Coding & Development / JavaScript

Write vanilla JavaScript to select, modify, or traverse DOM elements based on described behavior.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Frontend Interactivity, DOM Manipulation, Vanilla JS
Updated: May 2026
Why This Prompt Exists
Most DOM manipulation is repetitive — selecting elements, adding event listeners, updating content.

You get:

- inefficient selectors (querySelectorAll on every click)
- no event delegation (listeners on every element)
- memory leaks (listeners not removed)
- no null checks (element not found crashes)
- inline styles instead of CSS classes

But DOM manipulation is not random.

It is intentional interaction with the page.

- Selectors: getElementById, querySelector, querySelectorAll
- Traversal: parent, children, nextSibling, closest
- Modification: textContent, innerHTML, classList, setAttribute
- Events: addEventListener with proper removal
- Performance: event delegation, debouncing, throttling

Without best practices, DOM code is slow and buggy.

This framework forces AI to write efficient, safe DOM scripts.

The Prompt

```
Assume the role of a frontend developer who writes efficient, safe DOM
manipulation code.

Your task is to generate a vanilla JavaScript DOM manipulation script.

Generate:

1. SELECTORS
   - How to find target elements
   - Null checks for missing elements

2. DOM READ/TRAVERSAL
   - Get current values or structure

3. DOM MODIFICATION
   - Changes to make (content, attributes, classes, styles)

4. EVENT HANDLERS (if needed)
   - Events to listen for
   - Event delegation for dynamic elements
   - Cleanup (removeEventListener)

5. PERFORMANCE OPTIMIZATIONS
   - Debouncing or throttling if needed
```

   - Batch DOM updates

6. ERROR HANDLING
   - Handle missing elements gracefully

INPUTS:

Trigger (what starts the action):
[E.G., "Button click," "Page load," "Form submit," "Scroll"]

Target Elements (describe):
[E.G., "All elements with class 'accordion-header'"]

Action to Perform:
[E.G., "Toggle visibility of the next sibling element"]

Initial State (if any):
[E.G., "First accordion open by default"]

Dynamic Content (elements added after page load):
[YES / NO]

RULES:
- Use const/let (not var)
- Check if element exists before manipulating
- Use classList for toggling classes (not style directly)
- Use event delegation for dynamic elements
- Remove event listeners when no longer needed
- Batch DOM updates for performance

```
- Use textContent (not innerHTML) when possible (security)
```

How To Use It

- Describe the trigger clearly — what user action starts the script.
- Be specific about target elements (IDs, classes, attributes).
- If elements are added dynamically, set Dynamic Content to YES.
- Test the script with and without the target elements present.
- Use browser DevTools to debug selector issues.

Example Input

Trigger: Button click (button with class “toggle-btn”)

Target Elements: The parent container of the button (div with class “accordion-item”)

Action to Perform: Toggle class “active” on the parent, which changes the display of a child element with class “accordion-content”

Initial State: First accordion has class “active” by default

Dynamic Content: NO (accordions exist on page load)

Why It Works
Most DOM manipulation is inefficient and buggy.

This framework improves outcomes by forcing:

- null checks (prevents crashes)
- event delegation (handles dynamic content)
- classList (cleaner than style manipulation)
- performance optimizations (debounce, batch updates)
- cleanup (prevents memory leaks)

Great DOM scripts don’t just work — they’re efficient, safe, and maintainable.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, JavaScript frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The JavaScript Async/Await Converter