Javascript _Day - 8 Hands On _Sana Naveen

Problem 1

Problem Statement :
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.

Expectations:
• Proper DOM manipulation.
• Clean and readable code.

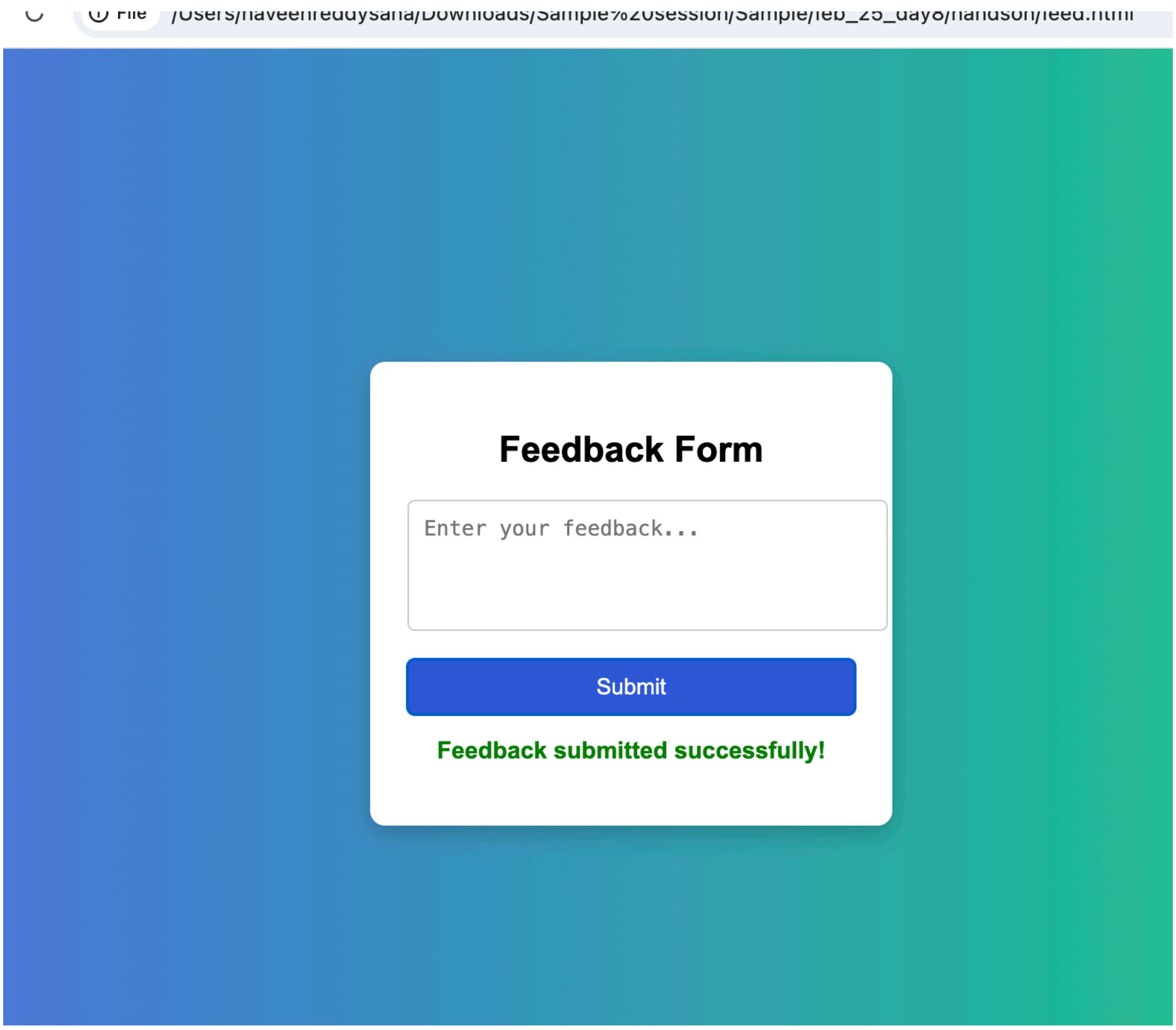
Learning Outcome
• Understand browser event handling.
• Learn DOM manipulation basics.

Code :

```
<!DOCTYPE html>
<html>
<head>
<style>

body {
   font-family: Arial, sans-serif;
   background: linear-gradient(to right, #4e73df, #1cc88a);
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
}

.container {
   background: white;
   padding: 25px;
   border-radius: 10px;
   width: 300px;
```

```
    box-shadow: 0 5px 15px rgba(0,0,0,0.2);
    text-align: center;
}

textarea {
    width: 100%;
    padding: 10px;
    border-radius: 5px;
    border: 1px solid #ccc;
    resize: none;
    margin-bottom: 15px;
    font-size: 14px;
}

button {
    width: 100%;
    padding: 10px;
    border: none;
    border-radius: 5px;
    background-color: #4e73df;
    color: white;
    font-size: 15px;
    cursor: pointer;
    transition: 0.3s;
}

button:hover {
    background-color: #2e59d9;
}

#ps {
    margin-top: 15px;
    font-weight: bold;
}

.success {
    color: green;
}

.error {
    color: red;
}

</style>
</head>

<body>

<div class="container">
    <h2>Feedback Form</h2>
    <textarea id="text" rows="4" placeholder="Enter your feedback..."></textarea>
    <button id="feed">Submit</button>
    <p id="ps"></p>
</div>

<script>
function dis() {
    let textarea = document.getElementById('text').value;
    let ps = document.getElementById('ps');
    if(textarea.trim() === "") {
        ps.innerHTML = "Please enter feedback!";

    }
    else{

    ps.innerHTML = "Feedback submitted successfully!";
    }
    textarea.value = "";
}

document.getElementById('feed').addEventListener("click", dis);
</script>

</body>
</html>
```

Output Screenshot :


File
/Users/naveenreddysana/Downloads/Sample%20session/Sample/feb_25_day8/handson/feed.html
Enter your feedback...
Submit

Feedback Form

Feedback submitted successfully!

Explanation :

This code builds a simple feedback form UI using HTML, CSS, and JavaScript.
CSS is used to create a centered card layout with a gradient background and styled button.
JavaScript adds a click event listener to the Submit button. When clicked, it checks the textarea input using trim(). If empty, it shows an error message; otherwise, it displays a success message using innerHTML.
It demonstrates basic DOM manipulation, event handling, and form validation.

Problem 2

Problem Statement :
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.

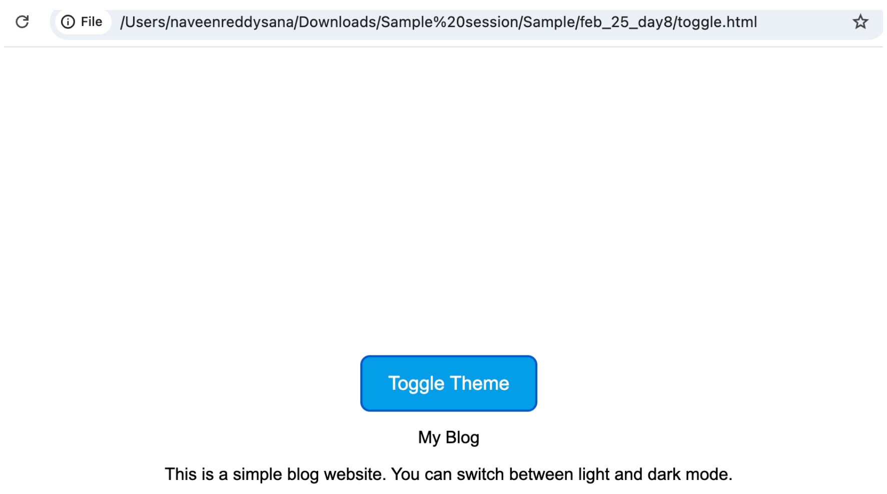
Technical Constraints
• Use Vanilla JavaScript.
• Use addEventListener() for events.
• No CSS frameworks.

Code :

```
<!DOCTYPE html>
<html>
<head>
   <title>Mini Blog</title>
   <style>
      body {
   background-color: white;
   color: black;
   transition: 0.3s;
   margin: 0;
   height: 100vh;
   display: flex;
   justify-content: center;
   align-items: center;
   font-family: Arial, sans-serif;
}

.container {
   text-align: center;
}
button {
   padding: 15px 25px;   /* increases size */
   font-size: 18px;      /* bigger text */
   border-radius: 8px;
   border: none;
   cursor: pointer;
   background-color: rgb(16, 159, 235);
   color: white;
}
      </style>
</head>

<body>

<div class="container">
   <button id="to">Toggle Theme</button>

   <p>My Blog</p>
```

```
    <p>
      This is a simple blog website. You can switch between light and dark mode.
    </p>
  </div>

  <script>
    //localStorage.setItem("theme", "dark")
    let b = document.body;
    function change() {
      if (b.style.backgroundColor === "black") {
          b.style.backgroundColor = "white";
          b.style.color = "black";
          localStorage.setItem("theme", "light")
      } else {
          b.style.backgroundColor = "black";
         b.style.color = "white";
         localStorage.setItem("theme", "dark")
      }
    }
    document.getElementById('to').addEventListener("click", change);
    </script>

  </body>
  </html>
```

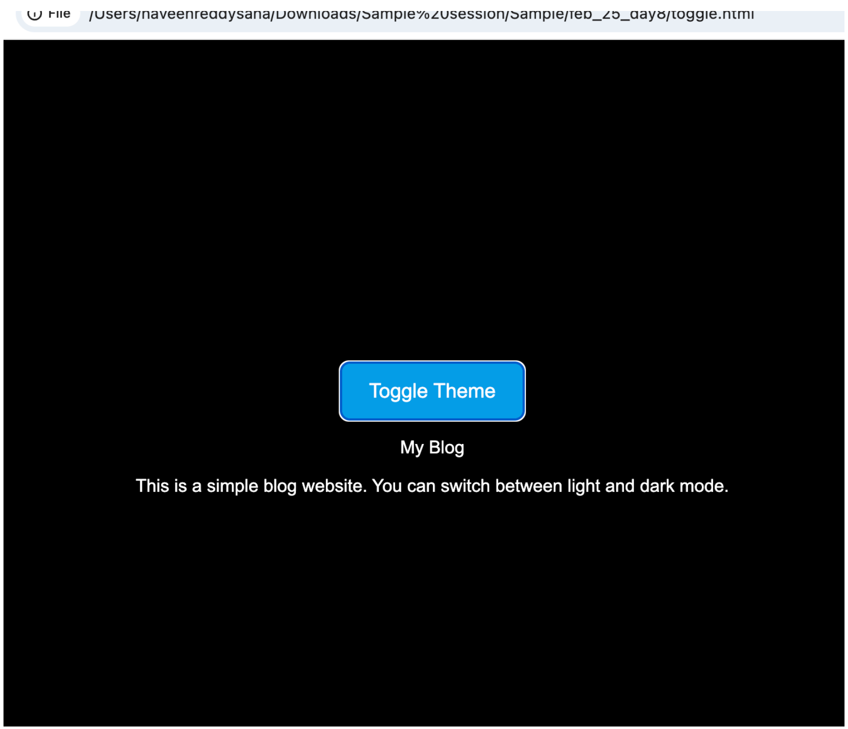
OutputScreenshot :



File /Users/naveenreddysana/Downloads/Sample%20session/Sample/feb_25_day8/toggle.html

My Blog

This is a simple blog website. You can switch between light and dark mode.



File /Users/naveenreddysana/Downloads/Sample%20session/Sample/feb_25_day8/toggle.html

My Blog

This is a simple blog website. You can switch between light and dark mode.

Explanation :

This code builds a centred mini blog page with a theme toggle button.
When the button is clicked, JavaScript changes the <body> background and text color between light and dark modes. The selected theme is saved using localStorage, allowing the preference to persist in the browser.
It demonstrates DOM manipulation, event handling, conditional logic, and localStorage usage.

Problem 3

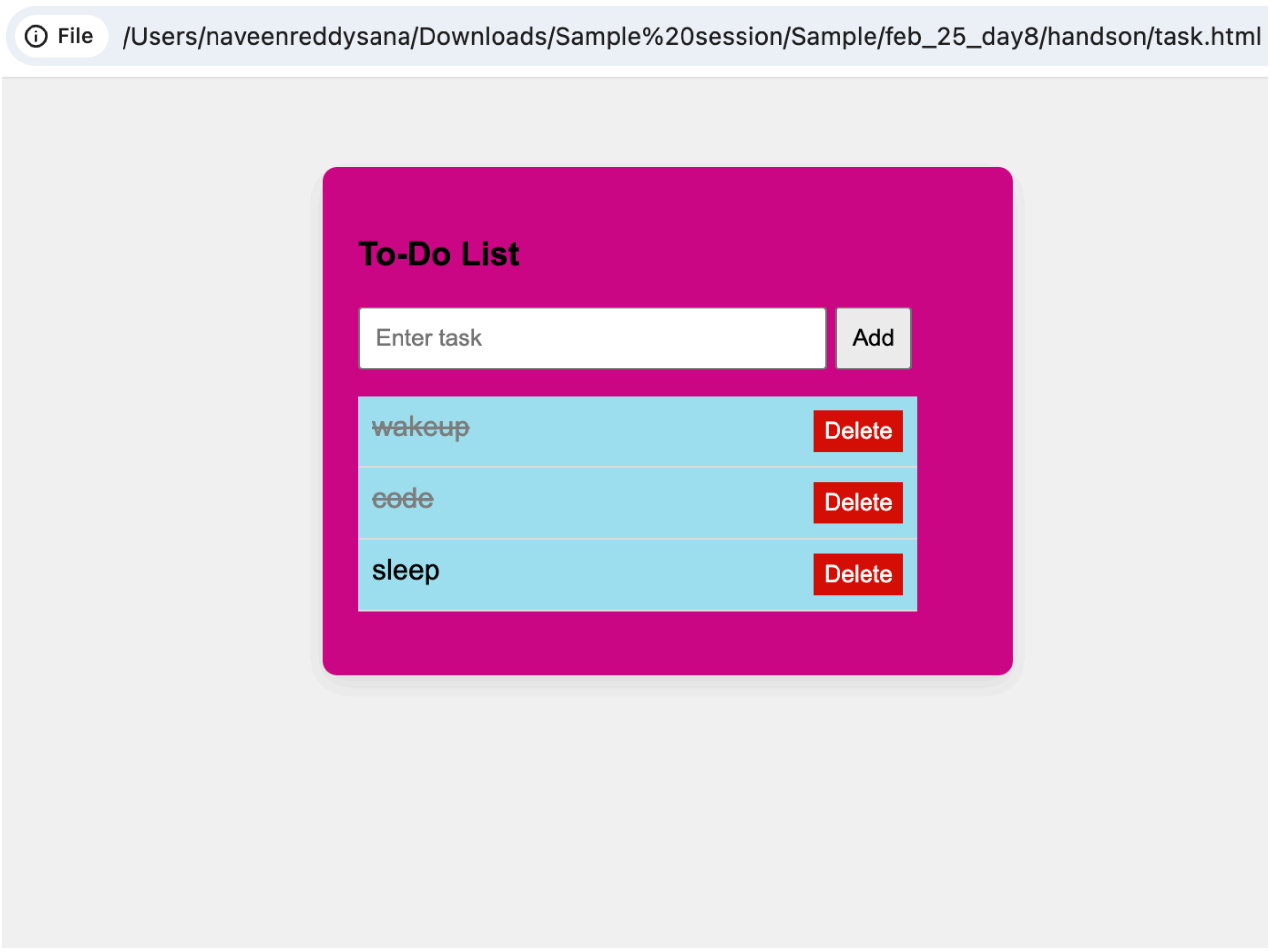
Problem Statement :
A productivity application requires an interactive To-Do list where users can add, delete, and mark tasks as complete without refreshing the page.
 Requirements
• Add new tasks dynamically.
• Delete tasks on button click.
• Mark tasks as completed.
• Use event delegation for better performance.

 Technical Constraints
• Must use DOM manipulation.
• Event delegation required.
• Code should follow modular structure.

Code :

```
<!DOCTYPE html>
<html>
<head>
   <title>To-Do App</title>
   <style>
      body {
         font-family: Arial;
         background: #f4f4f4;
         display: flex;
         justify-content: center;
         margin-top: 50px;
      }

      .todo-container {
         background: rgb(204, 12, 134);
         padding: 20px;
         width: 350px;
         border-radius: 8px;
         box-shadow: 0 5px 15px rgba(0,0,0,0.1);
      }

      input {
         width: 70%;
         padding: 8px;
      }

      button {
         padding: 8px;
         cursor: pointer;
      }

      ul {
         list-style: none;
         padding: 0;
         margin-top: 15px;
      }

      li {
         display: flex;
         justify-content: space-between;
         padding: 8px;
         background-color: rgb(159, 224, 238);
         border-bottom: 5px solid #ddd;
         width: 300px;
      }

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

      .delete-btn {
         background: rgb(215, 13, 13);
         color: white;
         border: none;
         padding: 4px 6px;
         cursor: pointer;
      }
   </style>
</head>

<body>

<div class="todo-container">
   <h3>To-Do List</h3>
   <input type="text" id="taskInput" placeholder="Enter task">
   <button id="addBtn">Add</button>

   <ul id="taskList"></ul>
</div>

<script>

// ===== MODULE =====
const TodoApp = (() => {

   const taskInput = document.getElementById("taskInput");
   const addBtn = document.getElementById("addBtn");
   const taskList = document.getElementById("taskList");

   // Add Task
   function addTask() {
      const taskText = taskInput.value.trim();
      if (taskText === "") return;

      const li = document.createElement("li");

      li.innerHTML = `
         <span class="task-text">${taskText}</span>
         <button class="delete-btn">Delete</button>
      `;

      taskList.appendChild(li);
      taskInput.value = "";
   }

   // Event Delegation
   function handleTaskClick(e) {
      if (e.target.classList.contains("delete-btn")) {
         e.target.parentElement.remove();
      }

      if (e.target.classList.contains("task-text")) {
         e.target.classList.toggle("completed");
      }
   }

   // Initialize
   function init() {
      addBtn.addEventListener("click", addTask);
      taskList.addEventListener("click", handleTaskClick);
   }

   return { init };

})();

// Start App
TodoApp.init();

</script>

</body>
</html>
```

Output ScreenShot :



ⓘ File /Users/naveenreddysana/Downloads/Sample%20session/Sample/feb_25_day8/handson/task.html

Explanation :
This code implements a modular To-Do application using HTML, CSS, and Vanilla JavaScript. The UI is styled with CSS to create a centered card layout and visually distinct task items. The JavaScript follows the Module Pattern (IIFE) to encapsulate variables and functions, preventing global scope pollution.
Tasks are added dynamically using DOM manipulation (createElement, innerHTML, appendChild). Event delegation is implemented by attaching a single click event listener to the <ul> (taskList) to handle both task deletion and completion toggling based on the clicked element’s class. This approach improves performance and scalability by avoiding multiple event listeners on individual elements.

Problem 4

Problem Statement :
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.

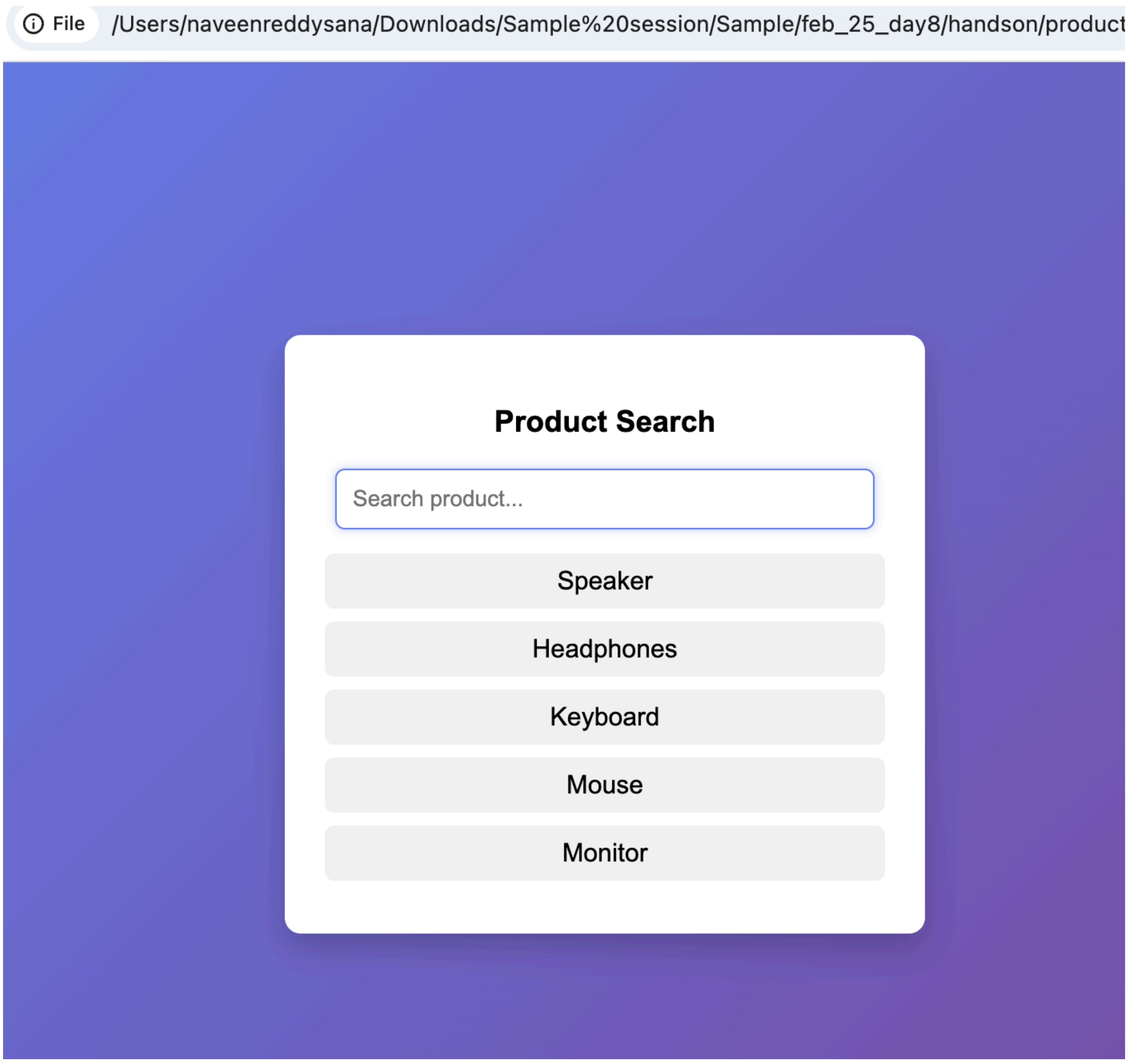
Technical Constraints
• Must use DOM manipulation.
• Event delegation required.
• Code should follow modular structure.

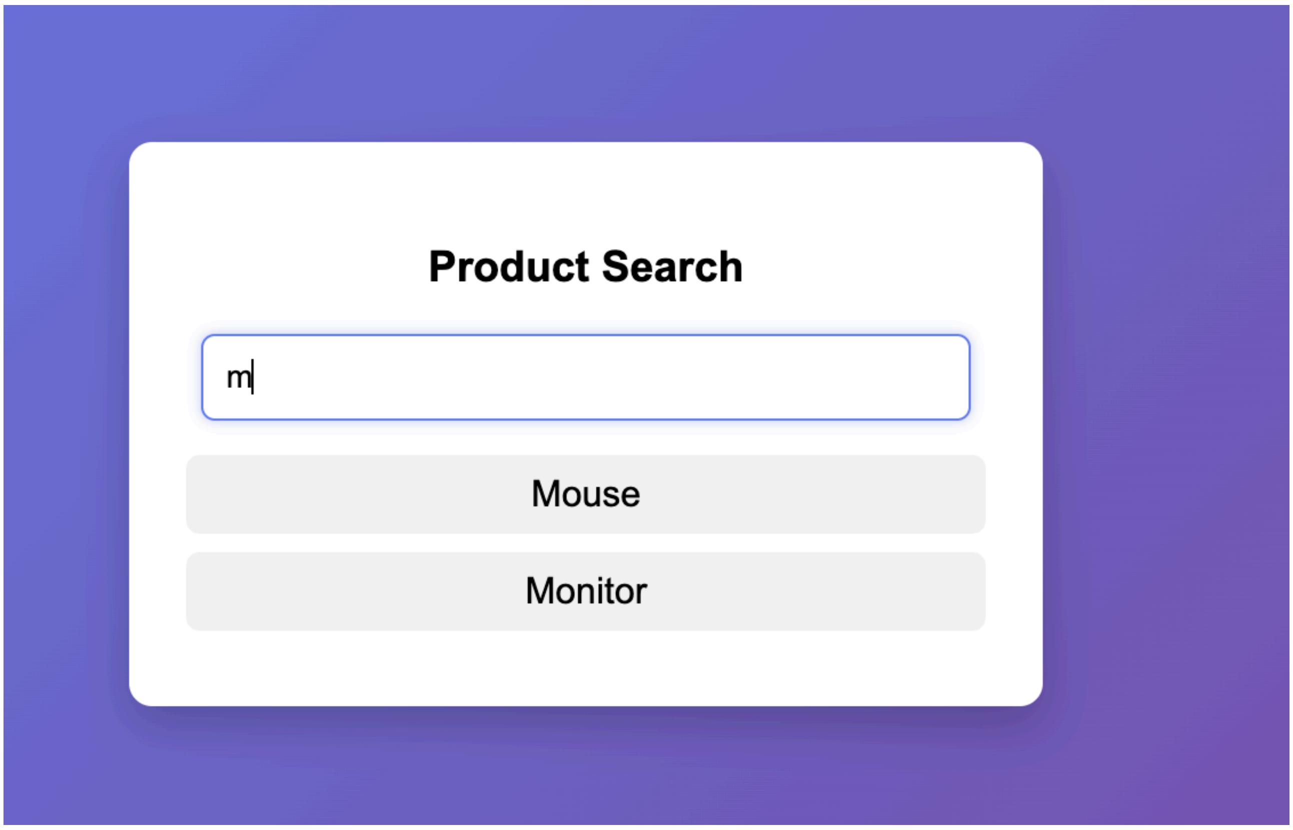
Code :

```
<!DOCTYPE html>
<html>
<head>
<style>
  body {
    font-family: Arial, sans-serif;
    background: linear-gradient(135deg, #667eea, #764ba2);
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
  }

  .container {
    background: white;
    padding: 25px;
    width: 350px;
    border-radius: 10px;
    box-shadow: 0 8px 20px rgba(0,0,0,0.2);
    text-align: center;
  }

  input {
    width: 90%;
    padding: 10px;
    border-radius: 6px;
    border: 1px solid #ccc;
    margin-bottom: 15px;
    font-size: 14px;
    outline: none;
    transition: 0.3s;
  }

  input:focus {
    border-color: #667eea;
    box-shadow: 0 0 5px rgba(102,126,234,0.4);
  }
```

```
    .product {
        background: #f4f4f4;
        padding: 8px;
        margin-bottom: 8px;
        border-radius: 6px;
        transition: 0.3s;
    }

    .product:hover {
        background: #e2e8f0;
        transform: scale(1.02);
    }

    .no-result {
        color: red;
        font-weight: bold;
    }
</style>
</head>

<body>

<div class="container">
    <h3>Product Search</h3>
    <input type="text" id="in" placeholder="Search product...">
    <div id="im"></div>
</div>

<script>
let products = [
    { id: 1, name: "Speaker", price: 1500 },
    { id: 2, name: "Headphones", price: 2000 },
    { id: 3, name: "Keyboard", price: 1200 },
    { id: 4, name: "Mouse", price: 800 },
    { id: 5, name: "Monitor", price: 10000 }
];

function display(list){
    let data = "";
    list.forEach(p => {
        data += `<div class="product">${p.name}</div>`;
    });
    document.getElementById("im").innerHTML = data;
}

display(products);

function check(){
    let s = document.getElementById('in').value.toLowerCase();

    if(s === ""){
        display(products);
    }
    else{
        let d = products.filter(a =>
            a.name.toLowerCase().includes(s)
        );

        if(d.length === 0){
            document.getElementById("im").innerHTML =
                `<div class="no-result">No products found</div>`;
        } else {
            display(d);
        }
    }
}

document.getElementById("in").addEventListener("input", check);
</script>

</body>
</html>
```

Output Screenshot :


File
/Users/naveenreddysana/Downloads/Sample%20session/Sample/feb_25_day8/handson/product
Product Search
Search product...
Speaker
Headphones
Keyboard
Mouse
Monitor



Explanation :
This code creates a dynamic product search app using Vanilla JavaScript.
Products are stored in an array and displayed using DOM manipulation. As the user types in the input field, the input event triggers filtering with filter() and includes(), updating the results instantly without page refresh. If no match is found, a “No products found” message is shown.
CSS is used to style the layout, input field, and product cards for a clean, modern UI.