FGCT6021 – Mobile Application Development
Lab 3 – Topic 3: JavaScript & DOM Manipulation

Student Name: Bhupendra Thapa
Course: Computer Science

Repository: https://github.com/Bhupu12/learning-journal-pwa.git

---

1. Overview

This week, building on my previous work from Lab 2, I enhanced my Learning Journal PWA by integrating JavaScript and DOM manipulation to make the site interactive. I added a reusable navigation menu, an interactive dark/light theme toggle, live date/time display on the homepage, collapsible project details, and a journal entry form with word-count validation.

---

2. File and Folder Structure

The project retains the folder structure from Lab 2:

learning-journal-pwa/

│

├── index.html ← Homepage

├── journal.html ← Weekly journal & form

├── projects.html ← Project showcase with collapsible details

├── about.html ← About me page

│

├── css/

│ └── style.css ← Styling (mobile-first + dark mode)

│

├── js/

│ └── script.js ← JavaScript for navigation injection, theme toggle, collapsibles, date/time, form validation

│

└── images/ ← (Media assets folder)

---

3. Tasks Completed

- **Reusable Navigation Menu**
  I moved the <nav> HTML into a template string inside script.js and programmatically injected it into each page's . This ensures consistent navigation and easier future updates.
- **Interactive Features**
  - **Theme Switcher**: A button labelled " 🌓 " toggles between light and dark mode by applying a .dark-mode CSS class to <body>.
  - **Live Date/Time**: On the homepage (index.html), I added and . Using the Date() object and setInterval, the current date and time update live.
  - **Collapsible Project Cards**: In projects.html, each project card has a "View Details" button (class toggle-btn) and a hidden paragraph (class project-details). When the button is clicked, JavaScript toggles display of the details and changes the button text between "View Details" / "Hide Details".
  - **Journal Form Validation**: On journal.html, I added a form (id="journal-form") and textarea (id="journal-input"). On submit, JavaScript counts words and prevents submission if fewer than 10 words, alerting the user accordingly.
- **DOM Manipulation and Event Handling**
  I used document.getElementById() to access specific elements (nav container, date/time display, form, textarea) and document.querySelectorAll() for button collections. I listened to events such as click (theme toggle, collapsibles) and submit (journal form). I also used DOMContentLoaded event to ensure all elements are loaded before script execution.

---

4. Reflection

1. Which DOM selection methods did you use and why?
I used getElementById() whenever I needed to target a specific element with an ID (for example, the navigation container, date/time display, journal form and textarea). It is fast and precise for unique elements. I also used querySelectorAll() (or querySelector())

when I needed to target a group of elements (e.g., all buttons with class toggle-btn) or select by class, which made it easier to loop through a collection of elements.

2. What was the most challenging part about linking JavaScript with your HTML?
The most challenging part was making sure my JavaScript ran *after* the DOM had loaded; initially, I got errors like document.getElementById(...) is null. I solved this by wrapping my code in document.addEventListener("DOMContentLoaded", ...). Also, ensuring the correct file paths (i.e., js/script.js), and that the navigation element containers existed on every page required careful checking.

3. How did you test and debug your JavaScript code?
I used the browser's DevTools Console to monitor error messages, console.log() statements to inspect variables and the flow of the code, and the Elements tab to verify that my navigation and dynamically inserted elements were present in the DOM. I tested each feature step-by-step: first the navigation injection, then theme toggle, then date/time, then collapsible cards, then form validation. Each time I encountered a bug, I traced the error back to wrong element ID, missing file path, or missing event listener.

---

5. Summary

In this lab I expanded my Learning Journal into an interactive PWA prototype by integrating dynamic JavaScript and DOM manipulation. The website is now more engaging, modular, and ready for future enhancements (such as offline caching, installable PWA features). This work strengthens the foundation for forthcoming weeks, where I will add further functionality and polish.

---

6. Next Steps

- Implement local Storage to persist journal entries.
- Add service worker for offline support.
- Enhance UI with animations or richer project interactivity.
- Continue populating the project page with new work as I progress through the module.

End of submission — Thank you!
Bhupendra Thapa