IES GASPAR MELCHOR DE Jovellanos
Calle Móstoles, 64
28941 Fuenlabrada
Madrid
Teléfono: 916971565

Comunidad de Madrid
Consejería de educación e investigación

MEMORIA DE AMPLIACIÓN: UNIDAD 06 (DOM Y EVENTOS)

2ºDaw

Sergio Sánchez Plaza

Miguel Sánchez Martín

➢ BLOQUE A: MANIPULACIÓN AVANZADA DEL DOM (RA4)

1. Elementos del DOM manipulados

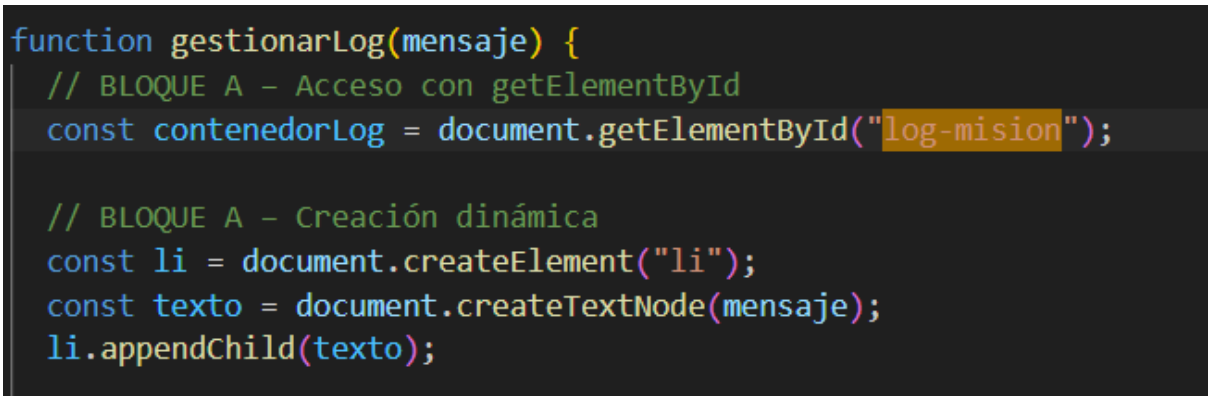
En esta ampliación de la practica del RA4-RA5 se han manipulados elementos del DOM mediante los siguientes métodos:

getElementById:

Se utiliza ampliamente para acceder a elementos clave del juego:

- #log-mision (contenedor del historial de mensajes)
- #nave-status (estado de plasma y casco)
- #nave-visual (imagen de la nave)
- #relleno-integridad y #relleno-combustible (barras de progreso)
- #btn-reiniciar-juego (botón de reinicio)
- #seccion-inventario-dinamica y #game-over-dinamico (elementos creados dinámicamente)

  Ejemplo de uso:



```
function gestionarLog(mensaje) {
  // BLOQUE A – Acceso con getElementById
  const contenedorLog = document.getElementById("log-mision");

  // BLOQUE A – Creación dinámica
  const li = document.createElement("li");
  const texto = document.createTextNode(mensaje);
  li.appendChild(texto);
```

getElementsByClassName

- Se emplea para acceder a todos los botones de destino galáctico (.btn-viaje) y desactivarlos al finalizar la partida.

Ejemplo de uso:

```
// Desactivar botones viaje - getElementsByClassName
const viajes = document.getElementsByClassName("btn-viaje");
for (let i = 0; i < viajes.length; i++) {
  viajes[i].disabled = true;
}
```

getElementsByName:
Se ha usado ese método en el formulario de diagnóstico para acceder a:

- Todos los radio buttons de la pregunta 1 (name="p1") → determinar cuál está seleccionado.

- Todos los checkboxes de la pregunta 3 (name="p3") → contar cuántos correctos están marcados

Ejemplo de uso

```
const radiosP1 = document.getElementsByName("p1");
let q1 = null;
for (let radio of radiosP1) {
  if (radio.checked) {
    q1 = radio.value;
    break;
  }
}
```

2. **Creación dinámica de contenido:**

En esta ampliación se ha implementado la creación dinámica de varios elementos nuevos en el DOM, utilizando exclusivamente métodos DOM puros (createElement, createTextNode, appendChild e insertBefore), sin recurrir a innerHTML para estos contenidos generados en tiempo de ejecución.

En la sección del juego (juego.js)

** **

- Método: document.createElement("div")
- Propósito: Contenedor que muestra el inventario (chatarra y células) y el botón de usar suministros.
- Inserción: contenedorLog.parentNode.insertBefore(seccionInventario, contenedorLog)
- Se crea solo la primera vez que se llama a gestionarLog() y luego se reutiliza.

```
let seccionInventario = document.getElementById(
  "seccion-inventario-dinamica",
);

if (!seccionInventario) {
  seccionInventario = document.createElement("div");
  seccionInventario.id = "seccion-inventario-dinamica";
  seccionInventario.setAttribute(
    "style",
    "margin:12px 0; padding:10px; background:rgba(30,50,80,0.4); border-radius:6px; border:1px solid #4db8ff33;",
  );

  contenedorLog.parentNode.insertBefore(seccionInventario, contenedorLog);
}
```

**** (dentro del inventario)

- Método: document.createElement("strong")
- Propósito: Título en negrita "📦 INVENTARIO:"
- Inserción: seccionInventario.appendChild(strong)

```
const strong = document.createElement("strong");
strong.setAttribute("style", "color:#4db8ff;");
strong.appendChild(document.createTextNode("📦 INVENTARIO: "));
seccionInventario.appendChild(strong);
```

**
** (dentro del inventario)

- Método: document.createElement("br")
- Propósito: Salto de línea entre el título y los valores de inventario
- Inserción: seccionInventario.appendChild(br)

```
const br = document.createElement("br");
seccionInventario.appendChild(br);
```

Nodo de texto con valores de inventario

- Método: document.createTextNode(...)
- Propósito: Muestra "🛠️ Chatarra: X 🔋 Células: Y"
- Inserción: seccionInventario.appendChild(txtInv)

```
const txtInv = document.createTextNode(
  `🛠️ Chatarra: ${miNave.inventario.chatarra}   🔋 Células: ${miNave.inventario.celulas}  `,
);
seccionInventario.appendChild(txtInv);
```

<button id="btn-usar-recursos-din">

- Método: document.createElement("button")
- Propósito: Botón "USAR SUMINISTROS" para aplicar reparaciones y recargas

- Inserción: seccionInventario.appendChild(btnUsar)

```
const btnUsar = document.createElement("button");
btnUsar.id = "btn-usar-recursos-din";
btnUsar.appendChild(document.createTextNode("USAR SUMINISTROS"));
```

<li> (cada mensaje del log)

- Método: document.createElement("li")
- Propósito: Cada entrada nueva del historial de misiones (éxitos, daños, eventos, etc.)
- Inserción: contenedorLog.insertBefore(li, contenedorLog.firstChild) → aparece arriba

```
// BLOQUE A – Creación dinámica
const li = document.createElement("li");
const texto = document.createTextNode(mensaje);
li.appendChild(texto);
```

Nodo de texto dentro de cada <li> del log

- Método: document.createTextNode(mensaje)
- Propósito: Contenido del mensaje (ej: "🚀 SALTO: Viajando a Orión...")
- Inserción: li.appendChild(texto)

** **

- Método: document.createElement("div")
- Propósito: Pantalla superpuesta grande "GAME OVER" cuando la nave se destruye
- Inserción: contenedor.appendChild(div)

```
// BLOQUE A – Creación dinámica de mensaje GAME OVER (sin innerHTML)
let contenedor = document.querySelector("#game-container");
let existe = document.getElementById("game-over-dinamico");

if (!existe) {
  const div = document.createElement("div");
  div.id = "game-over-dinamico";
```

Nodo de texto "GAME OVER"

- Método: document.createTextNode("GAME OVER")
- Propósito: Texto principal de la pantalla de fin de partida
- Inserción: div.appendChild(texto)

```
const texto = document.createTextNode("GAME OVER");
div.appendChild(texto);

contenedor.appendChild(div);
```

En la sección del formulario (formulario.js – versión modificada)

<h3> (dentro de #resultado-final)

- Método: document.createElement("h3")
- Propósito: Título “INFORME DE DIAGNÓSTICO” en el resultado final
- Inserción: feedbackGlobal.appendChild(h3)

```
const h3 = document.createElement("h3");
h3.textContent = "INFORME DE DIAGNÓSTICO";
feedbackGlobal.appendChild(h3);
```

<p> (dentro de #resultado-final)

- Método: document.createElement("p")
- Propósito: Muestra el porcentaje y estado (APTO / NO APTO)
- Inserción: feedbackGlobal.appendChild(p)

```
const p = document.createElement("p");
p.textContent = `Estado: ${porcentaje}% (${aprobado ? "APTO" : "NO APTO"})`;
feedbackGlobal.appendChild(p);
```

Nodo de texto dentro del <h3> y <p>

- Método: .textContent = ... (equivalente a crear nodo texto e insertarlo)
- Propósito: Contenido dinámico del informe (“INFORME DE DIAGNÓSTICO”, “Estado: XX% (...)”)

Todos estos elementos se generan únicamente cuando son necesarios (primera llamada a gestionarLog, primer envío del formulario, game over, etc.), se insertan en posiciones precisas del DOM y se eliminan/reconstruyen cuando corresponde (cumpliendo también el punto 4 de eliminación dinámica).

3. Modificación dinámica del DOM:

En esta ampliación se ha realizado una modificación dinámica intensiva del DOM, siempre utilizando métodos puros y evitando innerHTML para los contenidos generados en tiempo de ejecución. Las modificaciones afectan tanto al simulador de exploración (sección de juego) como al formulario de diagnóstico, generando feedback visual inmediato, actualizando estados en tiempo real y mostrando resultados finales de forma dinámica.

Las operaciones de modificación se dividen en los siguientes apartados obligatorios:

Modificar contenido textual accediendo a nodos de tipo texto (nodeValue)

- Se modifica el contenido textual directamente accediendo al nodo de texto hijo (sin recrear el elemento):
    - Elemento **#nave-status** (estado de la nave en el juego): se actualiza el texto con el valor actual de plasma y casco mediante firstChild.nodeValue = ... Ejemplo: “Plasma: 7200 | Casco: 80%”. Esta técnica se aplica cada vez que se llama a actualizarInterfaz() (después de un viaje, reparación, evento aleatorio, etc.).
- **Añadir y modificar atributos usando:**
    - **setAttribute** Se utiliza ampliamente para añadir o cambiar atributos en múltiples elementos:
        - **#nave-visual** (imagen de la nave): se modifica src (cambio de imagen: rick.jpg → damaged.jpg → estrellado.jpg/estrellado1.jpg) y style (filtros, rotación, sombras).

```
const imagenNave = document.getElementById("nave-visual");
if (!imagenNave) return;

const juegoTerminado = miNave.integridad <= 0;

if (juegoTerminado) {
  // Elección aleatoria de imagen de destrucción
  const usarPrimera = Math.random() > 0.5;
  imagenNave.setAttribute(
    "src",
    usarPrimera ? "assets/estrellado.jpg" : "assets/estrellado1.jpg",
  );

  // Modificación de atributos estilo
  imagenNave.setAttribute(
    "style",
    "transform: rotate(25deg); filter: grayscale(1) sepia(0.5);",
  );
```

        - **#relleno-integridad** y **#relleno-combustible** (barras de progreso): se actualiza el atributo style con width: XX%.

```
if (vida) vida.setAttribute("style", `width:${miNave.integridad}%`)

const energia = document.getElementById("relleno-combustible");
if (energia) {
  let porc = (miNave.combustible / miNave.combustibleMax) * 100;
  energia.setAttribute("style", `width:${porc}%`);
}
```

        - Contenedores dinámicos (**#seccion-inventario-dinamica**, **#game-over-dinamico**): se establece style para posición, fondo, borde, sombra, z-index, etc.

```
div.id = "game-over-dinamico";
div.setAttribute(
  "style",
  "position:absolute; top:30%; left:50%; transform:translate(-50%,-50%); background:rgba(200,0,0,0.85);
);
```

- Botones (.btn-viaje, #btn-reiniciar-juego, #btn-usar-recursos-din): se añade o modifica disabled, style (brillo en mouseover, márgenes, padding).

```
const btnUsar = document.createElement("button");
btnUsar.id = "btn-usar-recursos-din";
btnUsar.appendChild(document.createTextNode("USAR SUMINISTROS"));
btnUsar.setAttribute(
  "style",
  "margin-left:15px; padding:6px 12px; background:#4db8ff; color:black; border:none;
);
```

- Elementos del formulario: se modifica style en bordes y fondos durante validación en tiempo real (ej. borde azul/rojizo en #piloto-nombre).

```
inputNombre.addEventListener("input", (e) => {
  const regexNombre = /^[a-zA-ZÁÉÍÓÚáéíóúñÑ]{3,}$/;
  if (regexNombre.test(e.target.value)) {
  // Estado válido
  inputNombre.setAttribute("style", "border: 2px solid #4db8ff;");
  feedbackNombre.textContent = "✓ ID de Capitán validado.";
  feedbackNombre.setAttribute("style", "color: #4db8ff;");
} else {
  // Estado inválido
  inputNombre.setAttribute("style", "border: 2px solid #ff6f61;");
  feedbackNombre.textContent = "✗ El ID debe contener al menos 3 letras.";
  feedbackNombre.setAttribute("style", "color: #ff6f61;");
}
});
```

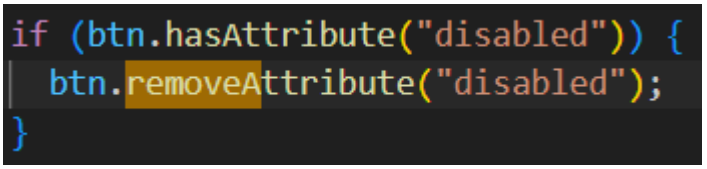
- **getAttribute** Se emplea para leer atributos existentes antes de tomar decisiones:
  - Atributo data-idx de los botones **.btn-viaje**: se lee con getAttribute("data-idx") para saber qué destino galáctico ha seleccionado el usuario.

```
const idx = parseInt(boton.getAttribute("data-idx"));
boton.disabled = true;
```

  - Comprobación de existencia de atributos (ej. hasAttribute("disabled")) antes de habilitar botones al reiniciar.

```
if (btn.hasAttribute("disabled")) {
  btn.removeAttribute("disabled");
}
```

- **removeAttribute** Se utiliza para eliminar atributos específicos cuando ya no son necesarios:
  - Atributo disabled de los botones **.btn-viaje** y #btn-reiniciar-juego al reiniciar la partida (en inicializarJuego()).



Estas modificaciones de atributos permiten:

- Cambios visuales inmediatos (daño en la nave, brillo en botones, barras de progreso)
- Control de interactividad (deshabilitar botones al perder, habilitarlos al reiniciar)
- Adaptación dinámica del interfaz según el estado del juego o formulario

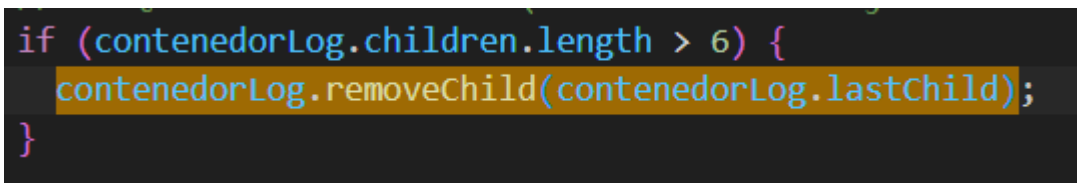
Con estas operaciones se genera feedback visual constante (cambio de imagen de la nave, colores de error/éxito, pantalla de game over, actualización de inventario) y se cumplen los requisitos de modificación dinámica del DOM de forma integrada y natural en el proyecto.

4. Eliminación dinámica de nodos

En esta ampliación se ha implementado la eliminación dinámica de nodos del DOM utilizando exclusivamente el método removeChild, cumpliendo el requisito del Bloque A. Estas eliminaciones se realizan para mantener el interfaz limpio, evitar acumulación innecesaria de elementos y reconstruir secciones dinámicas cuando el estado cambia (por ejemplo, al actualizar inventario, limpiar resultados o reiniciar la partida).

Las eliminaciones dinámicas afectan a los siguientes elementos y se aplican en contextos específicos del juego y del formulario:

- **Eliminación de mensajes antiguos en el log de misiones (#log-mision)**
  - Método: contenedorLog.removeChild(contenedorLog.lastChild)
  - Elementos eliminados: nodos <li> más antiguos del historial de mensajes.
  - Propósito: Limitar el número de mensajes visibles a 6 para evitar que el log se haga excesivamente largo y mantener la legibilidad.
  - Cuándo ocurre: cada vez que se añade un nuevo mensaje en gestionarLog() y se supera el límite.



- **Limpieza completa del contenedor de inventario (#seccion-inventario-dinamica)**

- Método: bucle while (seccionInventario.firstChild) { seccionInventario.removeChild(seccionInventario.firstChild); }
- Elementos eliminados: todos los nodos hijos existentes (,
, nodo de texto con valores de inventario, <button>, etc.).
- Propósito: Reconstruir el inventario desde cero cada vez que se actualiza (después de un viaje, uso de recursos o evento), garantizando que muestre siempre los valores actuales sin duplicados ni residuos.
- Cuándo ocurre: al inicio de cada llamada a gestionarLog() (cuando se actualiza el inventario).

```
while (seccionInventario.firstChild) {
  seccionInventario.removeChild(seccionInventario.firstChild);
}
```

- **Eliminación de la pantalla de GAME OVER (#game-over-dinamico)**
  - Método: gameOver.parentNode.removeChild(gameOver)
  - Elemento eliminado: el completo (incluyendo su nodo de texto hijo “GAME OVER”).
  - Propósito: Quitar la superposición roja de fin de partida al reiniciar el juego, devolviendo el interfaz a su estado inicial limpio.
  - Cuándo ocurre: en inicializarJuego() (al pulsar el botón de reinicio o usar el atajo Enter).

```
const gameOver = document.getElementById("game-over-dinamico");
if (gameOver && gameOver.parentNode) {
  gameOver.parentNode.removeChild(gameOver);
}
```

- **Limpieza del contenedor de resultados del formulario (#resultado-final)**
  - Método: bucle while (feedbackGlobal.firstChild) { feedbackGlobal.removeChild(feedbackGlobal.firstChild); }
  - Elementos eliminados: el <h3> y <p> anteriores del informe de diagnóstico.
  - Propósito: Evitar que se acumulen resultados viejos al enviar el formulario varias veces; se limpia antes de insertar el nuevo informe.
  - Cuándo ocurre: cada vez que se envía el formulario (en el evento submit) y al resetearlo (evento reset).

Estas operaciones de eliminación dinámica permiten:

- Mantener un interfaz limpio y sin elementos residuales (log controlado, inventario siempre actualizado, resultados del formulario frescos).
- Mejorar el rendimiento al no acumular nodos innecesarios.
- Proporcionar una experiencia fluida al reiniciar o actualizar secciones (sin pantallas superpuestas persistentes ni logs infinitos).

Todas las eliminaciones se realizan de forma segura (comprobando existencia del nodo y su padre) y afectan directamente al juego y al formulario, cumpliendo la aplicación práctica

obligatoria del enunciado: mostrar mensajes dinámicos integrados, insertar feedback visual y generar resultados finales dinámicamente.

➢ BLOQUE B: GESTIÓN AVANZADA DE EVENTOS (RA5)

3. Eventos avanzados utilizados

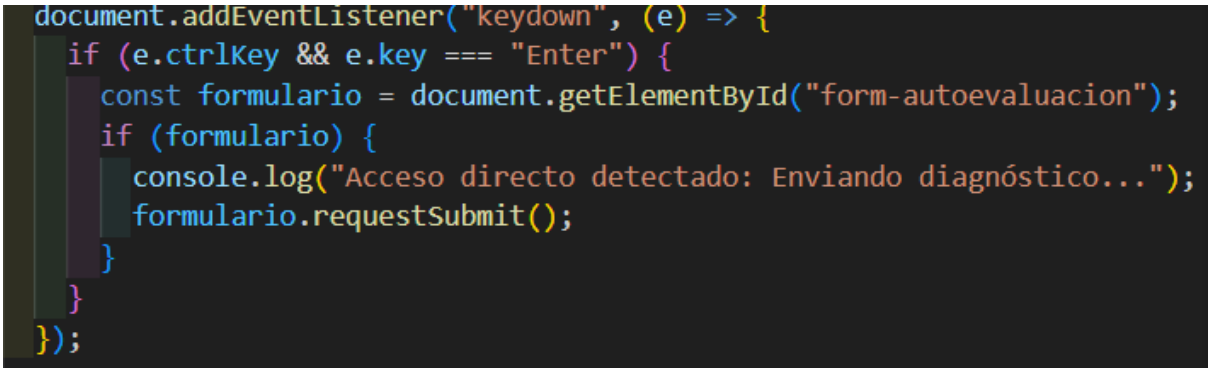
En esta ampliación se ha pasado de un manejo básico de eventos a un sistema más complejo que mejora la experiencia de usuario:

- **Eventos de interfaz y carga (load y DOMContentLoaded)**: Se utiliza DOMContentLoaded en juego.js para iniciar la lógica en cuanto el árbol DOM está listo, sin esperar a imágenes pesadas. Por otro lado, en app.js se emplea load para asegurar que todo el sistema (incluyendo recursos externos) esté disponible antes de activar la navegación.
- **Eventos de ratón (mouseover y mouseout)**: Implementados en los botones de misión. Estos eventos permiten detectar cuándo el usuario sitúa el puntero sobre una opción, aplicando un filtro de brillo dinámico mediante e.target.style.filter, lo que ofrece una interfaz más viva.

```
registrarEvento(boton, "mouseover", function (e) {
  boton.style.filter = "brightness(1.5)";
});

registrarEvento(boton, "mouseout", function () {
  boton.style.filter = "brightness(1)";
});
```

- **Eventos de teclado (keydown)**: Se ha registrado un escuchador en el objeto document para capturar la pulsación de teclas en cualquier parte de la aplicación, permitiendo la creación de atajos globales.



```
document.addEventListener("keydown", (e) => {
  if (e.ctrlKey && e.key === "Enter") {
    const formulario = document.getElementById("form-autoevaluacion");
    if (formulario) {
      console.log("Acceso directo detectado: Enviando diagnóstico...");
      formulario.requestSubmit();
    }
  }
});
```

4. Uso y propiedades del objeto Event

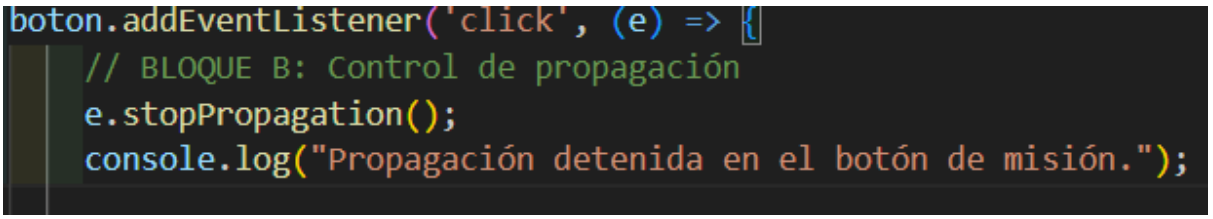
El objeto Event se ha utilizado para extraer información contextual de las interacciones:

- **Propiedades de identificación (target y currentTarget)**:
    - **target:** Usado en las validaciones del formulario para obtener el valor del input específico que cambió.
    - **currentTarget:** Fundamental en la navegación de app.js para asegurar que, aunque el usuario pulse en un icono interno, siempre capturemos el data-target del botón padre que tiene el manejador asignado.
- **Propiedades de estado (ctrlKey y key)**: En formulario.js, se analizan estas propiedades para habilitar un atajo de teclado profesional. Se verifica si e.ctrlKey es verdadero mientras e.key es "Enter" para disparar el método requestSubmit().

5. Control del flujo y propagación

Para evitar conflictos entre elementos y comportamientos nativos del navegador, se han aplicado:

- **Anulación de comportamientos por defecto (preventDefault())**: Aplicado en el evento submit. Esto detiene el envío tradicional del formulario al servidor, permitiendo que JavaScript valide los datos y muestre el feedback sin que la página se refresque.
- **Detención de la propagación (stopPropagation())**: Implementado en los botones de misión del juego. Al pulsar un botón, se evita que el evento "burbujee" hacia los elementos superiores del DOM (como el contenedor de la sección), previniendo que se activen otros escuchadores de eventos por error.

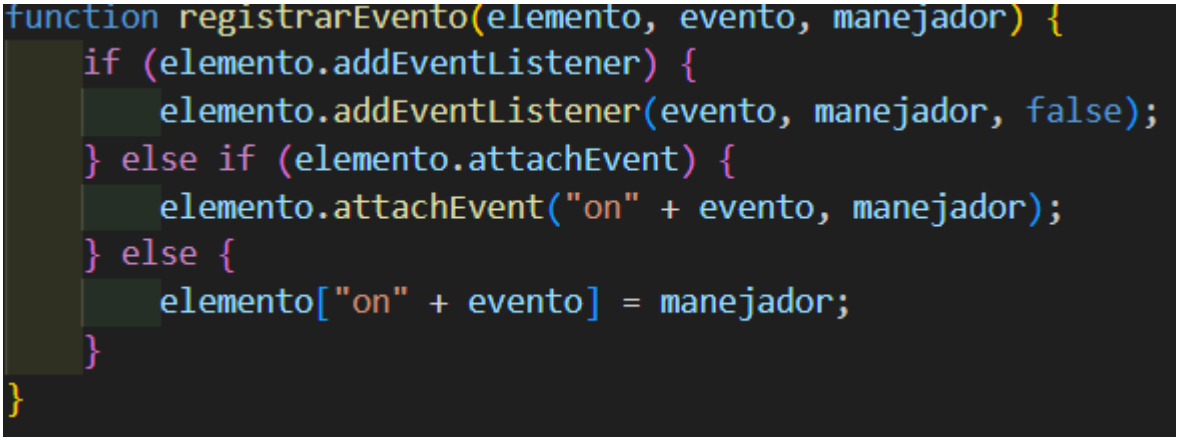


➢ BLOQUE C: PROGRAMACIÓN COMPATIBLE (RA5)

6. Compatibilidad entre navegadores (Cross-browser)

Se ha integrado una solución para garantizar que el proyecto funcione en el mayor número de entornos posible, siguiendo los estándares de programación defensiva.

- **Implementación de la función genérica**: Se ha creado en app.js la función registrarEvento(elemento, evento, manejador). Esta función actúa como un "wrapper" que decide qué método de registro usar según las capacidades del navegador del usuario.



```
function registrarEvento(elemento, evento, manejador) {
    if (elemento.addEventListener) {
        elemento.addEventListener(evento, manejador, false);
    } else if (elemento.attachEvent) {
        elemento.attachEvent("on" + evento, manejador);
    } else {
        elemento["on" + evento] = manejador;
    }
}
```

- **Justificación técnica**: Aunque addEventListener es el estándar actual (W3C), versiones antiguas de algunos navegadores (como Internet Explorer 8 o inferiores) no lo soportan, utilizando en su lugar attachEvent. Al implementar esta función, el proyecto detecta automáticamente el soporte:
    1. Si existe addEventListener, lo usa.
    2. Si no, busca attachEvent.
    3. Como último recurso, asigna el manejador directamente a la propiedad `on-` del elemento. Esto asegura que la aplicación sea robusta y accesible.