



BOOTCAMP

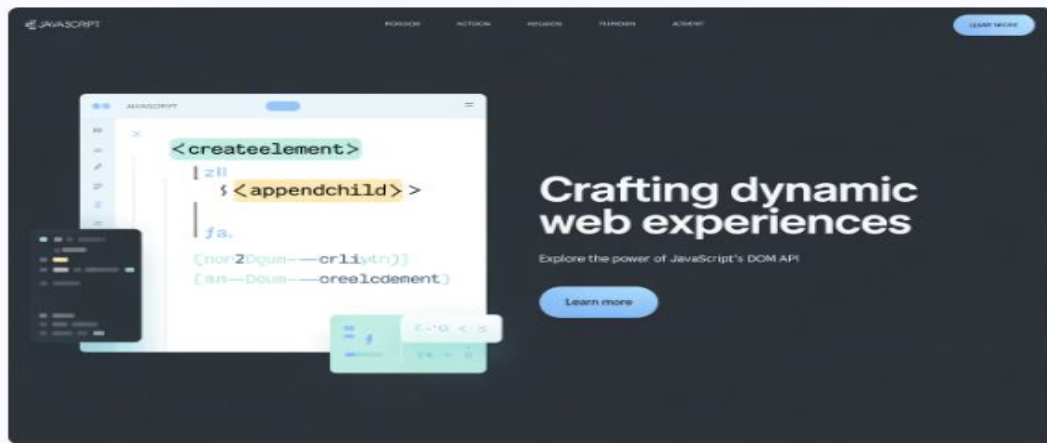
Generación T

 streambe

Crear Nodos en el DOM

JavaScript ofrece métodos específicos para crear diferentes tipos de nodos que luego pueden insertarse en el documento.

- Elementos: utilizando **document.createElement()**
- Texto: mediante **document.createTextNode()**
- Los nodos creados deben añadirse al DOM con métodos como **appendChild()**



```
// Crear un nuevo elemento párrafo
const parrafo = document.createElement('p');

// Crear un nodo de texto
const texto = document.createTextNode('Hola DOM!');

// Añadir el texto al párrafo
parrafo.appendChild(texto);

// Añadir el párrafo al body
document.body.appendChild(parrafo);
```

Manipular El Texto Y Contenido

Por lo tanto, si quisiéramos modificar el **contenido de HTML** de un documento, debemos usar `innerHTML`. Este devuelve el texto contenido en las etiquetas HTML. De esta manera, interpretará cada *tag* y lo traducirá en la pantalla.

```
h1.innerHTML = "<p>¡Hola!<p>"
```

En síntesis, con la Propiedad `textContent` podemos modificar todos los elementos o tags HTML que muestren texto. Con `innerHTML`, podemos modificar el contenido del tag; por ejemplo, reemplazándolo por otro tag.



Innet HTML

innerHTML permite definir el **código html interno** del elemento seleccionado. El navegador lo interpreta como código HTML y no como contenido de texto, permitiendo desde un string crear una nueva estructura de etiquetas y contenido.

Innet HTML

Al pasar un string con formato de etiquetas html y contenido a través de la propiedad innerHTML, el navegador **genera nuevos nodos con su contenido** dentro del elemento seleccionado.

Innet HTML

```
//CODIGO HTML DE REFERENCIA
<div id="contenedor"></div>

//CODIGO JS
let container = document.getElementById("contenedor")
// cambio el código HTML interno
container.innerHTML = "<h2>Hola mundo!</h2><p>Lorem ipsum</p>"

//Resultado en el DOM
<div id="contenedor">
  <h2>Hola mundo!</h2>
  <p>Lorem ipsum</p>
</div>
```

Innet Text

La propiedad **innerText** de un nodo nos permite modificar su nodo de texto. Es decir, acceder y/o modificar el contenido textual de algún elemento del DOM.

```
//CODIGO HTML DE REFERENCIA
<h1 id="titulo">Hola Mundo!</h1>

//CODIGO JS
let titulo = document.getElementById("titulo")
console.log( titulo.innerText ) // "Hola Mundo!"

// cambio el contenido del elemento
titulo.innerText = "Hola Coder!"
console.log( titulo.innerText ) // "Hola Coder!"
```


Class Name

A través de la propiedad **className** de algún nodo seleccionado podemos acceder al atributo **class** del mismo y definir cuáles van a ser sus clases:


```
//CODIGO HTML DE REFERENCIA
<div id="contenedor"></div>

//CODIGO JS
let container = document.getElementById("contenedor")
// cambio el código HTML interno
container.innerHTML = "<h2>Hola mundo!</h2>"
// cambio el atributo class
container.className = "container row"
//Resultado en el DOM
<div id="contenedor" class="container row">
  <h2>Hola mundo!</h2>
</div>
```

Agregar o quitar Nodos

Crear Elementos

Para crear diferentes elementos y luego añadirlos a un documento, debemos usar el método `createElement`.

 **Importante:** Con `createElement` se crea un Objeto que luego agregaremos al DOM.

Veamos su sintaxis:

```
let elemento = document.createElement("elemento") // Pasamos un String como parámetro, que i
```

Ahora veamos un ejemplo. Si quisiéramos crear una etiqueta `h1`, primero debemos guardar su referencia en una Variable:

```
let h1 = document.createElement("h1")
```

Luego, le podemos agregar texto usando `textContent`:

```
h1.textContent = "Plataforma 5"
```

Por último, podemos crear un elemento `div`:

```
let div = document.createElement("div")
```

Creación de elementos

Para crear elementos se utiliza la función `document.createElement()`, y se debe indicar el nombre de etiqueta HTML que representará ese elemento.

Luego debe agregarse como hijo el nodo creado con `append()`, al body o a otro nodo del documento actual.

```
// Crear nodo de tipo Elemento, etiqueta p
let parrafo = document.createElement("p");
// Insertar HTML interno
parrafo.innerHTML = "<h2>¡Hola Coder!</h2>";
// Añadir el nodo Element como hijo de body
document.body.append(parrafo);
```

Eliminar elementos

Se pueden eliminar nodos existentes y nuevos. El método `remove()` permite eliminar un nodo seleccionado del DOM:

```
let parrafo = document.getElementById("parrafo1");  
//Elminando el propio elemento  
parrafo.remove();  
  
let paises = document.getElementsByClassName("paises");  
//Eliminando el primer elemento de clase paises  
paises[0].remove();
```

Obtener datos de Inputs

Para obtener o modificar datos de un formulario HTML desde JS, podemos hacerlo mediante el DOM. Accediendo a la propiedad `value` de cada input seleccionado:

```
//CODIGO HTML DE REFERENCIA
```

```
<input id = "nombre" type="text">
```

```
<input id = "edad" type="number">
```

```
//CODIGO JS
```

```
document.getElementById("nombre").value = "HOMERO";
```

```
document.getElementById("edad").value = 39;
```

Ejemplo aplicado:

Creando opciones desde un Array

```
//Obtenemos el nodo donde vamos a agregar los nuevos elementos
let padre = document.getElementById("personas");
//Array con la información a agregar
let personas = ["HOMERO","MARGE", "BART", "LISA","MAGGIE"];
//Iteramos el array con for...of
for (const persona of personas) {
    //Creamos un nodo <li> y agregamos al padre en cada ciclo
    let li = document.createElement("li");
    li.innerHTML = persona
    padre.appendChild(li);
}
```

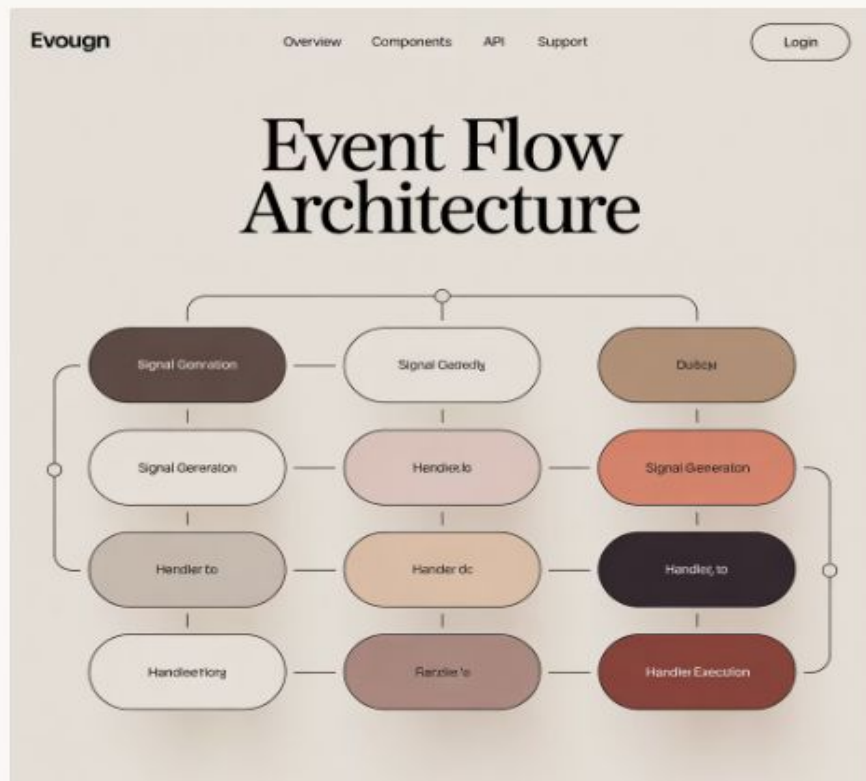
Eventos!

¿Cómo Funcionan los Eventos?

El navegador actúa como un vigilante constante, monitoreando todas las interacciones posibles:

- Genera señales cuando detecta una acción
- Asocia cada acción con un manejador específico
- Ejecuta el código correspondiente automáticamente

Este sistema permite crear interfaces reactivas sin necesidad de recargar la página.



Tipos Comunes de Eventos



Eventos de Ratón

click, dblclick, mouseover, mouseout y mousedown capturan diferentes interacciones del puntero con elementos de la página.



Eventos de Teclado

keydown, keyup y keypress permiten responder a las pulsaciones de teclas por parte del usuario.



Eventos de Formulario

submit, change y focus controlan las interacciones con formularios y campos de entrada.



Eventos de Ventana

load, resize y scroll detectan cambios en la ventana del navegador o la carga de la página.

Tipos De Eventos

Eventos Más Comunes

Evento	Descripción
<code>onchange</code>	Modificación de un elemento HTML
<code>onclick</code>	Usuario hace <i>click</i> sobre un elemento HTML
<code>mouseover</code>	Usuario mueve el <i>mouse</i> sobre un elemento HTML
<code>mouseout</code>	Usuario quita el <i>mouse</i> del elemento HTML
<code>keydown</code>	Usuario presiona una tecla
<code>load</code>	<i>Browser</i> termina de cargar la página

Detectores y Manejadores de Eventos

Conceptos Clave

- **Detectores:** Observa si ocurre un evento específico
- **Manejador:** Función que se ejecuta cuando se detecta el evento
- **addEventListener:** Método para conectar eventos con funciones



```
// Sintaxis básica
elemento.addEventListener('evento', función);

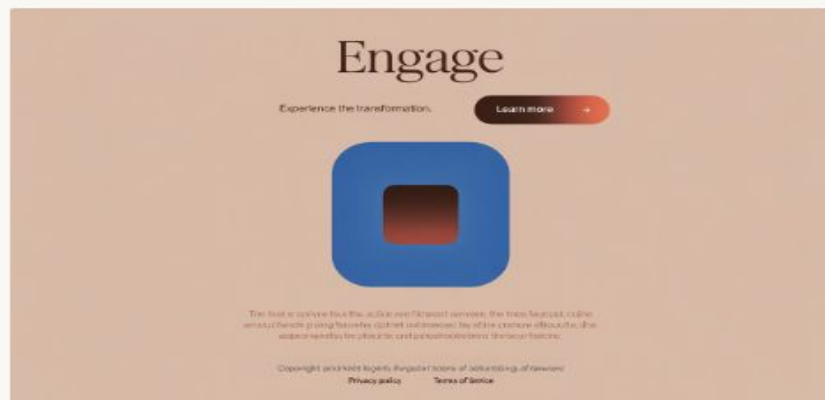
// Ejemplo práctico
button.addEventListener('click', function() {
  console.log('¡Botón pulsado!');
});
```

Ejemplo Práctico: Click en un Botón

```
// HTML
<button id="miBoton">Cambiar Color</button>
<div id="cuadrado"></div>

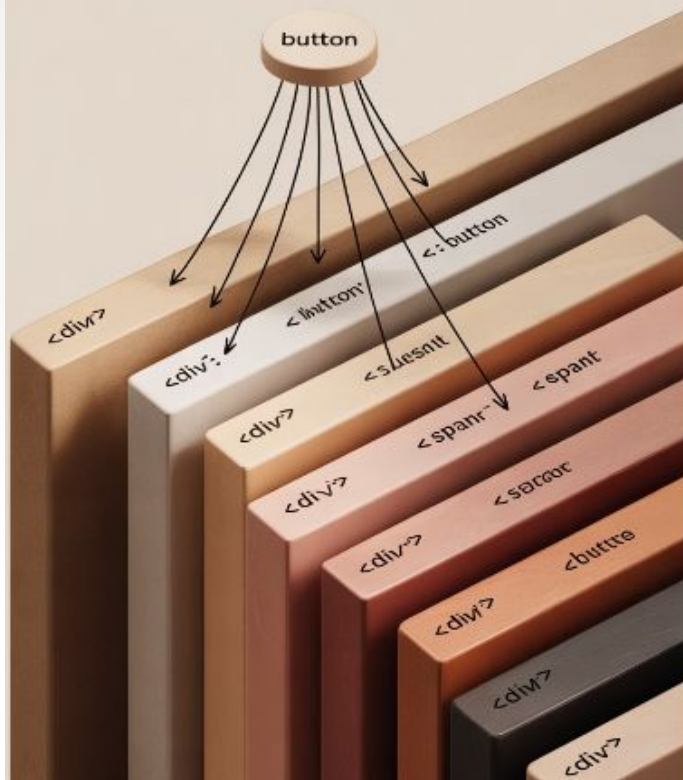
// CSS
#cuadrado {
  width: 100px;
  height: 100px;
  background-color: blue;
}

// JavaScript
document.getElementById('miBoton')
  .addEventListener('click', function() {
    const cuadrado =
      document.getElementById('cuadrado');
    cuadrado.style.backgroundColor = 'red';
  });
```



Este ejemplo muestra cómo un evento simple puede crear interactividad instantánea, cambiando propiedades visuales sin recargar la página.

Event bubbling



Burbujeo de Eventos (Event Bubbling)



Propagación Ascendente

Un evento se dispara primero en el elemento más específico (p. ej., un botón)

Ascenso por el DOM

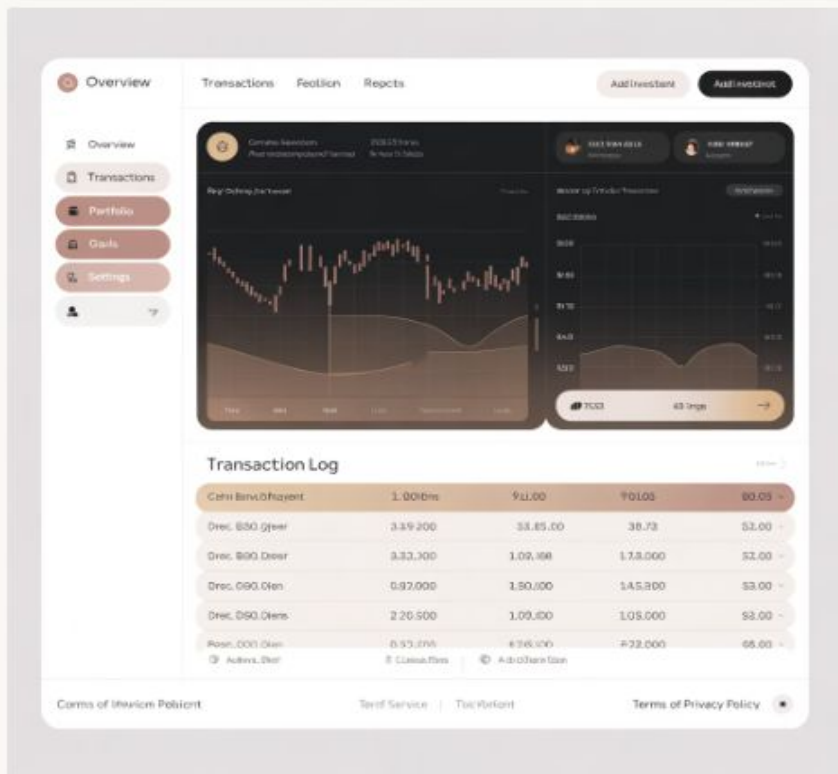
Luego se propaga hacia arriba, activando manejadores en elementos padres



Control con stopPropagation()

Podemos detener este burbujeo usando `event.stopPropagation()` si es necesario

¿Por Qué Son Importantes los Eventos?



Experiencias Interactivas

Transforman sitios estáticos en aplicaciones dinámicas que responden instantáneamente

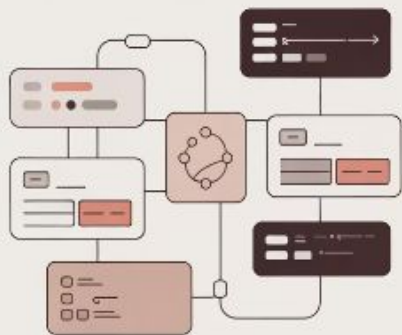
Interfaz Receptiva

Permiten feedback inmediato sin recargas de página completa

Arquitectura Moderna

Son fundamentales en frameworks como React, Vue y Angular

Orchestrate your interactions



Real-time updates

Real-time updates are essential for modern web applications. They allow users to see changes as they happen, improving the overall user experience.

Enjoy its free



Customizable events

Customizable events allow you to track specific user actions and trigger personalized responses. This helps you understand user behavior and optimize your application.

Enjoy its free



Simplified debugging

Simplified debugging makes it easier to identify and fix issues in your application. It provides clear logs and error messages, helping you quickly resolve problems.

Don't let it be

Resumen



Señales Digitales

Los eventos son notificaciones que indican que algo relevante ha ocurrido en el navegador



Código Reactivo

Permiten ejecutar funciones específicas en respuesta a acciones del usuario o del sistema



Web Interactiva

Son esenciales para crear experiencias web modernas, fluidas y centradas en el usuario

¡Dominar los eventos es crucial para cualquier desarrollador frontend!



Muchas gracias.

Nuestras Redes

www.generaciont.org

www.streambe.com

www.instagram.com/generaciont_ar

www.tiktok.com/@generaciont

generaciont@generaciont.org

Cel: [11 61331747](tel:1161331747)



[11 6133-1747](tel:1161331747)



GENERACIÓN T

generaciont@generaciont.org



GENERACIÓN T