

UNIVERSIDAD NACIONAL DEL CENTRO DEL PERU

FACULTAD DE INGENIERIA DE SISTEMAS



MONOGRAFIA:

Exploración Técnica de HTML Avanzado: Estructura,
Semántica y Funcionalidad Dinámica

AUTORES:

- ALIAGA ESQUIVEL, Giampier
- FERNANDEZ ROJAS, Gabriel
- GUTIERREZ MEZA, Ginao
- MACHADO CONTRERAS, Christian
- PANIAGUA ROJAS, Samira
- VARGAS HUAMAN, José

HUANCAYO – PERU

2025

ÍNDICE

INTRODUCCIÓN	1
Objetivo General	1
Objetivos Específicos.....	1
1 CAPÍTULO I. ELEMENTOS Y ATRIBUTOS GLOBALES	3
1.1 id (Identificador único).....	3
1.2 class (Clase de estilo o grupo).....	3
1.3 style (Estilo en línea).....	3
1.4 title (Información contextual).....	4
1.5 data-* (Atributos personalizados)	4
1.6 Ejemplo 1	4
1.7 Ejemplo 2	6
2 CAPÍTULO II. ELEMENTOS MULTIMEDIA Y SUS ATRIBUTOS	9
2.1 Elemento <audio>	9
2.1.1 Definición y función del elemento	9
2.1.2 Atributos principales	9
2.1.3 Formatos de audio soportados.....	9
2.1.4 Compatibilidad entre navegadores	10
2.2 Elemento <video>	10
2.2.1 Definición y función del elemento	10
2.2.2 Atributos principales	10
2.2.3 Formatos de video soportados.....	11
2.2.4 Compatibilidad entre navegadores	11
2.3 Ejemplo de código.....	11
2.3.1 Resultado.....	13
3 CAPITULO III. FORMULARIOS AVANZADOS: VALIDACIÓN Y NUEVOS TIPOS DE ENTRADA.....	14
3.1 Validación en HTML5.....	14
3.2 Nuevos Tipos de Entrada.....	15
3.2.1 Campo de correo electrónico (<input type="email">)	15
3.2.2 Campo de Fecha (<input type="date">).....	15
3.2.3 Campo de rango (<input type="range">).....	15
3.2.4 Campo numérico (<input type="number">)	15
3.2.5 Otros tipos	15
3.3 Ejemplo de formulario con validación nativa (simple)	16
3.3.1 Explicación del Código	18
3.3.2 Resultado.....	19

3.4	Ejemplo de formulario avanzado (complejo).....	19
3.4.1	Explicación del código	22
3.4.2	Resultado.....	23
4	CAPÍTULO IV. TÉCNICAS DE ACCESIBILIDAD Y ETIQUETADO.....	24
4.1	Directrices de accesibilidad para el contenido Web (WCAG):	24
4.2	Normativas legales:	24
4.3	Principios de Accesibilidad Web:	25
4.4	Etiquetado Aria.....	26
4.4.1	Ejemplos.....	26
5	CAPÍTULO V. BUENAS PRÁCTICAS DE SEO EN HTML	27
5.1	Etiquetas SEO	27
5.1.1	Encabezados (<h1> a <h6>).....	27
5.1.2	Atributo Alt de las imágenes ()	27
5.1.3	Enlaces internos y externos (<a>)	27
5.1.4	Etiqueta canonical (<link rel=""canonical"">).....	28
5.1.5	Meta robots (<meta name=""robots"">)	28
5.1.6	Etiquetas Open Graph (<meta property=""og:..."">)	29
5.1.7	Etiqueta de título (<title>)	29
5.1.8	Meta descripción (<meta name=""description"">)	29
5.1.9	Etiqueta hreflang (<link rel=""alternate"" hreflang="">)	29
5.1.10	Negritas (y).....	30
6	CAPÍTULO VI. USO DE TEMPLATE Y SLOT PARA CONTENIDO DINÁMICO	31
6.1	Fundamentos teóricos.....	31
6.2	Casos de uso	32
6.3	Ventajas y limitaciones.....	32
6.4	Ejemplo 1	32
6.5	Ejemplo 2	35
7	CAPÍTULO VII. MICRODATOS Y MARCADO SEMÁNTICO.....	42
7.1	¿Qué son los microdatos?.....	42
7.2	Sintaxis básica de microdatos	42
7.2.1	Ejemplo Básico	42
7.3	Uso de Schema.org.....	42
7.3.1	Ejemplo con productos.....	43
7.4	Ventajas del uso de microdatos	43
7.5	Comparación con otros métodos de marcado semántico	43
7.5.1	Tabla comparativa	43
8	CAPÍTULO VIII. INTEGRACIÓN DE SVG Y CANVAS PARA GRÁFICOS.....	44

8.1	Integración de SVG.....	44
8.1.1	Ejemplo Simple de Integración de SVG	44
8.1.2	Ejemplo Complejo de Integración de SVG	46
8.2	Integración de Canvas	49
8.2.1	Ejemplo Simple de Integración de Canvas.....	49
8.2.2	Ejemplo Complejo de Integración de Canvas	53
9	CAPÍTULO IX. INTEGRACIÓN CON JAVA SCRIPT: EVENTOS Y MANIPULACIÓN DEL DOM.....	57
9.1	¿Qué es el DOM?	57
9.2	Eventos en JavaScript.....	57
9.2.1	Métodos para manejar eventos	57
9.3	Manipulación del DOM	57
9.4	Ejemplo práctico	57
10	CAPÍTULO X. API DE HTML 5 – DRAG AND DROP, GEOLOCATION, WEB STORAGE (LOCALSTORAGE Y SESSIONSTORAGE)	59
10.1	API Drag and Drop	59
10.1.1	Conceptos básicos	59
10.1.2	Eventos clave.....	59
10.1.3	Ejemplo	59
10.2	API GeoLocation.....	64
10.2.1	Fundamentos	64
10.2.2	Métodos principales	64
10.2.3	Ejemplo	64
10.3	API Web Storage.....	69
10.3.1	Diferencias clave:.....	69
10.3.2	Métodos Principales	69
10.3.3	Ejemplo	70
11	CONCLUSIÓN	73
12	REFERENCIA BIBLIOGRÁFICA.....	74

INTRODUCCIÓN

El HyperText Markup Language (HTML) constituye la piedra angular de la World Wide Web, proporcionando la estructura y el significado del contenido que visualizamos a diario en nuestros navegadores. Si bien un conocimiento básico de HTML es fundamental para cualquier desarrollador web, el dominio de sus características avanzadas abre un abanico de posibilidades para la creación de sitios web dinámicos, interactivos y altamente optimizados. La evolución constante del lenguaje ha introducido elementos, atributos y APIs que permiten a los desarrolladores ir más allá de la presentación estática de información, integrando funcionalidades complejas y mejorando significativamente la experiencia del usuario.

Esta monografía está enfocada en HTML avanzado, explorando las capacidades que van más allá del marcado básico para adentrarse en características que potencian la interactividad, accesibilidad, semántica y funcionalidad de las páginas web modernas. El HTML avanzado representa la convergencia de tecnologías web actuales, permitiendo a los desarrolladores crear experiencias digitales ricas y accesibles sin depender exclusivamente de tecnologías adicionales.

En un contexto donde la experiencia del usuario se ha convertido en un factor determinante para el éxito de cualquier presencia digital, dominar las técnicas avanzadas de HTML ofrece ventajas competitivas significativas. Estas técnicas no solo mejoran la usabilidad y la interactividad, sino que también optimizan aspectos críticos como el rendimiento, la accesibilidad y el posicionamiento en motores de búsqueda.

Objetivo General

Analizar y aplicar conceptos avanzados del lenguaje HTML mediante el estudio de sus principales características, elementos estructurales, técnicas de interactividad, accesibilidad y optimización web, con el propósito de desarrollar páginas web modernas, funcionales y bien estructuradas.

Objetivos Específicos

- Identificar y utilizar elementos y atributos globales de HTML (data-*, id, class, style, title, entre otros) para enriquecer la estructura y funcionalidad de los documentos web.
- Explorar mecanismos avanzados como `<template>` y `<slot>`, e integrar contenido multimedia, SVG, Canvas y formularios modernos, para construir interfaces web dinámicas e interactivas.
- Aplicar funcionalidades avanzadas a través de la integración con JavaScript, utilizando APIs como Drag and Drop, Geolocalización y Web Storage, para potenciar la interacción y el almacenamiento en aplicaciones web.
- Implementar técnicas de accesibilidad web y estrategias de SEO, utilizando atributos aria-*, marcado semántico y microdatos, para mejorar la experiencia del usuario y la visibilidad en motores de búsqueda.

El dominio de HTML a este nivel es esencial en el panorama actual del desarrollo web, ya que permite la creación de sitios más eficientes, accesibles y optimizados para diversos dispositivos. Según [1], "el trabajo con HTML permite crear sitios web altamente eficientes tanto para los algoritmos de buscadores, gracias a las etiquetas, como para los usuarios, mediante el diseño visual de páginas web desarrollado con el lenguaje CSS". Esta eficiencia no solo mejora la experiencia del usuario, sino que también facilita el posicionamiento en motores de búsqueda.

Además, a este nivel se tiene un enfoque en la compatibilidad móvil y la estandarización entre navegadores. [2] destaca que "HTML pretende estandarizar las funciones y comportamientos de

los principales navegadores, reduciendo la necesidad de código específico para cada uno de ellos". Esta característica es crucial en un entorno donde los usuarios acceden a la web desde una gran variedad de dispositivos.

CAPÍTULO I. ELEMENTOS Y ATRIBUTOS GLOBALES

En HTML, los atributos globales son propiedades que pueden aplicarse a casi cualquier elemento del documento. Estos atributos son esenciales para proporcionar identificadores únicos, clases de estilo, descripciones adicionales y metadatos personalizados que enriquecen tanto la estructura como la funcionalidad del contenido web.

En el desarrollo web avanzado, el uso estratégico de estos atributos permite crear interfaces más dinámicas, accesibles, organizadas y fáciles de mantener. A continuación, se detallan los principales atributos globales utilizados en HTML avanzado:

1.1 id (Identificador único)

El atributo id se emplea para asignar un identificador único a un elemento del DOM (Document Object Model). Esto es fundamental para seleccionar elementos desde hojas de estilo CSS o scripts JavaScript, permitiendo así aplicar estilos personalizados o realizar manipulaciones específicas en tiempo de ejecución.

Ventajas:

- Permite navegación interna con anclas (``).
- Facilita la vinculación con etiquetas `<label>` en formularios.
- Es esencial para la manipulación precisa del DOM con JavaScript (`getElementById`).

1.2 class (Clase de estilo o grupo)

El atributo class permite agrupar elementos con características comunes. Un mismo nombre de clase puede aplicarse a múltiples elementos, lo que resulta útil para aplicar estilos CSS reutilizables o comportamientos similares con JavaScript.

En contexto avanzado:

- Se utiliza ampliamente con frameworks como Bootstrap, Tailwind o Materialize.
- Se puede combinar con JavaScript para alternar clases dinámicamente (`classList.add/remove/toggle`).

1.3 style (Estilo en línea)

El atributo style se utiliza para aplicar reglas CSS directamente dentro del elemento HTML. Aunque su uso se recomienda solo para estilos rápidos o específicos (debido a que rompe la separación de contenido y presentación), puede ser útil en situaciones donde el estilo se debe aplicar de forma dinámica a través de JavaScript.

Ejemplo avanzado de uso con JavaScript:

```
elemento.style.backgroundColor = "red";
```

1.4 title (Información contextual)

Este atributo proporciona información adicional o descripciones contextuales sobre un elemento. Al pasar el cursor sobre el elemento, el navegador muestra un "tooltip" con el texto proporcionado. Esto puede mejorar la experiencia del usuario y también es útil desde el punto de vista de la accesibilidad.

Ejemplos de uso avanzado:

- En elementos <abbr> para mostrar significados de abreviaturas.
- En íconos o botones donde se necesita brindar una explicación breve.

1.5 data-* (Atributos personalizados)

Los atributos data-* permiten almacenar información personalizada directamente en los elementos HTML. Son muy útiles en aplicaciones web modernas, ya que permiten pasar datos entre el HTML y JavaScript sin necesidad de usar otras estructuras más complejas.

Ventajas en HTML avanzado:

- Permite separar la lógica del diseño visual.
- Facilita la creación de componentes reutilizables.
- Muy utilizado en frameworks modernos y en el desarrollo de Single Page Applications (SPAs).

1.6 Ejemplo 1

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Lista de Productos</title>
  <style>
    .producto {
      border: 1px solid #ccc;
      padding: 15px;
      border-radius: 8px;
      margin-bottom: 10px;
    }
    .oferta {
      background-color: #ffe0b2;
    }
  </style>
</head>
<body>

  <h1>Catálogo de Productos</h1>
```



```

<div id="lista-productos">
  <div id="prod1" class="producto oferta" data-precio="49.99" data-
stock="true" title="Haz clic para más info">
    <h3>Audífonos Bluetooth</h3>
    <p style="color:green;">Precio: S/ <span
class="precio">49.99</span></p>
  </div>

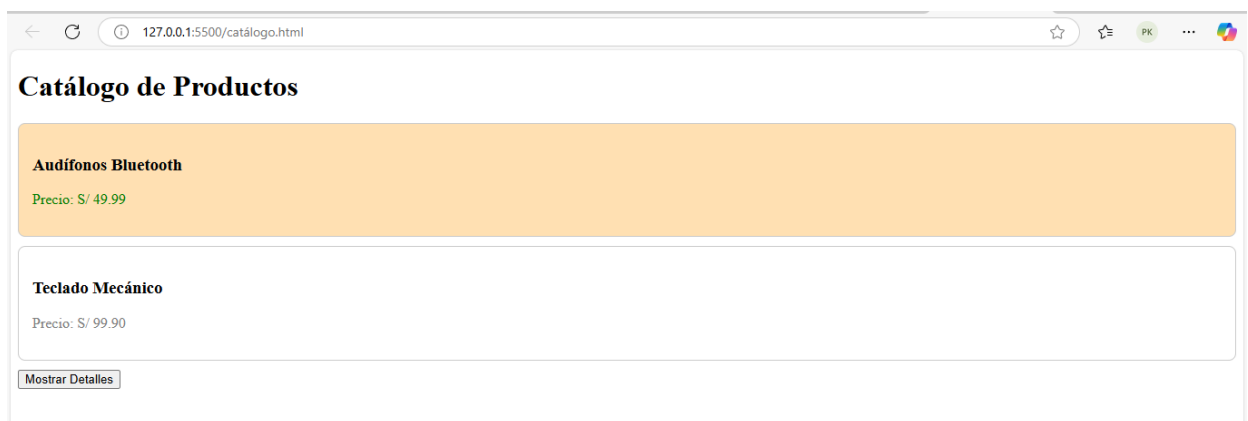
  <div id="prod2" class="producto" data-precio="99.90" data-
stock="false" title="Este producto está agotado">
    <h3>Teclado Mecánico</h3>
    <p style="color:gray;">Precio: S/ <span
class="precio">99.90</span></p>
  </div>
</div>

<button onclick="mostrarDetalles()">Mostrar Detalles</button>

<script>
function mostrarDetalles() {
  const productos = document.querySelectorAll('.producto');
  productos.forEach(producto => {
    const nombre = producto.querySelector('h3').textContent;
    const precio = producto.dataset.precio;
    const stock = producto.dataset.stock === "true" ? "Disponible"
: "Agotado";
    alert(`Producto: ${nombre}\nPrecio: S/ ${precio}\nEstado:
${stock}`);
  });
}
</script>

</body>
</html>

```



1.6.1.1 Conceptos aplicados:

Atributo	Uso
id	Identifica cada producto de forma única (prod1, prod2).
class	Aplica estilos comunes y condicionales como .oferta.
style	Estilo en línea para cambiar el color del texto del precio.
title	Muestra un tooltip al pasar el cursor sobre el producto.
data-*	Guarda información personalizada como el precio y stock disponible.

1.6.1.2 Explicación breve del JavaScript

- Se seleccionan todos los elementos con clase .producto.
- Se accede a los atributos personalizados data-precio y data-stock usando dataset.
- Se muestra la información de cada producto en una ventana emergente (alert).

1.7 Ejemplo 2

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Lista de Tareas</title>
  <style>
    .tarea {
      padding: 10px;
      margin: 5px 0;
      border: 1px solid #ccc;
      border-radius: 4px;
    }

    .alta { background-color: #ffe5e5; }
    .media { background-color: #fffbe5; }
    .baja { background-color: #e5ffe5; }

    button {
      margin-left: 10px;
    }
  </style>
</head>
<body>
  <h1>Mis Tareas</h1>
```

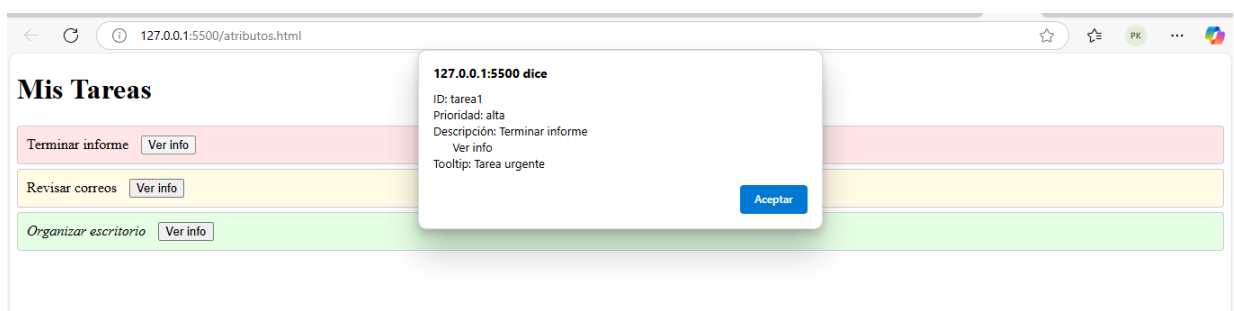
```

<div id="lista-tareas">
  <div class="tarea alta" data-priority="alta" title="Tarea urgente"
id="tarea1">
    Terminar informe
    <button onclick="mostrarInfo(this)">Ver info</button>
  </div>
  <div class="tarea media" data-priority="media" title="Tarea
intermedia" id="tarea2">
    Revisar correos
    <button onclick="mostrarInfo(this)">Ver info</button>
  </div>
  <div class="tarea baja" data-priority="baja" title="Tarea opcional"
id="tarea3" style="font-style: italic;">
    Organizar escritorio
    <button onclick="mostrarInfo(this)">Ver info</button>
  </div>
</div>

<script>
function mostrarInfo(boton) {
  const tarea = boton.parentElement;
  const id = tarea.id;
  const prioridad = tarea.dataset.priority;
  const titulo = tarea.title;
  const contenido = tarea.textContent.trim();

  alert(`ID:   ${id}\nPrioridad:   ${prioridad}\nDescripción:
${contenido}\nTooltip: ${titulo}`);
}
</script>
</body>
</html>

```



1.7.1.1 Conceptos aplicados:

Atributo	Uso
id	Identifica cada tarea de forma única (id="tarea1", etc.).
class	Define estilos según la prioridad (alta, media, baja).
style	Aplica un estilo directo (ej. italic en la última tarea).
title	Muestra un tooltip al pasar el mouse por encima.
data-*	Guarda información personalizada (en este caso, data-priority).
id="tarea1"	Identificador único de la tarea.
class="tarea alta"	Permite aplicar estilos CSS (clase general tarea, y alta para prioridad).
data-priority="alta"	Atributo personalizado que guarda la prioridad.
title="Tarea urgente"	Aparece como tooltip al pasar el mouse.
style="..." (en la 3.ª tarea)	Estilo directo en línea (ej: font-style: italic).

CAPÍTULO II. ELEMENTOS MULTIMEDIA Y SUS ATRIBUTOS

2.1 Elemento <audio>

2.1.1 Definición y función del elemento

El elemento <audio> de HTML permite incrustar contenido de audio en una página web, como música, efectos de sonido o grabaciones. Este elemento proporciona una forma sencilla de incluir archivos de audio directamente en las páginas web, sin necesidad de plugins externos. Permite la integración de archivos de sonido de diferentes formatos y ofrece opciones de control de reproducción para mejorar la experiencia del usuario [4].

2.1.2 Atributos principales

Según [4]:

- **src:** Este atributo especifica la ubicación del archivo de audio. Es obligatorio para que el archivo se reproduzca correctamente.
- **controls:** Añade controles de reproducción al reproductor de audio, como play, pause, volumen y barra de progreso. Este atributo es opcional, pero altamente recomendable para que los usuarios puedan interactuar con el contenido.
- **autoplay:** Hace que el archivo de audio se reproduzca automáticamente al cargar la página. Este atributo también es opcional, pero puede ser útil en situaciones específicas como música de fondo.
- **loop:** Hace que el audio se repita continuamente una vez que finaliza. Es útil para efectos de sonido o música de fondo que debe reproducirse sin fin.
- **muted:** Silencia el audio de manera predeterminada. Es útil en situaciones donde se desea que el usuario active el sonido manualmente.
- **preload:** Define la estrategia de precarga del archivo de audio. Los valores pueden ser: auto, el navegador carga el archivo completamente al inicio; metadata, solo se cargan los metadatos (como duración y tamaño); none, no se carga nada hasta que el usuario inicie la reproducción.

2.1.3 Formatos de audio soportados

Según [5] elemento <audio> es compatible con varios formatos de audio, lo que permite que se utilicen diferentes tipos de archivo según las necesidades y compatibilidad de los navegadores. Los formatos más comunes son:

- **MP3:** El formato más ampliamente soportado y popular.
- **WAV:** Formato de audio sin compresión que ofrece alta calidad, aunque los archivos tienden a ser más grandes.
- **OGG:** Un formato de código abierto que es particularmente popular en entornos libres.

2.1.4 Compatibilidad entre navegadores

El soporte del elemento <audio> varía según el navegador y el sistema operativo. A continuación, se muestra un resumen general de la compatibilidad:

- **Google Chrome:** Soporta los formatos MP3, WAV y OGG.
- **Mozilla Firefox:** Compatible con MP3, WAV y OGG.
- **Safari:** Soporta principalmente MP3, pero también WAV y OGG en versiones recientes.
- **Microsoft Edge:** Compatible con MP3, WAV y OGG.

Para garantizar que el audio sea compatible en todos los navegadores, es recomendable incluir varias fuentes en el elemento <audio>[6].

2.2 Elemento <video>

2.2.1 Definición y función del elemento

El elemento <video> en HTML5 permite incrustar contenido de video en una página web. Es una forma estándar de reproducir video de forma nativa sin necesidad de utilizar complementos o plugins externos. Con este elemento, los desarrolladores pueden añadir videos interactivos y controlar su reproducción, ya sea para presentaciones, tutoriales, o contenido multimedia en general [5].

2.2.2 Atributos principales

Según [7] los atributos principales son:

- **src:** Este atributo especifica la ubicación del archivo de video. Es obligatorio para que el archivo se reproduzca correctamente.
- **controls:** Añade controles de reproducción al reproductor de video, como play, pause, volumen y barra de progreso. Este atributo es opcional, pero recomendado para que los usuarios puedan interactuar con el contenido.
- **width y height:** Estos atributos definen las dimensiones del reproductor de video en píxeles. Son opcionales, pero es útil establecerlos para un diseño consistente.
- **poster:** Especifica una imagen que se muestra antes de que el video comience a reproducirse, útil para proporcionar una vista previa o una miniatura del video.
- **muted:** Silencia el audio del video por defecto. Es útil si se quiere evitar que el sonido se reproduzca inmediatamente al cargar el video, permitiendo al usuario activar el sonido cuando lo desee.
- **preload:** Define la estrategia de precarga del archivo de video, como: auto, el navegador carga el video completamente al inicio; metadata, solo se cargan los metadatos del vídeo; none, no se carga nada hasta que el usuario inicie la reproducción.

2.2.3 Formatos de video soportados

Según [5] el elemento `<video>` es compatible con varios formatos de video, lo que permite que los desarrolladores ofrezcan diferentes tipos de archivo para garantizar la compatibilidad con varios navegadores. Los formatos más comunes son:

- MP4: El formato más popular y compatible, ampliamente soportado en la mayoría de los navegadores.
- WebM: Un formato de código abierto que proporciona una buena calidad de video y es compatible con navegadores como Google Chrome y Firefox.
- OGV: Otro formato de código abierto, similar a WebM, pero menos utilizado.

2.2.4 Compatibilidad entre navegadores

Según [8] (2024) la compatibilidad de los navegadores con el elemento `<video>` varía dependiendo del formato de video:

- Google Chrome: Soporta MP4, WebM y OGV.
- Mozilla Firefox: Compatible con MP4, WebM y OGV.
- Safari: Principalmente compatible con MP4, aunque también soporta WebM en versiones recientes.
- Microsoft Edge: Compatible con MP4, WebM y OGV.
- Opera: Soporta MP4, WebM y OGV.

2.3 Ejemplo de código

Se presenta un código donde se utiliza tanto un elemento de audio como un elemento de video:

```
<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Ejemplo de Audio y Video</title>

<style>

  body {

    font-family: Arial, sans-serif;

    text-align: center;

    padding: 20px;
```

```

    }

    h1 {
        color: #193CB8;
    }

    .container {
        margin: 20px 0;
    }

    video, audio {
        width: 80%;
        max-width: 600px;
        margin: 10px 0;
    }
</style>
</head>
<body>

    <h1>Ejemplo de Elementos Multimedia: <audio> y <video></h1>

    <div class="container">

        <h2>Reproducción de Video</h2>

        <video
            poster="https://via.placeholder.com/600x300.png?text=Vista+Previa+del+Video"
            loop preload="auto">
            controls
            autoplay
            <source src="https://www.w3schools.com/html/movie.mp4" type="video/mp4">
            <source src="https://www.w3schools.com/html/movie.ogg" type="video/ogg">
            Tu navegador no soporta el elemento <code>video</code>.
        </video>
    </div>

    <div class="container">

        <h2>Reproducción de Audio</h2>

        <audio controls autoplay loop preload="auto">

```



```
<source src="https://www.soundhelix.com/examples/mp3/SoundHelix-Song-1.mp3"
type="audio/mp3">

<source src="https://www.soundhelix.com/examples/mp3/SoundHelix-Song-1.ogg"
type="audio/ogg">

  Tu navegador no soporta el elemento <code>audio</code>.

</audio>

</div>

</body>

</html>
```

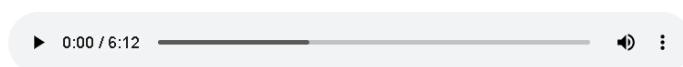
2.3.1 Resultado

Ejemplo de Elementos Multimedia:

Reproducción de Video



Reproducción de Audio



CAPITULO III. FORMULARIOS AVANZADOS: VALIDACIÓN Y NUEVOS TIPOS DE ENTRADA

3.1 Validación en HTML5

HTML5 no solo introdujo nuevos tipos de campo, sino también atributos que permiten declarar restricciones de validación directamente en el HTML. Estos atributos habilitan la validación del lado del cliente integrado en el navegador, el cual se ejecuta al intentar enviar el formulario o al cambiar de campo, sin necesidad de JavaScript. A continuación, se detallan los atributos más utilizados:

- **required:** Indica que el campo es obligatorio. Si un campo con `required` está vacío al momento de enviar el formulario, el navegador bloqueará el envío y mostrará un mensaje de error predeterminado (por ejemplo: "Este campo es obligatorio") [9]. Este atributo aplica a casi todos los tipos de `input` (texto, email, número, etc.), `select` y `textarea`. Además, los navegadores suelen aplicar estilos CSS por defecto a los campos requeridos que no han sido llenados correctamente, utilizando la pseudoclase `:invalid` [10].
- **pattern:** Permite especificar una expresión regular que el valor del campo debe cumplir para considerarse válido. Este atributo es útil para validaciones personalizadas, como formatos de código postal, nombres de usuario, etc., que no están cubiertos por los tipos de `input` existentes. Por ejemplo, `pattern="[A-Za-z]{3,}"` en un campo de texto obligaría a que el valor contenga al menos tres letras (solo alfabéticas). Si el valor no coincide con la expresión regular, el formulario no se enviará y el campo se marcará como inválido. Es conveniente usar también el atributo `title` en conjunto con `pattern` para proporcionar un mensaje de ayuda personalizado que indique al usuario el formato esperado [11].
- **Atributos de rango numérico/fechas (min, max, step):** Estos atributos restringen los valores aceptables de forma numérica. `min` y `max` definen los extremos permitidos. Por ejemplo, en un `<input type="date">`, `min="2020-01-01"` y `max="2025-12-31"` restringirían las fechas entre el año 2020 y 2025. Si el usuario introduce una fecha fuera de ese rango, el campo se considera inválido. El atributo `step` define el intervalo de variación. Por ejemplo, `step="2"` permitiría los valores 0, 2, 4, pero no 3 o 5. Es importante definir estos atributos de manera consistente para evitar confundir al usuario [12].
- **maxlength / minlength:** Especifican la cantidad máxima y mínima de caracteres permitidos en campos de texto. Por ejemplo, en un campo de contraseña podríamos usar `minlength="8"` para requerir al menos 8 caracteres. Si el usuario ingresa menos, el campo será marcado como inválido al intentar enviar el formulario [13].
- **type="submit" y validación:** La validación nativa se activa normalmente al hacer clic en un botón `<button type="submit">` o `<input type="submit">` asociado al formulario. Si existen campos que violan alguna de las restricciones (por ejemplo, `required` no lleno, formato inválido), el navegador detendrá el envío y resaltará el primer campo con error. También puede mostrar mensajes de error por defecto cerca de cada campo [14].
- **novalidate:** Si se coloca en la etiqueta `<form>`, deshabilita toda la validación nativa al enviar, permitiendo enviar los datos aunque los campos estén

incompletos o mal formateados. Este atributo se usa cuando se desea manejar las validaciones manualmente con scripts o en el servidor [15].

3.2 Nuevos Tipos de Entrada

HTML5 introdujo mejoras significativas en la construcción de formularios web, incorporando nuevos tipos de entrada (input types) y atributos de validación que facilitan la recolección de datos del lado del cliente. A continuación, se detallan los principales tipos de entrada avanzados y sus capacidades de validación nativa.

3.2.1 Campo de correo electrónico (<input type="email">)

Este tipo de entrada está diseñado para capturar direcciones de correo electrónico. Los navegadores que lo soportan validan automáticamente el valor introducido para verificar que cumple el formato general de un email (por ejemplo, que contenga un símbolo “@” y un dominio). Si el contenido no es una dirección de correo válida, el campo se considera inválido y el navegador mostrará un mensaje de error predeterminado [16]. Además, se puede usar la pseudoclase CSS :invalid para estilizar el campo dependiendo de si su valor es válido o no [17].

3.2.2 Campo de Fecha (<input type="date">)

Este campo proporciona un control específico para la selección de fechas. En navegadores modernos, suele renderizarse como un calendario emergente o selector de fecha. El valor capturado por este campo se normaliza en formato YYYY-MM-DD (año-mes-día) según el estándar ISO 8601 [18]. Además, los atributos min y max permiten definir un rango de fechas permitido. Si el navegador no soporta este tipo de entrada, el campo se mostrará como un campo de texto [19].

3.2.3 Campo de rango (<input type="range">)

Este input crea un control deslizante (slider) para seleccionar valores numéricos dentro de un rango continuo. Es apropiado cuando el valor exacto no es crucial, sino más bien una percepción relativa, como ajustar el volumen o el nivel de satisfacción [20]. El desarrollador puede configurar los valores mínimos (min), máximos (max) y el intervalo de pasos (step).

3.2.4 Campo numérico (<input type="number">)

Este campo permite la introducción de valores numéricos y ofrece controles con flechas para aumentar o disminuir el valor. Los atributos min, max y step permiten limitar y definir el paso de los valores numéricos aceptados [21]. A diferencia del tipo range, el campo numérico permite la edición manual del valor.

3.2.5 Otros tipos

HTML5 también introdujo controles especializados como type="color" (selector de color), type="url" (para URLs), type="tel" (para números telefónicos). Cada uno busca optimizar la experiencia de ingreso de datos específicos, ya sea ofreciendo teclados virtuales apropiados en dispositivos móviles o validando el formato básico [22].

3.3 Ejemplo de formulario con validación nativa (simple)

A continuación, se presenta un formulario sencillo que emplea algunos de los nuevos tipos de entrada y atributos de validación. Este formulario de ejemplo pide al usuario su nombre y correo electrónico, ambos obligatorios, y requiere que el correo tenga un formato válido:

```
<!DOCTYPE html>

<html lang="es">

  <head>

    <meta charset="UTF-8" />

    <title>Formulario de contacto</title>

    <style>

      body {

        font-family: Arial, sans-serif;

        padding: 20px;

      }

      form {

        max-width: 400px;

        margin: auto;

      }

      label,

      input {

        display: block;

        margin-bottom: 10px;

      }

      input[type="text"],

      input[type="email"] {

        width: 100%;

        padding: 8px;

        box-sizing: border-box;

      }

      button {

        padding: 10px 15px;
```

```

    background-color: #007bff;

    color: white;

    border: none;

    cursor: pointer;

}

button:hover {

    background-color: #0056b3;

}

</style>

</head>

<body>

<form id="formulario1">

    <label for="nombre">Nombre:</label>

    <input

        type="text"

        id="nombre"

        name="nombre"

        required

        minlength="2"

        pattern="[A-Za-zÑñ ]+"

        title="Sólo se permiten letras y espacios."

    />

    <label for="correo">Correo electrónico:</label>

    <input type="email" id="correo" name="correo" required />

    <button type="submit">Enviar</button>

</form>

<script>

    document

```

```

.getElementById("formulario1")
.addEventListener("submit", function (event) {
    if (!this.checkValidity()) {
        event.preventDefault();
        alert("Por favor, complete el formulario correctamente.");
    } else {
        alert("Formulario enviado correctamente.");
    }
});
</script>
</body>
</html>

```

3.3.1 Explicación del Código

a. Estructura HTML

- Se utiliza HTML5 para construir la estructura básica de la página.
- El formulario incluye dos campos principales: uno para el nombre y otro para el correo electrónico.
- Cada campo está asociado a una etiqueta descriptiva y contiene validaciones como longitud mínima, tipo de dato y patrón de entrada.
- El formulario se encuentra dentro del elemento <form> con un identificador único para su manejo con JavaScript.

b. Estilos CSS

- Se aplican estilos internos que definen la apariencia visual del formulario.
- El formulario se centra en la página y se limita a un ancho máximo para mantener un diseño ordenado.
- Los campos de texto ocupan todo el ancho disponible y tienen relleno para facilitar la escritura.
- El botón de envío tiene un color azul que cambia al pasar el cursor, lo cual mejora la experiencia visual.

c. Lógica con JavaScript

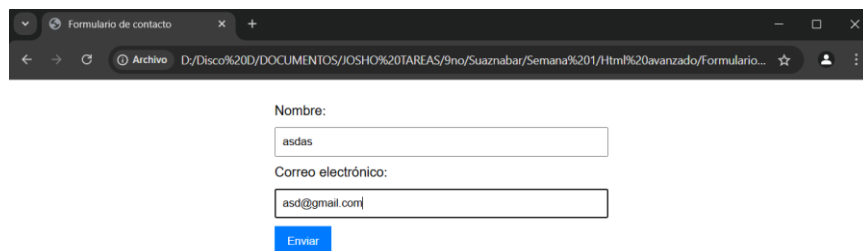
- Se añade un script que gestiona el evento de envío del formulario.
- Antes de enviarse, se verifica automáticamente si los campos cumplen con las validaciones establecidas en HTML.

- Si hay errores, se interrumpe el envío y se muestra una alerta al usuario.
- Si los datos son válidos, se muestra un mensaje confirmando el envío exitoso del formulario.

d. Interacción del Usuario

- El usuario completa los campos solicitados.
- Si algún dato no es correcto o está incompleto, el sistema lo notifica de inmediato mediante una alerta.
- La validación ocurre en el navegador del usuario, lo que permite una respuesta rápida sin necesidad de conexión con un servidor externo.

3.3.2 Resultado



3.4 Ejemplo de formulario avanzado (complejo)

A continuación, se presenta un formulario más completo que incorpora varios de los tipos de entrada y atributos mencionados, simulando un pequeño formulario de registro o encuesta. Incluiremos campos de distintos tipos: texto, número, fecha, rango y color, demostrando sus comportamientos:

```
<!DOCTYPE html>

<html lang="es">

  <head>

    <meta charset="UTF-8" />

    <title>Registro de usuario</title>

    <style>

      body {

        font-family: Arial, sans-serif;

        padding: 20px;

      }

    </style>

  </head>

  <body>

    <div>

      <div>

        <input type="text" value="Nombre:" />

      </div>

      <div>

        <input type="text" value="Correo electrónico:" />

      </div>

      <div>

        <input type="button" value="Enviar" />

      </div>

    </div>

  </body>

</html>
```

```
form {  
    max-width: 500px;  
    margin: auto;  
}  
  
label {  
    display: block;  
    margin-top: 15px;  
}  
  
input[type="text"],  
input[type="number"],  
input[type="date"],  
input[type="range"],  
input[type="color"] {  
    width: 100%;  
    padding: 8px;  
    margin-top: 5px;  
    box-sizing: border-box;  
}  
  
input[type="range"] {  
    width: 100%;  
}  
  
button {  
    margin-top: 20px;  
    padding: 10px 15px;  
    background-color: #28a745;  
    color: white;  
    border: none;  
    cursor: pointer;  
}  
  
button:hover {  
    background-color: #218838;
```



```

    }

</style>

</head>

<body>

<form id="formulario2">

    <label for="usuario">Nombre de usuario:</label>

    <input

        type="text"

        id="usuario"

        name="usuario"

        required

        pattern="[A-Za-z0-9_]{4,}"

        title="Debe tener al menos 4 caracteres (letras, números o guion bajo)."

    />

    <label for="edad">Edad:</label>

    <input type="number" id="edad" name="edad" min="1" max="120" />

    <label for="nacimiento">Fecha de nacimiento:</label>

    <input type="date" id="nacimiento" name="nacimiento" />

    <label for="satisfaccion">Nivel de satisfacción (0 a 100):</label>

    <input

        type="range"

        id="satisfaccion"

        name="satisfaccion"

        min="0"

        max="100"

        step="10"

    />

```

```

<label for="colorfav">Color favorito:</label>

<input type="color" id="colorfav" name="colorfav" />

<button type="submit">Registrar</button>
</form>

<script>
document
.getElementById("formulario2")
.addEventListener("submit", function (event) {
    if (!this.checkValidity()) {
        event.preventDefault();
        alert("Por favor, complete todos los campos correctamente.");
    } else {
        alert("Registro enviado correctamente.");
    }
});
</script>
</body>
</html>

```

3.4.1 Explicación del código

a. Estructura HTML

- El código utiliza HTML5 para crear un formulario de registro con distintos tipos de campos.
- Se incluyen entradas para nombre de usuario, edad, fecha de nacimiento, nivel de satisfacción y color favorito.
- Cada campo está acompañado de su respectiva etiqueta descriptiva para guiar al usuario.
- Se definen validaciones básicas: por ejemplo, el nombre de usuario requiere al menos cuatro caracteres y solo permite letras, números o guion bajo.
- El formulario está contenido dentro de un elemento <form> identificado como formulario2 para facilitar su manipulación con JavaScript.

b. Estilos CSS

- Se utilizan estilos internos para controlar la apariencia del formulario y hacerlo visualmente atractivo.
- El formulario se presenta centrado en la página, con un ancho máximo definido para asegurar una buena legibilidad.
- Todos los campos de entrada se distribuyen de manera vertical, ocupando el ancho completo disponible.
- El botón de registro está resaltado en color verde, con un cambio de tonalidad al pasar el cursor, brindando un efecto visual dinámico.

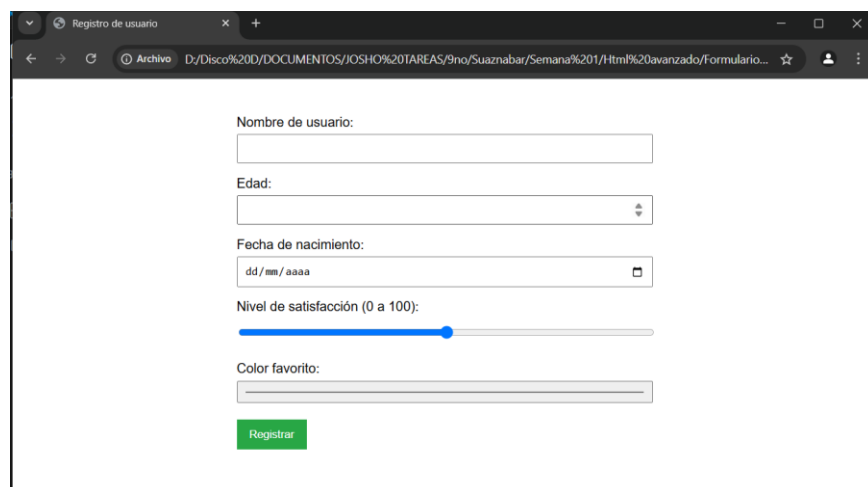
c. Lógica con JavaScript

- Se implementa un pequeño script que intercepta el envío del formulario.
- Antes de enviar los datos, se verifica que todos los campos cumplan con las validaciones establecidas en el HTML.
- Si algún campo no es válido, se interrumpe el envío y se muestra una alerta solicitando al usuario que corrija los errores.
- Si los datos ingresados son correctos, se muestra un mensaje confirmando el envío del registro.

d. Interacción del Usuario

- El usuario completa los distintos campos del formulario con su información personal.
- Si algún dato no cumple los requisitos, el sistema lo detecta y notifica al usuario mediante un mensaje emergente.
- Todo el proceso se realiza de forma local en el navegador, garantizando una validación inmediata sin necesidad de conexión a un servidor.

3.4.2 Resultado



The screenshot shows a web browser window with the title "Registro de usuario". The address bar displays the file path: "D:\Disco%20D\DOCUMENTOS\IOSHO%20TAREAS\9no\Suaznabar\Semana%201\Htm%20avanzado\Formulario...". The form itself is centered and contains the following fields and controls:

- Nombre de usuario:** A text input field.
- Edad:** A text input field with a small downward arrow icon on the right.
- Fecha de nacimiento:** A date picker input field showing the format "dd/mm/aaaa".
- Nivel de satisfacción (0 a 100):** A horizontal range slider with a blue track and a blue handle.
- Color favorito:** A color picker input field.
- Registrar:** A green button with white text.

CAPÍTULO IV. TÉCNICAS DE ACCESIBILIDAD Y ETIQUETADO

La accesibilidad web desempeña un papel esencial en el diseño y desarrollo de páginas en línea, ya que permite que todas las personas, incluidas aquellas con alguna discapacidad, puedan acceder a la información y servicios de manera justa. En un mundo cada vez más digital, garantizar la accesibilidad no solo responde a obligaciones legales, sino que también mejora la experiencia del usuario, haciéndola más equitativa y enriquecedora. A medida que la vida cotidiana se apoya cada vez más en las tecnologías digitales, asegurar el acceso inclusivo se vuelve crucial para fomentar la participación social y brindar igualdad de oportunidades.

El concepto de accesibilidad en la web se refiere a crear sitios que puedan ser utilizados por cualquier persona, sin importar sus capacidades físicas o cognitivas. Esto abarca a usuarios con dificultades visuales, auditivas, motrices o de aprendizaje. Más allá de cumplir con normativas, el objetivo es ofrecer una experiencia satisfactoria para todos. Aplicar correctamente las técnicas de accesibilidad permite que los contenidos sean utilizables y funcionales en distintos dispositivos y con diversas tecnologías de apoyo, como lectores de pantalla o teclados adaptados.

Este texto aborda las principales normas y pautas relacionadas con la accesibilidad web, así como las prácticas recomendadas para implementarlas eficazmente. Se analiza el marco establecido por las Directrices de Accesibilidad para el Contenido Web (WCAG), las leyes aplicables, y las estrategias útiles para integrar estas prácticas en el diseño de sitios web. También se enfatiza la utilidad de herramientas de evaluación y la necesidad de realizar pruebas con usuarios para detectar y corregir barreras de accesibilidad. Finalmente, se incluyen ejemplos representativos que demuestran cómo aplicar estas técnicas puede mejorar significativamente la experiencia de navegación para todos los usuarios.

4.1 Directrices de accesibilidad para el contenido Web (WCAG):

La WCAG son las pautas más reconocidas para la accesibilidad web, se basa en 4 principios fundamentales:

- **Perceptible:** La información y el contenido de la interfaz deben ser presentados de tal manera que los usuarios puedan percibirlos. Incluye el uso de texto alternativo para imágenes, transcripciones para contenido multimedia, contraste adecuado entre el texto y el fondo.
- **Operable:** Los componentes de la interfaz deben ser operables por todos los usuarios. Esto implica la accesibilidad mediante el teclado, la visibilidad del enfoque y la posibilidad de utilizar elementos interactivos sin la necesidad de precisión motora excesiva.
- **Comprensible:** El contenido y la interfaz deben ser comprensibles para los usuarios. Esto implica un lenguaje claro, una estructura de documento lógico y una navegación coherente.
- **Robusto:** El contenido debe ser robusto y compatible con una variedad de tecnologías asistivas. Incluye el uso de HTML semántico y la validación del código para evitar errores que puedan afectar la accesibilidad.

4.2 Normativas legales:

Existen leyes que exigen que las organizaciones proporcionen igualdad de acceso a su sitio web:

- **Ley de Americanos con discapacidades:** Prohíbe la discriminación contra personas con discapacidades y exige que las empresas proporcionen acceso equitativo a los bienes y servicios que ofrecen.
- **Ley de Accesibilidad de Canadá:** Exige que los sitios web sean accesibles y busca eliminar barreras para las personas con discapacidades y promover la inclusión en la sociedad digital
- **Directiva de Accesibilidad web de la Unión Europea:** Busca asegurar que la información pública esté disponible para todos los ciudadanos, incluyendo aquellos con discapacidades.

4.3 Principios de Accesibilidad Web:

Se basa en los principios fundamentales establecidos por la WCAG:

- **Perceptible:** Se refiere a la capacidad de los usuarios de percibir el contenido y la interfaz, incluye:
 - **Texto Alternativo:** Debe contar con atributos alt, que proporcionen descripciones textuales adecuadas.
 - **Transcripciones:** Contenido multimedia como audio y video deben contar con transcripciones que permitan que los usuarios con discapacidad auditiva puedan acceder a la información.
 - **Contraste de color:** El contraste entre el texto y el fondo debe ser suficiente para facilitar la lectura en usuarios con baja visión o daltonismo.
- **Operable:** Se refiere a la capacidad de los usuarios para interactuar con el contenido:
 - **Navegación con teclado:** Todos los elementos interactivos deben ser accesibles mediante teclado.
 - **Enfoque visibilidad:** Los elementos que tienen el enfoque del teclado deben ser claramente visibles para facilitar la navegación.
 - **Controles de formulario:** Los campos de formularios deben estar claramente etiquetados y proporcionar instrucciones claras.
- **Comprensible:** Capacidad de los usuarios para entender el contenido y la interfaz:
 - **Lenguaje claro:** Evitar tecnicismos y jergas innecesarias.
 - **Estructura del documento:** Contenido organizado de manera lógica, utilizando encabezados y estructuras jerárquicas.
 - **Consistencia:** La navegación y la estructura del sitio web deben ser consistentes en todas las páginas.
- **Robusto:** Capacidad del contenido para ser interpretado correctamente por una variedad de tecnologías asistivas:
 - **HTML Semántico:** Marca el contenido de manera que las tecnologías asistivas puedan interpretarlo correctamente.
 - **Validación del Código:** Identifica problemas que puedan interferir con el funcionamiento de las tecnologías asistivas.

4.4 Etiquetado Aria

Permite añadir atributos especiales al HTML para mejorar la accesibilidad de los elementos dinámicos e interactivos que no están bien soportados por HTML semántico tradicional. A continuación, se muestran los principales atributos aria-* y sus usos:

Atributo	Descripción
aria-label	Proporciona una etiqueta accesible personalizada para un elemento (invisible visualmente).
aria-labelledby	Asocia el elemento con otro cuyo contenido textual será su etiqueta accesible.
aria-describedby	Similar a aria-labelledby, pero se usa para describir el contenido o función del elemento.
aria-hidden	Indica si el elemento debe ser ignorado por tecnologías de asistencia.
aria-live	Indica que el contenido puede cambiar dinámicamente (útil para lectores de pantalla en contenido como notificaciones).
aria-expanded	Indica si un elemento (como un menú desplegable) está expandido o colapsado.
aria-controls	Indica qué elemento(s) son controlados por el actual.
aria-disabled	Similar a disabled, pero accesible para lectores de pantalla.
aria-pressed	Indica el estado de un botón "presionado" (por ejemplo, un botón tipo toggle).
role	Define el tipo de widget (por ejemplo, button, dialog, alert, etc.).

4.4.1 Ejemplos

- Etiqueta accesible personalizada

```
32  
33 <button aria-label="Cerrar ventana">X</button>  
34
```

- Elemento controlado y expandido

```
36 <button aria-controls="menu1" aria-expanded="false">Menú</button>  
37 <ul id="menu1" hidden>  
38   <li>Inicio</li>  
39   <li>Contacto</li>  
40 </ul>
```

- Ocultar elemento para lectores de pantalla

```
42  
43 <div aria-hidden="true">Este contenido es decorativo y no será leído.</div>  
44
```

- Contenido dinámico con Aria-live

```
46 <div aria-live="polite" id="mensaje">  
47   <!-- Un lector de pantalla leerá automáticamente este mensaje cuando cambie -->  
48 </div>
```

CAPÍTULO V. BUENAS PRÁCTICAS DE SEO EN HTML

El SEO, Search Engine Optimization (Optimización de motores de búsqueda), es el proceso de optimizar un sitio web para que aparezca en las primeras posiciones de los resultados orgánicos de los buscadores. Como objetivo se busca aumentar la visibilidad y atraer más visitantes al sitio web de forma natural.

Una parte fundamental de este proceso es la correcta estructuración del código HTML, ya que permite a los motores de búsqueda comprender el contenido y la jerarquía de una página. Aplicar buenas prácticas de SEO en HTML no solo mejora el ranking en los buscadores, sino que también optimiza la experiencia del usuario y facilita la accesibilidad del sitio.

5.1 Etiquetas SEO

5.1.1 Encabezados (<h1> a <h6>)

Los encabezados son etiquetas HTML utilizadas para organizar el contenido de una página en niveles jerárquicos, donde la etiqueta <h1> representa el tema principal y <h6> los temas menos relevantes. Estas etiquetas permiten estructurar el contenido de forma lógica, facilitando su lectura tanto para los usuarios como para los motores de búsqueda.

Para los usuarios, los encabezados hacen que el contenido sea más fácil de escanear y comprender, ayudándolos a encontrar rápidamente la información que buscan. Para los motores de búsqueda, indican la importancia relativa de cada sección, siendo el <h1> un indicador clave del tema central de la página.

```
51  
52 <h1>Guía Completa sobre SEO</h1>  
53 <h2>Qué es el SEO</h2>  
54
```

5.1.2 Atributo Alt de las imágenes ()

El atributo alt es un texto alternativo que se incluye dentro de la etiqueta para describir el contenido de una imagen. Este atributo es esencial para mejorar la accesibilidad, ya que permite a los lectores de pantalla transmitir información sobre la imagen a personas con discapacidades visuales. Además, el texto alternativo se muestra si la imagen no se carga correctamente.

Para los motores de búsqueda, el atributo alt ayuda a comprender el contexto y contenido de las imágenes, lo que es clave para el posicionamiento en búsquedas de imágenes. También refuerza la relevancia temática de la página, especialmente si el texto alt incluye palabras clave relacionadas.

```
54  
55 <img src=""grafico-seo.png alt="Gráfico mostrando tendencias de SEO en 2025">  
56
```

5.1.3 Enlaces internos y externos (<a>)

La etiqueta <a> permite crear enlaces que conectan diferentes páginas, ya sea dentro del mismo sitio (enlaces internos) o hacia otros sitios web (enlaces externos). Estos enlaces no solo facilitan la navegación para los usuarios, guiándolos a contenido relevante o complementario, sino que también organizan y conectan las diferentes secciones de un sitio web.

En términos de SEO, los enlaces internos distribuyen la autoridad entre las páginas del sitio, ayudando a que los motores de búsqueda comprendan la estructura del contenido. Por otro lado, los enlaces externos, que conseguiremos a través del link building, permiten mejorar la relevancia al señalar fuentes confiables.

```
7 <link href="https://www.mi-sitio.com/guia-seo">
```

5.1.4 Etiqueta canonical (<link rel="canonical">)

La etiqueta canonical se utiliza para informar a los motores de búsqueda cuál es la versión principal o preferida de una página cuando existen múltiples versiones similares o duplicadas. Su propósito es evitar problemas de contenido duplicado, consolidar la autoridad SEO y asegurar que el tráfico se dirija a la URL correcta.

De cara al usuario es cierto que no tiene un impacto visible, pero garantiza que siempre se muestre la versión más relevante del contenido en los resultados de búsqueda.

```
59 <link rel="canonical" href="https://www.mi-sitio.com/guia-seo">
```

5.1.5 Meta robots (<meta name="robots">)

La metaetiqueta robots le dice a los motores de búsqueda si deben indexarla o seguir sus enlaces. Es especialmente útil para controlar qué contenido deseas que aparezca o no en los resultados de búsqueda.

```
49  
50 <meta name="robots" content="noindex nofollow">  
51
```

A continuación, te explico cada una de las posibles opciones:

- **index:** instruye a los motores de búsqueda a indexar la página, es decir, incluirla en su base de datos para que aparezca en los resultados de búsqueda. Este es el comportamiento por defecto si no se especifica la etiqueta.
- **noindex:** indica que la página no debe ser incluida en los resultados de búsqueda. Es útil para páginas que no deseas que sean públicas, como páginas de inicio de sesión o versiones duplicadas de contenido.
- **follow:** permite a los motores de búsqueda rastrear y transmitir autoridad a los enlaces de la página. Este es el comportamiento predeterminado, incluso si no se incluye en la etiqueta.
- **nofollow:** indica que los motores de búsqueda no deben rastrear ni transmitir autoridad SEO a los enlaces de la página. Es útil para evitar promocionar enlaces externos irrelevantes o de baja calidad.

Así mismo, éstas etiquetas son combinables entre sí para generar diversas combinaciones:

- **index, follow:** La página será indexada y sus enlaces serán rastreados.

- index, nofollow: La página será indexada, pero los enlaces no transferirán autoridad SEO.
- noindex, follow: La página no será indexada, pero los enlaces seguirán siendo rastreados. Aunque a la larga Google bot deja de rastrearlo y las páginas acaban siendo huérfanas a efectos prácticos.
- noindex, nofollow: Ni la página será indexada, ni los enlaces serán rastreados.

5.1.6 Etiquetas Open Graph (<meta property="og:...">)

Las etiquetas Open Graph son metaetiquetas diseñadas para controlar cómo se presenta el contenido de una página en redes sociales como Facebook, LinkedIn y Twitter. Definen elementos clave como el título, la descripción y la imagen de vista previa que aparecen cuando alguien comparte un enlace.

Para los usuarios, mejoran la experiencia visual en redes sociales, haciendo que los enlaces sean más atractivos y clicables. Para los motores de búsqueda, aunque no impactan directamente el SEO, contribuyen al tráfico social. Un ejemplo sería:

```
51 <meta property="og:title" content="Guía completa de etiquetas SEO">
```

5.1.7 Etiqueta de título (<title>)

La etiqueta <title> define el título de la página web, que aparece tanto en la pestaña del navegador como en los resultados de búsqueda. Si bien es cierto que existe mucha controversia ya que hay gente que indica que no afecta al posicionamiento orgánico, la realidad es que afecta al CTR que ves en Google Search Console, lo cual indirectamente afecta al SEO.

De cara a los usuarios, un título claro y atractivo mejora la experiencia, ayudándolos a identificar rápidamente de qué trata una página. Un ejemplo optimizado sería:

```
53 <title>Guía completa sobre SEO para principiantes</title>
```

5.1.8 Meta descripción (<meta name="description">)

La meta descripción proporciona un resumen breve del contenido de la página que se muestra en los resultados de búsqueda, justo debajo del título. De nuevo, aunque no afecta directamente el posicionamiento, influye en el CTR (tasa de clics), ya que puede motivar al usuario a hacer clic en el enlace.

Un resumen conciso y relevante mejora la experiencia del usuario al saber qué esperar del contenido. Para los motores de búsqueda, sirve como una guía para destacar la relevancia de la página en una consulta específica. Un ejemplo sería:

```
65 <meta name="description" content="Aprende a optimizar tu página con
66 las mejores etiquetas SEO. Mejora tu visibilidad en buscadores con
67 esta guía completa.">
```

5.1.9 Etiqueta hreflang (<link rel="alternate" hreflang="">)

La etiqueta hreflang se usa para indicar el idioma y la región objetivo de una página web, siendo clave para sitios multilingües o multirregionales. Esto evita

confusiones entre versiones similares y asegura que los usuarios vean la página más relevante según su idioma o ubicación.

Para los motores de búsqueda, ayuda a evitar contenido duplicado y garantiza que la versión adecuada aparezca en los resultados. Un ejemplo de uso sería:

```
69 <link rel="alternate" hreflang="es-MX" href="https://www.mi-sitio.com/mx/">
```

5.1.10 Negritas (y)

Las etiquetas y permiten resaltar texto en negrita, pero con diferencias. tiene un valor semántico, indicando importancia al contenido, mientras que solo añade énfasis visual sin significado adicional.

Para los usuarios, destacan información clave dentro del texto. Para los motores de búsqueda, puede ayudar a identificar palabras relevantes relacionadas con las palabras clave. Por ejemplo:

```
71 <strong>SEO para principiantes</strong>
```

CAPÍTULO VI. USO DE TEMPLATE Y SLOT PARA CONTENIDO DINÁMICO

HTML moderno ha evolucionado para incluir elementos que facilitan la creación de contenido dinámico y reutilizable. Los elementos `<template>` y `<slot>` son dos herramientas fundamentales introducidas como parte de la especificación de Web Components, que permiten la creación de interfaces de usuario modulares, mantenibles y extensibles.

a. El elemento `<template>`

El elemento `<template>`, introducido en HTML5, proporciona un mecanismo para declarar fragmentos de código HTML que no se renderizan inmediatamente cuando se carga la página, pero pueden ser instanciados posteriormente mediante JavaScript. Este elemento actúa como un "plano" o "molde" para contenido que se generará dinámicamente.

Según la especificación oficial de HTML del WHATWG (Web Hypertext Application Technology Working Group):

"El elemento `template` es un mecanismo para contener contenido del lado del cliente que no se renderiza cuando se carga la página, pero que posteriormente puede ser instanciado durante el tiempo de ejecución utilizando JavaScript." (WHATWG, 2023)

b. El elemento `<slot>`

El elemento `<slot>` es parte de la especificación de Web Components y funciona en conjunto con Shadow DOM y Custom Elements. Este elemento define puntos de inserción dentro de un componente web donde se puede colocar contenido proveniente del exterior, permitiendo una composición flexible de componentes.

La especificación de Web Components define `<slot>` como:

"Un elemento `slot` representa un punto en un shadow tree donde se puede insertar contenido del árbol ligero del elemento." [23]

6.1 Fundamentos teóricos

Arquitectura de componentes web

Los elementos `<template>` y `<slot>` son parte de una arquitectura más amplia conocida como Web Components, que consiste en un conjunto de tecnologías que permiten crear elementos personalizados reutilizables para aplicaciones web:

- Custom Elements: API para definir nuevos elementos HTML
- Shadow DOM: Encapsulación de estilo y estructura
- HTML Templates: Declaración de fragmentos de marcado con `<template>`
- HTML Slots: Composición de contenido con `<slot>`
- Esta arquitectura promueve principios de diseño como:
- Encapsulación: Aislamiento de la estructura, estilo y comportamiento
- Reusabilidad: Creación de componentes que pueden utilizarse en múltiples contextos

- Composición: Combinación de componentes para crear interfaces más complejas
- Mantenibilidad: Separación de responsabilidades y facilidad de actualización

6.2 Casos de uso

- Los elementos `<template>` y `<slot>` son especialmente útiles en:
- Interfaces de usuario repetitivas: Listas, tablas, galerías
- Componentes configurables: Tarjetas, modales, notificaciones
- Sistemas de diseño: Creación de bibliotecas de componentes coherentes
- Aplicaciones de página única (SPA): Plantillas para vistas y componentes
- Renderizado condicional: Mostrar diferentes plantillas según el estado

6.3 Ventajas y limitaciones

a. Ventajas

- Rendimiento optimizado: El contenido de `<template>` no se procesa hasta que se necesita
- Separación de responsabilidades: Clara distinción entre estructura, contenido y comportamiento
- Reutilización de código: Reducción de la duplicación de HTML
- Mantenibilidad mejorada: Cambios en un componente se reflejan en todas sus instancias
- Compatibilidad con frameworks: Concepto similar al que utilizan frameworks como React, Vue o Angular

b. Limitaciones

- Soporte del navegador: Aunque ampliamente compatible, algunos navegadores antiguos no soportan estas características
- Curva de aprendizaje: Requiere comprensión de conceptos avanzados como Shadow DOM
- Ecosistema: Menos maduro que el de frameworks establecidos
- Depuración: El Shadow DOM puede complicar la inspección de elementos

6.4 Ejemplo 1

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Tarjetas de Usuario</title>
  <style>
    user-card {
      display: block;
```

```

        margin: 20px 0;
    }
</style>
</head>
<body>
    <h1>Lista de Usuarios</h1>

    <!-- Aquí irán las tarjetas -->
    <user-card>
        <span slot="name">Ana Ramírez</span>
        <span slot="email">ana@email.com</span>
        
    </user-card>

    <user-card>
        <span slot="name">Carlos López</span>
        <span slot="email">carlos@email.com</span>
        
    </user-card>

    <!-- Template oculto -->
    <template id="user-card-template">
        <style>
            .card {
                border: 1px solid #ccc;
                border-radius: 8px;
                padding: 15px;
                max-width: 300px; /* Limita el ancho máximo de la tarjeta */
                display: flex;
                align-items: center;
                gap: 10px;
                background-color: #f9f9f9;
                flex-wrap: wrap; /* Para que el contenido no se desborde */
                overflow: hidden; /* Asegura que los elementos dentro de la
tarjeta no se desborden */
            }
            img {
                width: 100px; /* Ajusta el tamaño a 100px de ancho */
                height: 100px; /* Ajusta el tamaño a 100px de alto */
                max-width: 100%; /* Asegura que la imagen no se haga más grande
de lo necesario */
                max-height: 100%; /* Asegura que la imagen no se haga más grande
de lo necesario */
                border-radius: 50%;
                object-fit: cover; /* Mantiene la proporción de la imagen sin
distorsionarla */
                flex-shrink: 0; /* Evita que la imagen se reduzca más de lo
necesario */
            }
        </style>
    </template>

```

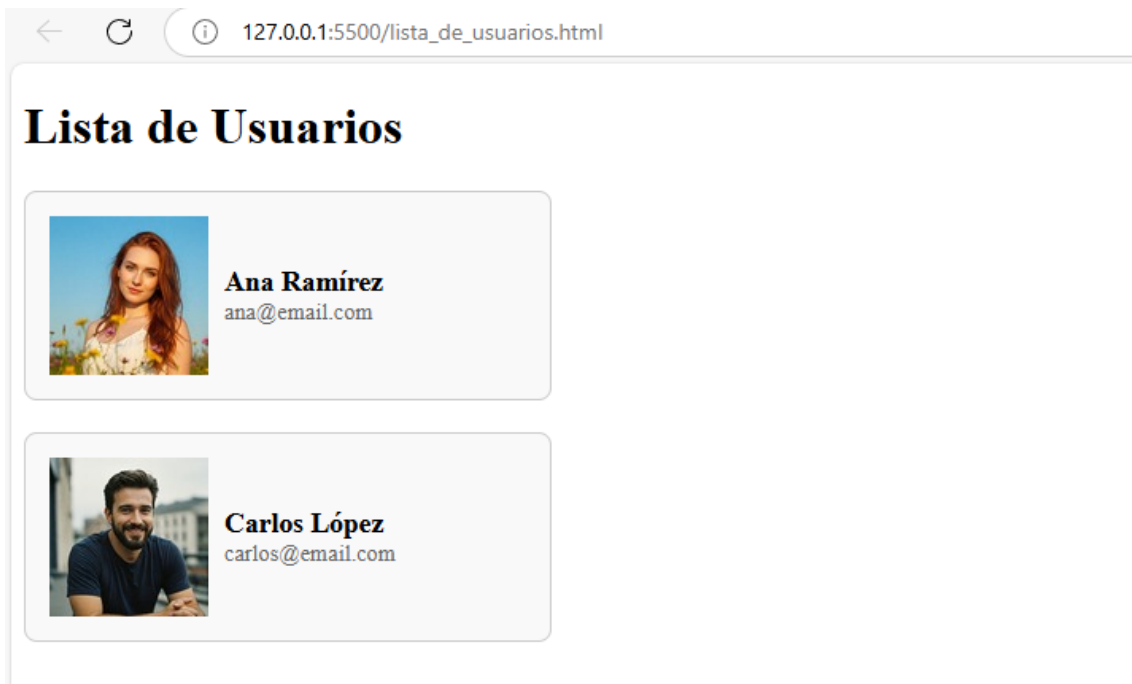
```

    }
    .info {
      display: flex;
      flex-direction: column;
      flex-grow: 1; /* Permite que el texto ocupe el espacio restante
*/
    }
    .name {
      font-weight: bold;
      font-size: 1.1em;
    }
    .email {
      font-size: 0.9em;
      color: #666;
    }
  }
</style>
<div class="card">
  <slot name="avatar"></slot>
  <div class="info">
    <div class="name"><slot name="name">Nombre no
disponible</slot></div>
    <div class="email"><slot name="email">Correo no
disponible</slot></div>
  </div>
</div>
</template>

<script>
  class UserCard extends HTMLElement {
    constructor() {
      super();
      const template = document.getElementById('user-card-template');
      const content = template.content.cloneNode(true);
      this.attachShadow({ mode: 'open' }).appendChild(content);
    }
  }

  customElements.define('user-card', UserCard);
</script>
</body>
</html>

```



6.4.1.1 Explicación del Código

a. HTML principal:

- Muestra una lista de usuarios (<user-card>) con nombre, correo electrónico e imagen.
- Cada tarjeta contiene datos que se insertan mediante elementos como o .

b. Template oculto (<template>):

- Define la plantilla de diseño que usarán todas las tarjetas.
- Dentro del <template>, hay estilos y estructura para mostrar correctamente cada tarjeta (imagen redonda, nombre en negrita, etc.).

c. JavaScript:

- Define un componente personalizado llamado <user-card>.
- Usa el contenido del template y lo inserta en un Shadow DOM (una especie de mini-DOM aislado para que el componente sea independiente del resto de la página).
- Esto permite que las tarjetas tengan su propio estilo sin interferencias externas.

6.5 Ejemplo 2

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```

<meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Template y Slot para Tarjetas de Producto</title>
  <style>
    /* Estilos generales */
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-
serif;
      line-height: 1.6;
      color: #333;
      max-width: 1200px;
      margin: 0 auto;
      padding: 20px;
    }

    .products-container {
      display: grid;
      grid-template-columns: repeat(auto-fill, minmax(300px,
1fr));
      gap: 20px;
      margin-top: 20px;
    }

    /* Estilos específicos para la tarjeta de producto */
    product-card {
      display: block;
      border: 1px solid #ddd;
      border-radius: 8px;
      overflow: hidden;
      transition: transform 0.3s ease, box-shadow 0.3s ease;
    }

    product-card:hover {
      transform: translateY(-5px);
      box-shadow: 0 10px 20px rgba(0,0,0,0.1);
    }
  </style>
</head>
<body>
  <h1>Catálogo de Productos</h1>

  <!-- Definición del template para la tarjeta de producto -->
  <template id="product-card-template">
    <div class="card">
      <div class="image-container">
        <slot name="product-image">

```



```

        <!-- Imagen predeterminada si no se proporciona
ninguna -->
        
    </slot>
</div>
<div class="content">
    <h2 class="title">
        <slot name="product-name">Producto sin nombre</slot>
    </h2>
    <p class="price">
        <slot name="product-price">$0.00</slot>
    </p>
    <div class="description">
        <slot name="product-description">Sin descripción
disponible</slot>
    </div>
    <div class="actions">
        <slot name="product-actions">
            <button class="default-button">Ver
detalles</button>
        </slot>
    </div>
</div>
</div>

<style>
    .card {
        background: white;
        height: 100%;
        display: flex;
        flex-direction: column;
    }

    .image-container {
        height: 200px;
        overflow: hidden;
    }

    .image-container img {
        width: 100%;
        height: 100%;
        object-fit: cover;
        transition: transform 0.5s ease;
    }

    .card:hover .image-container img {
        transform: scale(1.05);
    }

```

```

.content {
  padding: 15px;
  display: flex;
  flex-direction: column;
  flex-grow: 1;
}

.title {
  margin: 0 0 10px;
  font-size: 1.2rem;
}

.price {
  font-weight: bold;
  color: #e63946;
  font-size: 1.1rem;
  margin: 0 0 15px;
}

.description {
  flex-grow: 1;
  margin-bottom: 15px;
  font-size: 0.9rem;
  color: #666;
}

.actions {
  margin-top: auto;
}

button {
  background: #457b9d;
  color: white;
  border: none;
  padding: 8px 15px;
  border-radius: 4px;
  cursor: pointer;
  font-weight: 500;
  transition: background 0.3s ease;
}

button:hover {
  background: #1d3557;
}

.default-button {
  background: #ccc;
}

```

```

        color: #333;
    }

    .default-button:hover {
        background: #999;
    }
</style>
</template>

<!-- Contenedor para las tarjetas de producto -->
<div class="products-container">
    <!-- Instancia 1 de la tarjeta -->
    <product-card>
        
        <span slot="product-name">Laptop Ultradelgada Pro</span>
        <span slot="product-price">$1,299.99</span>
        <p slot="product-description">Potente laptop con procesador
de última generación, 16GB de RAM y 512GB SSD.</p>
        <div slot="product-actions">
            <button>Comprar ahora</button>
            <button>Añadir al carrito</button>
        </div>
    </product-card>

    <!-- Instancia 2 de la tarjeta -->
    <product-card>
        
        <span slot="product-name">Smartphone Galaxy X</span>
        <span slot="product-price">$899.99</span>
        <p slot="product-description">Teléfono inteligente con
pantalla AMOLED, cámara de 108MP y batería de larga duración.</p>
        <div slot="product-actions">
            <button>Ver especificaciones</button>
        </div>
    </product-card>

    <!-- Instancia 3 de la tarjeta (usando valores predeterminados)
-->
    <product-card>
        
        <span slot="product-name">Auriculares Inalámbricos</span>
        <span slot="product-price">$149.99</span>
    </product-card>
</div>

<script>

```

```

// Definición del componente personalizado ProductCard
class ProductCard extends HTMLElement {
  constructor() {
    super();

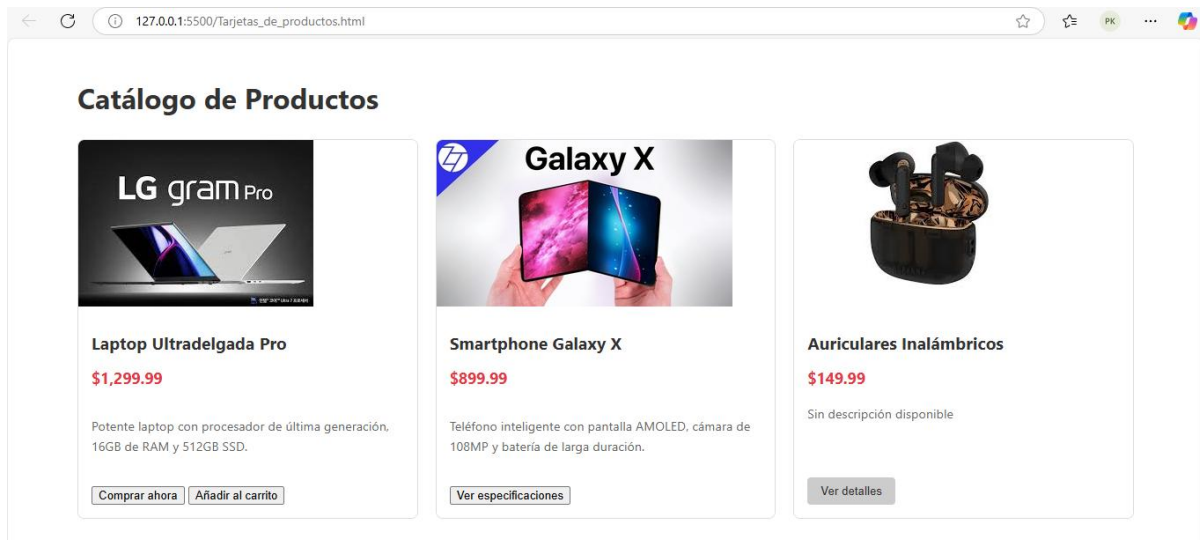
    // Crear shadow DOM
    this.attachShadow({ mode: 'open' });

    // Obtener el template
    const template = document.getElementById('product-card-
template');
    const templateContent = template.content;

    // Clonar el contenido del template y añadirlo al shadow
DOM
    this.shadowRoot.appendChild(templateContent.cloneNode
(true));
  }
}

// Registrar el componente personalizado
customElements.define('product-card', ProductCard);
</script>
</body>
</html>

```



6.5.1.1 Explicación del Código

a. Template del Componente (<template id="product-card-template">)

- Define cómo se ve cada tarjeta.

- Usa slots para que el contenido se pueda personalizar desde fuera (imagen, nombre, precio, descripción, acciones).
- Tiene estilos internos encapsulados para mantener el diseño aislado.

Tecnología	Uso
HTML5	Estructura semántica y slots
CSS Grid & Flex	Diseño de tarjetas responsive
Shadow DOM	Encapsulamiento visual
Custom Elements (Web Components)	Componente reutilizable <product-card>
JavaScript	Lógica para crear e insertar el componente

CAPÍTULO VII. MICRODATOS Y MARCADO SEMÁNTICO

En el desarrollo web moderno, el uso de tecnologías que mejoran la comprensión del contenido por parte de los motores de búsqueda y sistemas automatizados es crucial. Una de estas tecnologías es el marcado semántico, que permite añadir significado al contenido HTML. Dentro de este ámbito, los microdatos destacan como una forma estandarizada para incrustar metadatos dentro del contenido de las páginas web, mejorando la forma en que la información es interpretada por buscadores como Google. El uso de microdatos y marcado semántico avanzado permite una web más estructurada, accesible e inteligible tanto para humanos como para máquinas.

7.1 ¿Qué son los microdatos?

Los microdatos son una especificación del WHATWG (Web Hypertext Application Technology Working Group) para incrustar metadatos dentro de documentos HTML mediante atributos específicos. A diferencia de otros formatos como RDFa o JSON-LD, los microdatos se integran directamente dentro de las etiquetas HTML, lo que permite mantener el contenido y su significado estrechamente vinculados.

7.2 Sintaxis básica de microdatos

Los microdatos utilizan tres atributos fundamentales:

- `itemscope`: Indica que el elemento contiene un conjunto de propiedades relacionadas.
- `itemtype`: Especifica el tipo de elemento, definido por una URL (usualmente de Schema.org).
- `itemprop`: Define cada propiedad del tipo de elemento.

7.2.1 Ejemplo Básico

```
1 <div itemscope tipo de artículo="https://schema.org/Person">
2   <alcance itemprop="nombre">Cristián Machado</alcance>
3   <alcance itemprop="jobTítulo">Desarrollador Web</alcance>
4   <alcance itemprop="correo
electrónico">cris6mc@gmail.com</alcance>
5 </div>
6
```

Cristian MachadoDesarrollador Webcris6mc@gmail.com

7.3 Uso de Schema.org

Schema.org es una iniciativa conjunta de Google, Microsoft, Yahoo y Yandex, que proporciona un vocabulario estandarizado para describir distintos tipos de datos, como personas, productos, organizaciones, eventos, entre otros.

7.3.1 Ejemplo con productos

```
1 <div itemscope tipo de artículo="https://schema.org/Person"><div
  itemscope tipo de artículo="https://schema.org/Product">
2   <alcance itemprop="nombre">Smartphone Galaxy
    A52</alcance>
3   <alcance itemprop="descripción">Teléfono inteligente con
    pantalla AMOLED de 6.5".</alcance>
4   <alcance itemprop="ofertas" itemscope tipo de
    artículo="https://schema.org/Offer">
5     Precio: <alcance itemprop="precio">899</alcance> <alcance
    itemprop="precioCurrency">PLUMA</alcance>
6   </alcance>
7 </div>
```

Smartphone Galaxy A52Teléfono inteligente con pantalla AMOLED de 6.5". Precio: 899PEN

7.4 Ventajas del uso de microdatos

- Mejor SEO (Search Engine Optimization): Los microdatos ayudan a los motores de búsqueda a comprender el contenido y generar rich snippets (fragmentos enriquecidos) en los resultados.
- Mayor accesibilidad: Las tecnologías asistivas pueden interpretar mejor la estructura y significado del contenido.
- Interoperabilidad: Facilita el uso de la información en diferentes plataformas, como asistentes virtuales o redes sociales.

7.5 Comparación con otros métodos de marcado semántico

7.5.1 Tabla comparativa

Tecnología	Integración	Legibilidad	Popularidad
Microdatos	Dentro del HTML	Alta	Alta
RDFa	Dentro del HTML	Media	Baja
JSON-LD	En bloque separado (script)	Alta	Muy alta (preferida por Google)

Aunque JSON-LD es la forma recomendada por Google en muchos casos, los microdatos siguen siendo ampliamente utilizados, especialmente cuando se desea mantener el significado semántico junto al contenido visible.

Como conclusión, el uso de microdatos y marcado semántico avanzado es esencial para crear sitios web más comprensibles, accesibles y bien posicionados. Permiten que los motores de búsqueda interpreten mejor el contenido, mejorando su presentación en los resultados y facilitando la integración con tecnologías emergentes. Adoptar estas prácticas no solo mejora la experiencia del usuario, sino que también alinea los desarrollos web con los estándares actuales de la web semántica.

CAPÍTULO VIII. INTEGRACIÓN DE SVG Y CANVAS PARA GRÁFICOS

8.1 Integración de SVG

SVG (Scalable Vector Graphics) es un lenguaje de marcado basado en XML diseñado para describir gráficos vectoriales bidimensionales. A diferencia de las imágenes de tipo ráster, que están formadas por píxeles fijos, los gráficos SVG se definen mediante formas geométricas y coordenadas, lo que les permite escalarse a cualquier tamaño sin perder calidad ni nitidez. Esto lo hace ideal para gráficos en entornos web responsivos, ya que puede ser ampliado o reducido sin perder claridad visual [24].

Una de las ventajas significativas de SVG es que, al ser parte del DOM (Modelo de Objetos del Documento), sus elementos pueden ser accedidos y manipulados fácilmente mediante CSS o JavaScript. Esto permite realizar modificaciones dinámicas de los gráficos, como cambiar colores, tamaños o realizar animaciones interactivas. Además, cualquier texto dentro de la imagen sigue siendo legible y accesible para los motores de búsqueda y tecnologías de asistencia, lo que mejora la accesibilidad y el SEO de la página web [25].

8.1.1 Ejemplo Simple de Integración de SVG

Este ejemplo muestra un círculo que cambia de color cuando el usuario pasa el mouse sobre él, utilizando CSS para la interactividad.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>SVG Interactivo</title>
</head>
<body>
  <div>
    <h1>SVG Interactivo</h1>
    <svg>
      <circle
        class="interactive-circle"
        cx="50%"
        cy="50%"
        r="50%"
      />
    </svg>
  </div>
</body>
</html>
```

```
body {
  font-family: Arial, sans-serif;
  background-color: #f0f0f0;
  text-align: center;
  padding: 20px;
}

h1 {
  color: #333;
}

svg {
  border: 1px solid #ccc;
  margin-top: 20px;
}

.interactive-circle {
  transition: fill 0.3s ease;
}
```



```

    .interactive-circle:hover {
        fill: orange;
    }
</style>
</head>
<body>
    <h1>SVG con Círculo Interactivo</h1>

    <!-- SVG con efecto hover -->
    <svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
        <circle cx="100" cy="100" r="80" fill="blue" class="interactive-circle" />
    </svg>
</body>
</html>

```

8.1.1.1 Explicación del Código

a. Estructura HTML

- El código utiliza HTML5 para definir la estructura básica de la página web, que incluye un título y un SVG con un círculo interactivo.
- Se incorpora un título <h1> que explica el propósito de la página: "SVG con Círculo Interactivo".
- Dentro del elemento <svg>, se define un círculo con un radio de 80 píxeles, ubicado en el centro de un lienzo de 200x200 píxeles.
- El círculo está inicialmente coloreado de azul y se le asigna la clase interactive-circle para poder aplicar estilos dinámicos a través de CSS.

b. Estilos CSS

- Los estilos definidos en la sección <style> controlan la apariencia visual del contenido.
- El fondo de la página se establece en un color gris claro (#f0f0f0), y el texto se centra horizontalmente para mejorar la presentación.
- El círculo dentro del SVG tiene un borde gris claro y una transición suave definida para cambiar el color de relleno al interactuar con él.
- La clase interactive-circle tiene una propiedad transition que aplica un cambio suave en el color de relleno cuando el usuario pasa el cursor sobre el círculo, cambiando su color de azul a naranja.

c. Interactividad y Lógica CSS

- El SVG es interactivo gracias al efecto hover aplicado en el círculo.
- Cuando el usuario pasa el cursor sobre el círculo, el color de relleno cambia de azul a naranja con una transición de 0.3 segundos, proporcionando una respuesta visual dinámica.
- Esta interactividad está totalmente gestionada con CSS, sin necesidad de JavaScript, haciendo que el comportamiento sea ligero y eficiente.

d. Interacción del Usuario

- El usuario interactúa con el SVG simplemente pasando el cursor sobre el círculo.
- El efecto visual de cambio de color ocurre de forma inmediata, lo que mejora la experiencia del usuario sin requerir ninguna acción adicional.
- El diseño es simple y funcional, adecuado para mostrar cómo los SVGs pueden incluir elementos interactivos usando solo HTML y CSS.

e. Resultado:



8.1.2 Ejemplo Complejo de Integración de SVG

Este ejemplo utiliza CSS para animar un círculo que rota continuamente, mostrando cómo se pueden aplicar animaciones dentro de un gráfico SVG.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
```

```

<title>SVG + Imagen Giratoria</title>
<style>
  body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    text-align: center;
    padding: 20px;
  }

  h1 {
    color: #333;
  }

  svg {
    border: 1px solid #ccc;
    margin-top: 20px;
  }

  .rotating-group {
    transform-origin: center;
    animation: rotate 4s linear infinite;
  }

  @keyframes rotate {
    from {
      transform: rotate(0deg);
    }
    to {
      transform: rotate(360deg);
    }
  }
</style>
</head>
<body>
  <h1>SVG con Círculo e Imagen Girando</h1>

  <!-- SVG que contiene un círculo y una imagen -->
  <svg width="300" height="300" xmlns="http://www.w3.org/2000/svg">
    <g class="rotating-group">
      <!-- Círculo giratorio -->
      <circle
        cx="150"
        cy="150"
        r="100"
        fill="none"
        stroke="blue"
        stroke-width="5"
      />

```

```

<!-- Imagen HTML5 girando junto al círculo -->
<image
  href="https://upload.wikimedia.org/wikipedia/commons/thumb/6/61/HTML5_logo_and_wordmark.svg/512px-HTML5_logo_and_wordmark.svg.png"
  x="100"
  y="100"
  width="100"
  height="100"
/>
</g>
</svg>
</body>
</html>

```

8.1.2.1 Explicación del Código

a. Estructura HTML

- El código utiliza HTML5 para definir la estructura básica de la página web, que incluye un título y un SVG con un círculo interactivo.
- Se incorpora un título <h1> que explica el propósito de la página: "SVG con Círculo Interactivo".
- Dentro del elemento <svg>, se define un círculo con un radio de 80 píxeles, ubicado en el centro de un lienzo de 200x200 píxeles.
- El círculo está inicialmente coloreado de azul y se le asigna la clase interactive-circle para poder aplicar estilos dinámicos a través de CSS.

b. Estilos CSS

- Los estilos definidos en la sección <style> controlan la apariencia visual del contenido.
- El fondo de la página se establece en un color gris claro (#f0f0f0), y el texto se centra horizontalmente para mejorar la presentación.
- El círculo dentro del SVG tiene un borde gris claro y una transición suave definida para cambiar el color de relleno al interactuar con él.
- La clase interactive-circle tiene una propiedad transition que aplica un cambio suave en el color de relleno cuando el usuario pasa el cursor sobre el círculo, cambiando su color de azul a naranja.

c. Interactividad y Lógica CSS

- El SVG es interactivo gracias al efecto hover aplicado en el círculo.

- Cuando el usuario pasa el cursor sobre el círculo, el color de relleno cambia de azul a naranja con una transición de 0.3 segundos, proporcionando una respuesta visual dinámica.
- Esta interactividad está totalmente gestionada con CSS, sin necesidad de JavaScript, haciendo que el comportamiento sea ligero y eficiente.

d. Interacción del Usuario

- El usuario interactúa con el SVG simplemente pasando el cursor sobre el círculo.
- El efecto visual de cambio de color ocurre de forma inmediata, lo que mejora la experiencia del usuario sin requerir ninguna acción adicional.
- El diseño es simple y funcional, adecuado para mostrar cómo los SVGs pueden incluir elementos interactivos usando solo HTML y CSS.

e. Resultado:



8.2 Integración de Canvas

Canvas es una tecnología de gráficos introducida en HTML5 que proporciona un enfoque procedural para dibujar gráficos directamente sobre un lienzo de píxeles. A diferencia de SVG, que utiliza un enfoque declarativo, Canvas permite dibujar formas, líneas y otros elementos utilizando comandos de JavaScript. El elemento `<canvas>` actúa como un lienzo en blanco en el que se pueden trazar gráficos de forma dinámica, lo que lo hace ideal para crear gráficos interactivos, visualizaciones de datos o juegos en tiempo real.

Una de las características clave de Canvas es que, una vez que se ha dibujado un gráfico, el lienzo no conserva una estructura de objetos; es decir, los gráficos se representan como píxeles en el lienzo, lo que hace que no sean accesibles por el DOM ni por tecnologías de asistencia. Esto significa que Canvas no es ideal para gráficos que necesiten ser manipulados de manera accesible o ser interactivos a nivel de objetos específicos [26].

8.2.1 Ejemplo Simple de Integración de Canvas

Este ejemplo muestra un círculo que rebota dentro de un lienzo utilizando JavaScript para la animación.

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <title>Pelota Rebotando en Canvas</title>
    <style>
      body {
        font-family: Arial, sans-serif;
        background-color: #f0f0f0;
        text-align: center;
        padding: 20px;
      }

      h1 {
        color: #333;
      }

      canvas {
        background-color: #fff;
        border: 1px solid #000;
        margin-top: 20px;
      }
    </style>
  </head>
  <body>
    <h1>Pelota Rebotando</h1>
    <canvas id="miCanvas" width="400" height="400"></canvas>

    <script>
      const canvas = document.getElementById("miCanvas");
      const ctx = canvas.getContext("2d");
      let x = canvas.width / 2;
      let y = canvas.height - 30;
      let dx = 2;
      let dy = -2;
      const ballRadius = 10;

      function drawBall() {
        ctx.beginPath();
        ctx.arc(x, y, ballRadius, 0, Math.PI * 2);
        ctx.fillStyle = "#0095DD";
        ctx.fill();
        ctx.closePath();
      }

      function draw() {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        drawBall();
      }
    </script>
  </body>
</html>

```

```

// Rebote en los bordes
if (x + dx > canvas.width - ballRadius || x + dx < ballRadius) {
    dx = -dx;
}
if (y + dy > canvas.height - ballRadius || y + dy < ballRadius) {
    dy = -dy;
}

x += dx;
y += dy;

requestAnimationFrame(draw);
}

draw(); // Inicia la animación
</script>
</body>
</html>

```

8.2.1.1 Explicación del Código

a. Estructura HTML

- El código utiliza HTML5 para crear la estructura básica de la página web, que incluye un título y un elemento <canvas>.
- El título <h1> describe la funcionalidad de la página: "Pelota Rebotando".
- El elemento <canvas> tiene un identificador único miCanvas y un tamaño de 400x400 píxeles. Este lienzo es donde se dibuja la pelota y se gestionan sus movimientos.

b. Estilos CSS

- Se aplican estilos básicos a la página y el lienzo para darle una apariencia limpia y centrada.
- El fondo de la página se establece en un gris claro, y el lienzo tiene un fondo blanco con un borde negro para que sea visible.
- El texto en el título se muestra en un color gris oscuro para garantizar su legibilidad.

c. Lógica con JavaScript

- Se obtiene el contexto del lienzo con getContext("2d"), lo que permite dibujar en 2D dentro del <canvas>.
- Se definen varias variables: la posición inicial de la pelota (x y y), su velocidad en ambas direcciones (dx y dy), y su radio (ballRadius).

- La función `drawBall()` es responsable de dibujar la pelota en el lienzo. Se usa `ctx.arc()` para dibujar el círculo y `ctx.fill()` para darle color.
- La función `draw()` se encarga de la animación: limpia el lienzo con `clearRect()`, vuelve a dibujar la pelota y calcula su nueva posición basándose en su velocidad.
- Si la pelota toca los bordes del lienzo, la dirección de su movimiento se invierte, logrando el efecto de rebote.

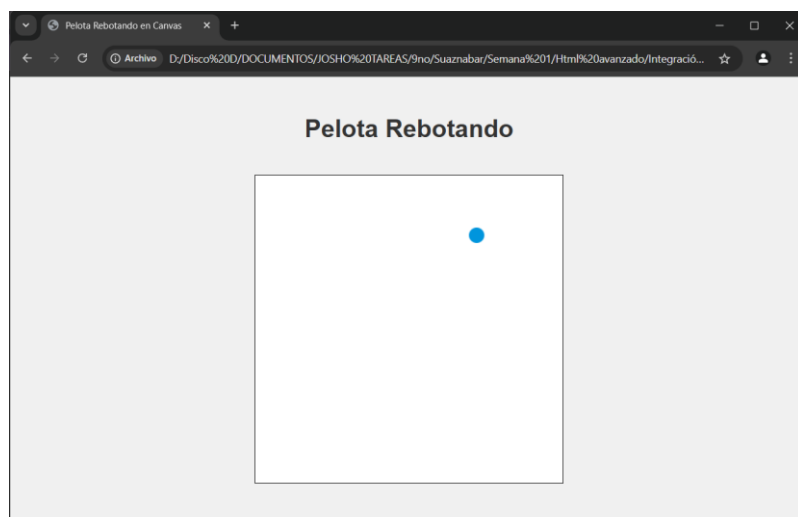
d. Interactividad y Animación

- La animación se ejecuta de manera continua gracias a la función `requestAnimationFrame(draw)`, que redibuja la pelota en cada cuadro.
- El rebote de la pelota está controlado por condiciones que verifican si la pelota ha tocado los bordes del lienzo. Si es así, la velocidad en la dirección correspondiente se invierte, causando el rebote.
- La pelota se mueve de forma continua y suave, dando la sensación de que está rebotando dentro del área del lienzo.

e. Interacción del Usuario

- El usuario observa cómo la pelota rebota automáticamente dentro del lienzo sin necesidad de intervención.
- La animación es completamente gestionada por JavaScript, creando una experiencia interactiva y fluida.
- El lienzo se actualiza en tiempo real para que la pelota siga un trayecto coherente y continuo, sin la necesidad de recargar la página.

f. Resultado:



8.2.2 Ejemplo Complejo de Integración de Canvas

El código dibuja un círculo inicial con partículas en un canvas HTML5 y permite crear más partículas al mover el mouse. Las partículas se mueven, caen con gravedad y desaparecen lentamente simulando desvanecimiento.

```
<canvas
  id="miCanvas"
  width="400"
  height="400"
  style="border: 1px solid #000000"
></canvas>
<script>
const canvas = document.getElementById("miCanvas");
const ctx = canvas.getContext("2d");
const particles = [];

function Particle(x, y) {
  this.x = x;
  this.y = y;
  this.size = Math.random() * 3 + 1.5;
  this.speedX = Math.random() * 2 - 1;
  this.speedY = Math.random() * -2 - 0.5;
  this.color = "rgba(0, 150, 255, 0.8)";
}

Particle.prototype.update = function () {
  this.x += this.speedX;
  this.y += this.speedY;
  this.speedY += 0.03; // Simula la gravedad
  if (this.size > 0.2) this.size -= 0.02; // Se reduce más lento
};

Particle.prototype.draw = function () {
  ctx.fillStyle = this.color;
  ctx.beginPath();
  ctx.arc(this.x, this.y, this.size, 0, Math.PI * 2);
  ctx.fill();
};

function createParticles(x, y, cantidad = 10) {
  for (let i = 0; i < cantidad; i++) {
    particles.push(new Particle(x, y));
  }
}

function crearFiguraCircular() {
  const centroX = canvas.width / 2;
  const centroY = canvas.height / 2;
```

```

const radio = 80;

for (let i = 0; i < 360; i += 10) {
  const ang = (i * Math.PI) / 180;
  const x = centroX + radio * Math.cos(ang);
  const y = centroY + radio * Math.sin(ang);
  createParticles(x, y, 5); // más partículas por punto
}
}

function animateParticles() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  for (let i = 0; i < particles.length; i++) {
    particles[i].update();
    particles[i].draw();
    if (particles[i].size <= 0.2) {
      particles.splice(i, 1);
      i--;
    }
  }
  requestAnimationFrame(animateParticles);
}

// Crear figura una vez al inicio
crearFiguraCircular();

// Partículas interactivas con el mouse
canvas.addEventListener("mousemove", function (e) {
  const rect = canvas.getBoundingClientRect();
  const x = e.clientX - rect.left;
  const y = e.clientY - rect.top;
  createParticles(x, y);
});

animateParticles();
</script>

```

8.2.2.1 Explicación del Código

a. Estructura HTML

- El código utiliza un elemento `<canvas>` en HTML, el cual se utiliza como lienzo para mostrar la animación.
- El canvas tiene un tamaño de 400x400 píxeles y un borde negro (`style="border: 1px solid #000000"`), lo que hace que sea visible en la página.

b. Lógica con JavaScript

- **Variables y Objetos:**
 - Se obtiene el contexto 2D del lienzo con `canvas.getContext("2d")`, lo cual permite dibujar sobre él.
 - Se crea un arreglo vacío `particles` que almacenará las partículas generadas.
- **Constructor de Partículas:**
 - `Particle` es una función constructora que define cada partícula. Las partículas tienen propiedades como:
 - **Posición (x, y)**
 - **Tamaño (size)** aleatorio entre 1.5 y 4.5 píxeles.
 - **Velocidad (speedX, speedY)** aleatoria para simular el movimiento.
 - **Color** en color RGBA con una opacidad del 80% (`rgba(0, 150, 255, 0.8)`).
- **Métodos de la Partícula:**
 - `update()`: Actualiza la posición de la partícula, simulando la gravedad al incrementar la velocidad en el eje Y y disminuyendo su tamaño gradualmente.
 - `draw()`: Dibuja la partícula como un círculo en el lienzo con el color y tamaño definidos.
- **Función `createParticles`:**
 - Crea varias partículas en la ubicación especificada por las coordenadas x y y. La cantidad de partículas se puede ajustar.
- **Función `crearFiguraCircular`:**
 - Crea una figura circular de partículas distribuidas de manera uniforme sobre el contorno de un círculo con centro en el medio del lienzo. Se genera cada partícula a intervalos de 10 grados.
- **Función `animateParticles`:**
 - Se encarga de animar las partículas: limpia el lienzo, actualiza la posición de cada partícula y la dibuja. Además, elimina las partículas que han alcanzado un tamaño muy pequeño (≤ 0.2).

c. Interactividad con el Usuario

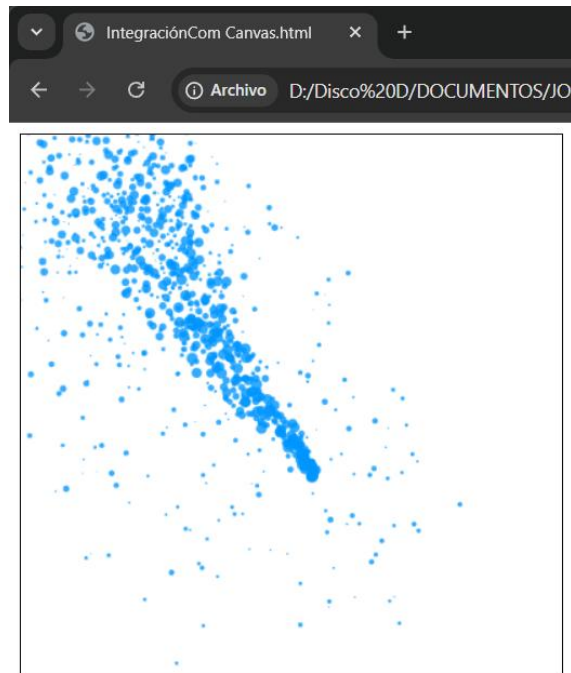
- **Evento de Mouse (`mousemove`):**
 - Al mover el mouse sobre el lienzo, se generan nuevas partículas en la posición del cursor, agregando un efecto interactivo al lienzo.
- **Animación:**
 - `animateParticles()` se ejecuta continuamente mediante `requestAnimationFrame(animateParticles)`, lo que garantiza una animación suave en el lienzo.

d. Comportamiento del Sistema

- Las partículas caen debido a la gravedad simulada y disminuyen su tamaño lentamente hasta desaparecer.
- Al mover el mouse sobre el lienzo, se generan nuevas partículas, proporcionando interactividad y un efecto visual dinámico.

- Inicialmente, se crea una figura circular de partículas en el centro del lienzo para dar un punto de partida visual.

e. **Resultado:**



CAPÍTULO IX. INTEGRACIÓN CON JAVA SCRIPT: EVENTOS Y MANIPULACIÓN DEL DOM

JavaScript es un lenguaje de programación fundamental en el desarrollo web moderno. Su integración con HTML permite crear páginas dinámicas e interactivas. Dos de los aspectos clave de esta integración son el manejo de eventos y la manipulación del DOM (Document Object Model). Estas técnicas permiten responder a las acciones del usuario, modificar elementos del sitio en tiempo real y construir experiencias ricas y reactivas. En este documento, exploraremos cómo funciona esta integración y presentaremos ejemplos prácticos que ilustran su aplicación.

9.1 ¿Qué es el DOM?

El DOM (Document Object Model) es una representación estructurada del contenido HTML como un árbol de nodos. Cada elemento HTML (etiqueta, texto, atributo) se convierte en un nodo accesible mediante JavaScript, lo que permite modificar la estructura, el estilo y el contenido de la página de forma dinámica.

9.2 Eventos en JavaScript

Los eventos son acciones que ocurren en la página web, como clics, movimientos del mouse, entrada de datos en formularios, entre otros. JavaScript permite detectar y responder a estos eventos mediante el uso de funciones conocidas como manejadores de eventos.

9.2.1 Métodos para manejar eventos

- Atributos HTML (obsoleto): `<button onclick='funcion()>`
- Propiedades del DOM: `element.onclick = funcion;`
- Método `addEventListener`: `element.addEventListener('click', funcion);` (recomendado)

9.3 Manipulación del DOM

Mediante JavaScript, se puede acceder y modificar elementos del DOM utilizando diferentes métodos:

- `getElementById(id)`: Selecciona un elemento por su ID.
- `getElementsByClassName(clase)`: Selecciona todos los elementos con la clase especificada.
- `querySelector(selector)`: Selecciona el primer elemento que coincida con el selector CSS.
- `innerHTML`, `textContent`: Permiten modificar el contenido de los elementos.
- `style`: Permite cambiar los estilos en línea de los elementos.

9.4 Ejemplo práctico

El siguiente ejemplo muestra cómo cambiar el contenido de un elemento al hacer clic en un botón:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <title>Eventos y DOM</title>
5  </head>
6  <body>
7  |   <h2 id="titulo">Texto original</h2>
8  |   <button onclick="cambiarTexto()">Cambiar Texto</button>
9
10 |   <script>
11 |     function cambiarTexto() {
12 |       document.getElementById("titulo").textContent = "Texto modificado con JavaScript";
13 |     }
14 |   </script>
15 </body>
16 </html>
17

```

Texto original

Cambiar Texto

Texto modificado con JavaScript

Cambiar Texto

Este ejemplo utiliza un atributo `onclick` para llamar a la función `cambiarTexto()`, que modifica el contenido del elemento con ID 'titulo'.

Como conclusión, la integración de JavaScript con HTML mediante eventos y manipulación del DOM es fundamental para el desarrollo de interfaces interactivas. Permite modificar el contenido y la estructura del sitio en tiempo real, respondiendo a las acciones del usuario. Dominar estas técnicas es esencial para crear experiencias web modernas, accesibles y dinámicas.

CAPÍTULO X. API DE HTML 5 – DRAG AND DROP, GEOLOCATION, WEB STORAGE (LOCALSTORAGE Y SESSIONSTORAGE)

Las APIs de HTML5 representan un conjunto de herramientas poderosas que permiten a los desarrolladores crear aplicaciones web modernas con funcionalidades avanzadas directamente en el navegador, sin necesidad de plugins externos. En esta monografía nos enfocaremos en tres APIs fundamentales:

- **Drag and Drop:** Permite implementar interfaces donde los usuarios pueden arrastrar y soltar elementos visuales
- **GeoLocation:** Proporciona acceso a la ubicación geográfica del dispositivo
- **Web Storage:** Ofrece mecanismos para almacenar datos persistentes (LocalStorage) o temporales (SessionStorage)

Estas tecnologías son esenciales para crear aplicaciones web interactivas, personalizadas según ubicación y con capacidad de trabajar offline.

10.1 API Drag and Drop

10.1.1 Conceptos básicos

La API Drag and Drop se compone de varios eventos que permiten controlar todo el proceso de arrastrar y soltar elementos:

- **Elemento arrastrable:** Cualquier elemento HTML con el atributo `draggable="true"`
- **Zona de destino:** Área que puede recibir elementos arrastrados
- **Transferencia de datos:** Mecanismo para pasar información entre el origen y destino

10.1.2 Eventos clave

EVENTO	DESCRIPCION
drag	Ocurre continuamente mientras el elemento se está arrastrando
dragenter	Cuando el elemento arrastrado entra en un objetivo válido
dragover	Se ejecuta constantemente mientras el elemento está sobre el objetivo
dragleave	Cuando el elemento arrastrado sale de un objetivo válido
drop	Se activa cuando el elemento se suelta sobre un objetivo válido
dragend	Ocurre cuando termina la operación de arrastre (éxito o cancelación)

10.1.3 Ejemplo

```
<!DOCTYPE html>
```

```
<html>
<head>
  <style>
    /* Estilo para el elemento arrastrable */
    #dragItem {
      width: 100px;
      height: 100px;
      background-color: #4CAF50;
      color: white;
      text-align: center;
      line-height: 100px;
      cursor: move;
    }

    /* Estilo para la zona de destino */
    #dropZone {
      width: 300px;
      height: 200px;
      border: 3px dashed #ccc;
      margin-top: 20px;
      padding: 10px;
      text-align: center;
    }

    /* Efecto visual cuando un elemento está sobre la zona */
    #dropZone.highlight {
      border-color: #ff5722;
      background-color: #ffe0b2;
    }
  </style>
</head>
```



```

<body>

  <!-- Elemento que se puede arrastrar -->
  <div id="dragItem" draggable="true" ondragstart="dragStart(event)">

    Arrástrame

  </div>


  <!-- Zona donde se puede soltar el elemento -->
  <div id="dropZone"

    ondragenter="dragEnter(event)"
    ondragover="dragOver(event)"
    ondragleave="dragLeave(event)"
    ondrop="drop(event)">

    Suelta aquí

  </div>


  <script>

    // Variable para almacenar el ID del elemento arrastrado
    let draggedItemId;


    // Se ejecuta cuando comienza el arrastre
    function dragStart(event) {

      // Almacenamos el ID del elemento arrastrado
      draggedItemId = event.target.id;


      // Establecemos los datos que se transferirán
      event.dataTransfer.setData("text/plain", event.target.id);


      // Efecto visual del arrastre
      event.dataTransfer.effectAllowed = "move";


      // Opcional: Cambiar aspecto del elemento durante el arrastre

```

```
event.target.style.opacity = "0.4";
}

// Cuando el elemento entra en la zona de destino
function dragEnter(event) {
    event.preventDefault();

    // Añadimos clase para resaltar visualmente
    event.target.classList.add("highlight");
}

// Mientras el elemento está sobre la zona
function dragOver(event) {
    event.preventDefault();

    // Especificamos el efecto deseado (move, copy, link, etc.)
    event.dataTransfer.dropEffect = "move";
}

// Cuando el elemento sale de la zona
function dragLeave(event) {
    event.preventDefault();

    // Quitamos el resaltado
    event.target.classList.remove("highlight");
}

// Cuando se suelta el elemento
function drop(event) {
    event.preventDefault();

    // Quitamos el resaltado
    event.target.classList.remove("highlight");
}
```

```

// Obtenemos los datos transferidos
const data = event.dataTransfer.getData("text/plain");

// Obtenemos el elemento arrastrado
const draggedElement = document.getElementById(data);

// Lo añadimos a la zona de destino
event.target.appendChild(draggedElement);

// Restauramos la opacidad
draggedElement.style.opacity = "1";

// Mostramos mensaje de éxito
event.target.innerHTML += `<p>Elemento ${data} soltado
correctamente!</p>`;
}
</script>
</body>
</html>

```

10.1.3.1 Análisis del código

a. Estructura HTML:

- `draggable="true"`: Hace que el div sea arrastrable
- Los eventos están definidos directamente en los elementos HTML

b. Estilos CSS:

- `.highlight`: Clase que se añade/remueve dinámicamente para feedback visual
- Se definen aspectos visuales tanto para el elemento arrastrable como para la zona de destino

c. Lógica JavaScript:

- `dataTransfer`: Objeto clave que permite transferir información entre origen y destino

- `effectAllowed` y `dropEffect`: Controlan los efectos visuales y el tipo de operación
- `preventDefault()`: Fundamental para evitar el comportamiento por defecto del navegador

d. Flujo completo:

- El usuario comienza arrastrando el elemento verde (`dragstart`)
- Al entrar en la zona destino, esta se resalta (`dragenter`)
- Mientras está sobre la zona, se mantiene el estado (`dragover`)
- Si sale sin soltar, se quita el resaltado (`dragleave`)
- Al soltar (`drop`), el elemento se añade a la zona y se muestra un mensaje

10.2 API GeoLocation

10.2.1 Fundamentos

La API GeoLocation permite a las aplicaciones web acceder a la ubicación geográfica del dispositivo del usuario, siempre con su consentimiento. Utiliza múltiples fuentes de información:

- GPS (más preciso)
- Redes WiFi
- Torres de telefonía móvil
- Dirección IP (menos preciso)

10.2.2 Métodos principales

Método	Descripción
<code>navigator.geolocation</code>	Objeto principal que contiene toda la funcionalidad
<code>getCurrentPosition()</code>	Obtiene la posición actual una sola vez
<code>watchPosition()</code>	Observa cambios en la posición (útil para aplicaciones en movimiento)
<code>clearWatch()</code>	Detiene el seguimiento iniciado con <code>watchPosition</code>

10.2.3 Ejemplo

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Prueba de Geolocalización</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin-top: 50px;
    }
    #location {
      margin-top: 20px;
      padding: 15px;
      border: 1px solid #ddd;
      border-radius: 5px;
      display: inline-block;
      min-width: 300px;
    }
    button {
      padding: 10px 20px;
      background-color: #4CAF50;
      color: white;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      font-size: 16px;
    }
    button:hover {
      background-color: #45a049;
    }
    .error {
      color: red;
```

```

    }
</style>
</head>
<body>
    <h2>Prueba de Geolocalización</h2>
    <button onclick="getLocation()">Obtener Mi Ubicación</button>

    <div id="location">
        <p>Presiona el botón para obtener tu ubicación</p>
    </div>

    <script>
        function getLocation() {
            const locationDiv = document.getElementById("location");
            locationDiv.innerHTML = "<p>Obteniendo ubicación...</p>";

            if (navigator.geolocation) {
                navigator.geolocation.getCurrentPosition(
                    showPosition,
                    showError,
                    {
                        enableHighAccuracy: true,
                        timeout: 10000,
                        maximumAge: 0
                    }
                );
            } else {
                locationDiv.innerHTML = '<p class="error">Tu navegador no soporta geolocalización</p>';
            }
        }
    </script>

```

```

function showPosition(position) {
    const locationDiv = document.getElementById("location");
    locationDiv.innerHTML = `
        <p><strong>Ubicación obtenida:</strong></p>
        <p>Latitud: ${position.coords.latitude}</p>
        <p>Longitud: ${position.coords.longitude}</p>
        <p>Precisión: ${position.coords.accuracy} metros</p>
        <p><small>Hora:                                ${new
Date(position.timestamp).toLocaleTimeString()}</small></p>
    `;
}

function showError(error) {
    const locationDiv = document.getElementById("location");
    let message = "";

    switch(error.code) {
        case error.PERMISSION_DENIED:
            message = "Permiso denegado por el usuario";
            break;
        case error.POSITION_UNAVAILABLE:
            message = "La información de ubicación no está disponible";
            break;
        case error.TIMEOUT:
            message = "La solicitud tardó demasiado tiempo";
            break;
        case error.UNKNOWN_ERROR:
            message = "Ocurrió un error desconocido";
            break;
    }

    locationDiv.innerHTML = `<p class="error">Error: ${message}</p>`;
}

```

```
}  
</script>  
</body>  
</html>
```

10.2.3.1 Explicación del código

a. Interfaz Simple

- **Un botón principal:** "Obtener Mi Ubicación" para iniciar el proceso
- **Área de resultados:** Un recuadro donde se muestra:
 - Mensaje inicial instructivo
 - Estado de la solicitud ("Obteniendo ubicación...")
 - Los datos de ubicación (latitud, longitud, precisión)
 - Mensajes de error si ocurren problemas

b. Funciones Clave

- **getLocation():** Función principal que inicia el proceso
 - Verifica si el navegador soporta geolocalización
 - Solicita los datos de ubicación al navegador
 - Maneja los casos de éxito y error
- **showPosition(position):**
 - Recibe el objeto position con los datos de ubicación
 - Muestra latitud, longitud, precisión y hora
 - Formatea los datos para una visualización clara
- **showError(error):**
 - Detecta el tipo de error (permiso denegado, timeout, etc.)
 - Muestra mensajes de error específicos y comprensibles

c. Cómo Probar

- Haz clic en el botón "Obtener Mi Ubicación"
- **Primera vez:** El navegador mostrará un aviso pidiendo permiso
 - Acepta el permiso para continuar
- **Si funciona correctamente** verás:
 - Tus coordenadas de latitud y longitud

- La precisión estimada en metros
- La hora en que se obtuvo la ubicación
- **Si hay problemas:**
 - Si deniegas el permiso, verás "Permiso denegado por el usuario"
 - Si el GPS está apagado, verás "La información de ubicación no está disponible"
 - Si tarda demasiado, verás "La solicitud tardó demasiado tiempo"

10.3 API Web Storage

La API Web Storage permite almacenar datos en el navegador del usuario de forma persistente (LocalStorage) o temporal (SessionStorage). Es más simple y potente que las cookies, con capacidad para guardar hasta 5-10MB por dominio.

10.3.1 Diferencias clave:

- LocalStorage:
 - Persiste incluso al cerrar el navegador
 - Accesible desde cualquier pestaña del mismo dominio
 - Ideal para preferencias de usuario o datos que deben mantenerse
- SessionStorage:
 - Solo dura mientras la pestaña está abierta
 - Accesible solo en la pestaña donde se guardó
 - Perfecto para datos temporales como formularios multi-paso

10.3.2 Métodos Principales

Evento	Descripción
setItem(key, value)	Almacena un dato con una clave específica
getItem(key)	Recupera el valor almacenado bajo una clave
removeItem(key)	Elimina un elemento específico del almacenamiento
clear()	Elimina todos los datos del almacenamiento

key(index)	Obtiene el nombre de la clave en la posición especificada
-------------------	---

10.3.3 Ejemplo

```

<!DOCTYPE html>
<html>
<head>
  <title>Demo Web Storage</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 600px;
      margin: 0 auto;
      padding: 20px;
    }
    .container {
      display: flex;
      gap: 20px;
      margin-bottom: 20px;
    }
    .panel {
      border: 1px solid #ddd;
      padding: 15px;
      flex: 1;
    }
    button {
      padding: 8px 12px;
      margin: 5px;
      cursor: pointer;
    }
  </style>

```

```

</head>
<body>
  <h1>Web Storage Demo</h1>

  <div>
    <input type="text" id="dataInput" placeholder="Ingresa un dato">
    <button onclick="saveData()">Guardar</button>
  </div>

  <div class="container">
    <div class="panel">
      <h3>LocalStorage</h3>
      <div id="localData"></div>
      <button onclick="clearLocal()">Limpiar</button>
    </div>

    <div class="panel">
      <h3>SessionStorage</h3>
      <div id="sessionData"></div>
      <button onclick="clearSession()">Limpiar</button>
    </div>
  </div>

  <script>
    // Mostrar datos al cargar
    updateDisplays();

    // Guardar en ambos storages
    function saveData() {
      const data = document.getElementById('dataInput').value;
      if (!data) return;

```

```

        localStorage.setItem('datoLocal', data);
        sessionStorage.setItem('datoSession', data);

        updateDisplays();
        document.getElementById('dataInput').value = "";
    }

    // Actualizar las visualizaciones
    function updateDisplays() {
        document.getElementById('localData').textContent =
            localStorage.getItem('datoLocal') || 'No hay datos';

        document.getElementById('sessionData').textContent =
            sessionStorage.getItem('datoSession') || 'No hay datos';
    }

    // Limpiar storages
    function clearLocal() {
        localStorage.clear();
        updateDisplays();
    }

    function clearSession() {
        sessionStorage.clear();
        updateDisplays();
    }
</script>
</body>
</html>

```

10.3.3.1 Explicación del Código

a. Interfaz Simple:

- Un campo de texto para ingresar datos
- Dos paneles para ver el contenido de cada storage
- Botones para guardar y limpiar

b. Funciones Clave:

- `setItem('clave', valor)`: Guarda datos
- `getItem('clave')`: Recupera datos
- `clear()`: Elimina todos los datos

c. Cómo Probar:

- Ingresa texto y haz clic en "Guardar"
- Verás el dato en ambos paneles
- Cierra y abre la pestaña: `LocalStorage` mantendrá el dato, `SessionStorage` no
- Usa los botones "Limpiar" para borrar cada storage

CONCLUSIÓN

El estudio avanzado de HTML permite no solo una comprensión profunda de su estructura y elementos, sino también la capacidad de crear experiencias digitales interactivas, accesibles y optimizadas para el usuario moderno. A lo largo de esta monografía, se ha analizado cómo el lenguaje ha evolucionado para integrar funcionalidades como formularios avanzados, gráficos SVG y Canvas, y técnicas de optimización como el SEO y la accesibilidad web.

Al dominar los elementos avanzados de HTML, los desarrolladores no solo mejoran la usabilidad y la estética de las páginas web, sino que también optimizan su rendimiento y visibilidad en motores de búsqueda, lo que resulta crucial en un entorno digital cada vez más competitivo. Las tecnologías web actuales, al permitir la creación de interfaces interactivas y dinámicas sin depender exclusivamente de tecnologías adicionales, abren nuevas posibilidades para la creación de aplicaciones web modernas.

En definitiva, la integración de estas técnicas avanzadas no solo enriquece la estructura y la funcionalidad de los documentos HTML, sino que también potencia la experiencia del usuario, haciéndola más accesible, intuitiva y fluida, independientemente del dispositivo o navegador utilizado. Esto convierte al dominio de HTML avanzado en una habilidad esencial para cualquier desarrollador web que busque estar a la vanguardia de las mejores prácticas de desarrollo web.

REFERENCIA BIBLIOGRÁFICA

- [1] EBAC, “¿Qué es HTML y para qué sirve?”, EBAC. [En línea]. Disponible en: <https://ebaonline.com/co/blog/html-que-es>
- [2] AppMaster, “¿Qué es HTML y para qué sirve?”, AppMaster. [En línea]. Disponible en: <https://appmaster.io/es/blog/que-es-html>
- [4] Mozilla Developer Network (MDN), “The <audio> HTML element.” [En línea]. Disponible en: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/audio> [Accedido: 4-may-2025].
- [5] Bitmovin, “HTML5 Video Tags - The Ultimate Guide [2024].” [En línea]. Disponible en: <https://bitmovin.com/blog/html5-video-tag-guide/> [Accedido: 4-may-2025].
- [6] DigitalOcean, “How To Add Audio Using HTML5.” [En línea]. Disponible en: <https://www.digitalocean.com/community/tutorials/how-to-add-audio-using-html5> [Accedido: 4-may-2025].
- [7] Mozilla Developer Network (MDN), “The <video> HTML element.” [En línea]. Disponible en: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video> [Accedido: 4-may-2025].
- [8] DigitalOcean, “How To Add Video Using HTML5.” [En línea]. Disponible en: <https://www.digitalocean.com/community/tutorials/how-to-add-video-using-html5> [Accedido: 4-may-2025].
- [9] W3C, "HTML5: A vocabulary and associated APIs for HTML and XHTML," W3C, 2025. [Enlace: <https://www.w3.org/TR/html5/>]
- [10] Mozilla Developer Network, "HTML5 Input Types," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input>]
- [11] W3C, "HTML5: The input element and the 'pattern' attribute," W3C, 2025. [Enlace: <https://www.w3.org/TR/html5/forms.html#text-type-inputs>]
- [12] Mozilla Developer Network, "HTML5 input types: number, range, date," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#number>]
- [13] Mozilla Developer Network, "Form validation," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form#validation>]
- [14] W3C, "HTML5: Form submission and validation," W3C, 2025. [Enlace: <https://www.w3.org/TR/html5/forms.html#novalidate>]
- [15] Mozilla Developer Network, "HTML5 input types: email," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#email>]
- [16] W3C, "HTML5: Email Input Type," W3C, 2025. [Enlace: <https://www.w3.org/TR/html5/forms.html#email-state-type-input>]
- [17] Mozilla Developer Network, "HTML5 date input," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#date>]
- [18] Mozilla Developer Network, "HTML5 input types: range," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#range>]

- [19] W3C, "HTML5 input types: number," W3C, 2025. [Enlace: <https://www.w3.org/TR/html5/forms.html#number-state-type-input>]
- [20] Mozilla Developer Network, "HTML5 input types: color, url, tel," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#color>]
- [21] W3C, "Scalable Vector Graphics (SVG)," W3C, 2021. [Enlace: <https://www.w3.org/TR/SVG2/>]
- [22] Mozilla Developer Network, "SVG: Scalable Vector Graphics," MDN, 2025. [Enlace: <https://developer.mozilla.org/en-US/docs/Web/SVG>]
- [23] World Wide Web Consortium (W3C). (2022). *Web Components*. <https://github.com/w3c/webcomponents/>
- [24] Mozilla Developer Network, "Canvas API," MDN, 2025. [Enlace: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API]
- [25] W3C, "Scalable Vector Graphics (SVG)," W3C, 2025. [Enlace: <https://www.w3.org/TR/SVG2/>]
- [26] Mozilla Developer Network, "Canvas API," MDN, 2025. [Enlace: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API]