FP-UNA_MIEK02

Tarea 2 de Sistemas de Información Web

CSS: INFORME TÉCNICO

Alumno: Antonio Rubén Resquin Castellano

Año 2025

1. Contenido

1. Introducción

CSS (*Cascading Style Sheets* o Hojas de Estilo en Cascada) es el lenguaje utilizado para describir la presentación de documentos HTML. Gracias a CSS podemos definir colores, tamaños, fuentes, márgenes, disposición de elementos y todos los aspectos visuales de una página web. Separar el contenido (HTML) de la presentación (CSS) permite mantener un código más limpio y facilitar el mantenimiento. En este informe técnico se explicarán conceptos fundamentales de CSS de forma sencilla, incluyendo cómo combinar selectores y aprovechar la herencia, cómo funciona la **cascada** y la especificidad (el orden de prioridad de las reglas CSS), una introducción a los principales *frameworks* de CSS (Bootstrap, Tailwind CSS y Foundation) con sus ventajas y desventajas, y por último se presentarán ejemplos prácticos de animaciones usando solo CSS (sin JavaScript) para ilustrar su potencial.

2. Técnicas para combinar selectores CSS y la herencia

A. COMBINAR SELECTORES CSS

Los **selectores CSS** determinan a qué elementos HTML se aplicarán ciertas reglas de estilo. Podemos combinar selectores de distintas maneras para ser muy específicos al seleccionar elementos, o para aplicar un estilo a múltiples elementos a la vez. A continuación, se describen algunas técnicas comunes para combinar selectores:

- **Selector descendiente**: se utiliza un espacio entre dos selectores para indicar una relación de ancestro-descendiente. Por ejemplo, article p selecciona todos los <p> que estén **dentro** de un <article> (no importa qué tan anidados) (Selectores CSS - Aprende desarrollo web | MDN).
- **Selector hijo directo**: utiliza el símbolo > entre dos selectores. Por ejemplo, nav > ul selecciona solo los <ul> que son hijos *directos* de un <nav> (Selectores CSS - Aprende desarrollo web | MDN) (no seleccionaría un <ul> más profundo).
- **Selectores de hermanos**: permiten seleccionar elementos que comparten el mismo padre. h1 + p selecciona el <p> inmediatamente siguiente a un <h1> (hermano adyacente), mientras que h1 ~ p selecciona *todos* los <p> que sean hermanos de un <h1> (en cualquier posición posterior) (Selectores CSS - Aprende desarrollo web | MDN).
- **Combinar clases o etiquetas**: se pueden combinar selecciones más precisas. Por ejemplo, ul.lista selecciona solo los <ul> que tengan la clase lista. Incluso es posible combinar múltiples clases: .btn.red seleccionaría elementos que tengan **ambas** clases btn *y* red a la vez.
- **Listas de selectores (selector múltiple)**: Separando selectores por coma, podemos aplicar la misma regla a distintos elementos. Por ejemplo: h1, h2, h3 { font-family: Arial; } aplicará la fuente Arial a **todos** los <h1>, <h2> y <h3> a la vez. Esto ayuda a no repetir código de estilo para cada tipo de elemento.

Ejemplo de uso de selectores combinados: supongamos que queremos estilizar los enlaces dentro de una barra de navegación sin afectar otros enlaces. Podemos combinar selectores de etiqueta y clase para lograrlo:

```
nav ul li a {
  text-decoration: none;        /* quita el subrayado de los enlaces dentro de nav */
  color: black;                 /* texto negro para esos enlaces */
}
```

En el código anterior, nav ul li a selecciona todos los <a> que estén dentro de un <li> dentro de un <ul> dentro de un <nav>. De este modo, solo los enlaces de la

barra de navegación recibirán esos estilos, sin afectar a otros enlaces en la página. Si quisiéramos solo los hijos directos, podríamos usar combinadores de hijo (>), y así sucesivamente según la relación que necesitemos especificar.

B. HERENCIA EN CSS

La **herencia** es un comportamiento por el cual algunos estilos aplicados a un elemento HTML **se propagan automáticamente** a sus elementos *hijo* (descendientes). Esto significa que ciertas propiedades CSS definidas en un elemento padre serán heredadas por sus hijos, a menos que estos hijos las sobrescriban con otro valor (Cascada y herencia - Aprende desarrollo web | MDN). Por ejemplo, si definimos un color de texto para el <body> de la página, todos los textos dentro del <body> aparecerán en ese color, salvo que algún elemento hijo tenga definido otro color propio.

No todas las propiedades son heredables; principalmente lo son aquellas relacionadas con texto y fuentes (como color, font-family, font-size, etc.), mientras que propiedades de caja y diseño (como width, margin, padding, bordes, fondo, etc.) **no** se heredan automáticamente (Cascada y herencia - Aprende desarrollo web | MDN). Veamos un ejemplo sencillo:

```
<!-- HTML de ejemplo para herencia -->
<div class="padre">Texto del padre
  <p class="hijo">Texto del hijo (hereda el color)</p>
</div>
.padre {
  color: blue; /* color azul para el texto del elemento padre */
}
.hijo {
  /* Sin especificar color, hereda el color azul del padre */
  font-weight: bold; /* esta propiedad se aplica solo al hijo */
}
```

En este caso, el tiene el texto color azul, y su elemento hijo <p class="hijo"> **hereda** ese color azul al no tener un color propio definido (Herencia en CSS - CSS en español). Sin embargo, el <p> marca su texto en **negrita** con font-weight: bold sin afectar al padre. Si agregáramos una regla CSS .hijo { color: red; }, entonces el párrafo hijo pasaría a mostrarse en rojo, ya que estaríamos sobrescribiendo el color heredado con un valor específico en el elemento hijo. En resumen, la herencia nos permite evitar repetir reglas en cada elemento hijo, ya que muchos estilos generales (colores de texto, tipografías, etc.) fluyen desde los ancestros hacia los descendientes de forma automática (Cascada y herencia - Aprende desarrollo web | MDN).

Nota: Algunas propiedades pueden forzarse a heredarse usando la palabra clave inherit. Por ejemplo, p { border: inherit; } haría que un párrafo heredase el borde de su padre (normalmente los bordes no se heredan). Sin embargo, en la

práctica común se confía en la herencia natural de las propiedades típicas y se establecen explícitamente aquellas que no se heredan.

3. Prevalencia entre selectores CSS (Especificidad y cascada)

Cuando múltiples reglas CSS pueden aplicarse a un mismo elemento, el navegador debe decidir cuál de ellas usar. Este mecanismo se rige por la **cascada** y la **especificidad** de los selectores. En términos sencillos, la **especificidad** es una medida de qué tan específico es un selector: a mayor especificidad, mayor prioridad tiene esa regla sobre otras menos específicas (Cascada y herencia - Aprende desarrollo web | MDN). La cascada indica que, en igualdad de condiciones de especificidad, **la última regla definida** en la hoja de estilos es la que prevalecerá (Cascada y herencia - Aprende desarrollo web | MDN). Veamos estos conceptos por separado:

A. ESPECIFICIDAD DE SELECTORES

Cada tipo de selector tiene un peso específico. Por regla general, los selectores de ID tienen más peso que los de clase, y estos a su vez tienen más peso que los selectores de elemento (etiquetas HTML) (Cascada y herencia - Aprende desarrollo web | MDN). Por ejemplo, un selector #menu (ID) ganará sobre .menu (clase), y .menu ganará sobre nav (etiqueta), si los tres aplican estilos al mismo elemento. El navegador calcula una *puntuación* de especificidad para cada regla; sin entrar en números exactos, el orden de prioridad de mayor a menor es:

a. **Estilos en línea** (atributo style en el HTML, si lo hubiera).

b. **Selectores de ID** (#identificador).

c. **Selectores de clase**, **atributo** y **pseudoclases** (.clase, [atributo], :hover, etc.).

d. **Selectores de tipo (etiqueta)** y **pseudoelementos** (div, p, h1, ::after, etc.).

e. **Selector universal** (*) y estilos por defecto del navegador.

Como ejemplo, imaginemos las siguientes reglas y un elemento HTML que coincide con todas ellas:

```
<h1 id="titulo" class="titulo">Título de la página</h1>
h1 { color: blue; }       /* regla de elemento */
.titulo { color: red; }   /* regla de clase */
#titulo { color: green; } /* regla de ID */
```

En este caso, las tres reglas podrían aplicarse al <h1> dado que tiene tanto la etiqueta h1, la clase titulo y el id titulo. La **regla de ID** (#titulo) es la más específica, por lo que su estilo (color verde) prevalecerá sobre los demás. La regla de clase (color rojo) queda descartada porque tiene menor especificidad

que el ID, y la de elemento (color azul) es la menos específica de las tres (Cascada y herencia - Aprende desarrollo web | MDN). El resultado final es que el título se verá en color **verde** (definido por el ID). Si quitáramos la regla de ID, entonces prevalecería la de clase (y el texto sería rojo), ya que una clase es más específica que un selector de etiqueta.

B. CASCADA (ORDEN DE LAS HOJAS DE ESTILO)

Cuando dos reglas tienen **la misma especificidad**, el desempate lo decide el orden en que aparecen. La palabra "cascada" en CSS significa que las últimas declaraciones pueden sobrescribir a las anteriores en caso de conflicto (Cascada y herencia - Aprende desarrollo web | MDN). Por ejemplo, si tenemos dos reglas exactamente igual de específicas que aplican a un elemento, la que esté definida más tarde en el archivo CSS (más "abajo") será la que se aplique. Supongamos:

```
p { color: black; }
/* ... muchas líneas después ... */
p { color: blue; }
```

Ambos selectores p tienen la misma especificidad (seleccionan cualquier párrafo). En este caso el párrafo terminaría azul, porque la segunda regla aparece después y *cascada* sobre la primera, reemplazándola. Si invertimos el orden, cambiaría el resultado en consecuencia. Por esta razón, el orden de las reglas en el CSS es importante cuando apuntan a los mismos elementos (Cascada y herencia - Aprende desarrollo web | MDN).

En resumen, para saber qué estilo se aplica finalmente a un elemento, debemos considerar primero la especificidad de los selectores involucrados y luego el orden en que aparecen. Además, ¡existe la directiva !important que puede usarse junto a una declaración CSS para forzar que tenga la máxima prioridad, ignorando especificidad y orden normales. Sin embargo, **¡usar !important no es recomendable** salvo casos excepcionales, ya que dificulta mantener la cascada ordenada y puede causar conflictos difíciles de depurar.

4. *Frameworks* de CSS (Bootstrap, Tailwind CSS, Foundation)

Un **framework de CSS** es un conjunto de herramientas (principalmente hojas de estilo CSS ya predefinidas, a veces junto con componentes HTML/JS) que facilitan y agilizan el desarrollo de sitios web con diseños consistentes. En lugar de escribir todos los estilos desde cero, un framework nos proporciona clases ya listas para usar que aplican estilos comunes (por ejemplo, clases para crear un diseño de columnas, botones con cierto aspecto, barras de navegación responsivas, etc.). A continuación, se presenta una visión general de tres de los frameworks de CSS más populares, junto con sus ventajas y desventajas:

A. BOOTSTRAP

Bootstrap es uno de los frameworks CSS más populares y fue originalmente desarrollado por Twitter en 2011. Se destaca por su sistema de rejilla (*grid*) responsivo y una amplia variedad de componentes ya estilizados listos para usar (navegaciones, formularios, botones, modales, etc.). Incluye también código JavaScript para componentes interactivos.

Ventajas:

- Compatibilidad amplia con navegadores: Bootstrap funciona correctamente en todos los navegadores modernos (Los 5 mejores frameworks de CSS para tus proyectos).
- Gran comunidad y documentación: al ser tan popular, cuenta con mucha documentación, ejemplos y foros de ayuda mantenidos por su comunidad (Los 5 mejores frameworks de CSS para tus proyectos).
- Muchos componentes predefinidos: ofrece componentes de interfaz (menús, tarjetas, carruseles, etc.) y utilidades listas para usar, lo que acelera el desarrollo (Los 5 mejores frameworks de CSS para tus proyectos).
- Facilidad de uso: es relativamente sencillo comenzar a usarlo; con solo agregar las clases adecuadas a los elementos HTML, ya obtenemos un diseño atractivo sin escribir CSS desde cero (Los 5 mejores frameworks de CSS para tus proyectos).

Desventajas:

- Personalización limitada: no es sencillo sobreescribir o modificar profundamente los estilos por defecto de Bootstrap, por lo que a veces todos los sitios hechos con este framework lucen similares (Los 5 mejores frameworks de CSS para tus proyectos).
- Tamaño: incluir Bootstrap añade un archivo CSS grande al proyecto, lo que puede impactar el rendimiento si la página carga estilos que quizás no use (Los 5 mejores frameworks de CSS para tus proyectos).
- Dependencia de su estructura: para aprovecharlo al máximo, debes seguir su estructura HTML y clases; alejarse de ellas implica perder sus beneficios o requerir más trabajo de sobrescritura de CSS.

B. TAILWIND CSS

Tailwind CSS es un framework moderno que sigue un enfoque diferente llamado "*utility-first*". A diferencia de Bootstrap, Tailwind no proporciona componentes pre-estilizados, sino una serie de **clases utilitarias** muy atómicas que representan estilos individuales (por ejemplo, una clase para un margen específico, otra para

color de fondo, etc.). El desarrollador *compone* sus propios diseños combinando estas clases en el HTML. Tailwind escanea el HTML y genera un CSS final que solo contiene las clases utilizadas, lo que lo hace altamente eficiente.

Ventajas:

- Alta productividad y velocidad: al tener clases utilitarias para casi todo, es rápido construir diseños en el HTML sin tener que escribir CSS; esto puede acelerar mucho el desarrollo (Los 5 mejores frameworks de CSS para tus proyectos).
- CSS optimizado y ligero: Tailwind genera archivos muy pequeños porque elimina (purga) cualquier clase no utilizada, resultando en un CSS final mínimo (Los 5 mejores frameworks de CSS para tus proyectos).
- Extremadamente personalizable: se puede configurar prácticamente todo (colores, tamaños, espaciamientos, etc.) mediante un archivo de configuración, adaptándose a las necesidades de diseño del desarrollador (Los 5 mejores frameworks de CSS para tus proyectos).
- Reutilización de estilos: fomenta la consistencia, ya que en lugar de escribir estilos variados, se reutilizan las mismas clases utilitarias en todo el proyecto, manteniendo un estilo uniforme (Los 5 mejores frameworks de CSS para tus proyectos).

Desventajas:

- Curva de aprendizaje pronunciada: adoptar la mentalidad utility-first puede ser desafiante para principiantes; hay que familiarizarse con muchas clases abreviadas y conceptos nuevos, lo cual toma tiempo (Los 5 mejores frameworks de CSS para tus proyectos).
- Configuración previa necesaria: Tailwind no funciona simplemente enlazando un archivo CSS como Bootstrap. Requiere un proceso de *build* (por ejemplo, con Node.js) para generar el CSS a medida. Esto agrega un paso de configuración que puede ser complejo para proyectos pequeños o para quienes no están familiarizados (Los 5 mejores frameworks de CSS para tus proyectos).
- HTML recargado de clases: aunque es parte de su filosofía, el HTML puede verse saturado de clases utilitarias, lo que a algunos desarrolladores les parece menos limpio y hace más difícil leer el marcado (esta es una crítica común, aunque las herramientas y buenas prácticas pueden mitigar este problema).

C. FOUNDATION

Foundation es un framework CSS creado por Zurb, contemporáneo a Bootstrap (ambos surgieron a inicios de la década de 2010). Se promociona como un framework orientado a profesionales, altamente personalizable y modular. Al igual que Bootstrap, ofrece un sistema de grid responsivo y varios componentes de interfaz, pero con un enfoque más minimalista en estilos por defecto, pensado para ser un "esqueleto" flexible más que un producto terminado.

Ventajas:

- Fácil de usar y semanticidad: Foundation está diseñado para ser claro y legible, empleando HTML semántico. Es relativamente sencillo comenzar a utilizarlo y entender sus clases, y cuenta con una comunidad y voluntarios que mantienen el proyecto (Los 5 mejores frameworks de CSS para tus proyectos).
- Flexible y modular: se puede incluir únicamente los módulos que se necesitan (tipografía, grid, forms, etc.), lo que permite reducir peso. Sus estilos por defecto no son intrusivos, facilitando la personalización sobre ellos (Los 5 mejores frameworks de CSS para tus proyectos).
- Herramientas y extensibilidad: incluye su propia CLI (línea de comandos) para inicializar proyectos rápidamente, y ofrece complementos JavaScript opcionales (por ejemplo, tooltips, modales, carruseles) que se integran fácilmente si se requieren (Los 5 mejores frameworks de CSS para tus proyectos). En general, brinda un control completo al desarrollador de la UI, sin imponer un estilo visual específico.

Desventajas:

- Menor comunidad: aunque Foundation es utilizado en muchos sitios, su comunidad es más pequeña en comparación con Bootstrap. Por ello, puede haber menos recursos o ejemplos disponibles, y resolver dudas puede ser más difícil si no hay tanta gente usándolo (¿Cuáles son los frameworks CSS más populares 2024? - Bambu Mobile) (¿Cuáles son los frameworks CSS más populares 2024? - Bambu Mobile).
- Complejidad y aprendizaje: su gran flexibilidad trae consigo muchas opciones de configuración, lo que puede abrumar a usuarios novatos. Aprender a aprovechar todo su potencial requiere tiempo, y su curva de aprendizaje es más empinada en comparación con frameworks más restrictivos (Los 5 mejores frameworks de CSS para tus proyectos).
- Menor estandarización: debido a que Foundation permite más libertad y no impone tantos estilos por defecto, es posible que diferentes proyectos usándolo no se parezcan entre sí. Si bien esto no es intrínsecamente malo, significa que los desarrolladores no tienen un conjunto tan definido

de componentes visuales inmediatamente reconocibles como ocurre con Bootstrap, y podría tomar más trabajo lograr ciertos componentes desde cero.

Nota: Además de estos tres, existen otros frameworks CSS populares como **Bulma**, **Materialize**, **Semantic UI**, entre otros, cada uno con sus particularidades. La elección del framework depende de las necesidades del proyecto y preferencias del desarrollador. También es válido no usar ningún framework y escribir CSS puro, especialmente si el diseño es muy personalizado o si se quiere tener un control total sobre el código CSS generado.

5. Ejemplos de animaciones con CSS (sin JavaScript)

CSS permite crear animaciones y transiciones visuales sin necesidad de JavaScript, lo que simplifica lograr efectos interactivos básicos de forma eficiente. A continuación, se presentan tres ejemplos prácticos de animaciones hechas con CSS puro:

Ejemplo 1: Botón que cambia de color al pasar el cursor

En este ejemplo, crearemos un botón que al colocar el cursor encima (estado hover) cambie suavemente de color. Para ello aprovecharemos la pseudoclase :hover de CSS y la propiedad transition para animar el cambio de color.

HTML: definimos un botón con una clase para poder estilizarlo.

<button class="boton">¡Haz clic aquí!</button>

CSS: aplicamos estilos base al botón y definimos el estado hover con un color de fondo distinto. La propiedad transition logrará que el cambio sea gradual.

```
.boton {
  background-color: #3498db; /* color de fondo azul inicial */
  color: white;            /* texto blanco */
  padding: 10px 20px;      /* espacio interno (padding) */
  border: none;            /* sin borde */
  border-radius: 5px;      /* bordes redondeados */
  transition: background-color 0.3s ease; /* transición suave de color
en 0.3s */
}

.boton:hover {
  background-color: #2ecc71; /* color de fondo verde al pasar el
cursor */
}
```

Explicación: inicialmente el botón tiene fondo azul. La regla .boton:hover indica que cuando el usuario coloque el mouse sobre el botón, su fondo cambiará a verde. Gracias al transition, ese cambio de azul a verde tomará 0.3 segundos con una transición suave (ease). Si quitáramos transition, el color cambiaría

instantáneamente. Este efecto mejora la experiencia visual dando feedback inmediato al usuario cuando interactúa con el botón. Cabe destacar que todo esto se logra solo con CSS: la pseudoclase :hover es suficiente para detectar la interacción, sin necesidad de JavaScript.

Ejemplo 2: Cuadro que rebota

Ahora realizaremos una animación continua de un elemento, haciendo que una caja "rebote" verticalmente. Utilizaremos para ello las **@keyframes** de CSS, que permiten definir una secuencia de cambios de estilo, y la propiedad animation para aplicarla al elemento.

HTML: un elemento sencillo que representará la caja:

CSS: definimos la animación con @keyframes y luego la aplicamos a la clase del elemento.

```
.caja {
  width: 50px;
  height: 50px;
  background-color: #e74c3c;          /* color rojo para la caja */
  border-radius: 5px;                 /* bordes ligeramente redondeados
*/
  animation: rebotar 0.5s infinite alternate ease-in-out;
  /* "rebotar" es el nombre de la animación definida abajo.
     0.5s es la duración de cada ciclo.
     infinite indica que se repite sin fin.
     alternate hace que la animación vaya y vuelva (rebote).
     ease-in-out suaviza el movimiento. */
}

@keyframes rebotar {
  from { transform: translateY(0); }        /* empieza en la posición
original */
  to   { transform: translateY(-100px); } /* se desplaza 100px hacia
arriba */
}
```

Explicación: aquí hemos creado una animación llamada rebotar. La regla @keyframes define que durante la animación el elemento cambiará su posición vertical desde translateY(0) (posición inicial) hasta translateY(-100px) (100 píxeles hacia arriba). Al aplicar animation: rebotar 0.5s infinite alternate, la caja .caja ejecutará esa animación continuamente: en medio segundo irá de 0px a -100px, luego gracias a alternate invertirá la dirección y volverá de -100px a 0px en el siguiente medio segundo, y así sucesivamente de forma infinita. El resultado es un efecto de **rebote** vertical constante. La curva de animación ease-in-out hace que la velocidad sea más lenta al inicio y al final de cada medio ciclo, imitando un movimiento un poco más natural (acelerando y desacelerando).

Podemos ajustar la duración (0.5s), la distancia (-100px) o incluso añadir más puntos intermedios en @keyframes (por ejemplo un 50% con la caja más abajo para simular gravedad) para refinar el efecto, pero incluso con esta configuración básica logramos el rebote sin una sola línea de JavaScript.

Ejemplo 3: Texto que se desvanece

En el último ejemplo, mostraremos un texto que se desvanece gradualmente hasta desaparecer. Usaremos nuevamente @keyframes para animar la opacidad del texto de 1 a 0, logrando un efecto de desvanecimiento.

HTML: un párrafo de texto al que aplicaremos la animación:

Este texto se desvanece lentamente.

CSS: definimos la animación de desvanecer y la asociamos a la clase del párrafo.

```
.texto {
  animation: desvanecer 3s forwards;
  /* "forwards" mantiene el estado final de la animación (opacidad
0)
     después de completarse, evitando que el texto reaparezca */
}

@keyframes desvanecer {
  from { opacity: 1; }   /* opacidad inicial 100% (visible) */
  to   { opacity: 0; }   /* opacidad final 0% (transparente) */
}
```

Explicación: la animación desvanecer simplemente reduce la propiedad opacity de 1 a 0 en un lapso de 3 segundos. Al aplicar animation: desvanecer 3s forwards al elemento .texto, le indicamos que ejecute esa animación una vez durante 3 segundos. La palabra clave forwards asegura que al terminar la animación el elemento conserve la opacidad final (0), es decir, quede invisible. Sin forwards, el elemento volvería a su estado inicial (opacidad 1) inmediatamente después de completar la animación, lo que en este caso haría que el texto "parpadee" volviendo a aparecer. Con esta técnica podemos crear efectos de aparición/desaparición suaves. Por ejemplo, podríamos hacer que un elemento aparezca gradualmente usando desde opacity: 0 hasta opacity: 1, o incluso combinar con transforms para hacerlo deslizar mientras se desvanece, etc. Todo sin necesidad de JavaScript, aprovechando las capacidades de CSS moderno.

6. Conclusión

A lo largo de este informe hemos cubierto varios fundamentos de CSS de forma amigable para principiantes. Aprendimos a seleccionar elementos HTML con precisión combinando selectores y cómo la herencia puede propagar estilos automáticamente, ahorrando trabajo redundante. También discutimos cómo el navegador decide qué estilos aplicar cuando hay múltiples reglas (especificidad y orden en la cascada), evitando confusiones al depurar estilos. Exploramos algunas herramientas externas (frameworks CSS) que, si bien no son obligatorias, pueden acelerar el desarrollo y ofrecer soluciones ya probadas para diseños comunes, analizando sus pros y contras. Finalmente, vimos el poder de CSS para lograr interactividad y dinamismo visual mediante animaciones puras, desde simples transiciones al hover hasta movimientos continuos y efectos de desvanecimiento. Con este conocimiento, un desarrollador principiante debería sentirse más cómodo entendiendo y escribiendo CSS, y estará mejor preparado para profundizar en cada uno de estos temas a medida que construye sitios web más complejos.

7. Referencias:

(1). MDN Web Docs. (2024). Conceptos de cascada, especificidad y herencia en CSS. Recuperado de **https://developer.mozilla.org/es/docs/Web/CSS**

(2). Bootstrap. (2024). Documentación oficial de Bootstrap 5. Recuperado de https://getbootstrap.com/

Tailwind CSS. (2024). Guía de inicio rápido. Recuperado de **https://tailwindcss.com/docs**

(3). Zurb Foundation. (2024). Introducción a Foundation Framework. Recuperado de **https://get.foundation/docs/**

(4). W3Schools. (2024). CSS Animations - W3Schools. Recuperado de **https://www.w3schools.com/css/css3_animations.asp**

(5). CSS Tricks. (2024). Understanding CSS specificity. Recuperado de https://css-tricks.com/specifics-on-css-specificity/