```
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
```

Modular - created styles that can be stacked together, for example: button + danger + wide


Given the project's size, the current level of coupling between styles and elements is optimal. While more advanced tools like React, TypeScript, or TailwindCSS could improve future scalability, the current setup is the best choice for maintaining simplicity and structure within the project's scope.

Testing

The tests were selected based on the core requirements of the application, including loading media, adding, editing, and deleting entries, as well as handling edge cases such as invalid input or unexpected user behavior.

Tests were specifically chosen to cover a wide range of scenarios, such as validating media data, checking for XSS (Cross-Site Scripting) vulnerabilities, and verifying that search and sort features work independently and in combination with add, edit, and delete operations. Dialog handling (for opening, closing, and resetting forms) was also tested thoroughly to ensure a smooth user experience.

Both simple and complex user actions were tested, including sorting, searching, and performing CRUD operations, while ensuring state persistence throughout. Edge cases, such as attempting to load an incorrect or empty media file, submitting forms with invalid data, or performing operations on duplicate entries, were also included to verify robustness.

 For a complete table listing all tests please refer to **CS2003_P2_Testing.pdf**.

Evaluation

The submission successfully meets all the core requirements outlined in the project brief. Every aspect, including the design, implementation, and testing of the application, has been completed, tested and works as expected.

The clear consistency across all stages reflects planning and execution. Each component aligns seamlessly, contributing to a cohesive and reliable final product that meets the project goals effectively.

Conclusion

The project delivered all required functionality and advanced features. Testing confirmed that the application performs reliably and meets the project goals.

For me, the challenge was the lack of advanced tools like React, TypeScript, and TailwindCSS. While not necessary for a project of this size, these tools would be important for maintaining the app as it expands. With more time, integrating them would have made future development more manageable.