Guía de Trabajo: DOM en JavaScript

Objetivo

El objetivo del DOM es proporcionar un modelo estructurado de los documentos para que los lenguajes de programación puedan acceder, modificar y actualizar la estructura y el contenido de una página web en tiempo real.

Introducción al DOM

El DOM (Document Object Model) es una interfaz de programación que permite manipular documentos HTML y XML. Cuando un navegador carga una página web, crea una representación del documento en forma de árbol jerárquico, donde cada elemento, atributo o contenido de texto es un nodo.

---

1. Qué es el DOM

El DOM en Javascript se utiliza para interactuar con los elementos de una página web de manera dinámica. Es decir, que los desarrolladores pueden acceder a los elementos HTML, modificar su contenido, estilo y atributos, pero también agregar o eliminar elementos cuando sea necesario en la maquetación web

1.1. Componentes clave del DOM

Concepto	Descripción
Nodo	Representa cualquier parte del documento: etiquetas, texto, atributos, etc.
Elemento	Nodo que representa una etiqueta HTML (`<div>` , `<p>` , `<h1>` , etc.).
Propiedad	Información asociada al nodo, como `id` , `class` , o `innerHTML` .
Método	Acción que se puede realizar en un nodo, como `getElementById()` o `appendChild()` .

1.2. **El DOM permite**:

- Leer, modificar o eliminar elementos HTML.
- Cambiar el estilo de los elementos.
- Responder a eventos como clics y teclas presionadas.

Ejemplo 1

2. Seleccionar elementos en el DOM

Los métodos de selección permiten acceder a nodos específicos del árbol DOM. A continuación, una tabla con los métodos más comunes:

Método	Descripción	Ejemplo
`getElementById`	Selecciona un elemento por su ID.	`document.getElementById('myId')`
`getElementsByClassName`	Selecciona elementos por su clase. Devuelve una colección.	`document.getElementsByClassName('myClass')`
`getElementsByTagName`	Selecciona elementos por su etiqueta (`div` , `p` , etc.).	`document.getElementsByTagName('div')`
`querySelector`	Selecciona el primer elemento que coincide con un selector CSS.	`document.querySelector('.myClass')`
`querySelectorAll`	Selecciona todos los elementos que coincidan con un selector CSS.	`document.querySelectorAll('p')`

2.1. Principales diferencias

Método	Selector Aceptado	Devuelve	Soporta Selectores CSS Complejos
`querySelector`	ID, clase, etiqueta, combinaciones	Nodo único	Sí
`querySelectorAll`	ID, clase, etiqueta, combinaciones	NodeList (similar a arreglo)	Sí
`getElementById`	ID	Nodo único	No
`getElementsByClassName`	Clase	HTMLCollection	No
`getElementsByTagName`	Etiqueta	HTMLCollection	No

2.2. ¿Cuál usar y cuándo?

querySelector y querySelectorAll:

- Úsalos cuando necesites flexibilidad o seleccionar elementos usando selectores CSS avanzados.
- Ejemplo: clase > #id, div:nth-child(2), [data-atributo="valor"]

getElementById:

- Úsalo para una selección rápida si ya sabes el id único de un elemento.

getElementsByClassName:

- Ideal si necesitas seleccionar muchos elementos de la misma clase y no te importa usar una colección dinámica.

getElementsByTagName:

- Útil cuando necesitas trabajar con todas las etiquetas de un tipo específico (como todas las img o p).

Ejemplo 2

3: Manipulación del DOM

Teoría:

Los métodos de manipulación permiten cambiar el contenido, estilo o estructura de un documento.

Propiedad/Método	Acción	Ejemplo
`innerHTML`	Cambia el contenido HTML de un elemento.	`element.innerHTML = '<b>Hello</b>'`
`textContent`	Cambia solo el texto de un elemento (sin interpretar HTML).	`element.textContent = 'Hello'`
`style`	Modifica el estilo CSS en línea de un elemento.	`element.style.color = 'red'`

`removeChild()`	Elimina un nodo hijo del nodo padre.	`javascript const ultimoElemento = lista.lastElementChild; lista.removeChild(ultimoElemento);`
`appendChild()`	Agrega un nodo al final de la lista de hijos de un nodo especificado.	`javascript const nuevoElemento = document.createElement('li'); nuevoElemento.textContent = "Nuevo Elemento"; lista.appendChild(nuevoElemento);`

Ejemplo 3

4. Eventos en el DOM

Teoría:

Los eventos son acciones realizadas por el usuario o el navegador (clics, teclas presionadas, etc.). A través de JavaScript, podemos responder a estos eventos usando addEventListener.

Evento	Descripción
`click`	Ocurre cuando se hace clic en un elemento.
`mouseover`	Ocurre cuando el puntero pasa sobre un elemento.
`keydown`	Ocurre cuando se presiona una tecla.
`submit`	Ocurre cuando se envía un formulario.

Conclusión

En resumen, el DOM es una herramienta poderosa que permite a los desarrolladores web transformar documentos estáticos en experiencias dinámicas e interactivas. Su conocimiento es fundamental para construir aplicaciones modernas y optimizadas.