Week2_DAY-3_Hands On_Koduru_Ushasree

LEVEL – 1 (Problem 1): Interactive Feedback Button

Scenario:

A startup company wants a simple interactive webpage where users can click a button to submit feedback and instantly see a confirmation message without refreshing the page.

Requirements

- Create a webpage with a “Submit Feedback” button.
- When clicked, display a confirmation message dynamically.
- Use JavaScript event listeners to handle the click event.

Technical Constraints

- Use plain HTML, CSS, and JavaScript.
- Event handling must use addEventListener().
- No external frameworks allowed.

Code :-

p1.html > html > body > script > btn.addEventListener("click") callback

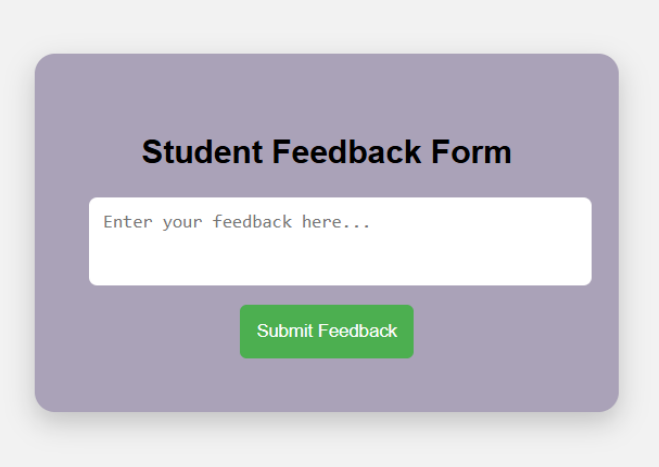
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Student Feedback</title>

<style>
body {
    margin: 0;
    height: 100vh;
    display: flex;
    justify-content: center;
    align-items: center;
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
}

.card {
    background-color: #aaa2b8;
    padding: 40px;
    width: 350px;
    text-align: center;
    border-radius: 15px;
    box-shadow: 0 8px 20px rgba(0,0,0,0.2);
}
```

```
    textarea {
        width: 100%;
        padding: 10px;
        border-radius: 6px;
        border: none;
        resize: none;
    }

    button {
        background-color: #4CAF50;
        color: white;
        border-radius: 5px;
        border: none;
        padding: 12px;
        margin-top: 10px;
        cursor: pointer;
    }

    button:hover {
        background-color: #568c59;
    }

    #output {
        margin-top: 15px;
        font-size: 16px;
        font-weight: bold;
        display: none;
    }
    </style>
    </head>

    <body>
```

```
<div class="card">
    <h2>Student Feedback Form</h2>

    <textarea id="feedbackInput" rows="3"placeholder="Enter your feedback here..."></textarea>

    <button id="feedbackBtn">Submit Feedback</button>

    <div id="output"></div>
</div>

<script>
const btn = document.getElementById("feedbackBtn");
const msgDiv = document.getElementById("output");
const feedbackInput = document.getElementById("feedbackInput");
```

```
76  btn.addEventListener("click", function() {
77
78      const feedbackText = feedbackInput.value.trim();
79
80      if (feedbackText === "") {
81          msgDiv.textContent = "Please enter your feedback!";
82          msgDiv.style.color = "red";
83      } else {
84
85          msgDiv.textContent = "Thank You! Feedback submitted successfully.";
86          msgDiv.style.color = "green";
87
88          feedbackInput.value = "";
89      }
90
91      msgDiv.style.display = "block";
92  });
93  </script>
94
95  </body>
96  </html>
```

Output :-



Technical Requirements :-

The application must be developed using plain HTML, CSS, and Vanilla JavaScript without using any external libraries or frameworks. Event handling should strictly be implemented using the addEventListener() method, and inline event attributes such as onclick must not be used. DOM elements must be accessed using standard DOM selection methods like getElementById().

LEVEL – 1 (Problem 2): Theme Toggle Application

Scenario:

A small blog website wants to allow users to switch between Light Mode and Dark Mode interactively.

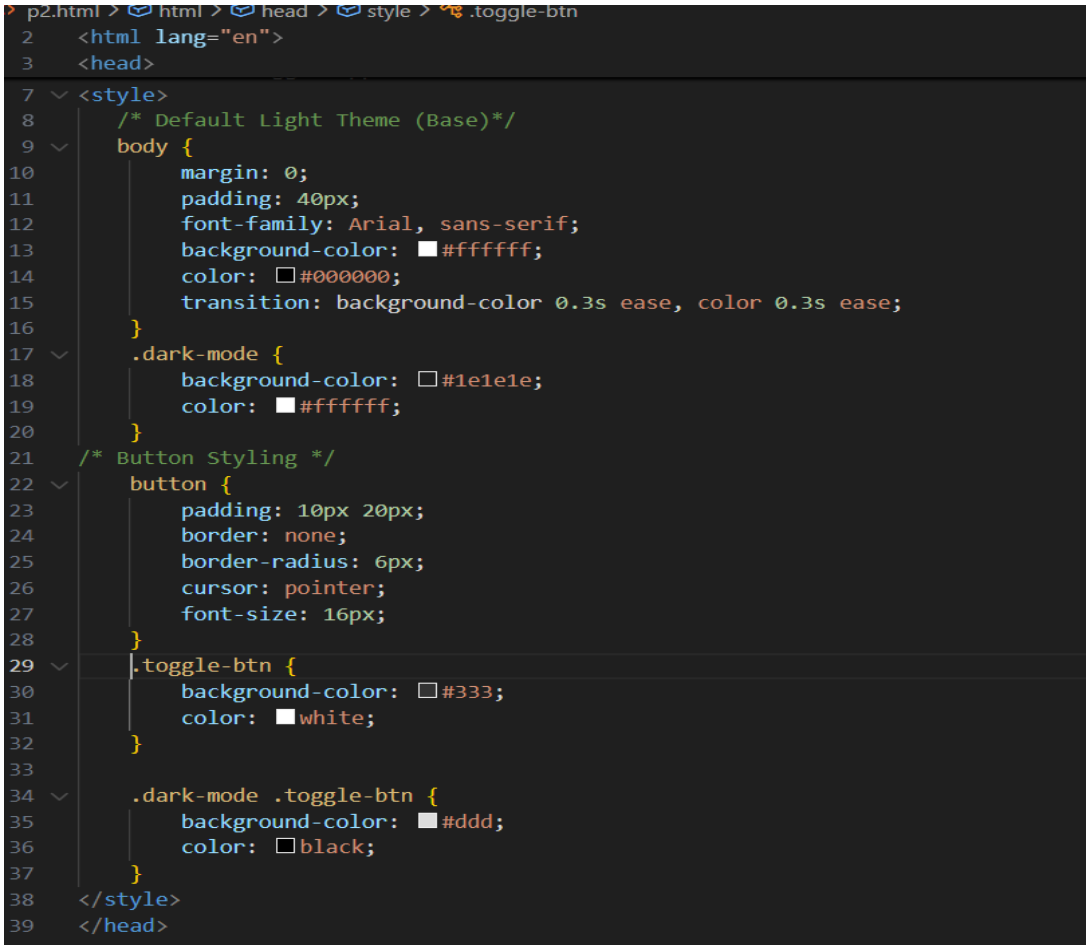
Requirements

- Create a toggle button to switch themes.
- Change background and text color dynamically.
- Store theme preference in localStorage.

Technical Constraints

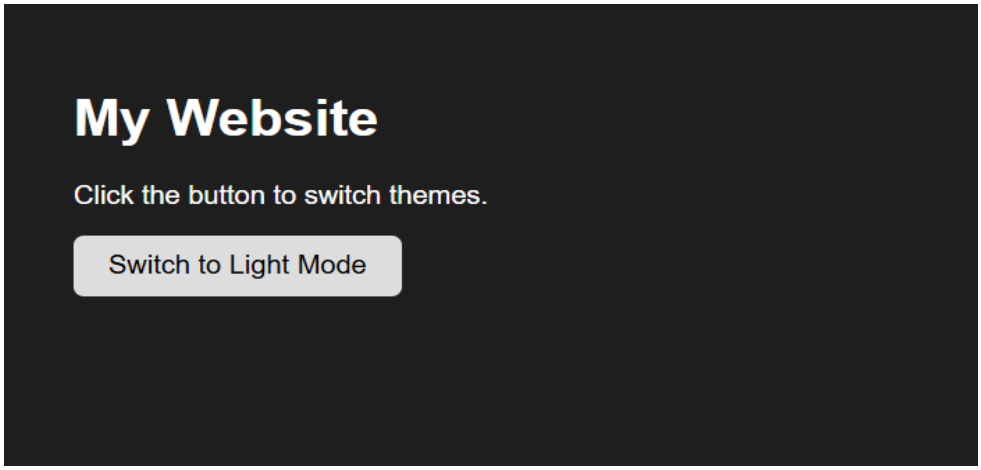
- Use Vanilla JavaScript.
- Use addEventListener() for events.
- No CSS frameworks.

Code :-



```
<body>
<h1>My Website</h1>
<p> Click the button to switch themes.</p>
<button id="themeToggle" class="toggle-btn">Switch to Dark Mode</button>
<script>
//   DOM Selection
const toggleBtn = document.getElementById("themeToggle");
// Apply Saved Theme On Load
function loadTheme() {
    const savedTheme = localStorage.getItem("theme");
    if (savedTheme === "dark") {
        document.body.classList.add("dark-mode");
        toggleBtn.textContent = "Switch to Light Mode";
    }
}
//   Function: Toggle Theme
function toggleTheme() {
    // Toggle dark-mode class on body
    document.body.classList.toggle("dark-mode");
    // Check if dark mode is active
    const isDark = document.body.classList.contains("dark-mode");
    if (isDark) {
        localStorage.setItem("theme", "dark");
        toggleBtn.textContent = "Switch to Light Mode";
    } else {
        localStorage.setItem("theme", "light");
        toggleBtn.textContent = "Switch to Dark Mode";
    }
}
toggleBtn.addEventListener("click", toggleTheme);
// Load Saved Theme When Page Opens
window.addEventListener("load", loadTheme);
</script>
</body>
</html>
```

Output :-



Technical Requirements :-

The theme toggle application must be implemented using Vanilla JavaScript along with standard HTML and CSS, without relying on any external CSS or JavaScript frameworks. User interactions must be handled using addEventListener(), and theme switching should dynamically modify styles such as background color and text color through DOM manipulation or class toggling. The selected theme preference must be stored and retrieved using localStorage to ensure persistence even after page reload.

LEVEL – 2 (Problem 3): Dynamic To-Do List

Scenario:

A productivity application requires an interactive To-Do list where users can add, delete, and mark tasks as complete without refreshing the page.

Requirements

- Add new tasks dynamically.
- Delete tasks on button click.
- Mark tasks as completed.
- Use event delegation for better performance.

Technical Constraints

- Must use DOM manipulation & Code should follow modular structure.
- Event delegation required.

Code :-

p3.html > html > head > style > button

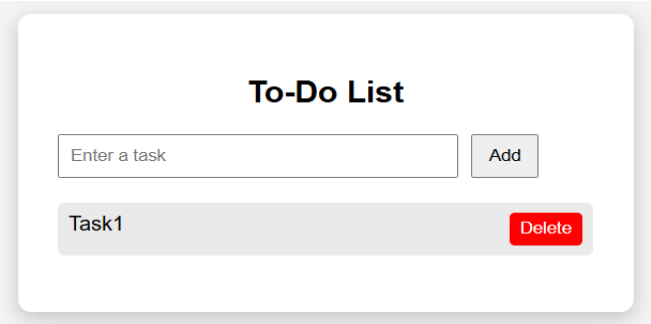
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Dynamic To-Do List</title>

<style>
    body {
        font-family: Arial, sans-serif;
        background: #f4f4f4;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
    }
    .container {
        background: white;
        padding: 30px;
        width: 400px;
        border-radius: 10px;
        box-shadow: 0 5px 15px rgba(0,0,0,0.2);
    }
    h2 {
        text-align: center;
    }
    input {
        width: 70%;
        padding: 8px;
    }
    button {
        padding: 8px 12px;
        cursor: pointer;
        margin-left: 5px;
    }
```

```css
ul {
    list-style: none;
    padding: 0;
    margin-top: 20px;
}

li {
    display: flex;
    justify-content: space-between;
    padding: 8px;
    margin-bottom: 8px;
    background: #eaeaea;
    border-radius: 5px;
}

.completed {
    text-decoration: line-through;
    color: gray;
}

.delete-btn {
    background: red;
    color: white;
    border: none;
    border-radius: 4px;
    padding: 5px 8px;
}
```

```html
<body>
<div class="container">
    <button id="addBtn">Add</button>
    <ul id="taskList"></ul>
</div>

<script>
// DOM Selection
const taskInput = document.getElementById("taskInput");
const addBtn = document.getElementById("addBtn");
const taskList = document.getElementById("taskList");
// Function: Create Task Element
function createTask(taskText) {
    const li = document.createElement("li");
    li.innerHTML = `
        <span class="task-text">${taskText}</span>
        <button class="delete-btn">Delete</button>`;
    return li;
}
// Function: Add Task
function addTask() {
    const text = taskInput.value.trim();
    if (text === "") return; // prevent empty tasks
    const taskItem = createTask(text);
    taskList.appendChild(taskItem);
    taskInput.value = ""; // clear input
}
// Event: Add Button Click
addBtn.addEventListener("click", addTask);
```

```
100   // Event Delegation (Imp)
101   taskList.addEventListener("click", function(event) {
102         // If Delete button clicked
103       if (event.target.classList.contains("delete-btn")) {
104           event.target.parentElement.remove();
105       }
106       // If Task text clicked (mark complete)
107       if (event.target.classList.contains("task-text")) {
108           event.target.classList.toggle("completed");
109       }
110   });
111   </script>
112   </body>
113   </html>
```

Output :-



Technical Requirements :-

The To-Do List application must be built entirely using HTML, CSS, and Vanilla JavaScript with no external frameworks. All tasks should be created, updated, and removed dynamically using DOM manipulation methods such as createElement(), appendChild(), and remove(). Event delegation must be implemented by attaching a single event listener to a parent container element and handling child element actions through event.target, rather than assigning individual listeners to each button. The code structure should follow a modular approach, separating concerns into reusable functions for adding tasks, deleting tasks, marking tasks as completed, and rendering updates to the UI.

LEVEL – 2 (Problem 4): Mini Product Search App

Scenario:

An e-commerce company needs a mini product search application that filters products dynamically as users type in the search box.

Requirements

- Display product list from a predefined array.
- Filter products dynamically using input event.
- Display "No Results Found" when applicable.

Technical Constraints

- Must use DOM manipulation.
- Event delegation required.
- Code should follow modular structure.

Code :-

p4.html > html > head > style > ul

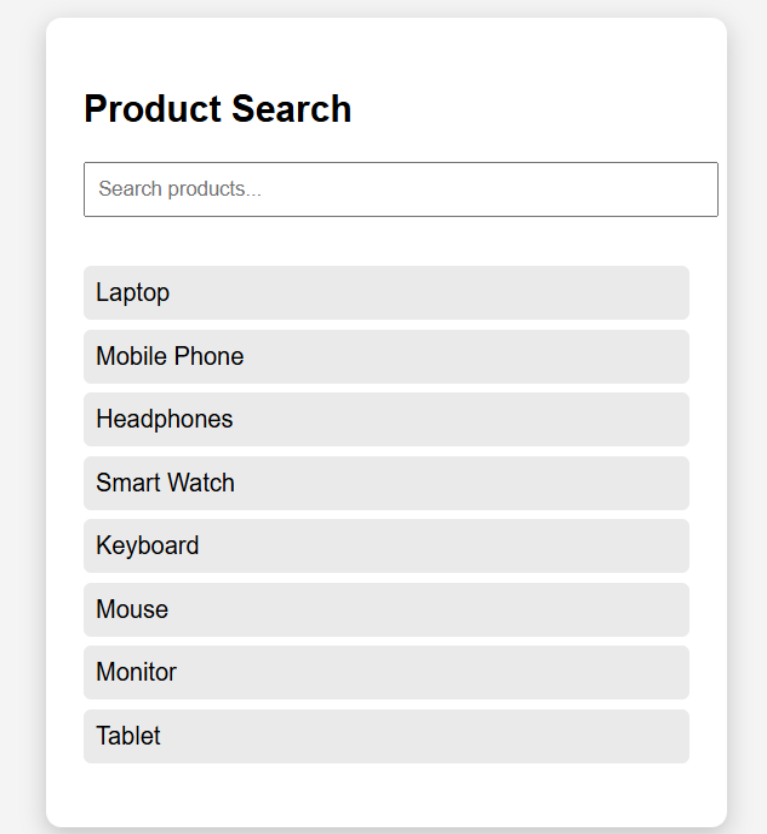
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Mini Product Search</title>

<style>
    body {
        font-family: Arial, sans-serif;
        background: #f4f4f4;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
    }

    .container {
        background: white;
        padding: 25px;
        width: 400px;
        border-radius: 10px;
        box-shadow: 0 5px 15px rgba(0,0,0,0.2);
    }

```

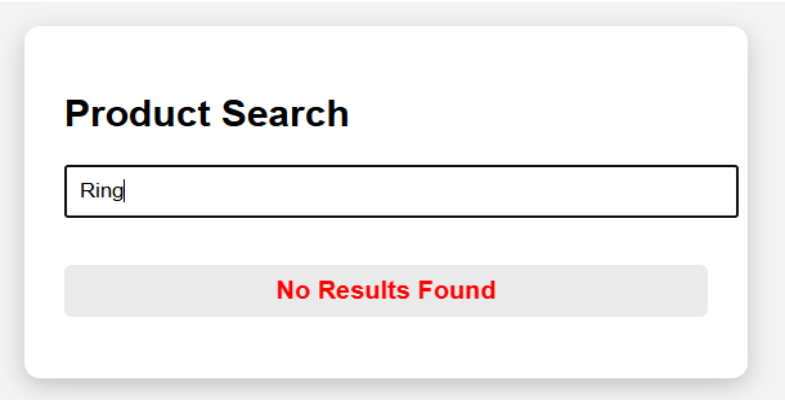
```
input {
        width: 100%;
        padding: 8px;
        margin-bottom: 15px;
    }
    ul {
        list-style: none;
        padding: 0;
    }
    li {
        padding: 8px;
        margin-bottom: 6px;
        background: #eaeaea;
        border-radius: 5px;
        cursor: pointer;
    }
    .no-results {
        color: red;
        font-weight: bold;
        text-align: center;
    }
</style>
</head>
<body>
<div class="container">
    <h2>Product Search</h2>
    <input type="text" id="searchInput" placeholder="Search products...">
    <ul id="productList"></ul>
</div>
<script>
// Application State
const products = ["Laptop", "Mobile Phone", "Headphones", "Smart Watch", "Keyboard", "Mouse", "Monitor","Tablet"];
```

```
// DOM Selection
const searchInput = document.getElementById("searchInput");
const productList = document.getElementById("productList");
// Function: Render Products
function renderProducts(items) {

    productList.innerHTML = ""; // Clear previous list

    if (items.length === 0) {
        productList.innerHTML = `<li class="no-results">No Results Found</li>`;
        return;
    }

    items.forEach(product => {
        const li = document.createElement("li");
        li.textContent = product;
        productList.appendChild(li);
    });
}
// Function: Filter Products
function filterProducts(query) {

    const filtered = products.filter(product =>
        product.toLowerCase().includes(query.toLowerCase())
    );

    renderProducts(filtered);
}
```

```
88    // Event: Real-Time Input
89    searchInput.addEventListener("input", function() {
90        const searchText = searchInput.value.trim();
91        filterProducts(searchText);
92    });
93
94    // Event Delegation Example
95    productList.addEventListener("click", function(event) {
96        if (event.target.tagName === "LI" &&
97            !event.target.classList.contains("no-results")) {
98            alert("You selected: " + event.target.textContent);
99        }
00    });
01
02    // Initial Render
03    renderProducts(products);
04    </script>
05    </body>
06    </html>
```

Output :-





Technical Requirements :-

The product search application must be developed using plain HTML, CSS, and Vanilla JavaScript without any third-party libraries. The product data should be stored in a predefined array that serves as the application's state. Filtering must be implemented dynamically using the input event so that results update in real time as the user types. DOM manipulation techniques must be used to render the filtered results and to display a "No Results Found" message when applicable. Event delegation should be applied where appropriate to handle interactions efficiently. The code should follow a modular structure with clearly defined functions for filtering, rendering, and managing state to ensure maintainability and scalability.