

# Module -26 Assignment

## **Q1.) Explain the DOM and its role in Web development.**

The **DOM (Document Object Model)** is a programming interface for web documents. It represents the structure of a document (like HTML or XML) as a tree of objects that can be manipulated programmatically using scripting languages such as JavaScript.

Here's a detailed breakdown of the DOM and its role in web development:

1. **Tree Structure:** The DOM represents the HTML document as a hierarchical tree structure, where each element, attribute, and piece of text in the HTML is a node in the tree.

### Example:

```
<html>
  <body>
    <h1>Hello, World!</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

### DOM Tree:

Document

```
└─ html
   └─ head
      └─ body
         └─ h1
            └─ p
```

### Role in Web Development:-

#### 1. **Dynamic Content Manipulation:**

- Developers can dynamically add, modify, or remove HTML elements and content using the DOM.

- Example: Change the text of an `<h1>` element.

Code: `document.querySelector("h1").textContent = "Welcome!"`;

## 2. Event Handling:

- The DOM allows attaching event listeners to elements, enabling interactivity.
- Example: Add a click event to a button.

Code: `document.querySelector("button").addEventListener("click", () => {  
 alert("Button clicked!");  
});`

## 3. Styling and Class Manipulation:

- Developers can dynamically update styles or classes of elements.
- Example: Change the background color of a `<div>` element.

Code: `document.querySelector("div").style.backgroundColor = "blue"`;

## 4. Accessing and Updating Attributes:

- Attributes like `id`, `src`, `href` can be accessed or updated using the DOM.
- Example: Change the `src` of an image.

Code: `document.querySelector("img").src = "new-image.jpg"`;

## 5. Creating Interactive Web Applications:

- By leveraging the DOM, developers can build interactive web applications like forms with validation, dynamic content loaders, and single-page applications (SPAs).

## Q2.) Explain the concept of event delegation and provide a scenario where it is beneficial.

**Event Delegation** is a technique in JavaScript where you add a single event listener to a parent element to manage events for its child elements, even if those child elements are dynamically added to the DOM. This leverages the concept of **event propagation** (specifically, **event bubbling**), where events triggered on child elements propagate up to their ancestors.

### Working:

1. When an event occurs on an element, it first goes through the **capture phase** (from the root to the target element), then the **target phase** (on the target element itself), and finally the **bubble phase** (from the target back up to the root).
2. By adding a listener to a common ancestor, you can "delegate" the handling of events for all child elements using the event object's target property to identify the clicked element.

### Use Of Event Delegation:

#### 1. **Improved Performance:**

Instead of attaching event listeners to multiple child elements, you attach one listener to their parent, reducing memory usage and DOM traversal overhead.

#### 2. **Dynamic Content Handling:**

Event delegation works well with elements that are added to the DOM dynamically after the page has loaded, as the parent's event listener will still catch events from new children.

**Example:** Imagine a dynamic list where new items are added via JavaScript, and you want to handle a click event on each item.

#### • **Without Event Delegation:**

You need to add a click event listener to each list item manually, which doesn't account for dynamically added items:

```
Code: document.querySelectorAll("li").forEach(item => {  
  
  item.addEventListener("click", () => {  
  
    console.log("List item clicked!");  
  
  });  
  
});
```

- **With Event Delegation:**

You add a single event listener to the parent <ul>:

```
Code: document.querySelector("ul").addEventListener("click", event => {  
  if (event.target.tagName === "LI") {  
    console.log("List item clicked:", event.target.textContent);  
  }  
});
```

### Q3.) Explain the concept of Event Bubbling in the DOM.

Event Bubbling is a fundamental concept in the DOM (Document Object Model) related to how events propagate. When an event is triggered on an element, it "bubbles" up through its ancestors in the DOM hierarchy, starting from the target element and moving up to the `<html>` element (the root of the document).

#### How Event Bubbling Works-

##### 1. Event Triggering:-

- When an event occurs on an element (e.g., a click on a button), the event is first processed on the **target element** where it originated.

##### 2. Bubbling Phase:-

- After being handled on the target, the event propagates upward to its parent element, then to the grandparent, and so on, until it reaches the root of the DOM (the `<html>` element).
- During this phase, any event listeners on the ancestor elements for that event type are triggered.

#### Example:-

##### HTML Structure:

```
<div id="parent">  
  <button id="child">Click Me!</button>  
</div>
```

##### JavaScript:

```
document.getElementById("parent").addEventListener("click", () => {  
  console.log("Parent clicked");  
});  
  
document.getElementById("child").addEventListener("click", () => {  
  console.log("Child clicked");  
});
```

#### **Q4.) Explain the purpose of the addEventListener method in JavaScript and how it facilitates event handling in the DOM.**

The addEventListener method in JavaScript is a powerful and flexible way to handle events in the DOM. It allows you to attach event handlers (functions that execute in response to events) to HTML elements. By using addEventListener, developers can create interactive and dynamic web applications.

##### **Purpose of addEventListener-**

##### **1. Attaching Event Handlers:**

- The primary purpose of addEventListener is to attach event handlers to DOM elements for specific events (like click, mouseover, keydown, etc.).

##### **2. Flexibility and Advanced Options:**

- It allows adding multiple event listeners for the same event on the same element.
- It supports options like controlling the event's capture or bubble phase.

##### **3. Dynamic and Modern Event Handling:**

- Unlike older methods like onclick, it adheres to modern standards and provides better compatibility and flexibility.

Code: element.addEventListener(event, listener, options);

Example:-

##### **1. Html code:**

```
<ul id="myList">
  <li>Item 1</li>
  <li>Item 2</li>
</ul>
<button id="addItem">Add Item</button>
```

##### **2. Javascript code:**

```
const list = document.getElementById("myList");
const addItem = document.getElementById("addItem");
// Event delegation
list.addEventListener("click", (event) => {
  if (event.target.tagName === "LI") {
    console.log("Clicked:", event.target.textContent);
  }
});
// Add new list items dynamically
```

```
addItem.addEventListener("click", () => {  
  const newItem = document.createElement("li");  
  newItem.textContent = `Item ${list.children.length + 1}`;  
  list.appendChild(newItem);  
});
```

**Q5.) Create an HTML page with a button. Use JavaScript to display an alert when the button is clicked.**

The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays a file tree for a project named 'PW Skills Full Stack Development Aug'24'. The file 'Answers5.html' is selected. The main editor area shows the content of 'Answers5.html', which is an HTML document with a button and JavaScript code. The code includes a meta charset, viewport, and title, followed by a button with the text 'Click Me!'. A JavaScript script is attached to the button's click event, displaying an alert with the message 'Button was clicked!'. The terminal panel at the bottom shows the command prompt output, indicating the current directory is '/e/GitHub Repo/PW Skills Full Stack Development Aug'24'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Button Click Alert</title>
7 </head>
8 <body>
9   <h1>Button Click Example</h1>
10  <button id="alertButton">Click Me</button>
11
12  <script>
13    // Select the button element
14    const button = document.getElementById("alertButton");
15
16    // Attach a click event listener
17    button.addEventListener("click", () => {
18      // Display an alert when the button is clicked
19      alert("Button was clicked!");
20    });
21  </script>
22 </body>
23 </html>
24
```

```
Chirag@DESKTOP-LHWAH8 MINGW64 /e/GitHub Repo (main)
$ ls -la
./ .git/ .github/ .java/ .java-projects/ .project [Bean & Brew Coffee]/ .project [FirstFlight Travels]/ .project [Reactive Resume ]/ .project [bookshelf Emporium]/ .project [React DevOps and Cloud Computing Aug'24]/ .project [Web Development Projects]/
./ .vscode/ .java-projects/ .java-projects/ .project [bookshelf Emporium]/ .project [Reactive Resume ]/ .project [Web Development Projects]/

Chirag@DESKTOP-LHWAH8 MINGW64 /e/GitHub Repo (main)
$ cd "PW Skills Full Stack Development Aug'24"

Chirag@DESKTOP-LHWAH8 MINGW64 /e/GitHub Repo/PW Skills Full Stack Development Aug'24 (main)
$
```

### Button Click Example

Click Me!

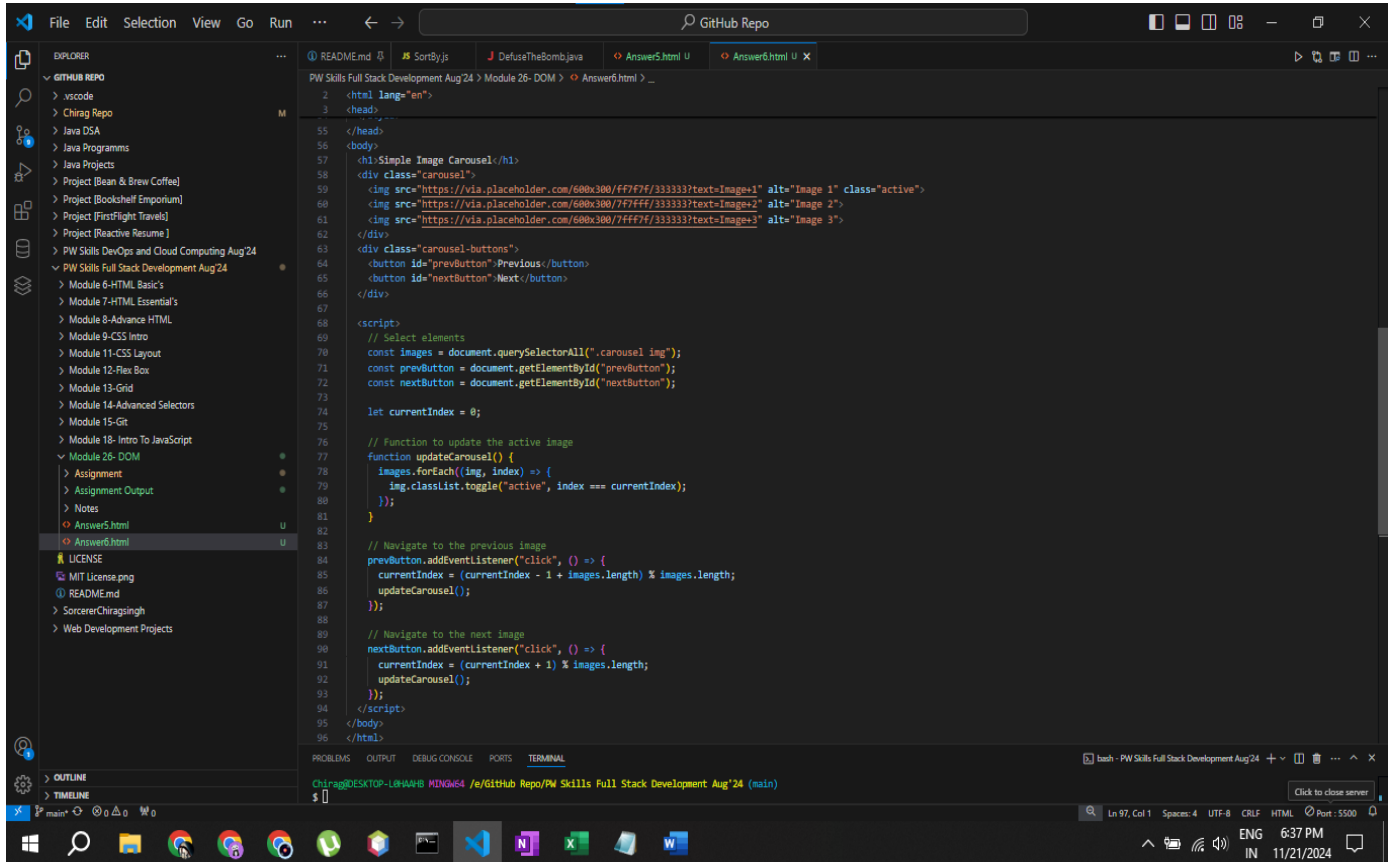
127.0.0.1:5500 says

Button was clicked!

OK



**Q6.) Create a simple image carousel using HTML and JavaScript, Design a basic HTML structure with images, and use JavaScript to implement functionality that allows users to navigate through the images.**



The screenshot shows a VS Code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'GitHub Repo' folder containing 'README.md', 'SortBy.js', 'DefuseTheBomb.java', 'Answer5.html', and 'Answer6.html'. The code editor displays the content of 'Answer6.html', which is a simple image carousel. The HTML structure includes a header, a main content area with three placeholder images, and a footer with navigation buttons. The JavaScript code implements the carousel functionality, including a function to update the active image and event listeners for the navigation buttons.

```
2 <html lang="en">
3 <head>
4 <title>Simple Image Carousel</title>
5 </head>
6 <body>
7 <h1>Simple Image Carousel</h1>
8 <div class="carousel">
9 
10 
11 
12 </div>
13 <div class="carousel-buttons">
14 <button id="prevButton">Previous</button>
15 <button id="nextButton">Next</button>
16 </div>
17 </body>
18 </html>
19
20 <script>
21 // Select elements
22 const images = document.querySelectorAll(".carousel img");
23 const prevButton = document.getElementById("prevButton");
24 const nextButton = document.getElementById("nextButton");
25
26 let currentIndex = 0;
27
28 // Function to update the active image
29 function updateCarousel() {
30   images.forEach((img, index) => {
31     img.classList.toggle("active", index === currentIndex);
32   });
33 }
34
35 // Navigate to the previous image
36 prevButton.addEventListener("click", () => {
37   currentIndex = (currentIndex - 1 + images.length) % images.length;
38   updateCarousel();
39 });
40
41 // Navigate to the next image
42 nextButton.addEventListener("click", () => {
43   currentIndex = (currentIndex + 1) % images.length;
44   updateCarousel();
45 });
46 </script>
47 </body>
48 </html>
```

### Simple Image Carousel

Image 1

Previous

Next

### Simple Image Carousel

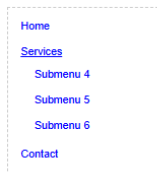
Image 2

Previous

Next

**Q7.) Build a dynamic dropdown menu using HTML and JavaScript. Create an HTML structure for a navigation menu with dropdowns. Use JavaScript to toggle the visibility of dropdowns when the user hovers over menu items.**

```
58 <html lang="en">
59 <body>
60 <ul class="menu">
61 <li>
62 <ul class="submenu">
63 <li><a href="#">Submenu 4</a></li>
64 <li><a href="#">Submenu 5</a></li>
65 <li><a href="#">Submenu 6</a></li>
66 </ul>
67 </li>
68 <li><a href="#">Contact</a>
69 </li>
70 </ul>
71 </body>
72 </html>
73
74 <script>
75 // Select all elements with submenu
76 const menuItems = document.querySelectorAll(".has-submenu");
77
78 menuItems.forEach(item => {
79   const submenu = item.nextElementSibling;
80
81   // Show submenu on mouseenter
82   item.addEventListener("mouseenter", () => {
83     submenu.style.display = "block";
84   });
85
86   // Hide submenu on mouseleave
87   item.addEventListener("mouseleave", () => {
88     submenu.style.display = "none";
89   });
90
91   // Add the same functionality for submenu hover
92   submenu.addEventListener("mouseenter", () => {
93     submenu.style.display = "block";
94   });
95
96   submenu.addEventListener("mouseleave", () => {
97     submenu.style.display = "none";
98   });
99
100 </script>
101 </body>
102 </html>
```



## Q8. Create a simple dynamic shopping list with the following features

- The item should appear in the list.
- Each item should be given a button that can be pressed to delete that item off the list.
- The input should be emptied and focused ready for you to enter another item.

Shopping List

Add Item

egg

Delete

milk

Delete

bread

Delete

The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'PW Skills Full Stack Development Aug'24' folder. The code editor shows the 'Answer8.html' file, which contains the following code:

```
1 <html lang="en">
2 <body>
3   <script>
4     function addItem() {
5       alert("Please enter an item.");
6       return;
7     }
8
9     // Create a new list item
10    const listItem = document.createElement("li");
11
12    // Add the item text
13    listItem.textContent = itemText;
14
15    // Create a delete button
16    const deleteButton = document.createElement("button");
17    deleteButton.textContent = "Delete";
18    deleteButton.addEventListener("click", () => {
19      shoppingList.removeChild(listItem);
20    });
21
22    // Append the delete button to the list item
23    listItem.appendChild(deleteButton);
24
25    // Add the list item to the shopping list
26    shoppingList.appendChild(listItem);
27
28    // Clear the input and focus it
29    itemInput.value = "";
30    itemInput.focus();
31  }
32
33  // Add item on button click
34  addItemButton.addEventListener("click", addItem);
35
36  // Add item on Enter key press
37  itemInput.addEventListener("keypress", (event) => {
38    if (event.key === "Enter") {
39      addItem();
40    }
41  });
42 </script>
43 </body>
44 </html>
```

The terminal at the bottom shows the command prompt with the path 'C:\Users\DESKTOP-LHWAWE\minkm4\e\Github Repo\PW Skills Full Stack Development Aug'24 (main)' and the prompt '\$ '.