Hands-On Lab: Manipulating DOM with JavaScript

##**Estimated Time: 45 minutes**

Introduction

The Document Object Model (DOM) represents the structure of an HTML document as an object, allowing you to interact with and modify the content, structure, and styles of web pages dynamically. JavaScript provides various DOM manipulation methods that enable developers to create dynamic and interactive web pages. In this lab, you will learn how to manipulate the DOM using JavaScript to build a to-do list application.

Objectives

By the end of this lab, you will:

- Understand how to use DOM manipulation methods such as `document.createElement()`, `appendChild()`, and `removeChild()`
- Add, edit, and remove elements dynamically on a web page
- Create an interactive to-do list application where tasks can be managed in real time

Exercise 1: Understanding starter code

In this exercise, you will create the basic structure for a to-do list app with three buttons: **Add Task**, **Remove Task**, and **Edit Task**. This code will set up the foundation of the app, and you will later enhance it in the subsequent exercises by adding functionality to these buttons.

- Create a new file named `index.html`
- Copy the code provided below and insert it into the `index.html` file

Basic HTML structure

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do List</title>
    <style>
        /* Styles the main container of the to-do list */
        .todo-container {
            width: 300px;
            margin: 50px auto;
            padding: 20px;
            border: 1px solid #ccc;
            border-radius: 5px;
        }

        /* Removes default list styling for the to-do list */
        .todo-list {
            list-style-type: none;
            padding: 0;
        }
        /* Styles each list item in the to-do list */
        li {
            margin: 10px 0;
            display: flex;
            justify-content: space-between;
            align-items: center;
        }
        /* Adds spacing for buttons in list items */
        button {
            margin-left: 10px;
        }
    </style>
</head>
```

```
<body>
<div class="todo-container">
    <h1>To-Do List</h1>
    <!-- Input field to add new tasks -->
    <input type="text" id="taskInput" placeholder="Add a new task">

    <!-- Button to add a new task to the list -->
    <button class="add-btn" onclick="addTask()">Add Task</button>

    <!-- Unordered list to display the tasks -->
    <ul class="todo-list" id="todoList"></ul>
</div>
<script>
    // Placeholder for functionality to be added in future exercises
</script>
</body>
</html>
```

Code explanation

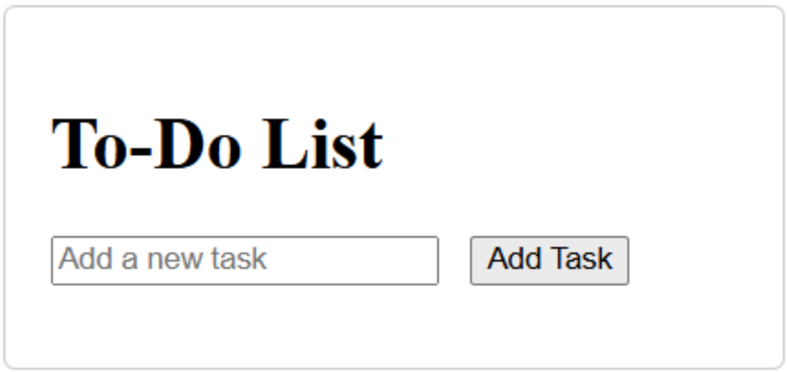
1. **HTML structure:**
    - The `<ul>` with the `id="todoList"` will hold the list of tasks.
    - The `<input>` field with `id="taskInput"` allows users to enter new tasks.
    - There is a button to trigger adding tasks, but the functionality will be implemented in the following exercises.
2. **CSS styling:**
    - The `.todo-container` class is used to style the to-do list container, centering it and adding padding and borders for a clean look.
    - The `.todo-list` class styles the unordered list, and `li` items are given margin and flexbox styling to align items neatly.
3. **JavaScript placeholder:**
    - The script section will contain the logic for interacting with the DOM, such as adding tasks, removing tasks, and editing tasks.

Launch the webpage using Live Server

1. Open your file explorer and locate your HTML file
2. Right-click the file and select **Open with Live Server**
3. A notification will appear indicating that the server has started on port 5500
4. Click the button below the notification or use the **Skills Network Toolbox**, then navigate to **Other > Launch Application**, enter port `5500`, and click the launch button highlighted in the screenshot below. This will open the webpage in a new browser tab.

Launch Application

The output should appear as shown in the screenshot below:



Exercise 2: Adding a task to the To-Do List

In this exercise, you will implement the functionality to add a new task to the list. The task will be added dynamically when the user interacts with the page. Here's how it works:

- When the user types a task into the input field and clicks the **Add Task** button, the task will be added to the list.
- Each task will be displayed with Edit and Delete buttons next to it.

Steps to implement the task:

1. **Selecting elements:**
   - Use `document.getElementById('taskInput')` to select the input field where the user will type their task
   - Use `document.getElementById('todoList')` to select the unordered list (`<ul>`) where the new tasks will be appended
2. **Creating new elements:**
   - Use `document.createElement("li")` to create a new list item (`<li>`) to hold the task
   - Use `document.createElement("span")` to create a `span` element that will hold the task text
   - Create buttons for **Edit** and **Delete** using `document.createElement("button")`
3. **Adding event handlers:**
   - Link the **Edit** button to an event handler by setting the `onclick` property to a function (e.g., `editTask(span)`)
   - Similarly, link the **Delete** button to an event handler (e.g., `removeTask(li)`)
4. **Appending elements to the DOM:**
   - Append the task text and the buttons to the list item (`<li>`) using the `appendChild()` method
   - Append the list item (`<li>`) to the unordered list (`<ul>`) using `appendChild()`
5. **Clearing the input field:**
   - After adding the task to the list, clear the input field using `input.value = ""`

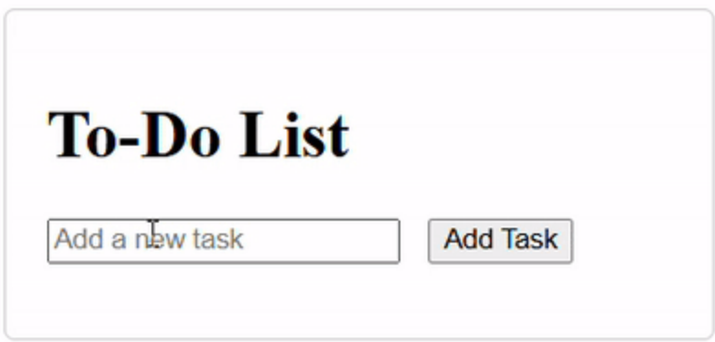
▼ Click here to view the solution code
Place the code provided below inside the script tag of the starter code

```
// Function to add a task
function addTask() {
    const input = document.getElementById("taskInput");
    const taskText = input.value.trim();
    if (taskText !== "") {
        const ul = document.getElementById("todoList");
        // Create new list item
        const li = document.createElement("li");
```

```
            // Create task text element
            const span = document.createElement("span");
            span.textContent = taskText;
            // Create edit button
            const editButton = document.createElement("button");
            editButton.textContent = "Edit";
            editButton.onclick = () => editTask(span);
            // Create remove button
            const removeButton = document.createElement("button");
            removeButton.textContent = "Delete";
            removeButton.onclick = () => removeTask(li);
            // Append buttons and text to the list item
            li.appendChild(span);
            li.appendChild(editButton);
            li.appendChild(removeButton);
            // Append list item to the list
            ul.appendChild(li);
            // Clear the input field
            input.value = "";
        }
        else {
                alert("Please enter a valid task.");
            }
    }
```

Output:

- Reload the webpage to test the functionality. If the live server is not running, start it and then launch the application.
- Once the page is loaded, type a task in the input field and click **Add Task**. The task will be added to the to-do list, appearing along with **Edit** and **Delete** buttons for each task.



Exercise 3: Editing a task in the To-Do List

In this exercise, you will implement the functionality to edit an existing task in the list. Here's how it works:

- When the user clicks the **Edit** button next to a task, they will be prompted to enter a new task text.
- If the user enters a valid new task text (non-empty), the task will be updated dynamically in the list.
- The task text will only be updated if the user provides a non-empty input.

Steps to implement the task:

1. **Set up the Edit function:**

- Create a function `editTask(span)` where `span` refers to the element containing the current task text
- This function will allow you to prompt the user for new text to replace the current task

2. **Prompt the user for new text:**
    - Inside the `editTask()` function, use the `prompt()` method to ask the user for a new task text. You can pass the current task (`span.textContent`) as the default value in the prompt.
3. **Check for valid input:**
    - After the user enters their new task, check if the input is not `null` and not empty. Use `newTask.trim()` to ensure that the task text is not just whitespace.
4. **Update the task:**
    - If the user provides valid input, update the task by assigning the new value to `span.textContent`. This will dynamically update the task in the list.

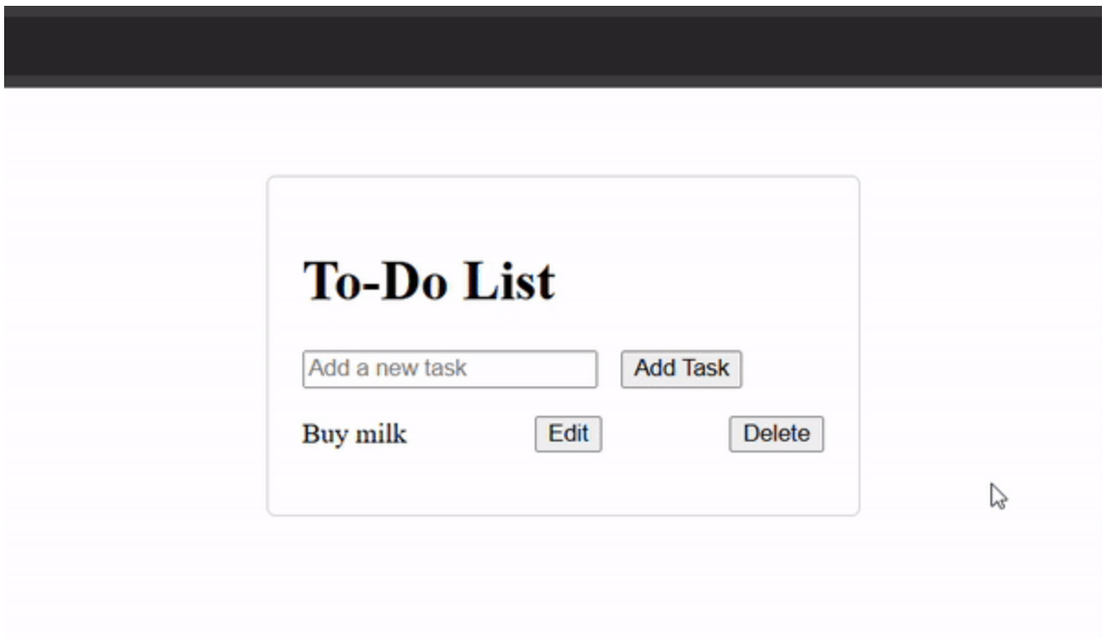
▼ Click here to view the solution code
Place the code provided below inside the script tag after the function addTask.

```
// Function to edit an existing task
function editTask(span) {
    // Prompt the user to enter a new task description
    const newTask = prompt("Edit your task:", span.textContent);

    // Update the task only if the input is not null or empty
    if (newTask !== null && newTask.trim() !== "") {
        span.textContent = newTask.trim(); // Set the new task text
    }
}
```

Output:

- Reload the webpage to test the functionality. If the live server is not running, start it and then launch the application.
- Once the page is loaded, clicking on the **Edit** button for a specific task will display a prompt to update the task. Make your edits and click **OK** to save the changes.



Exercise 4: Removing a task from the To-Do List

In this exercise, you will implement the functionality to remove a task from the list. Here's how it works:

- When the user clicks the **Delete** button next to a task, the task will be removed from the list.
- The task will be removed dynamically from the unordered list.

Steps to implement the task:

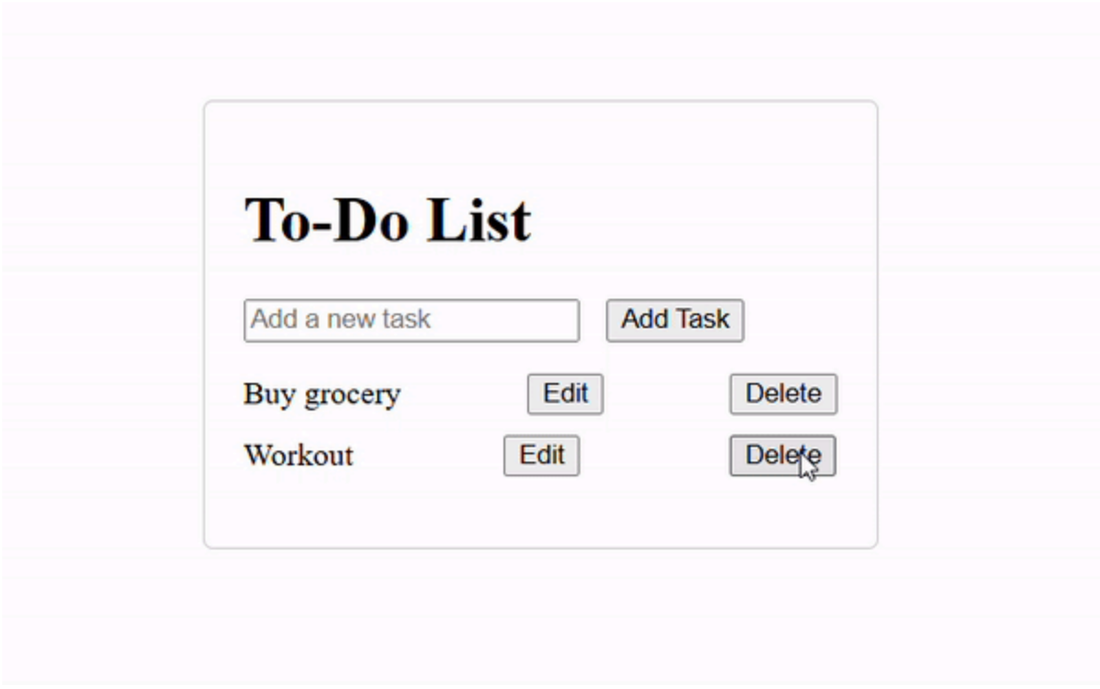
1. **Selecting the list:**
    - Use `document.getElementById('todoList')` to select the unordered list (`<ul>`) that contains the tasks
2. **Removing the task:**
    - When the user clicks the **Delete** button, use the `removeChild()` method to remove the specific task (represented as an `<li>` element) from the list.
3. **Removing the task element:**
    - Pass the `task` (the `<li>` element) to the `removeChild()` method to remove it from the DOM

▼ Click here to view the solution code
Place the code provided below inside the script tag after the function editTask.

```
// Function to remove a task from the to-do list
function removeTask(task){
    const ul = document.getElementById("todoList"); // Get the list container
    ul.removeChild(task); // Remove the specified task element
}
```

Output:

- Reload the webpage to test the functionality. If the live server is not running, start it and then launch the application.
- Once the page is loaded, clicking the **Delete** button will remove the corresponding task from the list.



▶ Click here to view the complete solution code

Conclusion

In this lab, you explored key DOM manipulation techniques to dynamically interact with a web page. By completing the exercises, you gained practical experience with methods such as `createElement()`,

`appendChild()`, and `removeChild()`. These techniques are essential for building interactive and user-friendly web applications, empowering you to create dynamic content that responds to user interactions.

Author

Rajashree Patil

© IBM Corporation. All rights reserved.