**Unit 5 Reflective Learning Dairy**

Based on the work already performed in unit 1, I came up with the following three ideas on how to improve my site using javascript:

1. Dynamic Project Gallery with Filters and Sorting

As described in the project section of my sitemap in unit 1 I would like to create a dynamic project gallery that allows users to filter projects by category (e.g., “Web Development,” “Mobile Apps,” “integration Projects”) and sort them by date or name. Using JavaScript, I can implement a switch for each filter type and a drop-down menu to select the sorting preference. This would make navigating my portfolio much smoother and more interactive compared to static HTML lists. I may also add small animations (such as fade-in/fade-out effects) to make the transitions visually engaging.

This feature relates strongly to my personas and scenarios. For the recruiter and team lead persona, it allows them to quickly find the type of project they care about (e.g., “Mobile Apps” if they’re hiring for app developers). For the student persona, it allows them to learn from or compare relevant projects without going through unrelated work.

1. Personalized Greeting and Theme Customization

Another idea is to use JavaScript to detect the time of day and display a personalized greeting such as “Good morning, welcome to my portfolio!” or “Good evening, thanks for visiting.” Additionally, I could allow visitors to toggle between light and dark mode or choose a preferred accent color, with their choice saved in local storage so that it persists across sessions.

This connects directly to usability and accessibility for my personas. All my persona’s benefits from a professional but welcoming site that feels tailored to them, which may leave a stronger impression. They also benefit from theme customization since they may prefer a dark mode while browsing at night. In the scenario where a visitor returns to my site multiple times, the fact that their theme preferences are remembered shows attention to detail and improves their overall user experience.

1. Back to Top Button & Active Navigation Highlight

I can also add a floating “Back to Top” button that appears at the bottom corner of the page when a user scrolls down. Clicking this button would smoothly scroll the user back to the top of the page. Along with this, I plan to dynamically highlight or underline the navigation link for the page the user is currently on. For example, if the visitor is on the “Projects” page, the “Projects” link in the navigation bar will be visually marked. This can be achieved with JavaScript by detecting the current URL or scroll position and adding/removing CSS classes to the appropriate navigation items.

This improvement makes navigation faster and more user-friendly, especially for recruiters who may be scrolling through long pages of my portfolio. Instead of manually scrolling back, the floating button provides a quick and accessible way to return to the top. The active navigation highlighting also helps visitors know exactly where they are on the site, improving clarity and orientation.

**Dynamic Project Gallery with Filters and Sorting Design**

1. **Data**
   * projects (global): An array of project objects that hold all project data (name, category, date, description, imageUrl)
   * filteredProjects (global) : an array to store currently displayed projects
   * selectedCategory (global) : a string that holds the current filter (e.g . “Mobile Apps”)
   * sortOrder (global): a string that holds the current sorting selection ( “dateAsc”, “dateDesc” etc.)
2. **Functons and Classes**
   * Project(name, category, date, description, imageUrl)

* Class constructor used to create a project object.
* Scope: global
* renderProjects(projectList)
  + Input: array of projects
  + Output: updates the DOM with project cards.
  + Scope: global
* filterProjects(category)
  + Input: category string
  + Output: filters the project by the specified category, stores it in filteredProjects and updates the UI via renderProjects(filteredProjects)
* sortProjects(criteria)
  + Input: sorting criteria ("dateAsc", "dateDesc", "nameAsc", "nameDesc")
  + Output: sorts the filteredProjects and updates the UI via renderProjects()
* searchProjects()
  + Input: keyword typed in the search bar
  + Output: filters the project by the keyword using regex and updates the UI
* initGallery()
  + called on page load to load all projects when the user first visits the page and attaches event listeners to filter buttons and sort dropdown.

1. **Program Flow**

**A diagram of a computer

AI-generated content may be incorrect.**

**Reflection**

When I began working with JavaScript for my portfolio site, my main goal was to make the pages more interactive and useful for the personas and scenarios I designed in Unit 1. For example, the recruiter persona wanted to quickly browse and sort through my projects, while the fellow developer persona wanted an easy way to filter or search projects based on their interests. This made it clear to me that I needed features such as filtering by category, sorting by name or date, and a search function.

**My Learning Process**

I found the design stage of this project much easier than I expected because I could map the features directly to the personas and scenarios, I created in Unit 1. Once I finalized the design, I revisited JavaScript fundamentals to adapt my knowledge to the front-end. I refreshed myself using W3Schools, focusing on DOM manipulation, object handling, event listeners, array methods (filter, sort), and regular expressions.

After that refresher, I began developing the features incrementally. I first implemented project rendering, then sorting and filtering, and finally a search bar that uses regex to find matches in project titles and descriptions. Breaking the tasks into smaller steps made the process more manageable and allowed me to build on successes.

**Obstacles I Faced**

Even with my background, I ran into obstacles. For example, I initially used inline event handlers in the HTML but realized this wasn’t the best practice for clean, maintainable code. I switched to adding all event listeners in JavaScript once the DOM was loaded, which improved both the structure and separation of concerns. Another challenge was handling regex search properly—my first implementation was case-sensitive, which excluded relevant results. Adding the /i flag fixed this.

A key challenge was ensuring that renderProjects() only ran after the DOM had fully loaded. Running it too early meant the script tried to manipulate elements that didn’t yet exist.

Another challenge was adapting my code to **XHTML 1.0 strict compliance**. Initially, I used custom data-category attributes to store project categories, but I discovered that these attributes are not valid in XHTML 1.0 strict. To meet the course requirements, I learned about the rel attribute and used it as a valid alternative for storing category values.

**Debugging Process**

For debugging, I used Visual Studio Code, which made it much easier to identify and fix both compilation and runtime errors. I relied on breakpoints and ran my code in debug mode to step through execution and inspect variable values at key points. This gave me deeper insight into the control flow and helped me catch logical mistakes early.

Overall, this part of the course was an opportunity not just to practice JavaScript syntax, but also to expand the way I apply it—moving from backend integrations into user-facing web interactivity. That shift has strengthened both my technical skills and my appreciation of how code connects directly to user experience.